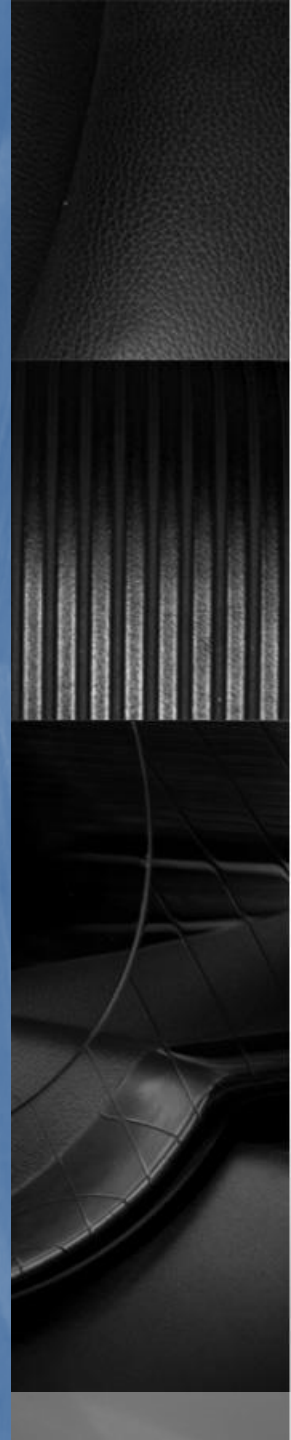
3D Graphics and Animation

Lighting, Material and Shading

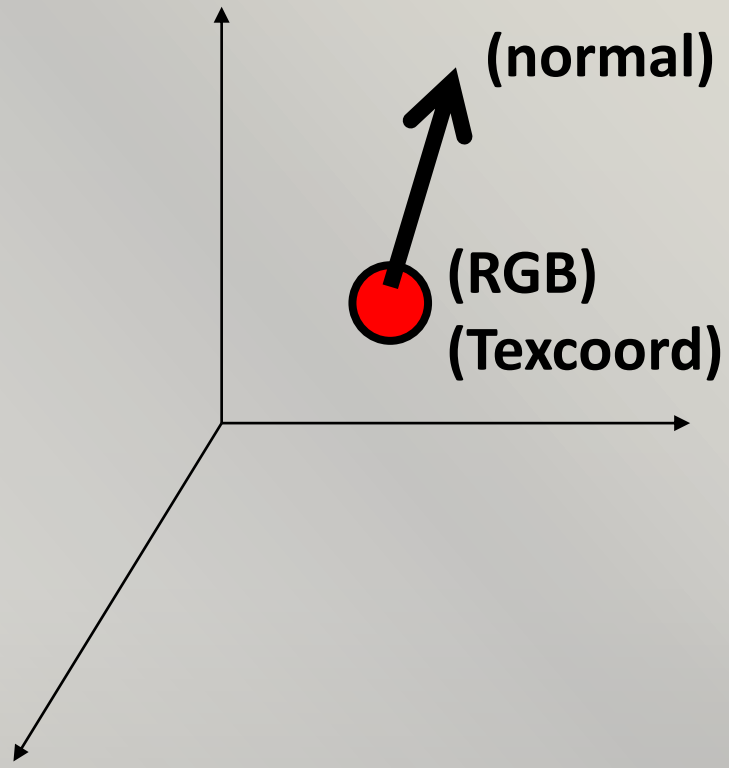


Recap

- In the last lecture, we discussed the camera model used in graphics, and the related mathematics in computing the projections.
- These are the basics of understanding how object vertices are processed spatially in the rendering pipeline
- This lecture, we will study the computation of color associated with these vertices

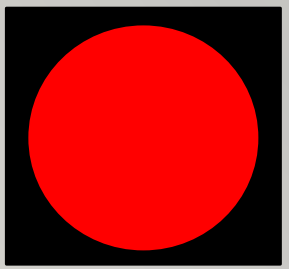
Vertices Data

- Associate with vertices are a number of properties, they may include
  - Normal vector
  - Material / Color
  - Texture coordinate



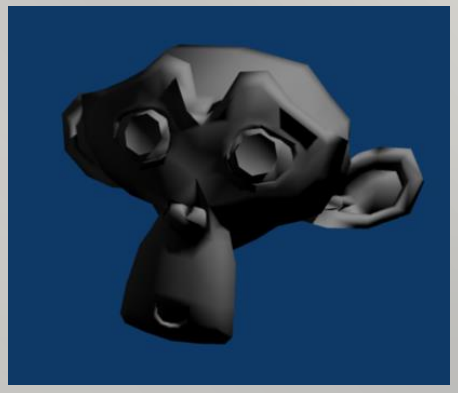
3D without Lighting

- If only color is considered without lighting, everything drawn will be in plain colors
- You even can't feel that it is a 3D object
  - Loss feeling of shape
- When lighting is added
  - You can understand the shape of object based on its **shading**



No Lighting

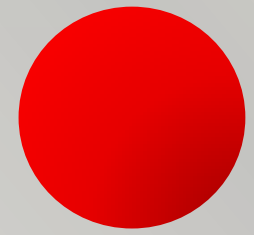
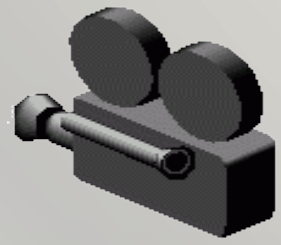
With Lighting



