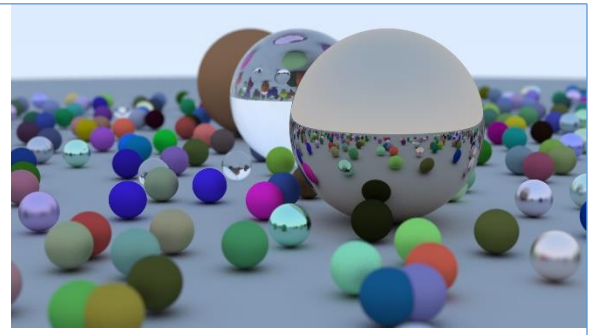
Comp4422



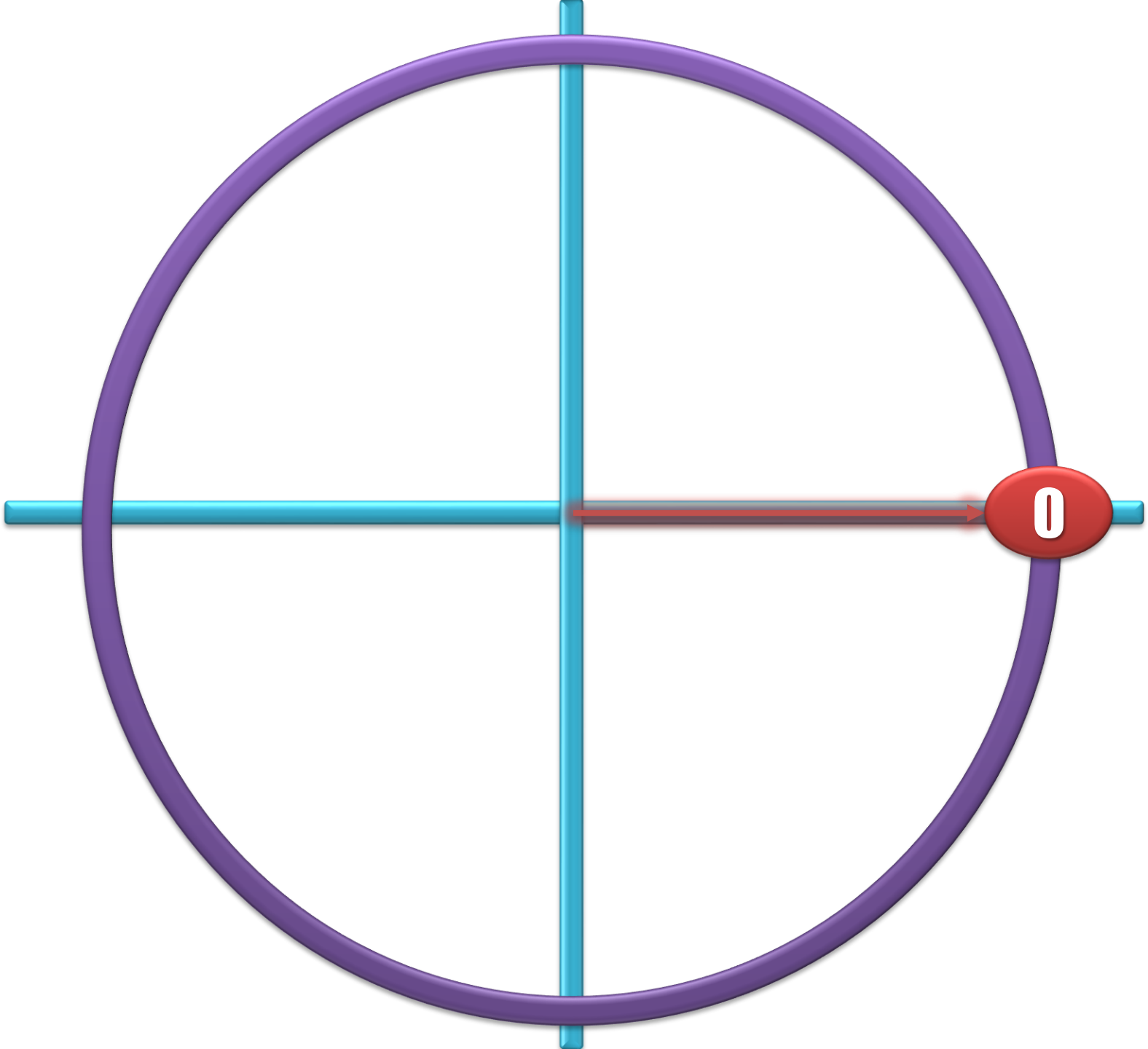
Computer Graphics

Lecture 08:

Shading

What you will learn…

- What a shader needs
- Normal Vectors
- Flat Shading
- Gouraud Shading
- Phong Shading

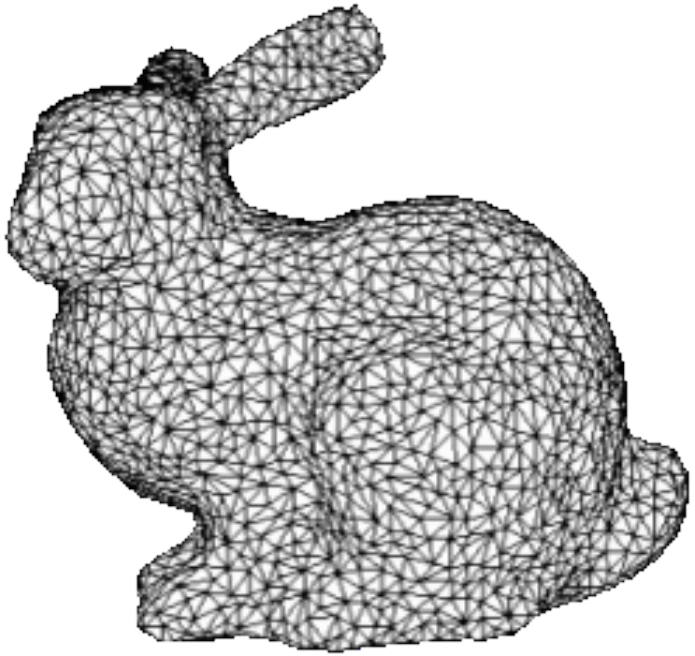

0

What a shader needs

1. The Model

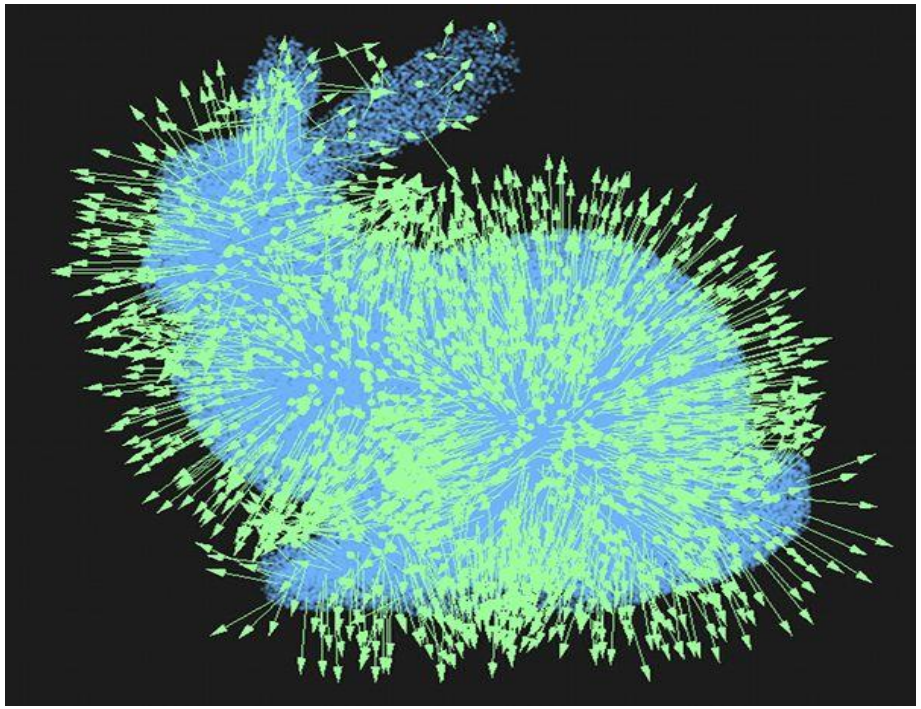
- Everything starts with a **model** of a real or abstract object

2. The Surface Mesh



- Then, we discretize the surface into a **mesh** of polygons (mostly triangles)

3. Normal Vectors



- Then, we must compute the **normal vectors** and store them somewhere

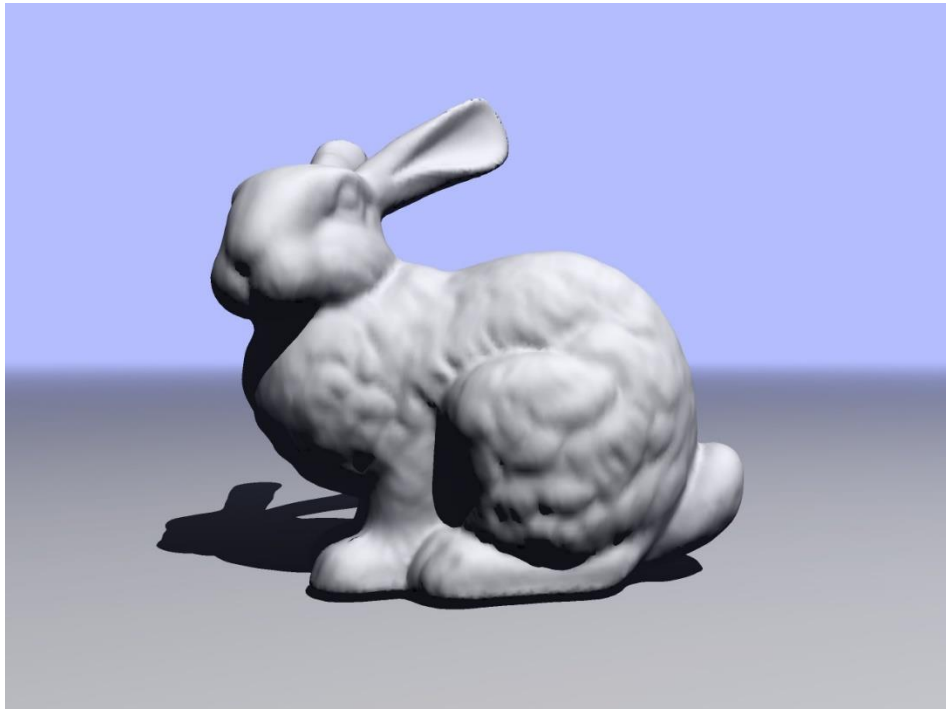
4. Materials

- Then, we add the **material spectral** properties.

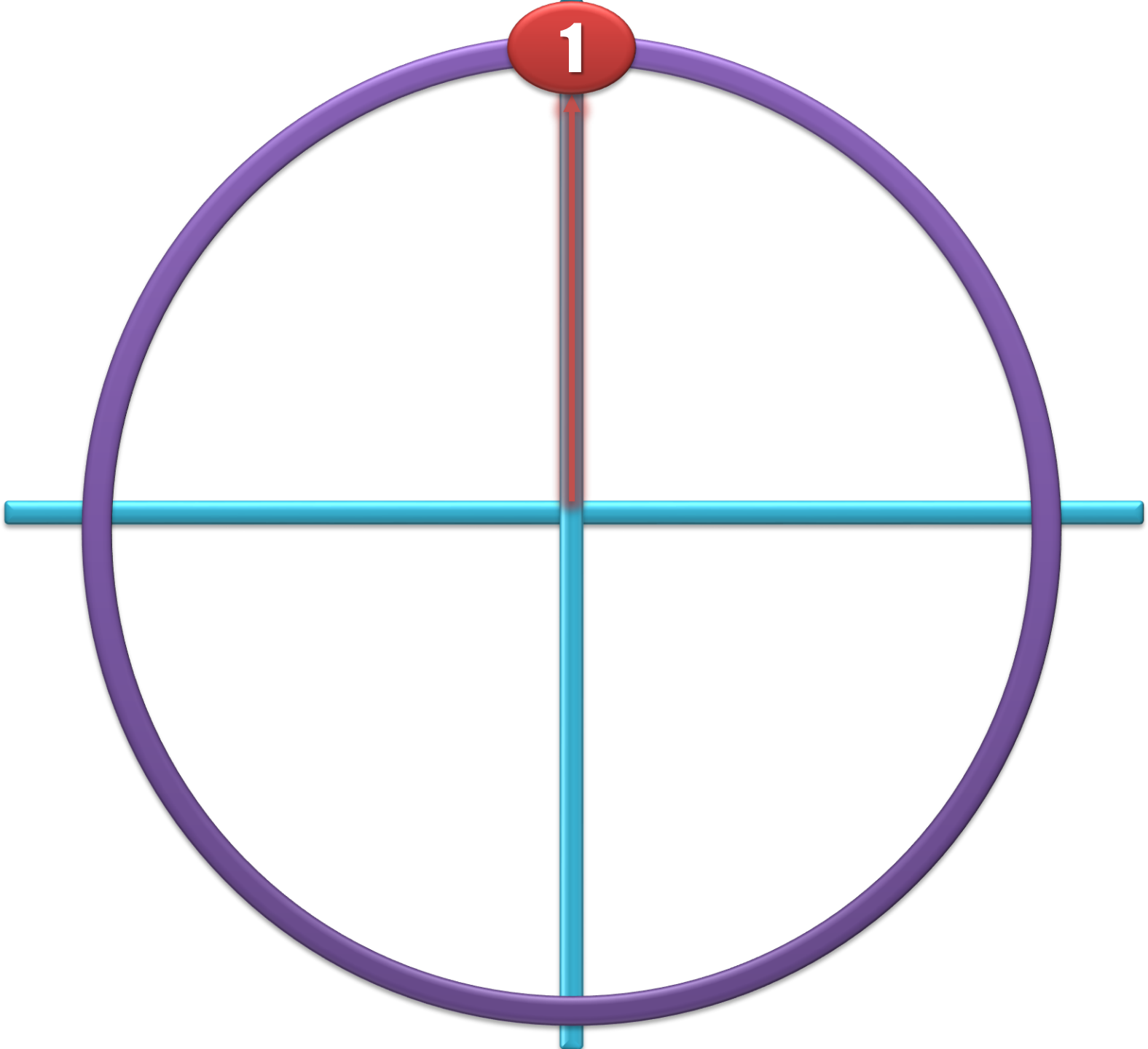
5. Lights

- Then, we add the **lights.**

6. Final Shading



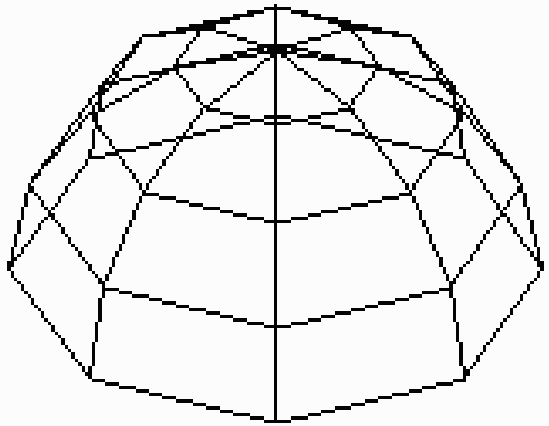
- Then, we **render** or **shade** it.