Lighting or Illumination

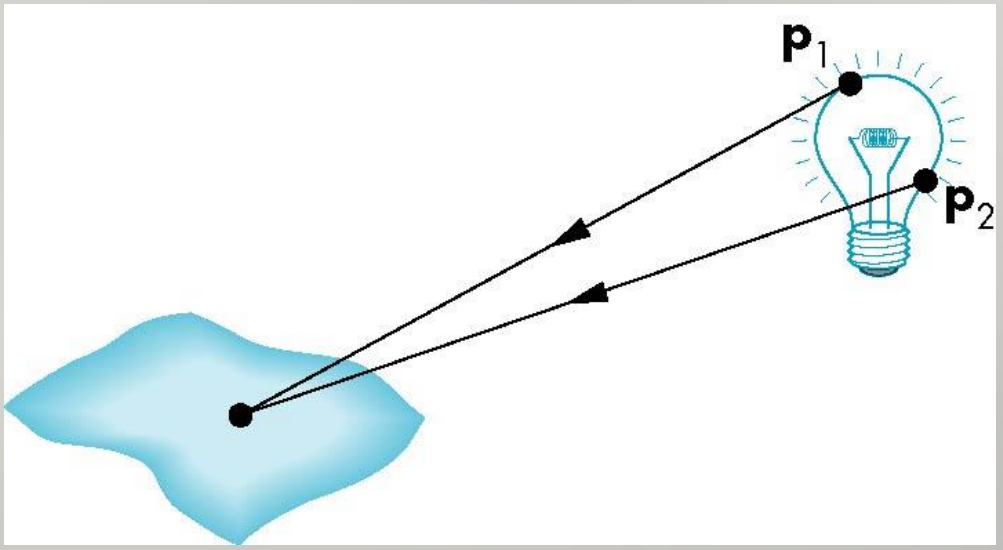
- When discuss about illumination, we need to consider
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation



Light Sources in Real World

- General light sources are difficult to work with because we must integrate light coming from all points on the source
- Usually we will use simplified light sources

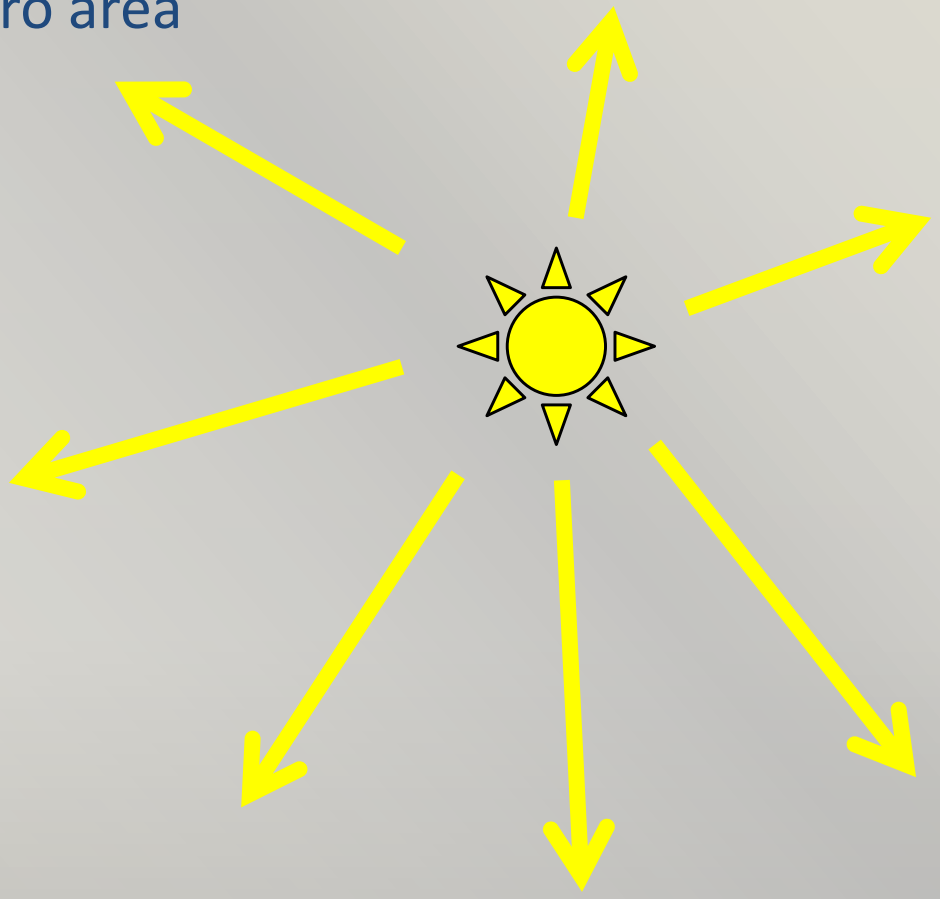


Kinds of lights

- Point light
- Directional light
- Spotlight
  - Restrict light from ideal point source
- Ambient light
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces

Point Light

- Light rays are originate from a single position
  - Note that the source has zero area
- Defined by
  - Location
  - Color
- It shines to all directions equally

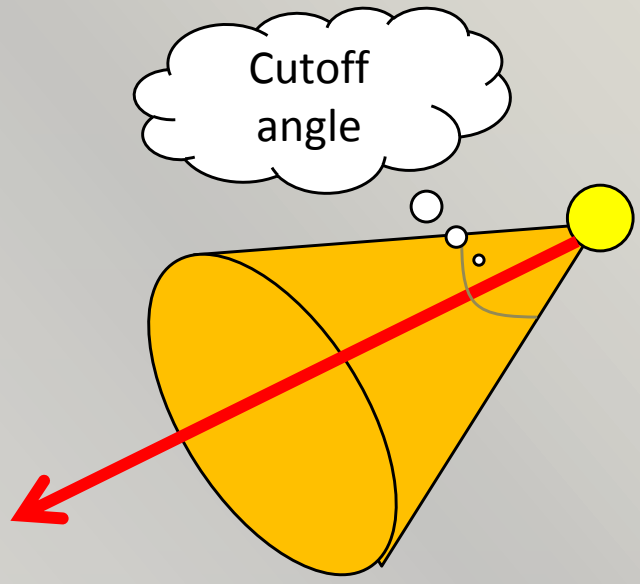


Directional Light

- Light rays are originate from a source far away
  - Similar to what the sun does
- All rays are parallel
- Defined by
  - Direction vector
  - Color
- The effective region is infinite

Spot Light

- Similar to a point light but having a limited effective region
- Defined by
  - Position of light source
  - Direction vector
  - Cutoff angle
  - Color



Ambient Light

- Ambient light is the result of multiple interactions between light sources and the objects in the environment
  - To really simulate this is very difficult and time consuming, so this term is added to skip out the related computations
- Can be considered as “background” light
- Defined by
  - Color

Attenuation

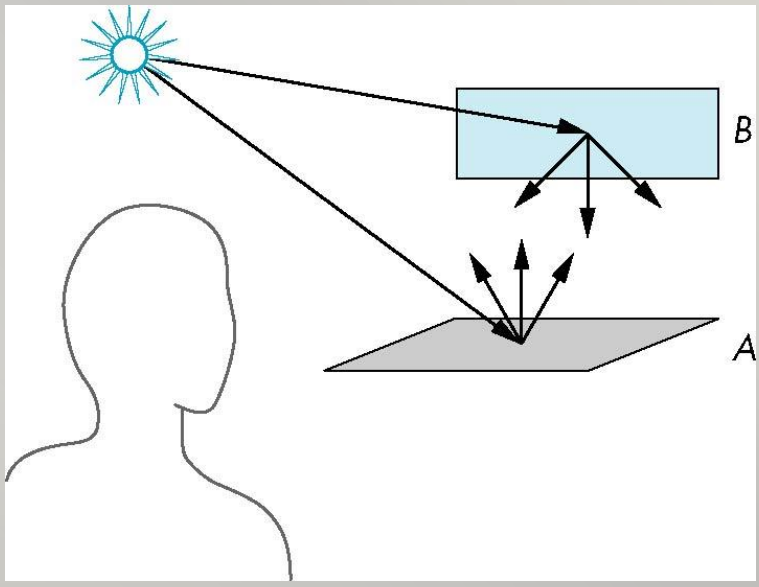
- The light from a point source that reaches a surface is inversely proportional to the square of the distance between them
- We can multiply a factor f to the light intensity arriving the surface

$$f = 1/(a + bd + cd^2)$$

- Here, d is the distance
- a,b,c are constant terms to control the effect of distance

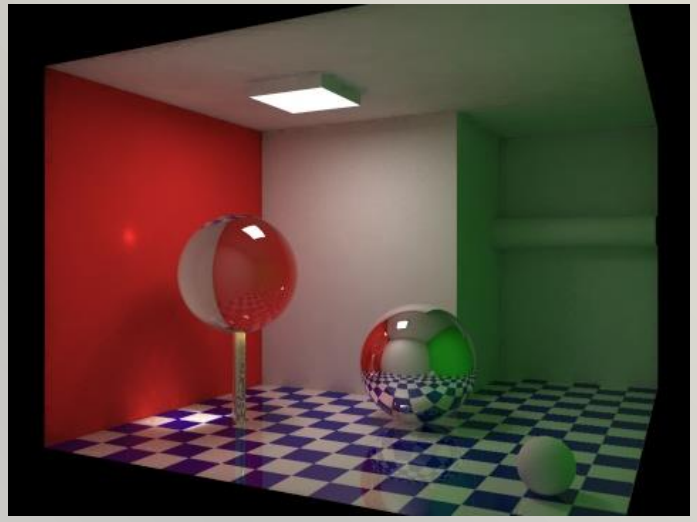
Reflection from Object Surface

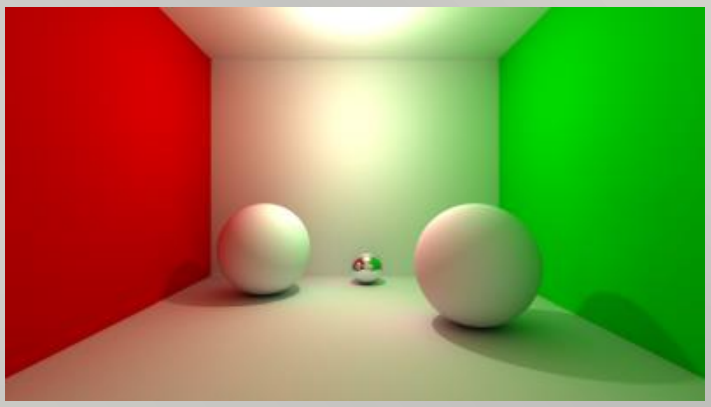
- In real world, when light arrives a surface
  - Some will be absorbed
  - Some will be reflected
- The reflected light will continue to travel
- It may hit other surface
  - Some will be absorbed
  - Some will be reflected
- This process repeats until all light are absorbed