1

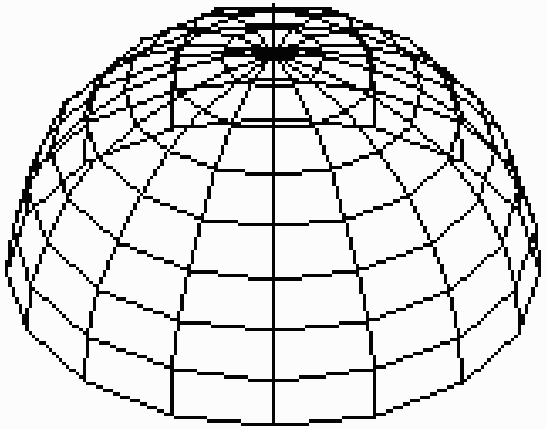
The Mesh

The Mesh

- Represents the geometry of the object
- A mesh can be:
  - Coarse
  - Dense



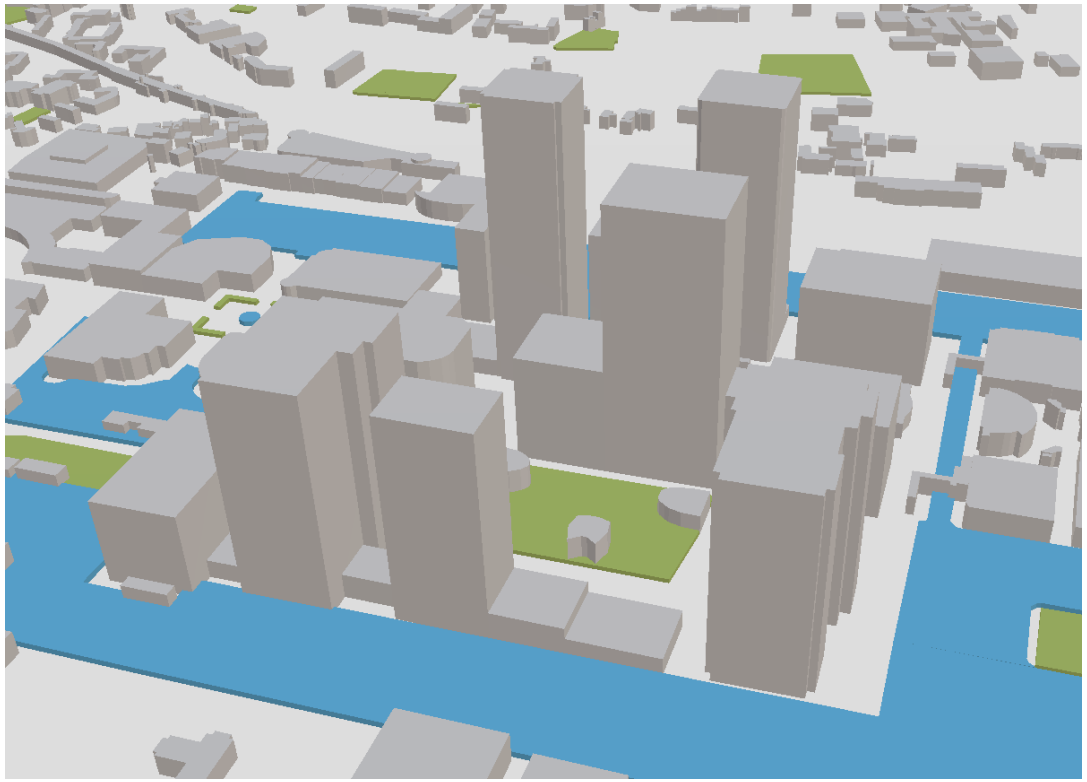
Coarse Mesh

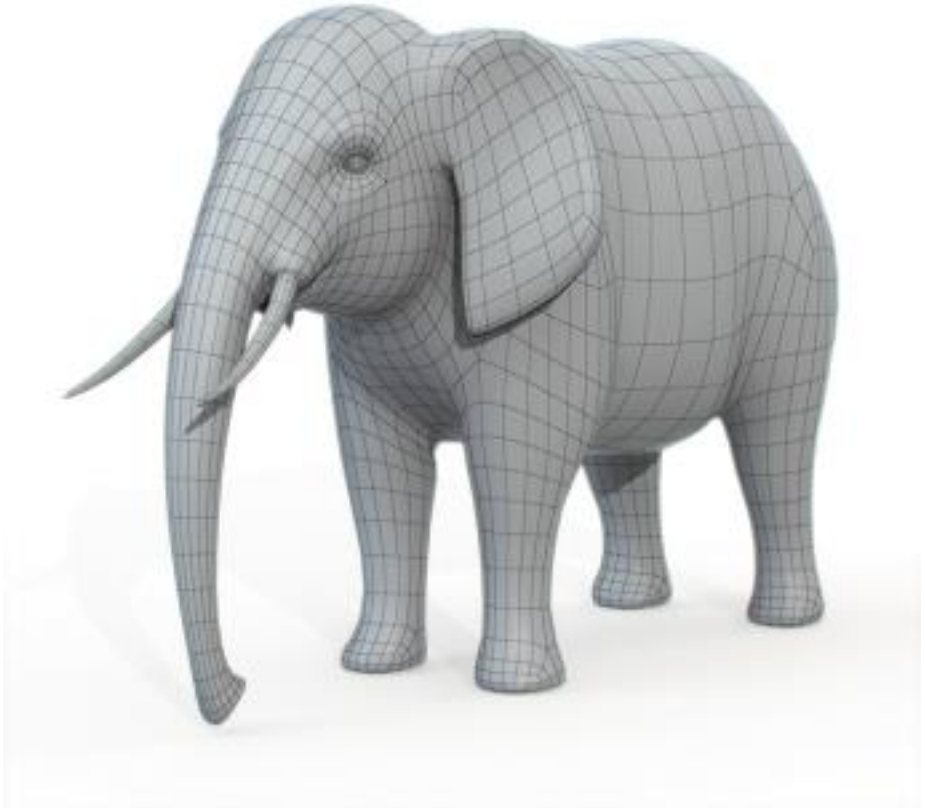


Dense Mesh

A Coarse Mesh is for…

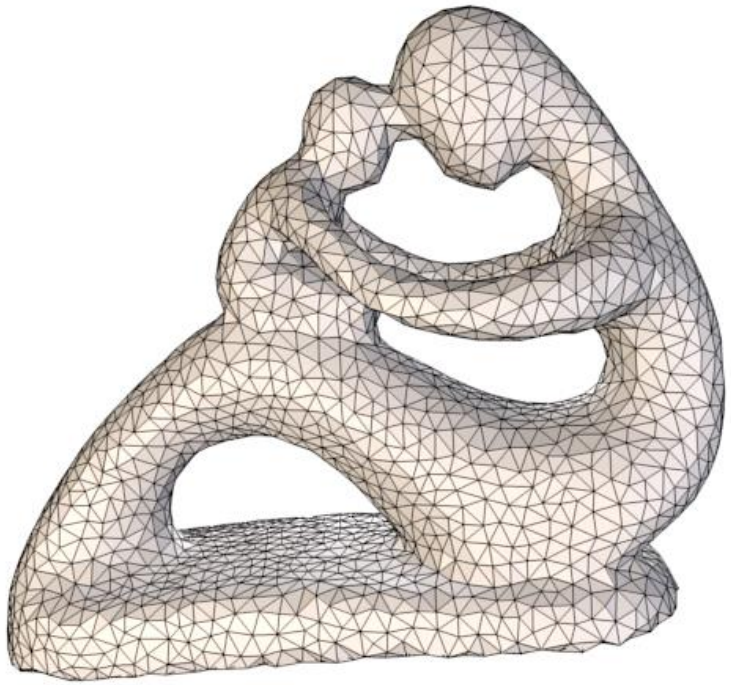
- Objects with low uniform curvature

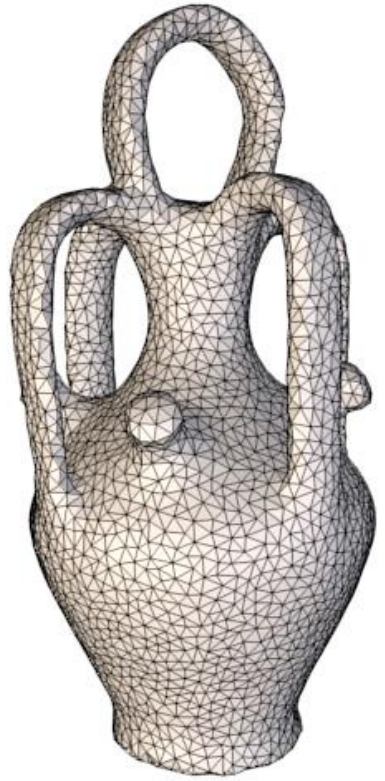




A Dense Mesh is for…

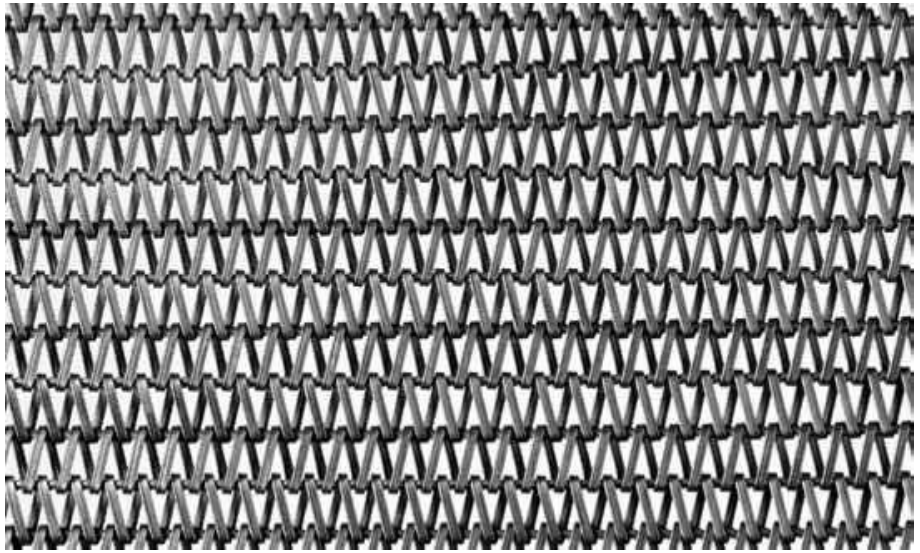
- Objects with high non-uniform curvature





A Dense Mesh is for…

- Deformable objects or rough surfaces

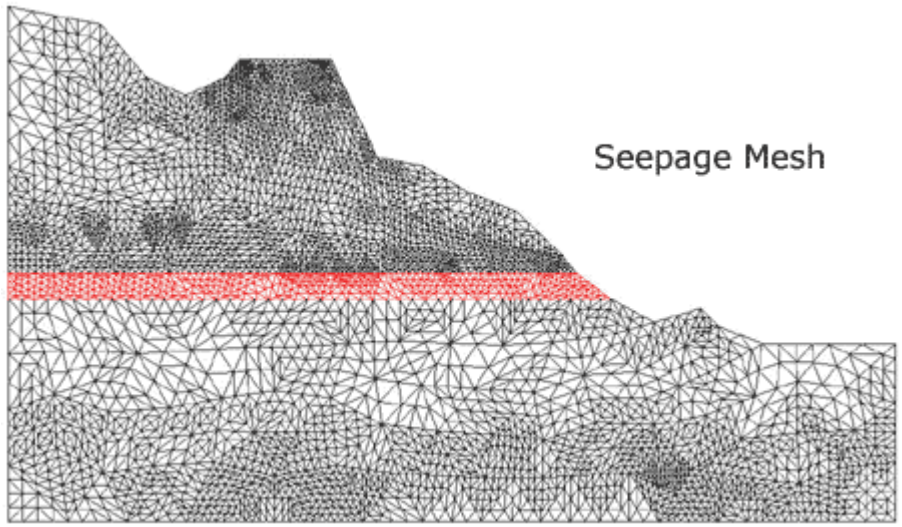


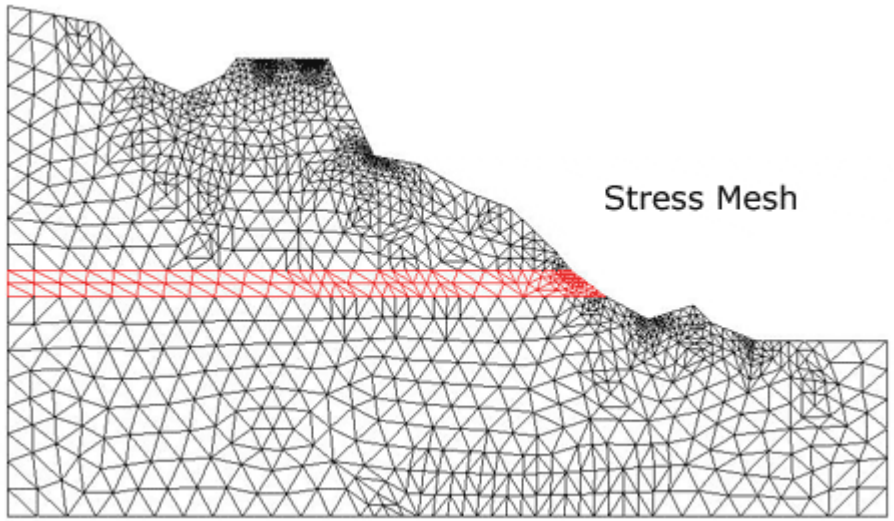
Meshing is important

In Large Models

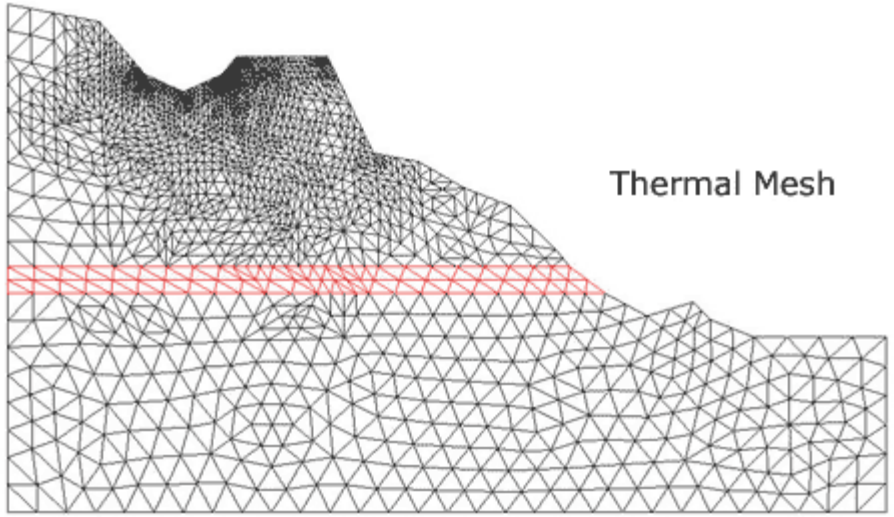
Meshes is important

In material analysis





In structural analysis

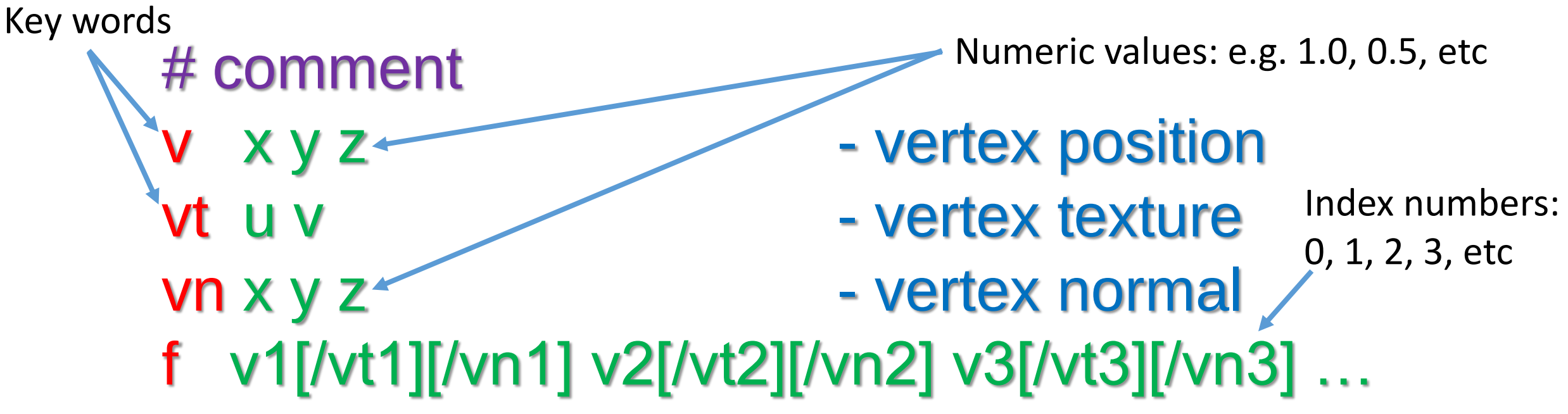


Storing a mesh .obj files

- Most commonly used files for storing and preserving mesh models: mesh**.obj**
- ASCII format (.obj)
- Binary format (.mod)
- Originally developed by Wavefront Tech for their Advanced Visualizer package

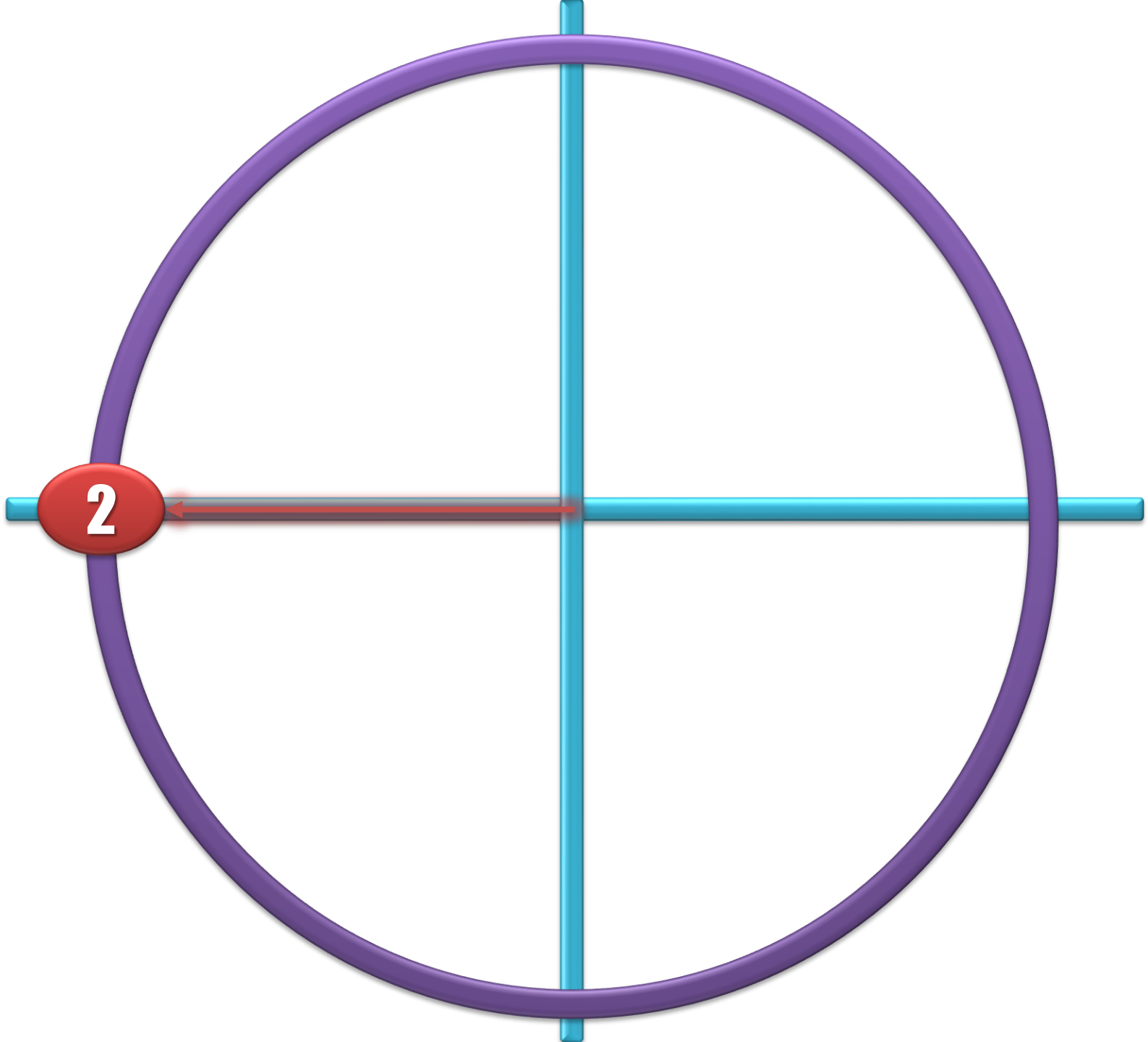
Storing a mesh: .obj files

- The basic types of a .obj file:



Materials

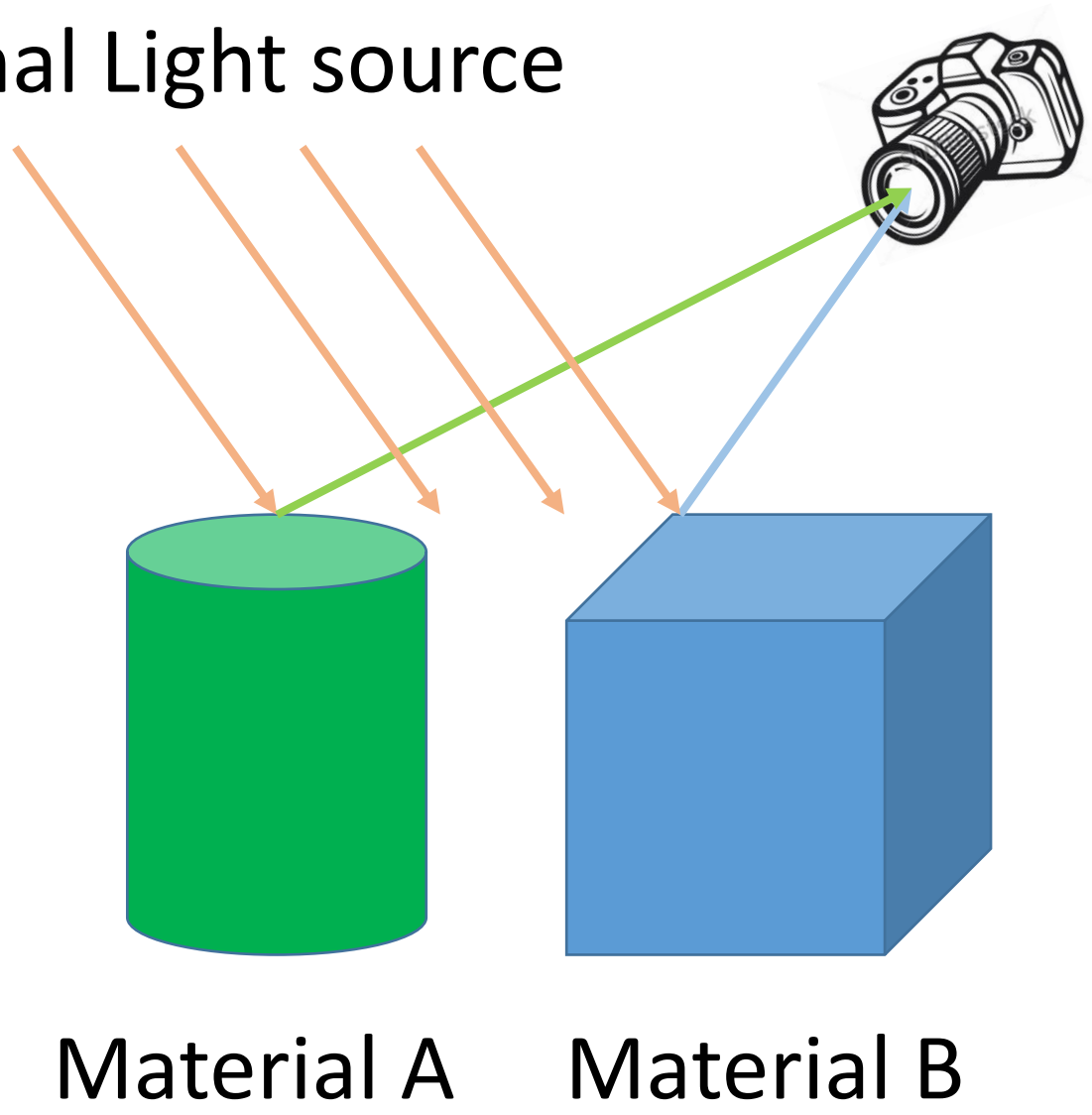
- In rendering are interested in the spectral qualities of materials; this means their color, and spectral reflections
- Model as triplets of RGB: constants k_d k_s
- Texture can be added for realism