Reflection from Object Surface

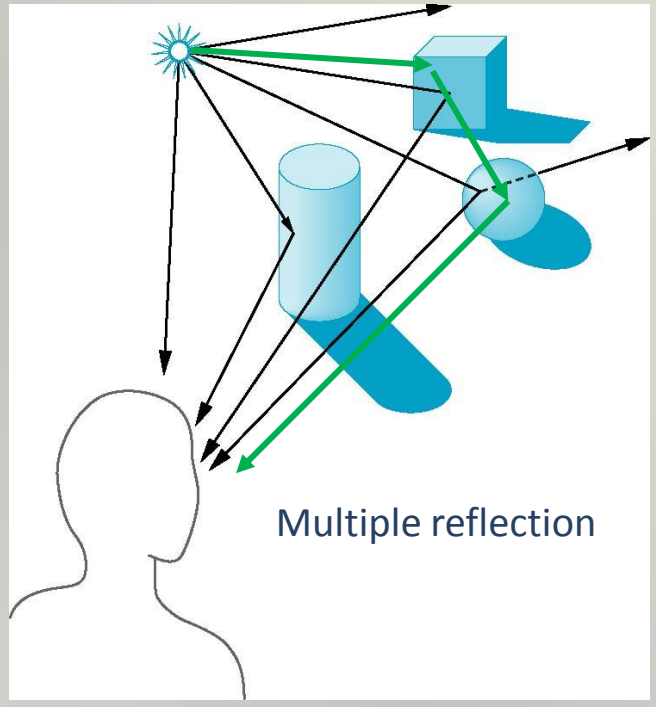
- To completely simulate what happened in the real world is basically impossible
  - Requires lots of computations
  - Too time consuming
- It is sometimes not worth the effort as the light intensity will decrease after several reflections





Reflection from Object Surface

- So, most of the real-time rendering engines consider **ONLY single reflection** but NOT multiple reflections
  - That also means only consider light from light source, but not those reflected from other surface

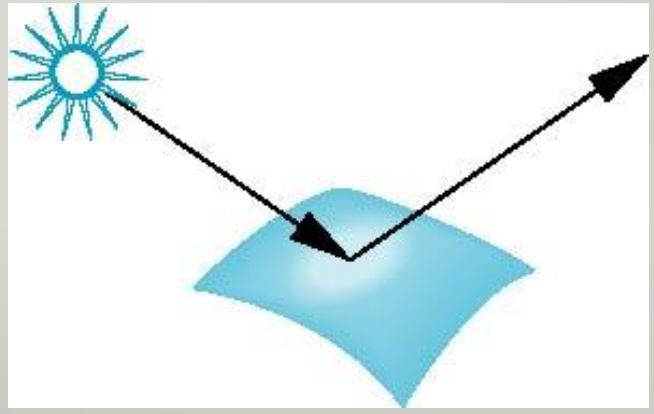


Material

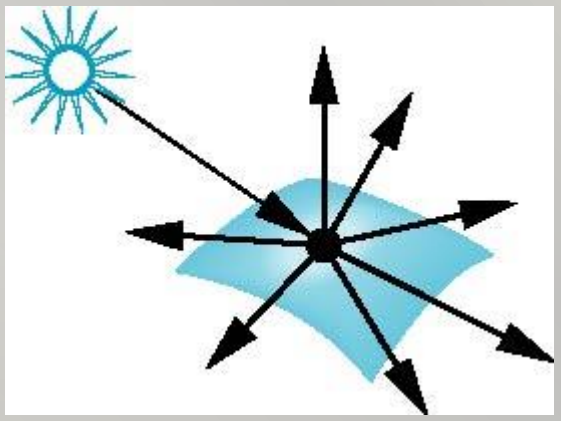
- Consider a single reflection of light, it can be
  - In various directions, or
  - Certain direction
- This will depend on the material properties of the surface the light hits
- A simple classification of surface materials in real world
  - Smooth surface
  - Rough surface

Material meets Lighting

- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflected the light
- A rough surface scatters light in all directions
  - “rough” here is in micro level