2

Lights

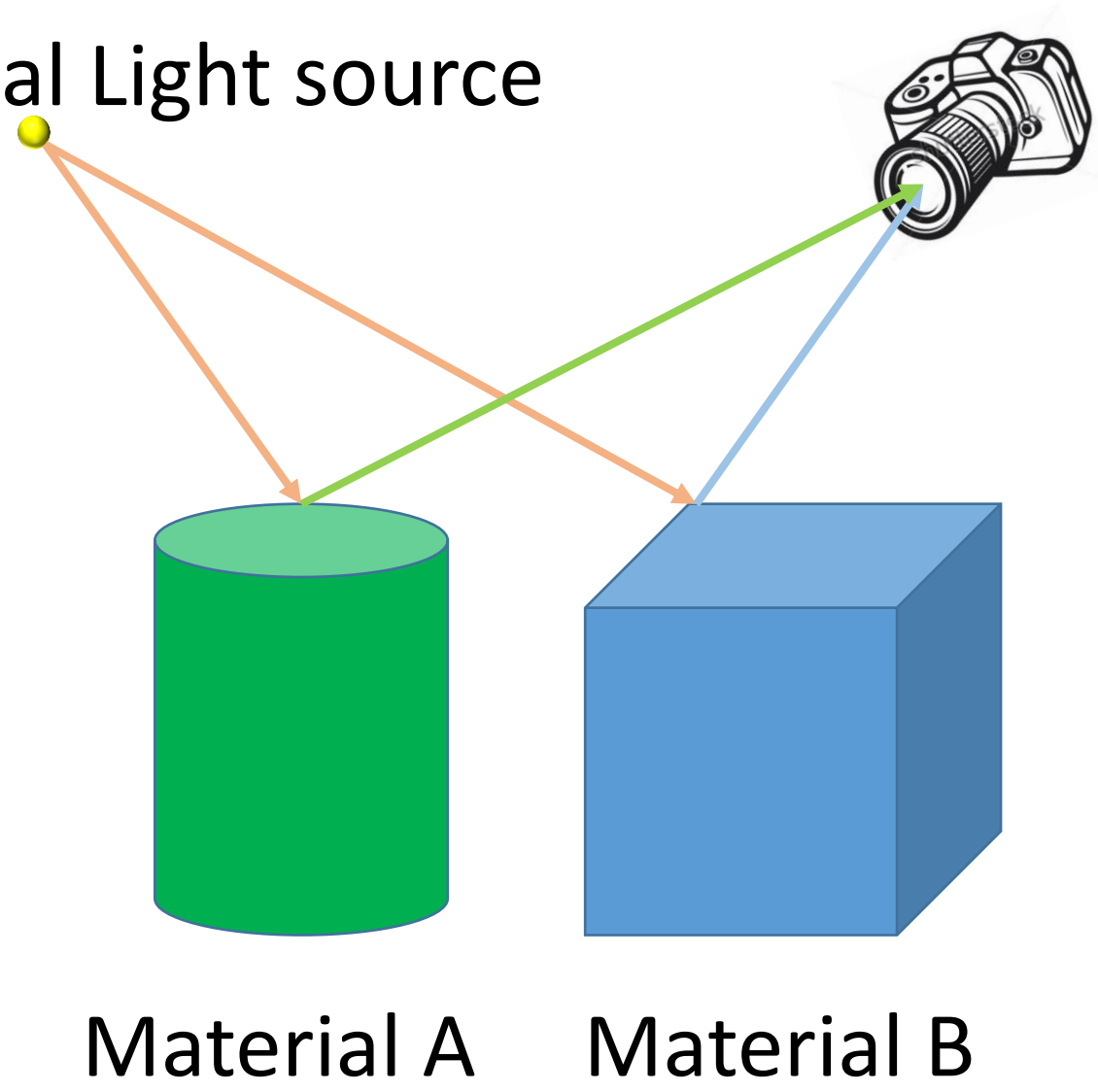
Lights

- Light sources can be
  - Directional
  - Positional
  - Areal



Lights

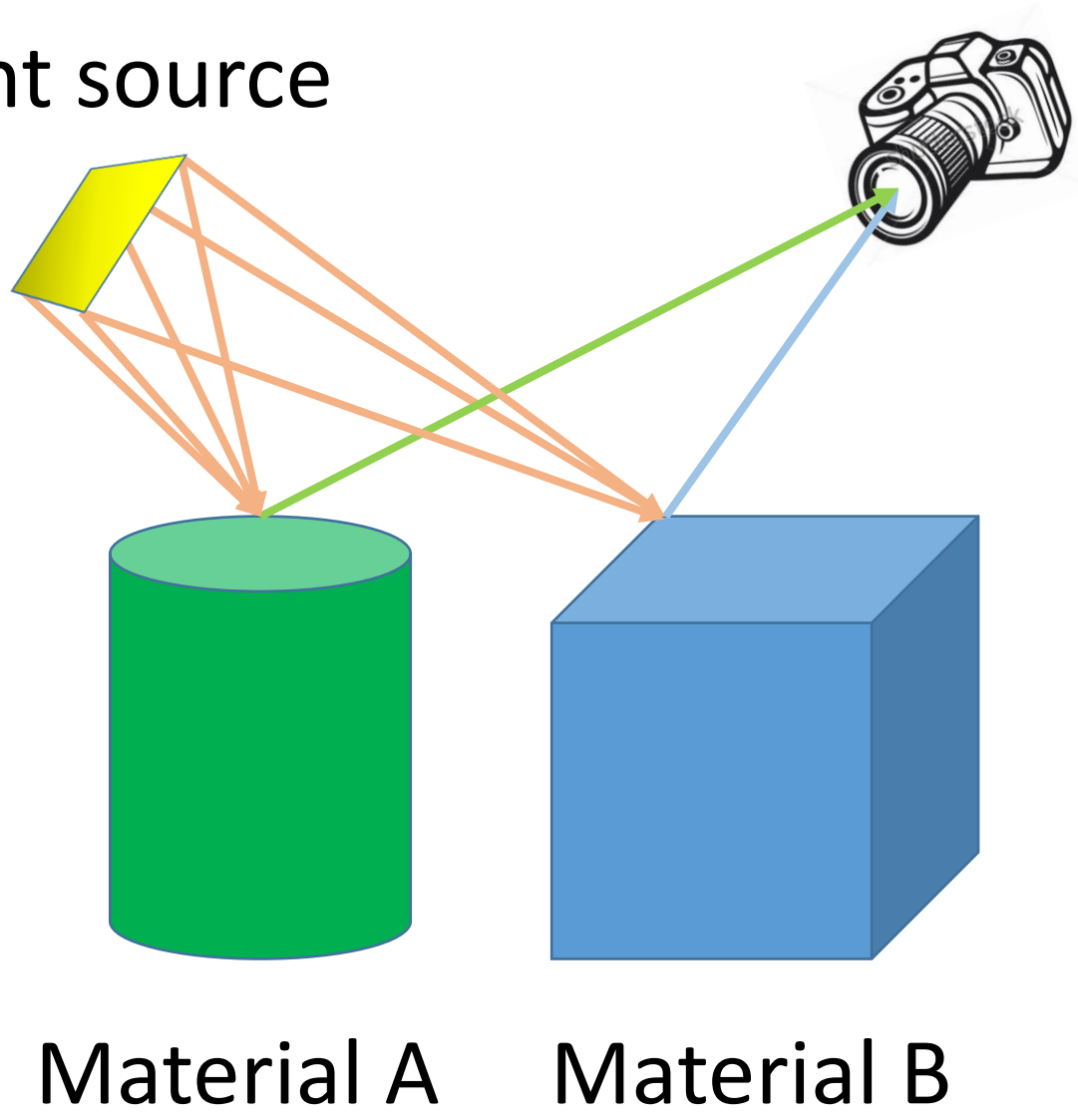
- Light sources can be
  - Directional
  - Positional
  - Areal

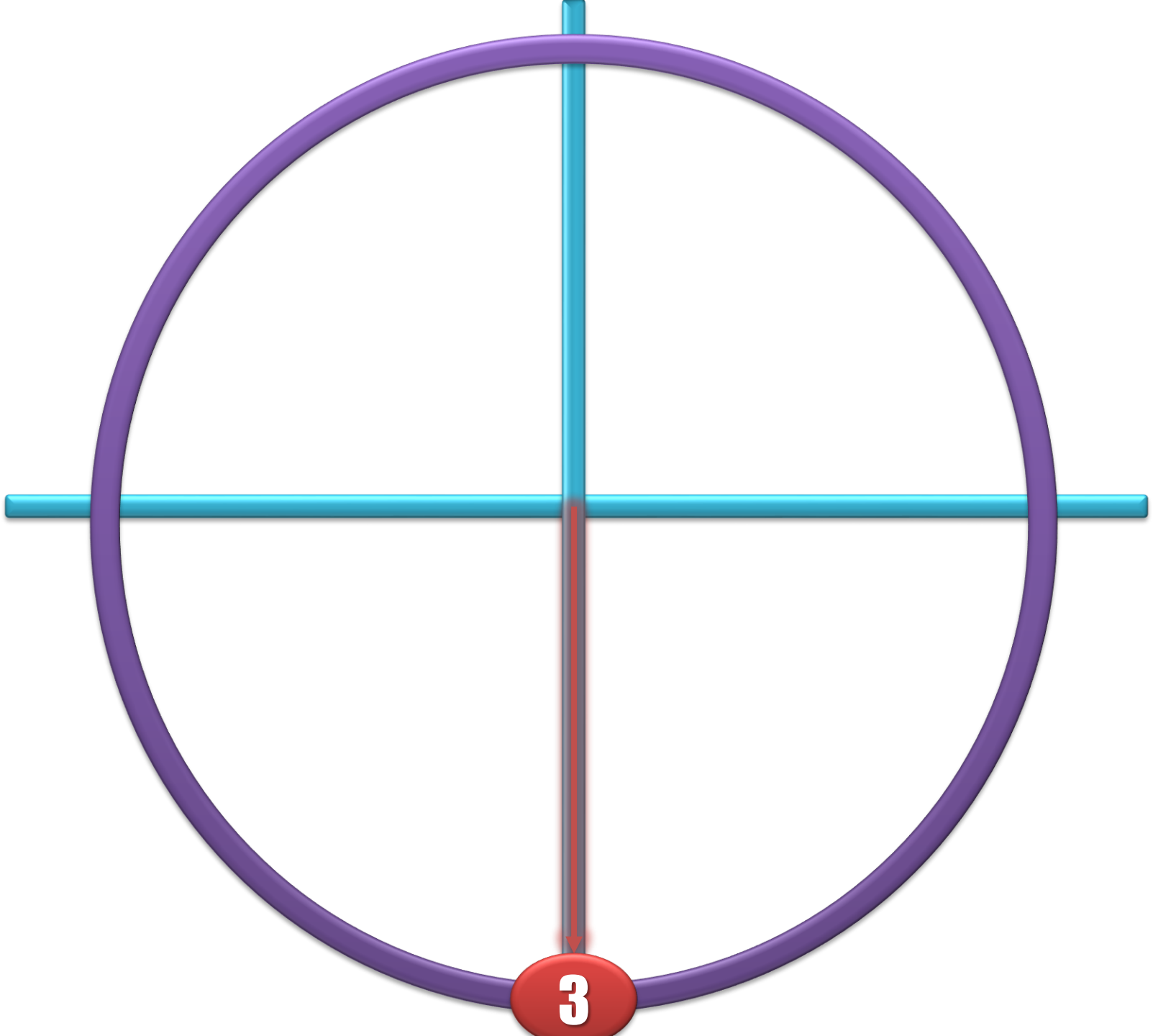


Lights

- Light sources can be
  - Directional
  - Positional
  - Areal

Areal Light source

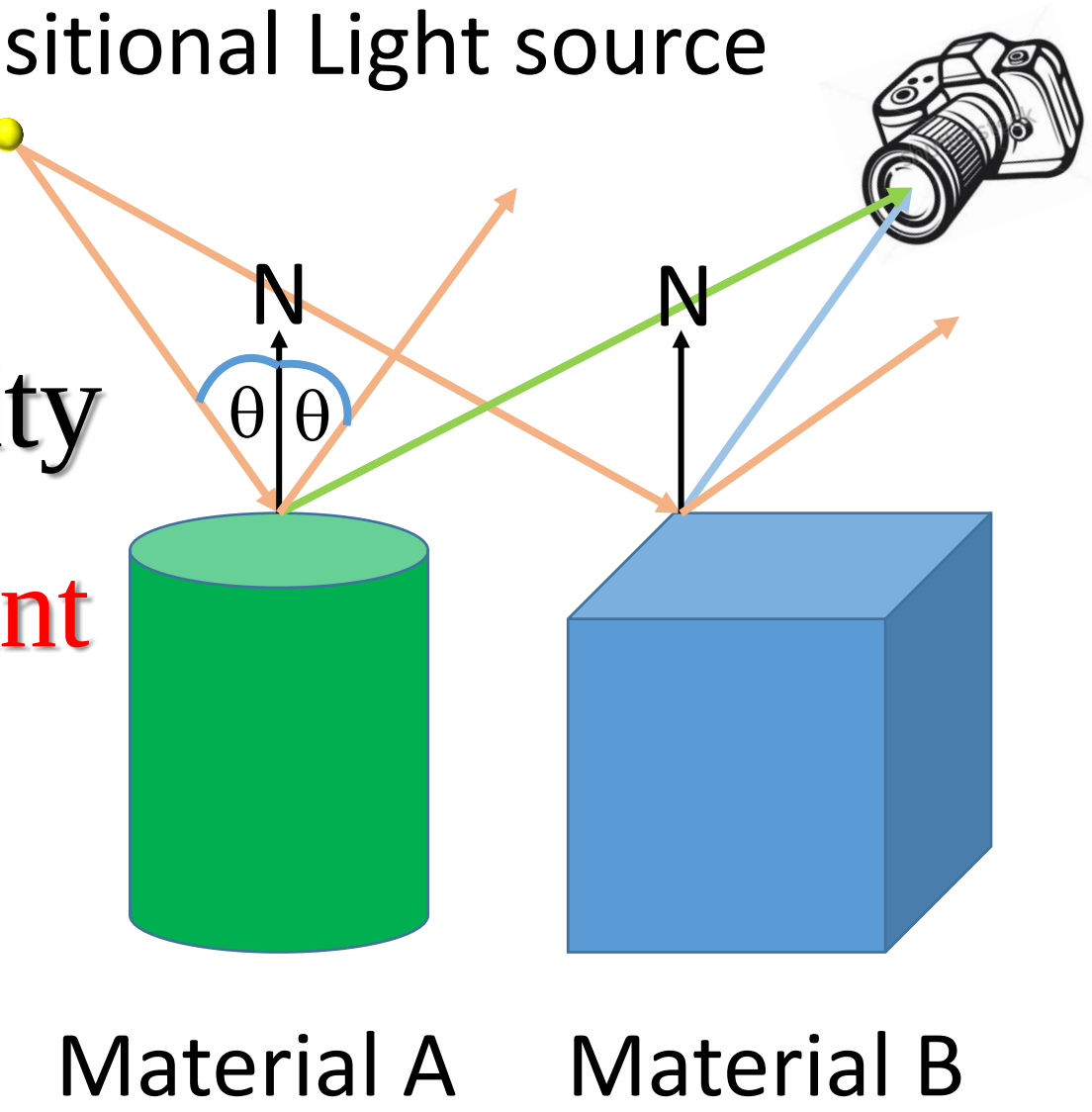



3

Normal Vectors

Normal Vector

- Most important quantity
  - Normal vector at a point



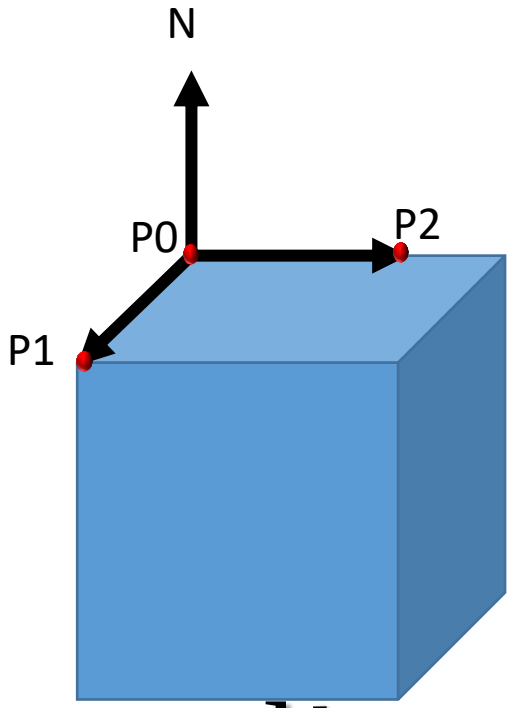
Normals

- Normals are vectors perpendicular to the surface that we want to illuminate
- Normals represent the orientation of the surface
- Normals are critical to model the light-object interaction

Calculating Normals

- Use the vector cross product

$$\mathbf{V1} = \mathbf{P1} - \mathbf{P0}$$
$$\mathbf{V2} = \mathbf{P2} - \mathbf{P0}$$
$$\mathbf{n} = \mathbf{V1} \times \mathbf{V2}$$

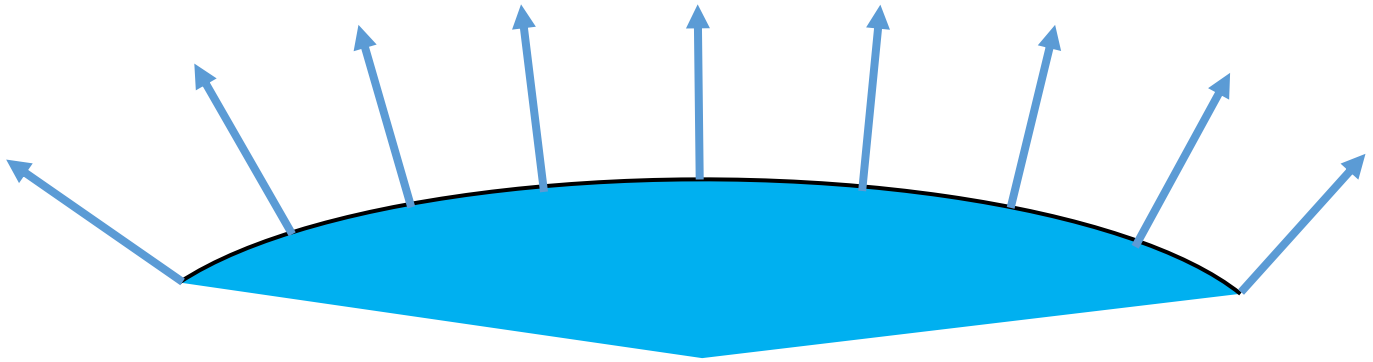


- We also need to normalize the normal!

$$\mathbf{n} = \sqrt{(n_0)^2 + (n_1)^2 + (n_2)^2} \qquad \mathbf{N} = \frac{\mathbf{n}}{||\mathbf{n}||}$$

How many normals?

- Surfaces have a normal vector at each point



- So, how many normal vectors?

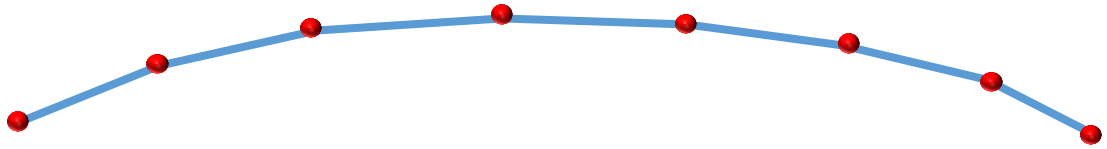
Answer:

- An infinite number of normal vectors!

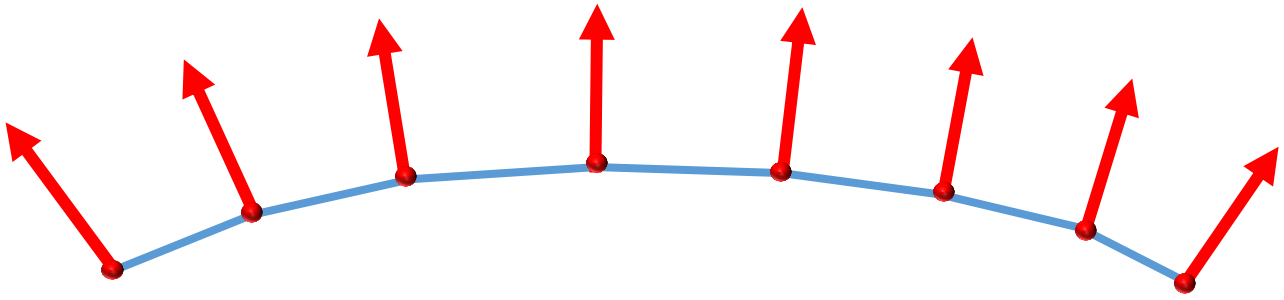
- That's too many!

How to solve the problem

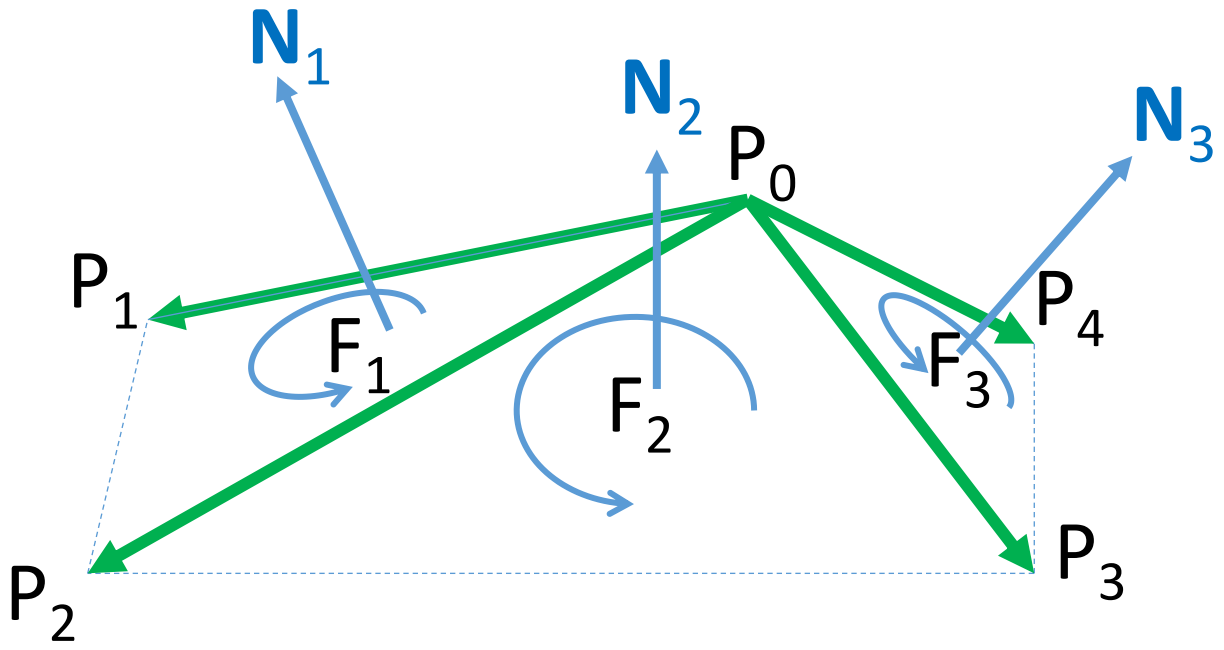
Step1: Discretize the surface



Step2: Discretize the normal vectors

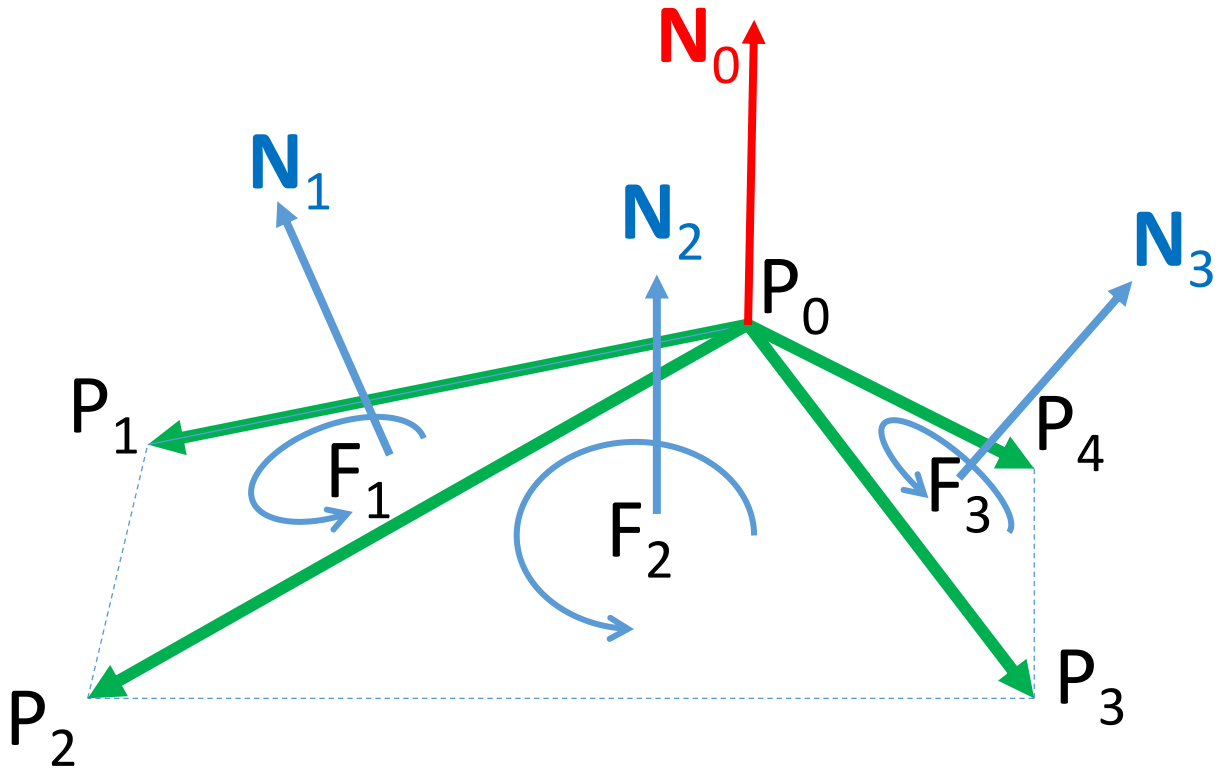


Normals are ⊥ to Faces



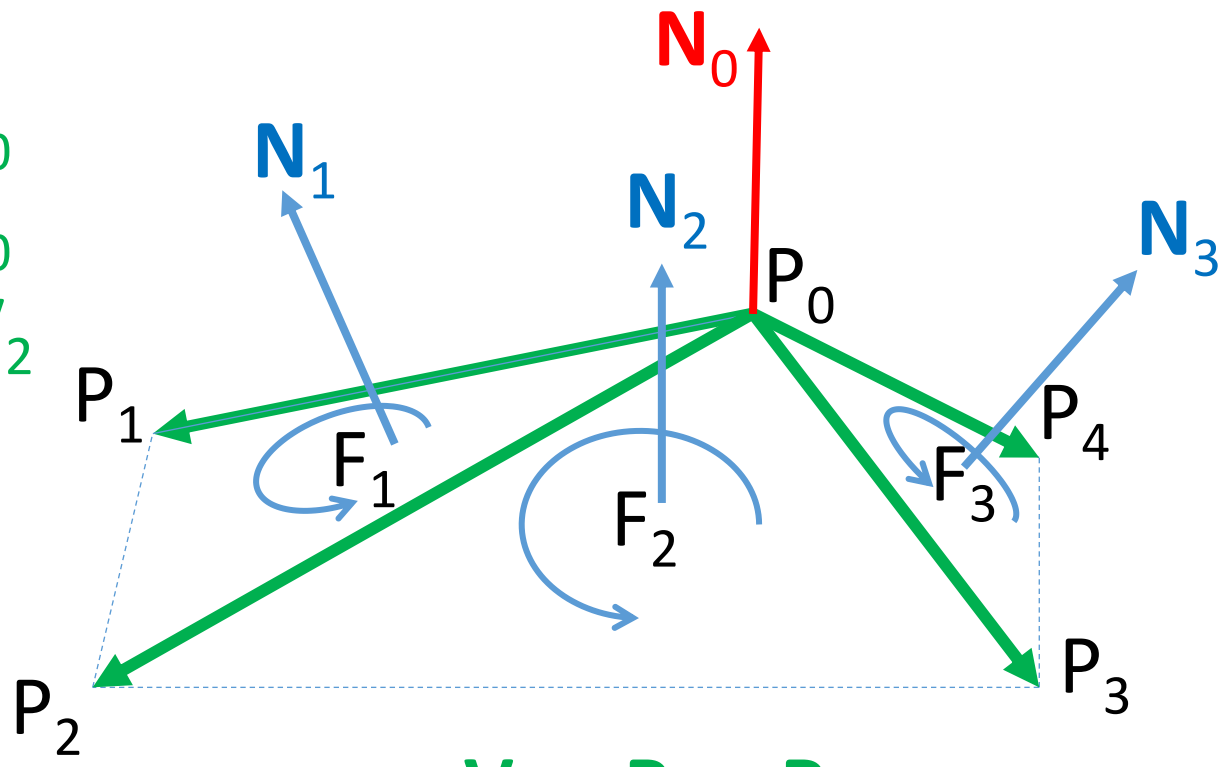
But, how to store the normal vectors?

Normal Vertex Vectors



How to compute normals?

$V_1 = P_1 - P_0$
$V_2 = P_2 - P_0$
$N_1 = V_1 \times V_2$



$V_3 = P_3 - P_0$
$V_4 = P_4 - P_0$
$N_3 = V_3 \times V_4$

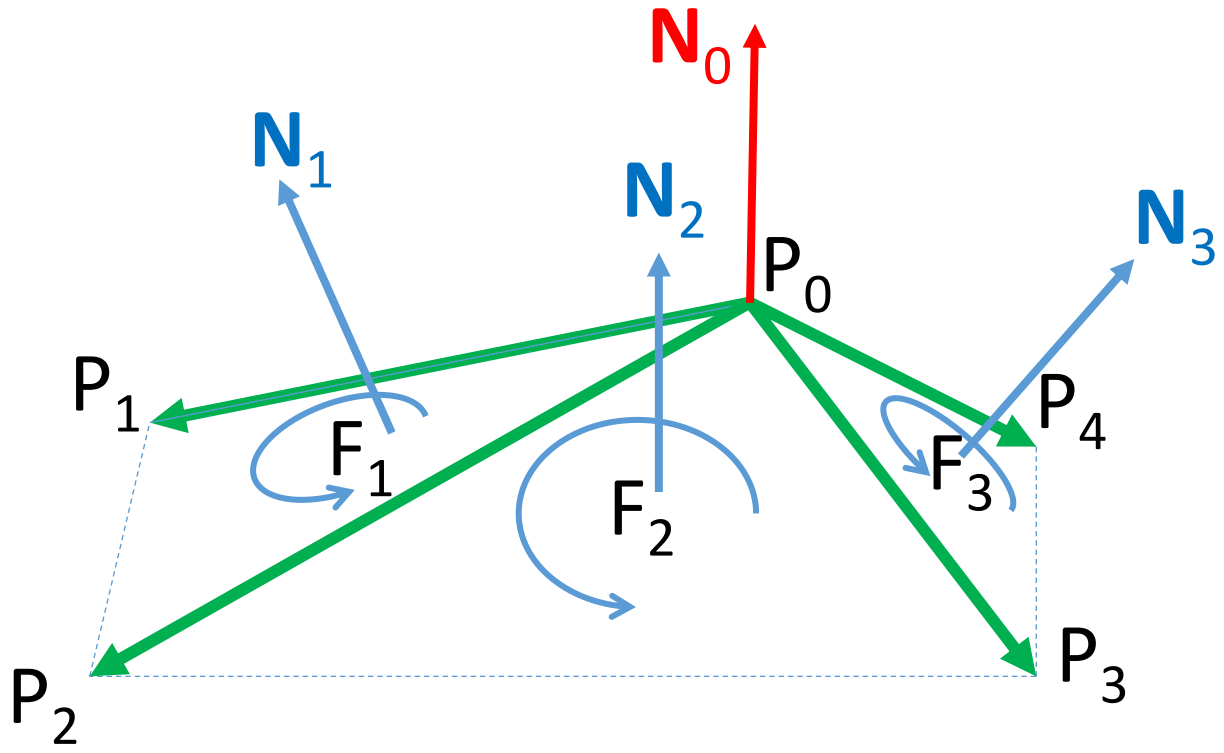
Must normalize!

$N_i = N_i / || N_i ||$

$V_2 = P_2 - P_0$
$V_3 = P_3 - P_0$
$N_2 = V_2 \times V_3$

$$|| N || = \sqrt{N_x^2 + N_y^2 + N_z^2}$$

How about N_0…?