smooth surface



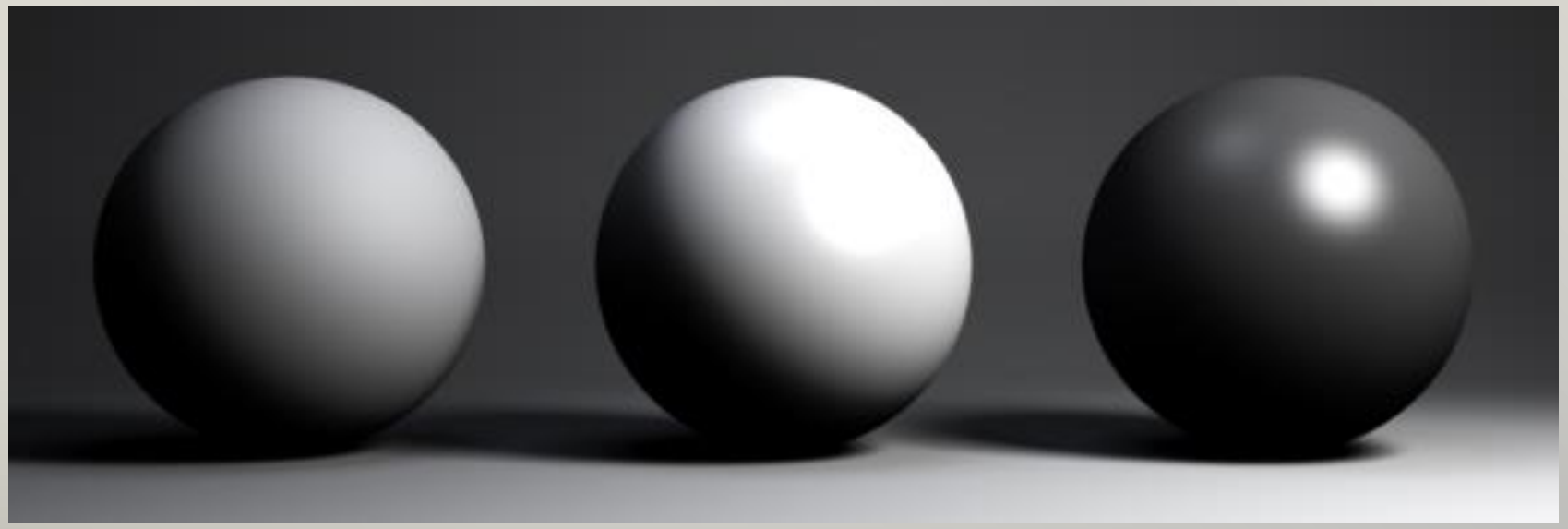
rough surface

Material meets Lighting

- Rough diffuse surface looks like plastic
- Shiny surface looks like metal, and we can find highlight regions

rough diffuse surface

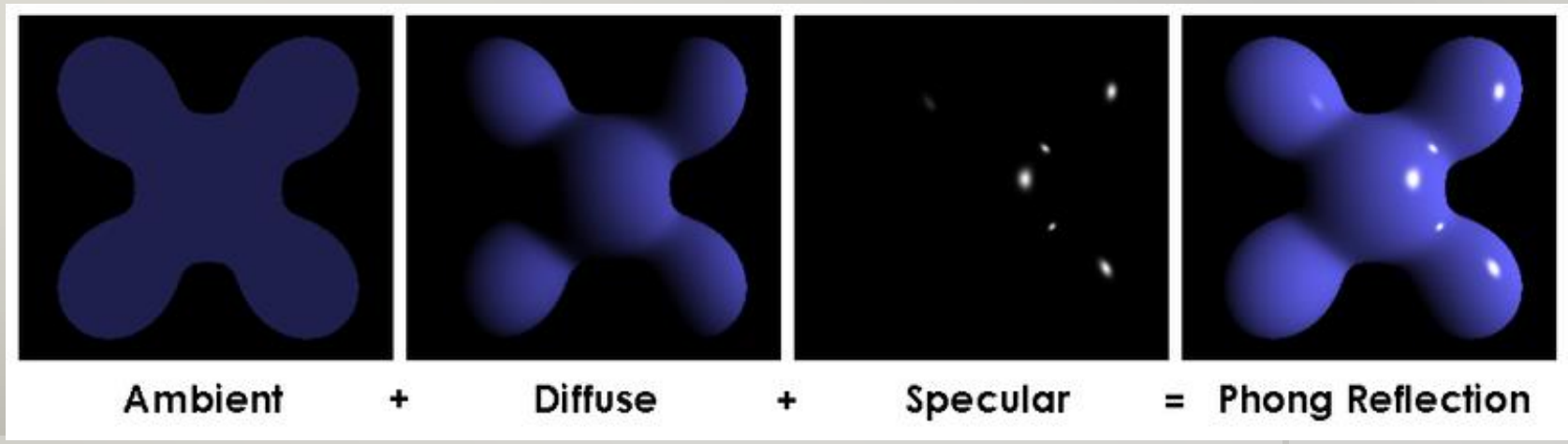
shiny surfaces



Bensimonds.com

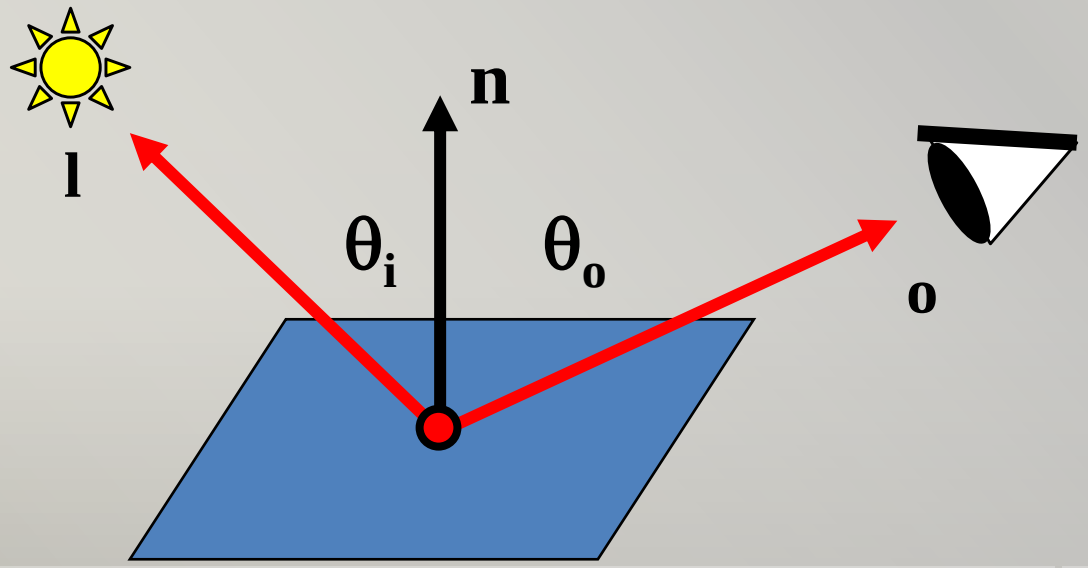
Lighting Model

- Based on above knowledge of materials, lighting model for surface reflection is developed
- 3 major components are included
  - Diffuse
  - Specular
  - Ambient