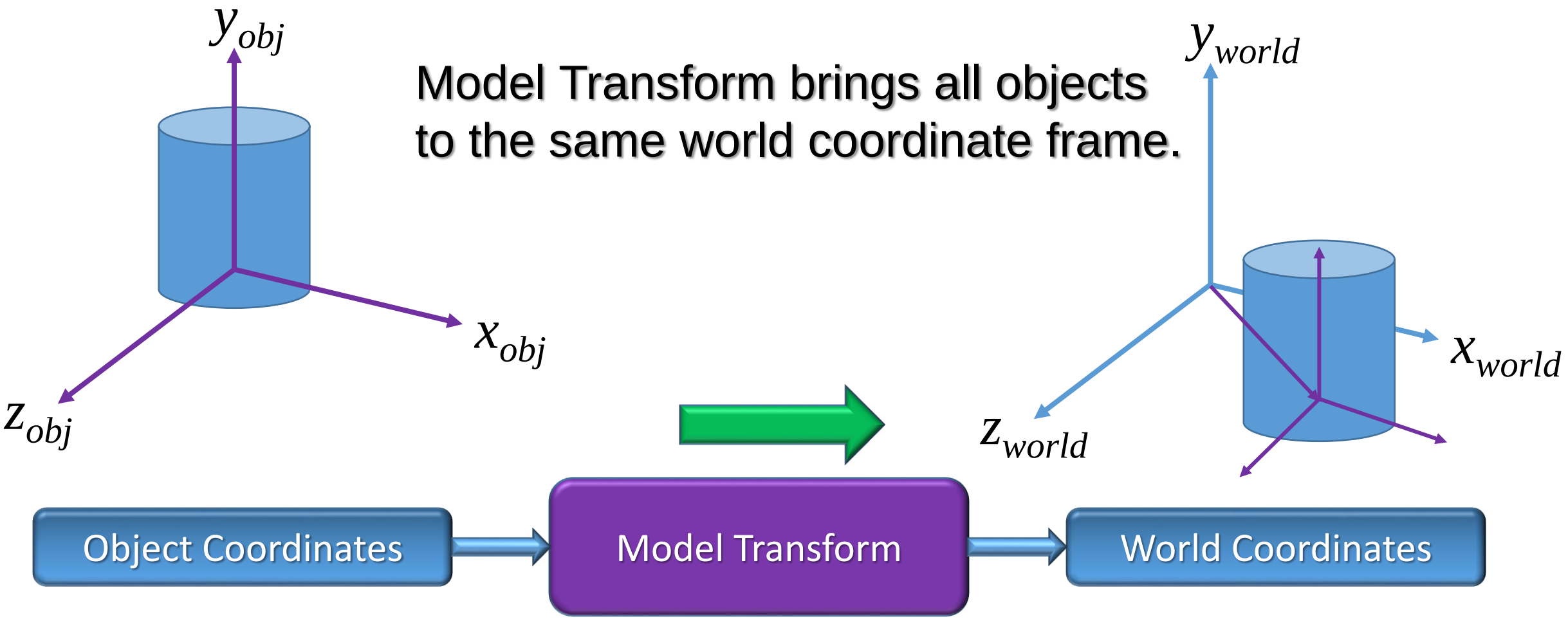


Average over all adjacent faces: $N'_0 = N_1 + N_2 + N_3 +$

Again, do not forget to normalize the normal: $N_0 = N'_0 / || N'_0 ||$

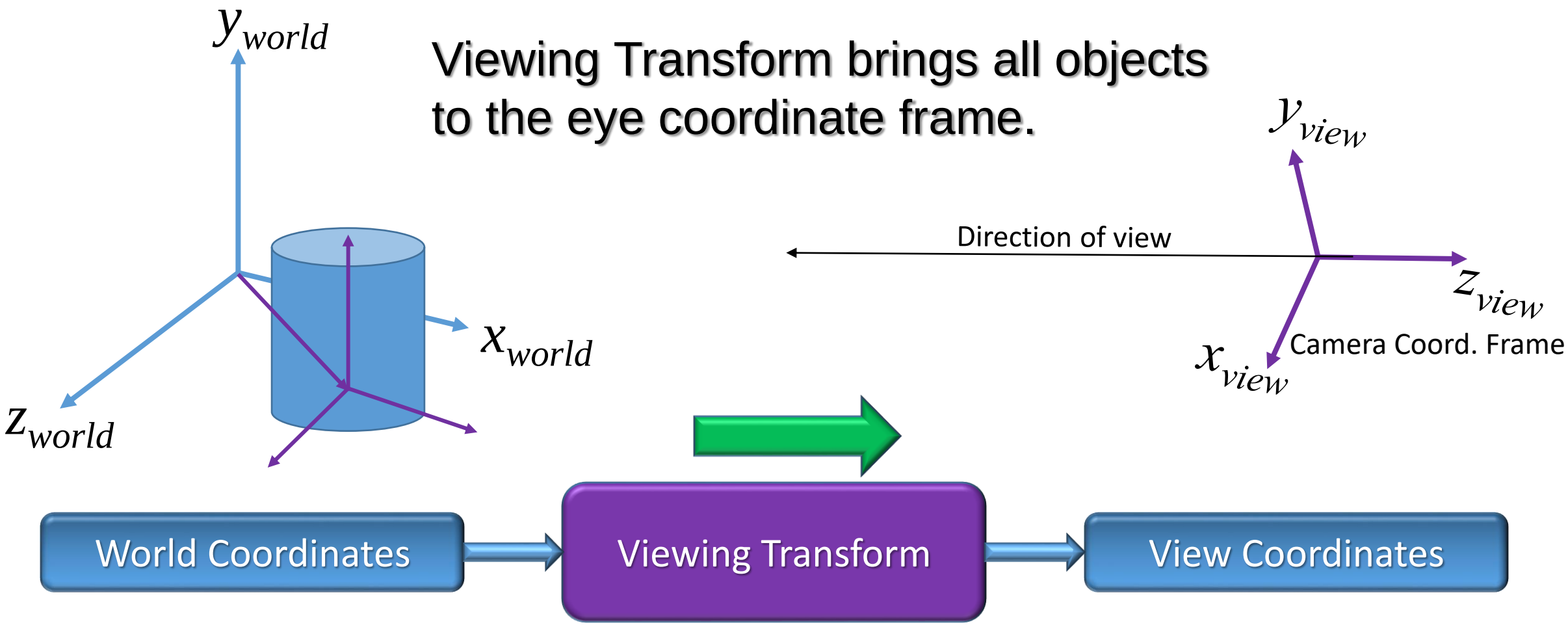
The Normal Problem

Recall…Model Transform

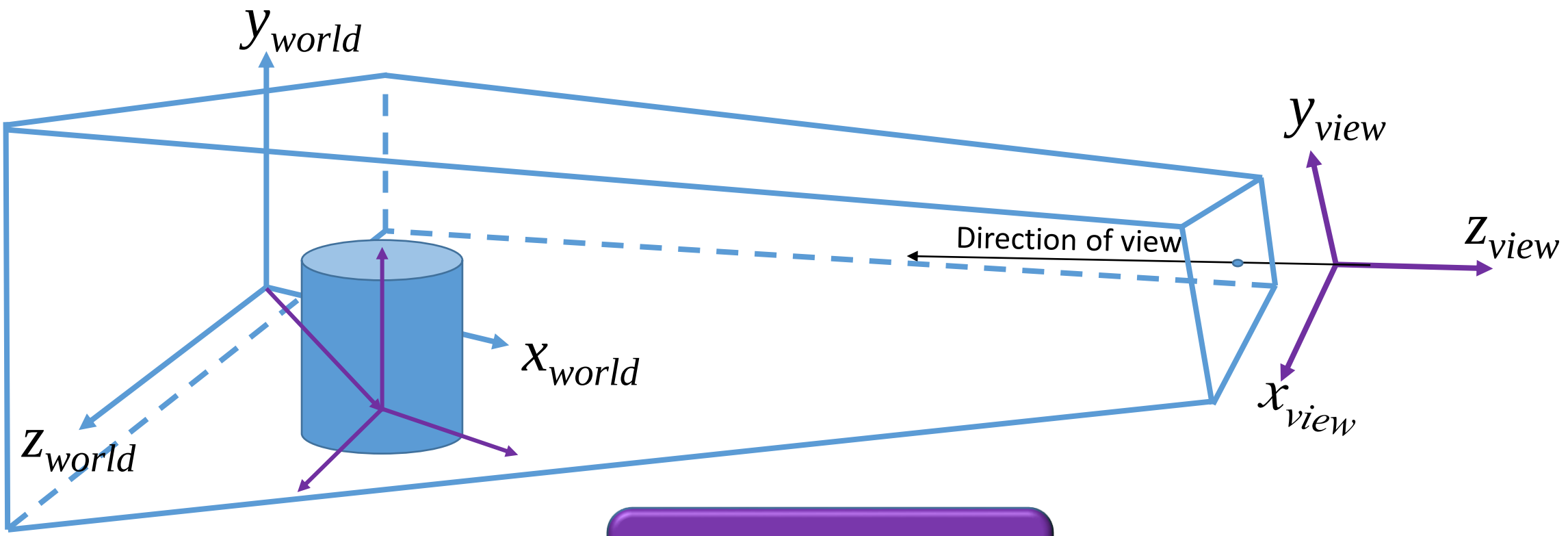


Recall…Viewing Transform

Viewing Transform brings all objects to the eye coordinate frame.

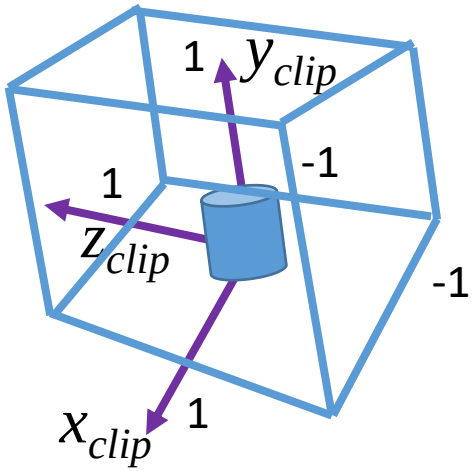


Recall: Viewing Frustrum

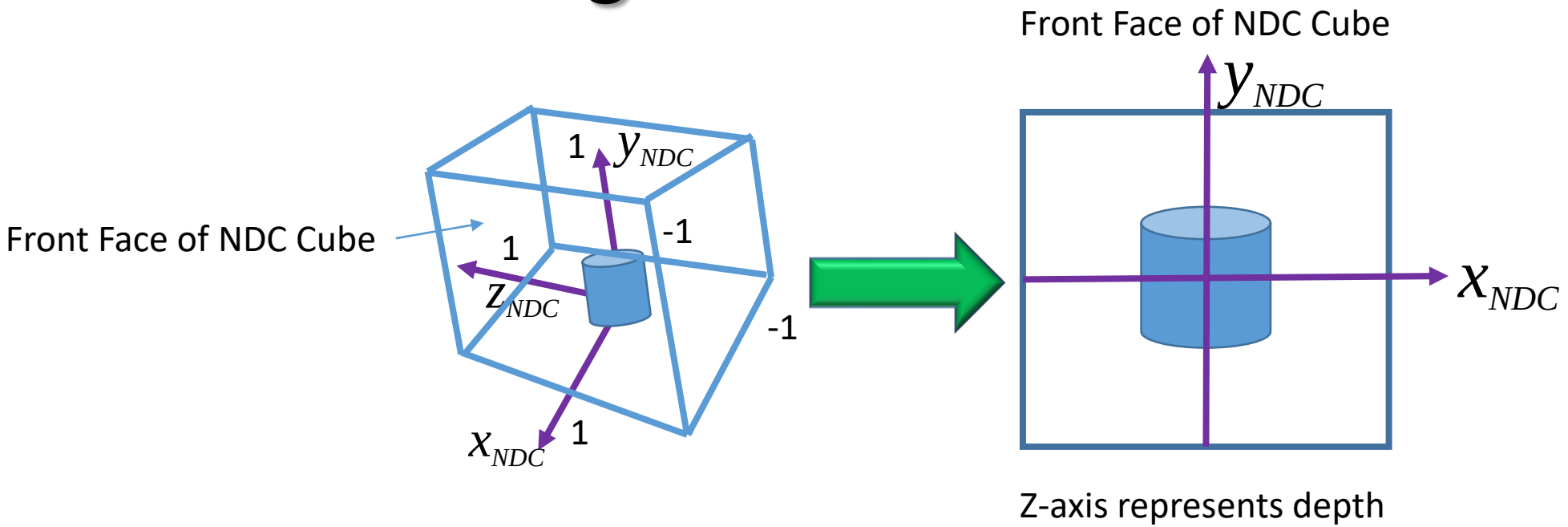


View Coordinates → Projection Transform → Clip Coordinates

Recall: Projection Transform

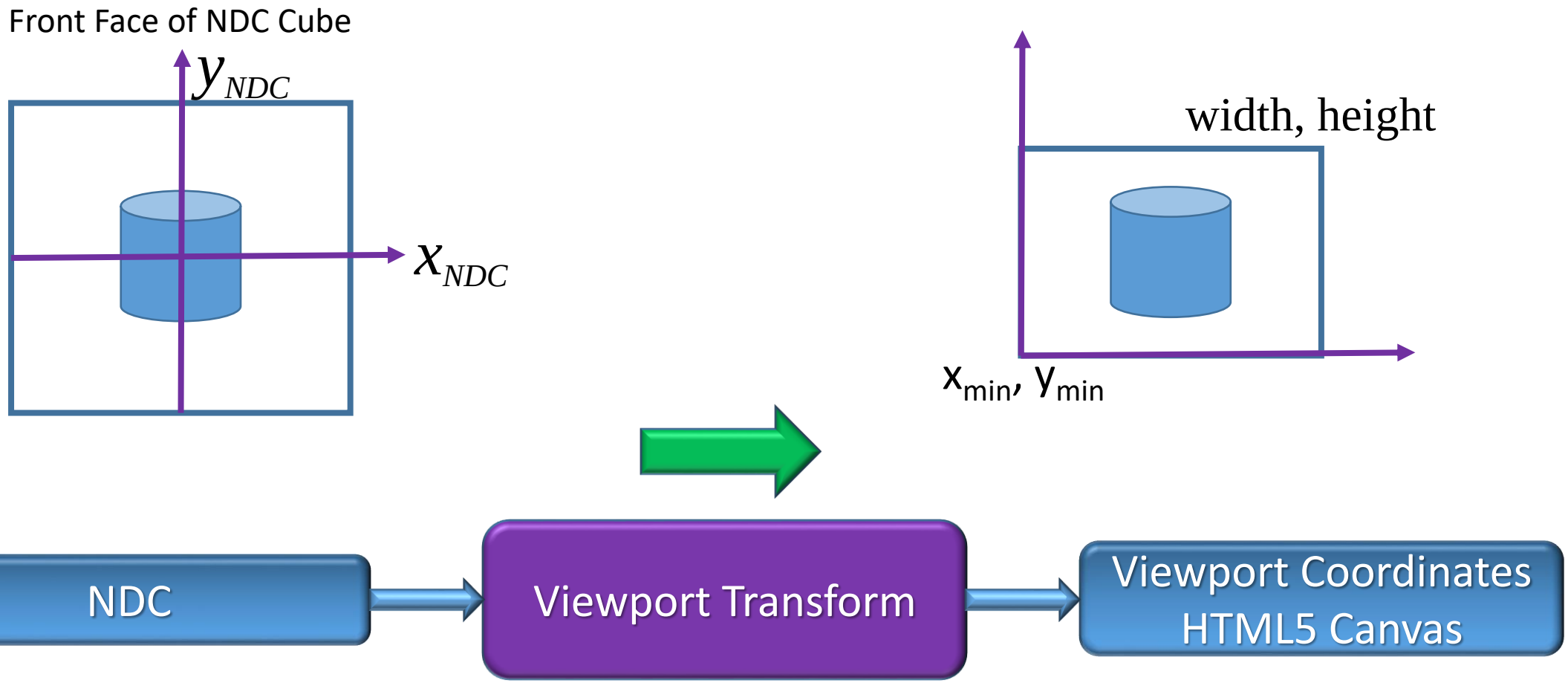


Recall: Projection Transform



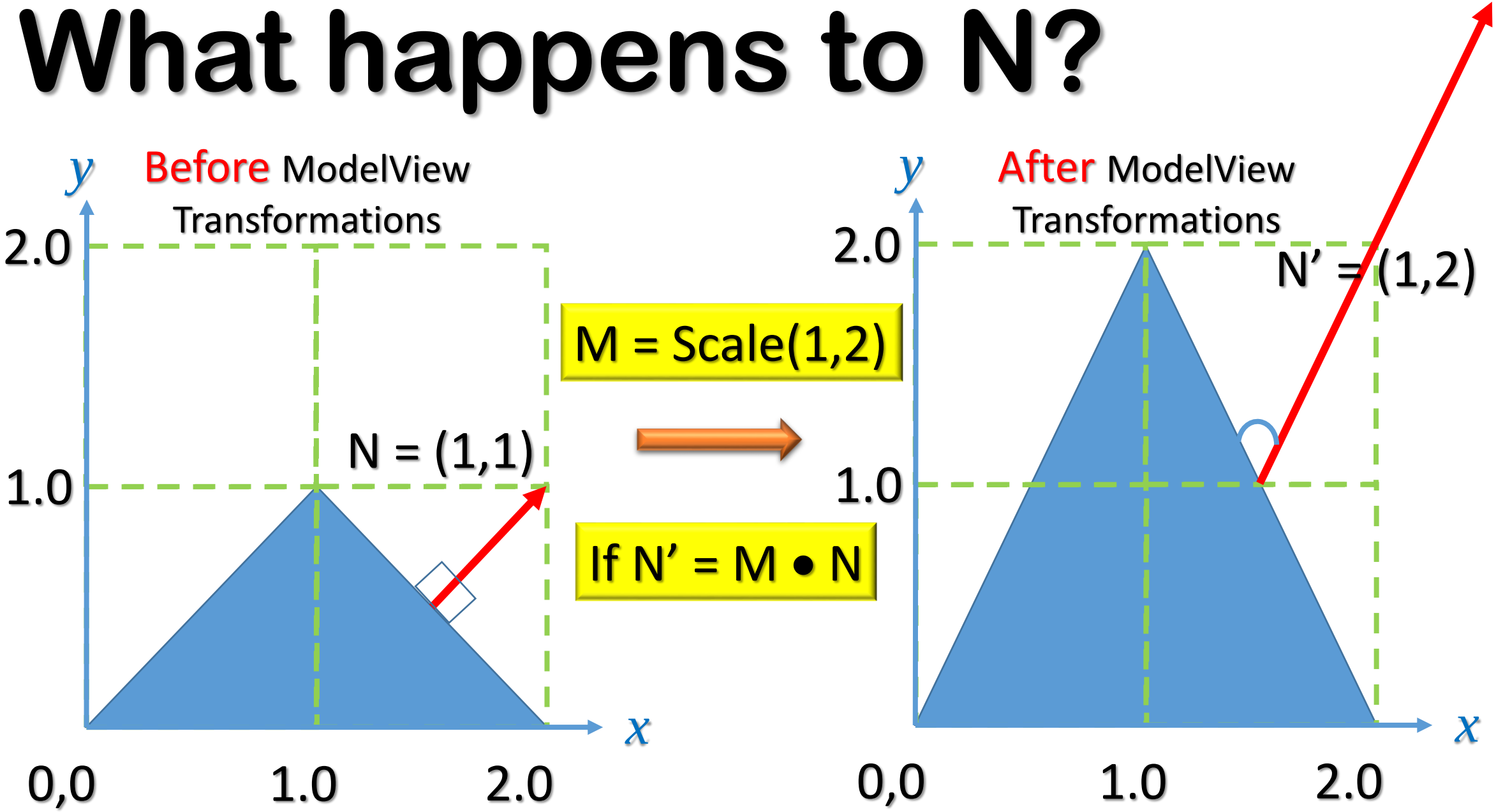
Cube: [-1,-1,-1] to [+1,+1,+1]

Recall: Viewport Transform



But, can we transform the Normal Vectors N by Model View Transformation M?

What happens to N?



Answer: **No!**

How to solve the Normal Transformation Problem?

Answer: we must use geometric properties

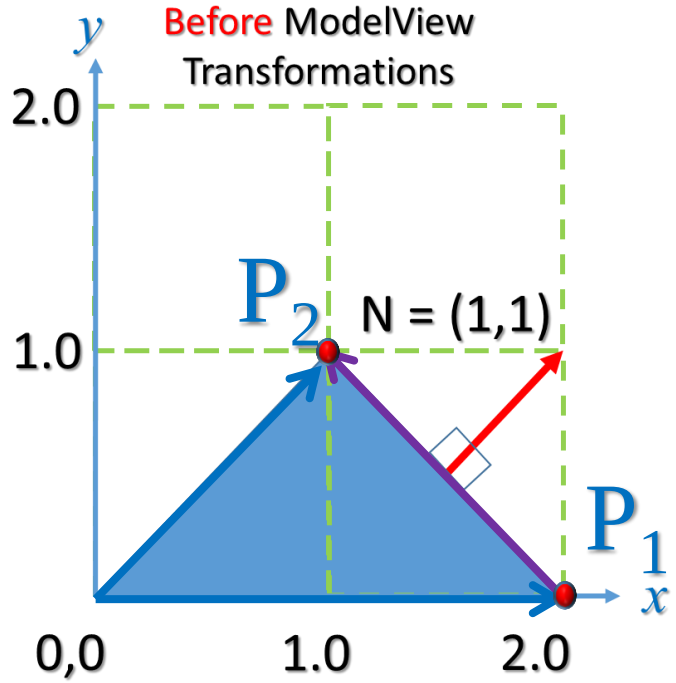
By Definition

- A normal vector must be perpendicular to a face vector S!

$$N \bullet S = 0$$

- But, what is S ?

$$S = P_2 - P_1$$



How does S transform?

- All the points are transformed by M

$$P_1' = M\,P_1$$

$$P_2' = M\,P_2$$

- Hence, $P_2' - P_1' = M\,(P_2 - P_1)$
- So, the surface vector S' transforms as:

$$S' = M \bullet S$$

Question: What is N' ?

- We must maintain perpendicularity

$$N' \bullet S' = 0 = N \bullet S$$

- Since $S' = M\,S$:

$$N' \bullet M\,S = 0$$

- Therefore, let $N' = Q\,N$

$$(Q\,N) \bullet (M\,S) = 0$$

Question: What is Q?

- Since

$$(Q\,N) \bullet (M\,S) = 0$$

- We can re-write the above as:

$$N^T\,Q^T\,M\,S = 0$$

- But, in order for this to hold, we must have: $Q^T\,M = 1$, **where 1=identity**

Question: What is Q?

- Since,

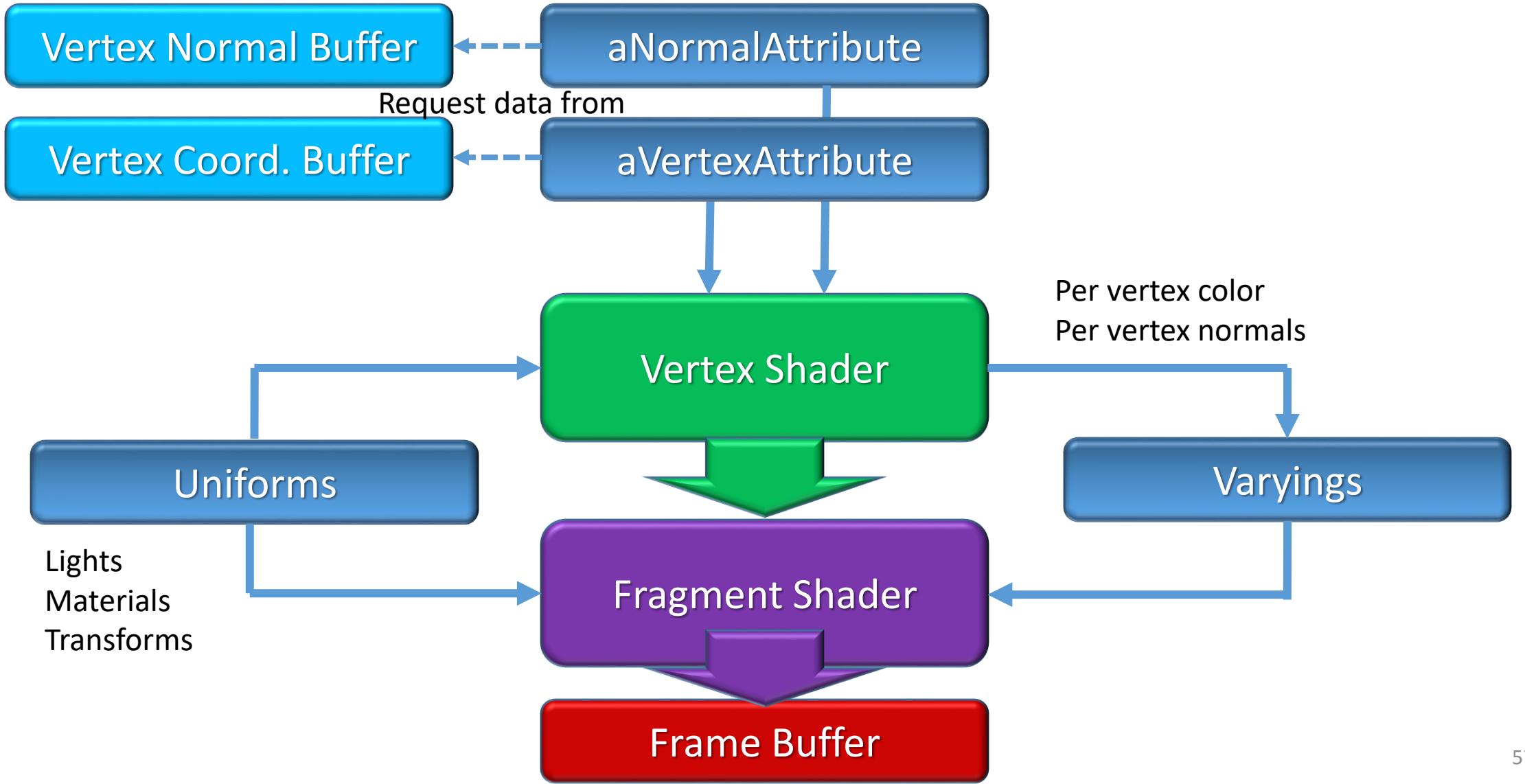
$$Q^T M = 1$$

- We must have:

$$Q = (M^T)^{-1}$$

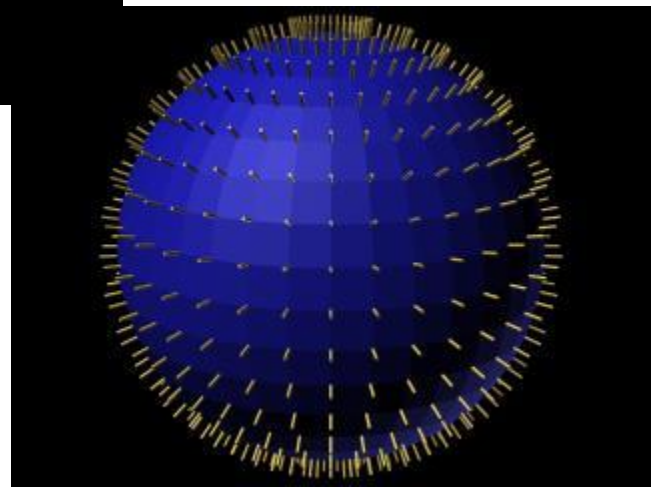
- So, **Q** is the inverse of the transpose of the Model-View transformation matrix!

Pipeline


Vertex Normal Buffer
aNormalAttribute
Request data from
Vertex Coord. Buffer
aVertexAttribute
Vertex Shader
Per vertex color
Per vertex normals
Uniforms
Varyings
Lights
Materials
Transforms
Fragment Shader
Frame Buffer

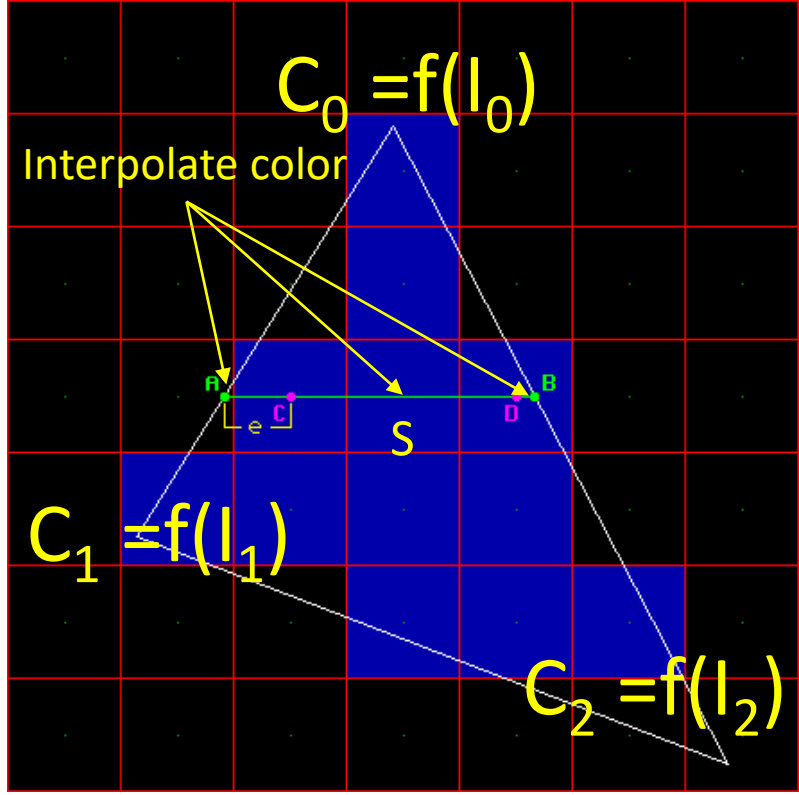
Light Reflection Models

Flat Shading



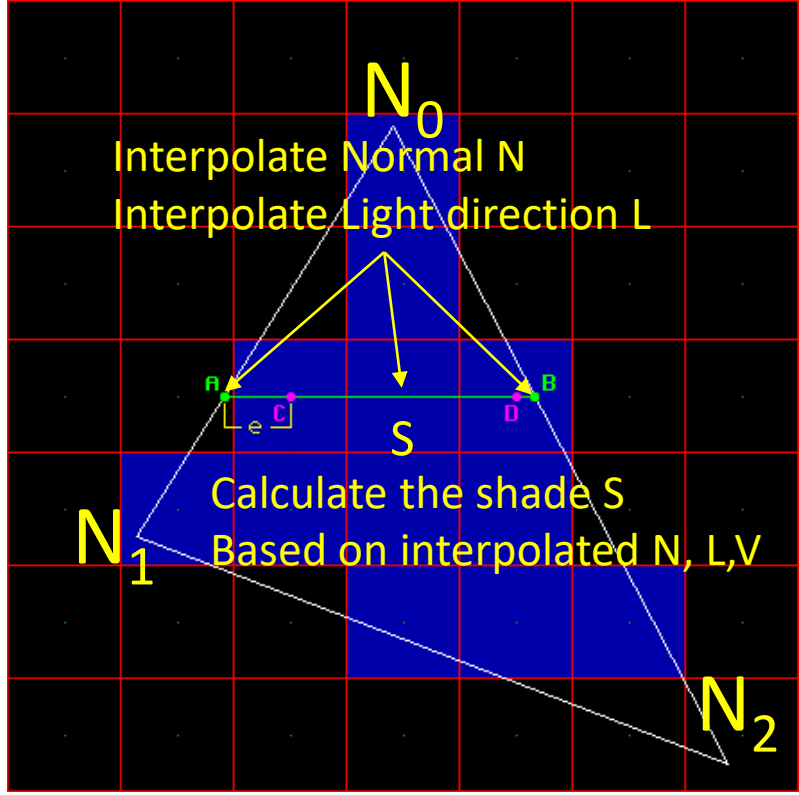
- Algorithm:
  - For each face in the mesh
    - N: Calculate the normal
    - L: Calculate the light direction
    - I: Calculate the intensity of the reflected light using an illumination model, e.g. $I = I_a + I_d + I_s$
    - C: Set the color of the entire face to a constant
    - S: Shade or render the entire face with color C.