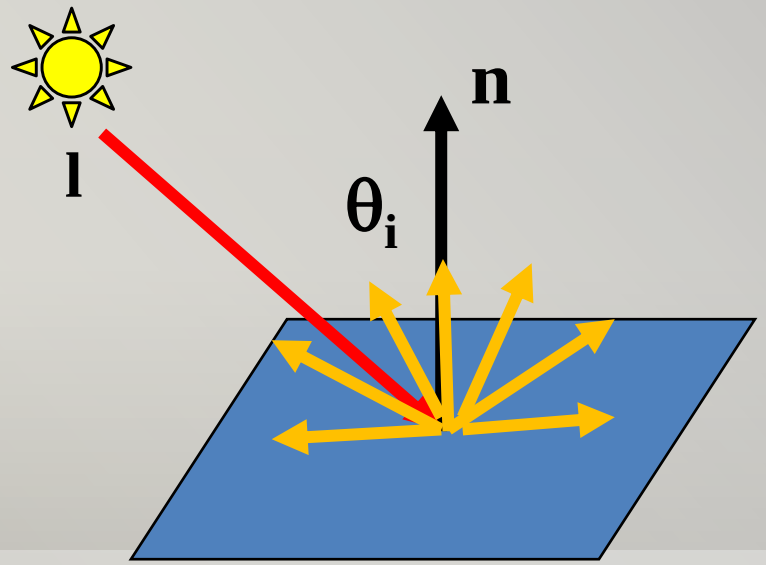
Some Notations and Terms

- Consider a surface at a point p
  - It's normal is $\mathbf{n}$ (unit vector)
  - The lighting source is on $\mathbf{l}$ (unit vector)
  - The observer(camera) is on $\mathbf{o}$ (unit vector)
- Reminded that $\cos\theta_i = \mathbf{l} \cdot \mathbf{n}$, $\cos\theta_o = \mathbf{l} \cdot \mathbf{o}$



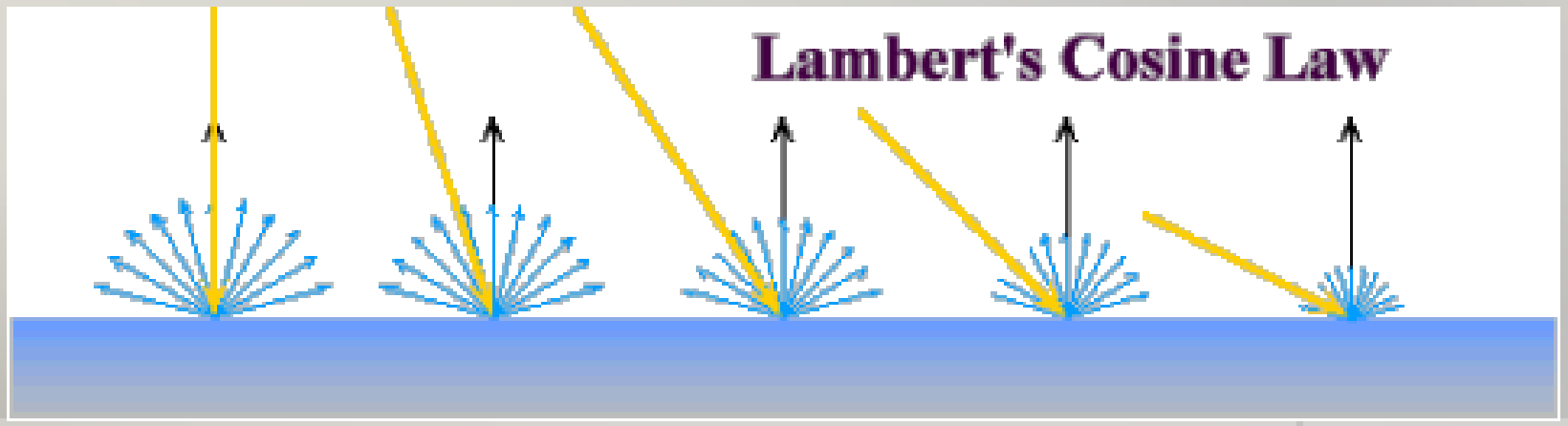
Diffuse Reflection

- Model reflection on rough surface
- Reflection are equally in all directions
  - So, NOT depends on observer's position
- The mathematical model is based on Lambertian Shading (1971)



Diffuse Reflection: Lambertian Law

- Amount of light reflected is proportional to the vertical component of incoming light
  - reflected intensity: $\cos \theta_i = \mathbf{l} \cdot \mathbf{n}$
- Reflection decreases with increase in angle between surface normal and source



Diffuse Component

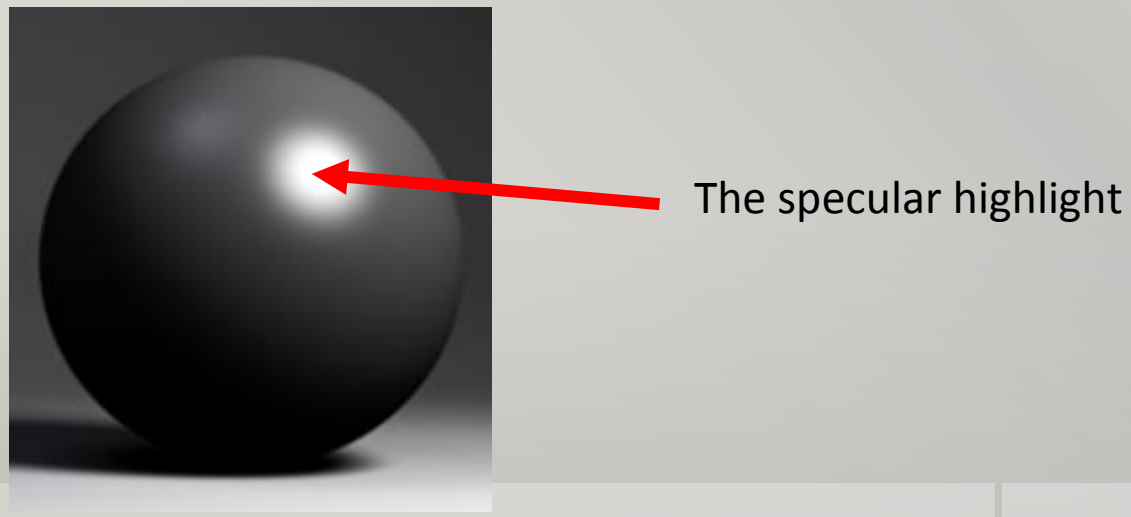
- As a result, we form the diffuse component $\mathbf{C_d}$ in the light model as follow:

$$\mathbf{C_d} = k_d\, I_d\, \mathbf{l} \cdot \mathbf{n}$$

- k_d and I_d are the coefficients comes from material and light for controlling intensity (strength) of diffuse reflection respectively

Specular Reflection

- To model highlight occurs on smooth surfaces
- Reflection are concentrated on certain directions ONLY
- Highlights will be observed within certain angle to the materials but not other

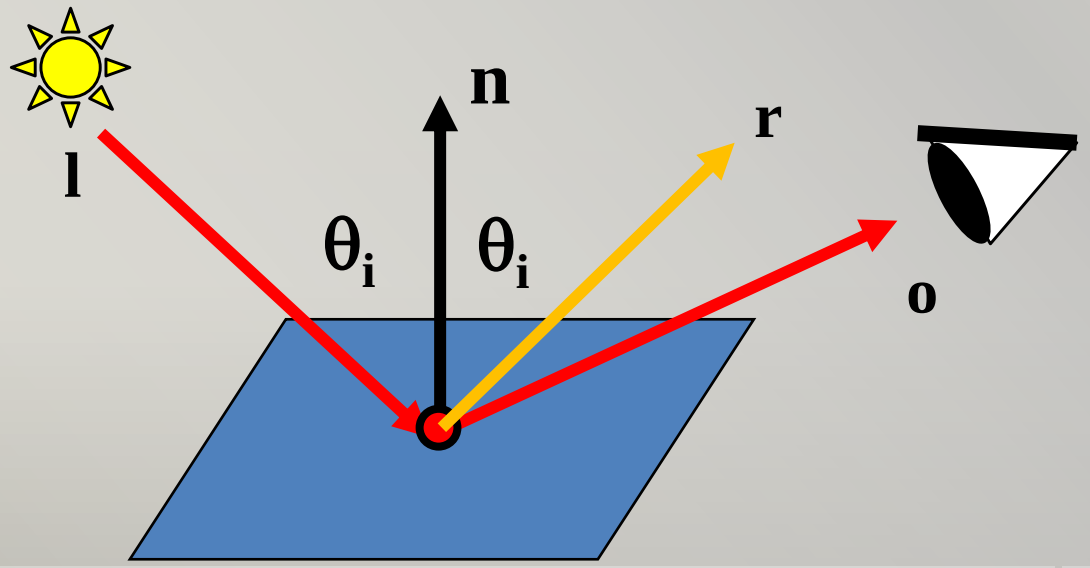


Specular Reflection

- It depends on the relative positions between
  - Light source,
  - object surface and
  - Observer
- The mathematical model is based on Phong model
  - proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased

Specular Reflection

- As we know, the ideal reflection should occur with the same angle (θ_i) to the normal, but on the other side
- This direction **r** has the strongest reflection from **l**