Gouraud Shading



- Algorithm:
  - For each vertex in the mesh
    - N: Calculate the normal
    - L: Calculate the light direction
    - I: Calculate the intensity of the reflected light using an illumination model, e.g. $I = I_a + I_d + I_s$
    - C: Set the color of the vertex as a function of I
    - S: Shade each pixel with an interpolated color S

Phong Shading



- Algorithm:
  - For each vertex in the mesh
    - N: Calculate the normal
    - L: Calculate the light direction
    - For each pixel
      - N: Interpolate the normal
      - L: Interpolate the light direction vector
      - I: Calculate the intensity of the reflected light using an illumination model, e.g. $I = I_a + I_d + I_s$
      - C: Set the color of the pixel as a function of I
      - S: Shade the pixel with as a function of C

GPU Shading

N = Normal Vector

L = Light Direction Vector

V = Viewer Direction Vector

R = Reflection Direction Vector

Ia = Ambient Light Component

Id = Diffuse Reflection Component

Is = Specular Reflection Component

Total Reflected Light:

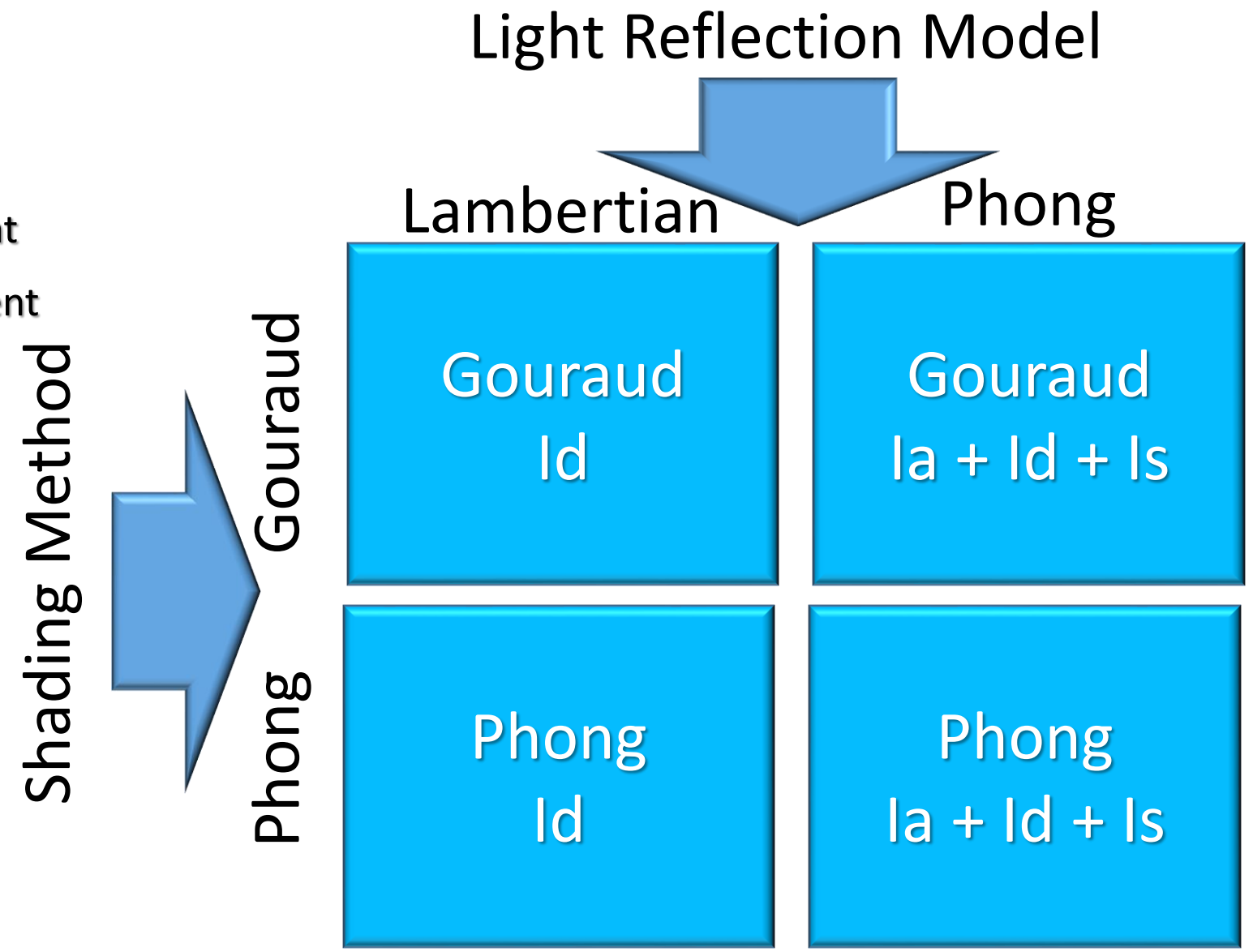
$I = Ia + Id + Is$

$Ia = Ka\ I_L$

$Id = Kd\ I_L\ N \bullet L$

$Is = Ks\ I_L\ R \bullet V$

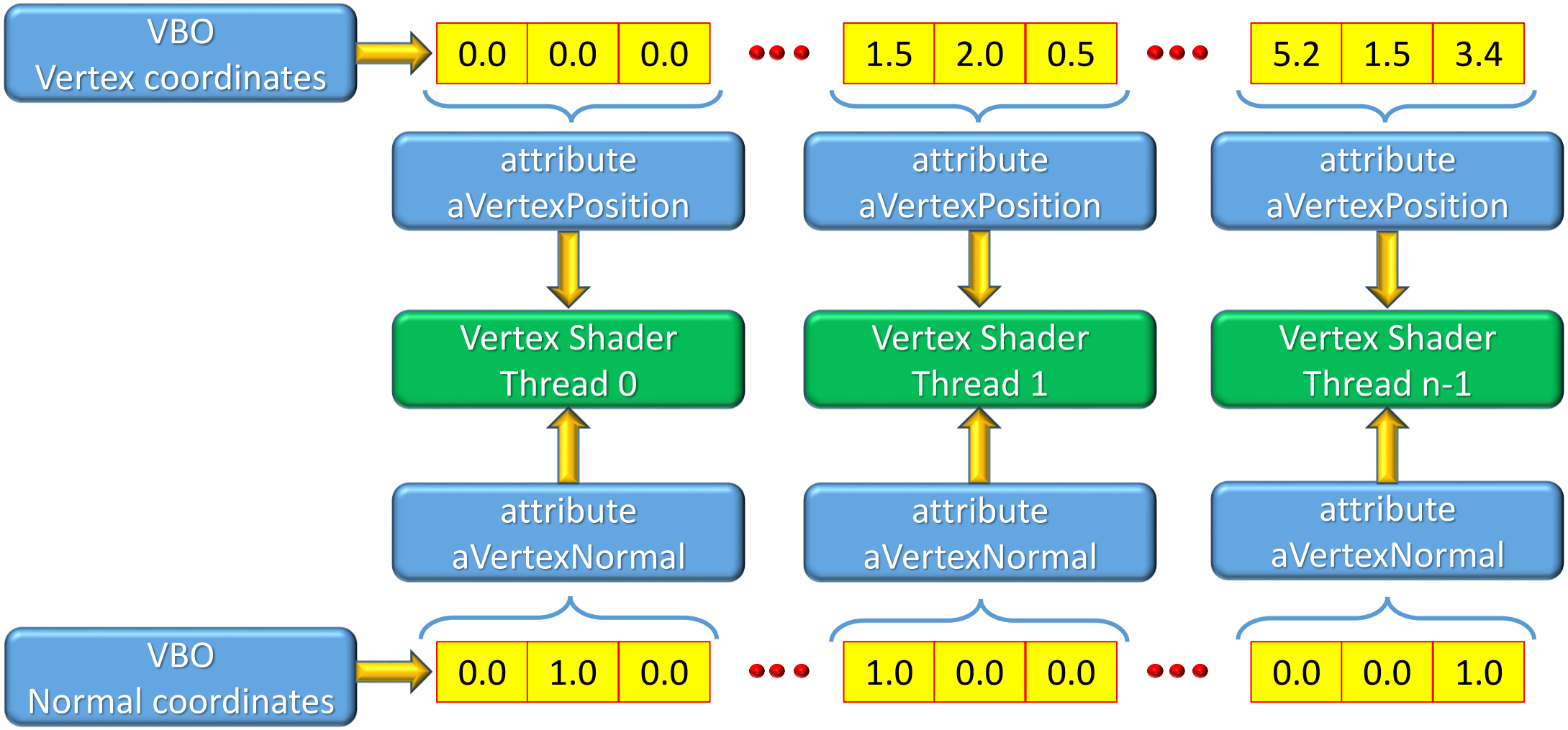
$R = 2(L \bullet N)N - L$

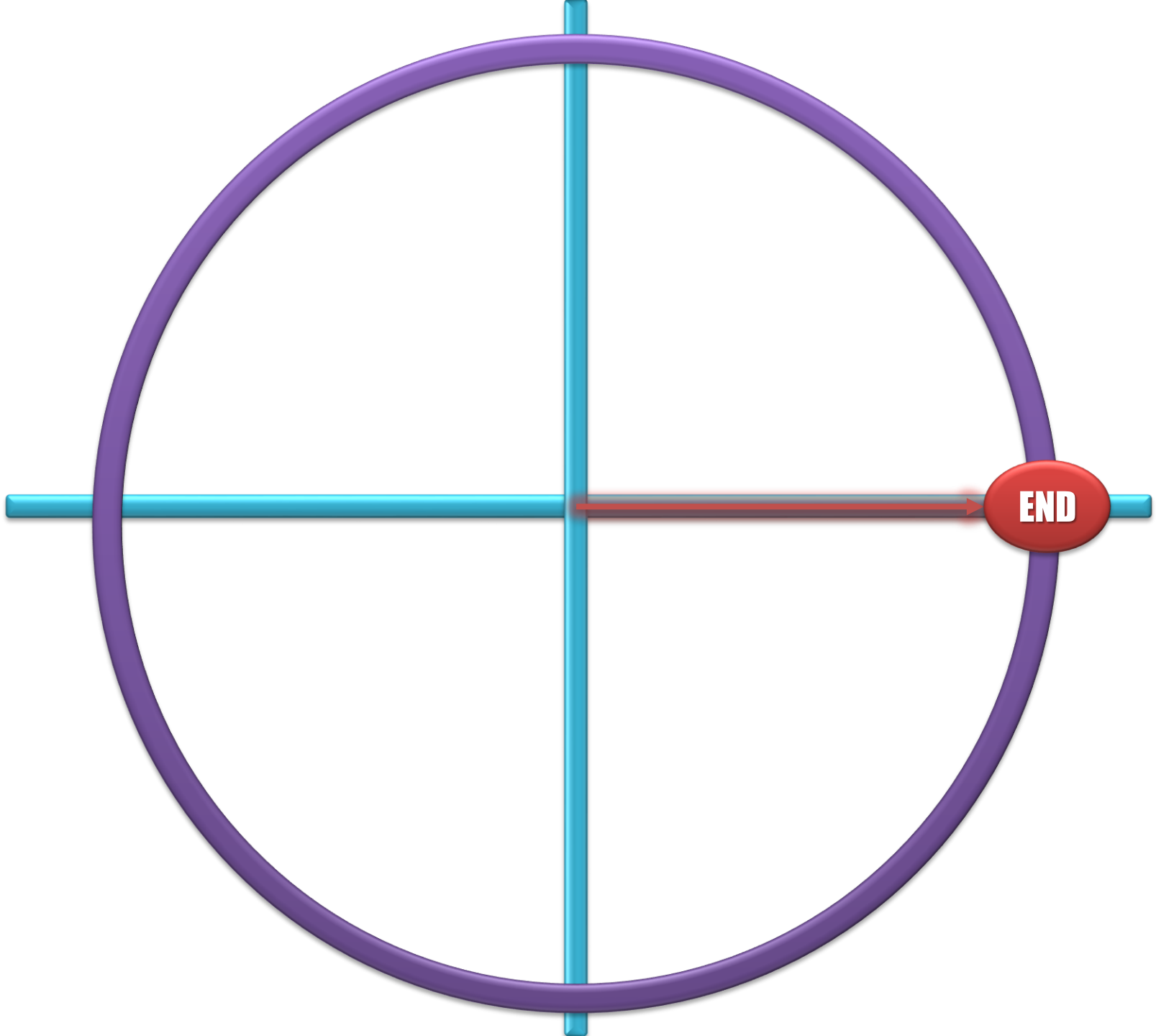


GPU Parallelism

- WebGL demands parallel thinking!
- We must deal with:
  - Browser – JavaScript Communication
  - JavaScript – Libraries and Functions
  - JavaScript – Vertex Shader Communication
  - Vertex Shader – Fragment Shader Comm.

Parallel Data Flow




END