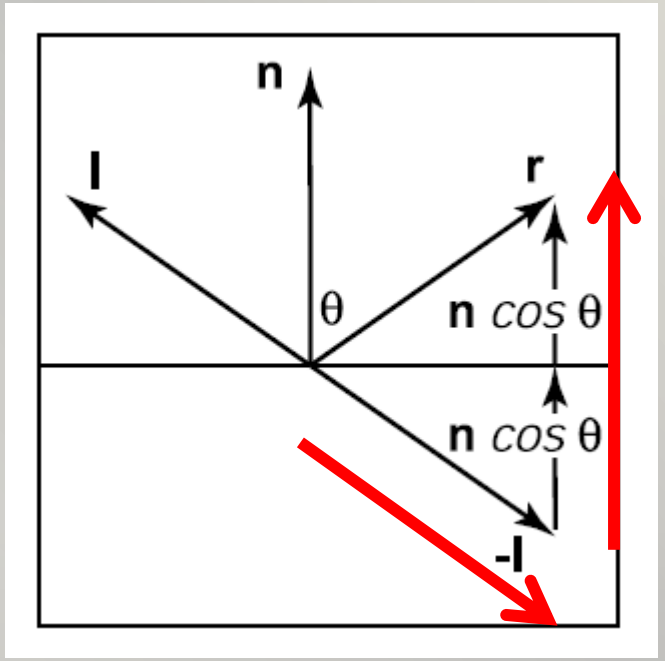


How to Compute Reflection vector

- The reflection vector **r** can be computed using **l** and **n** with the following vector computation:

$$\mathbf{r} = -\mathbf{l} + 2(\mathbf{n}\cos\theta)$$

$$= -\mathbf{l} + 2\mathbf{n}(\mathbf{l}\cdot\mathbf{n})$$

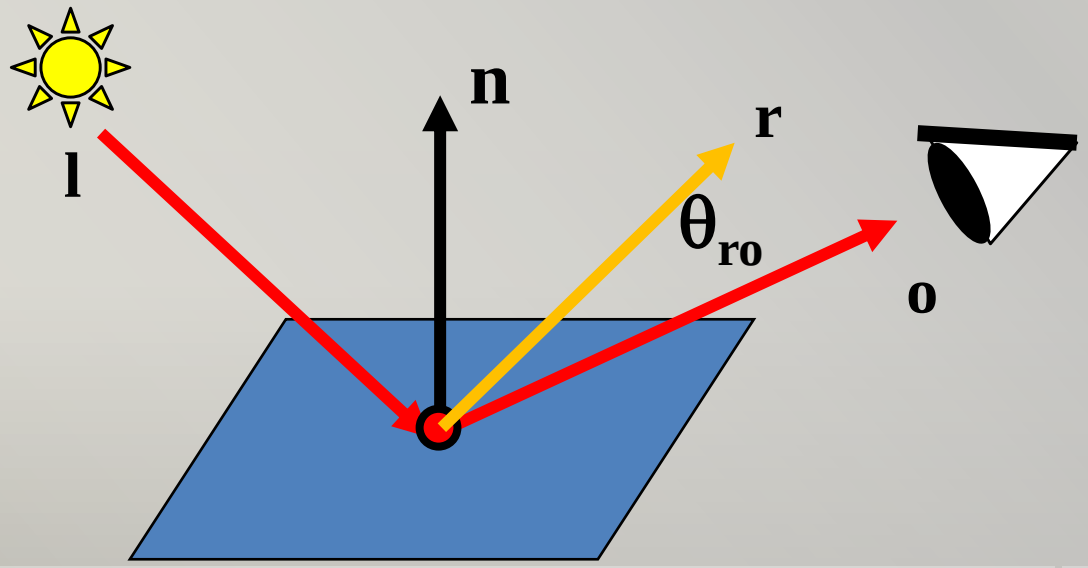


Specular Component

- The specular component $\mathbf{C_s}$ in the light model is fomulated as:

$$\mathbf{C_s} = k_s\, I_s\, (\mathbf{r} \cdot \mathbf{o})^{\alpha}$$

  - The term $\mathbf{r} \cdot \mathbf{o} = \cos \theta_{ro}$, corresponds to the deviation of observer from the maximum reflection

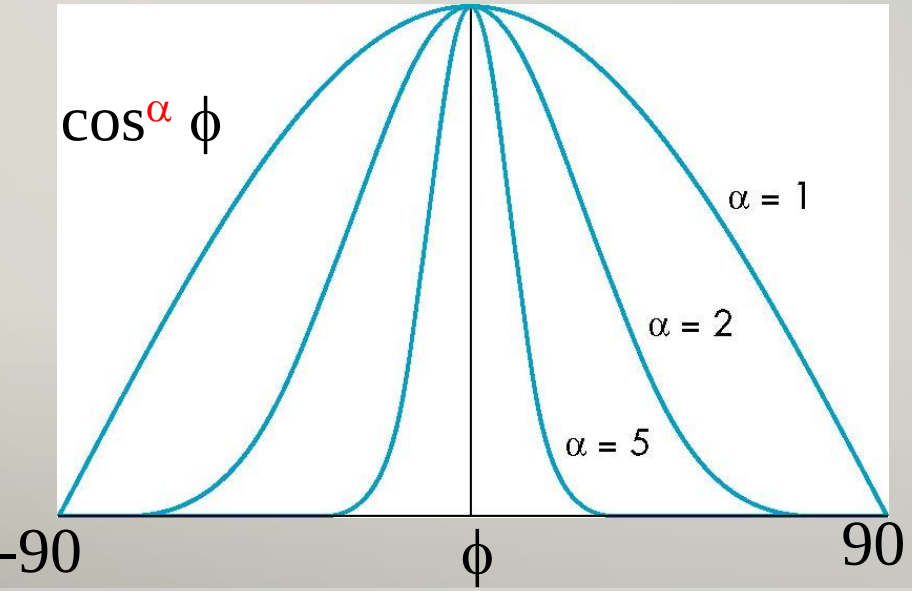


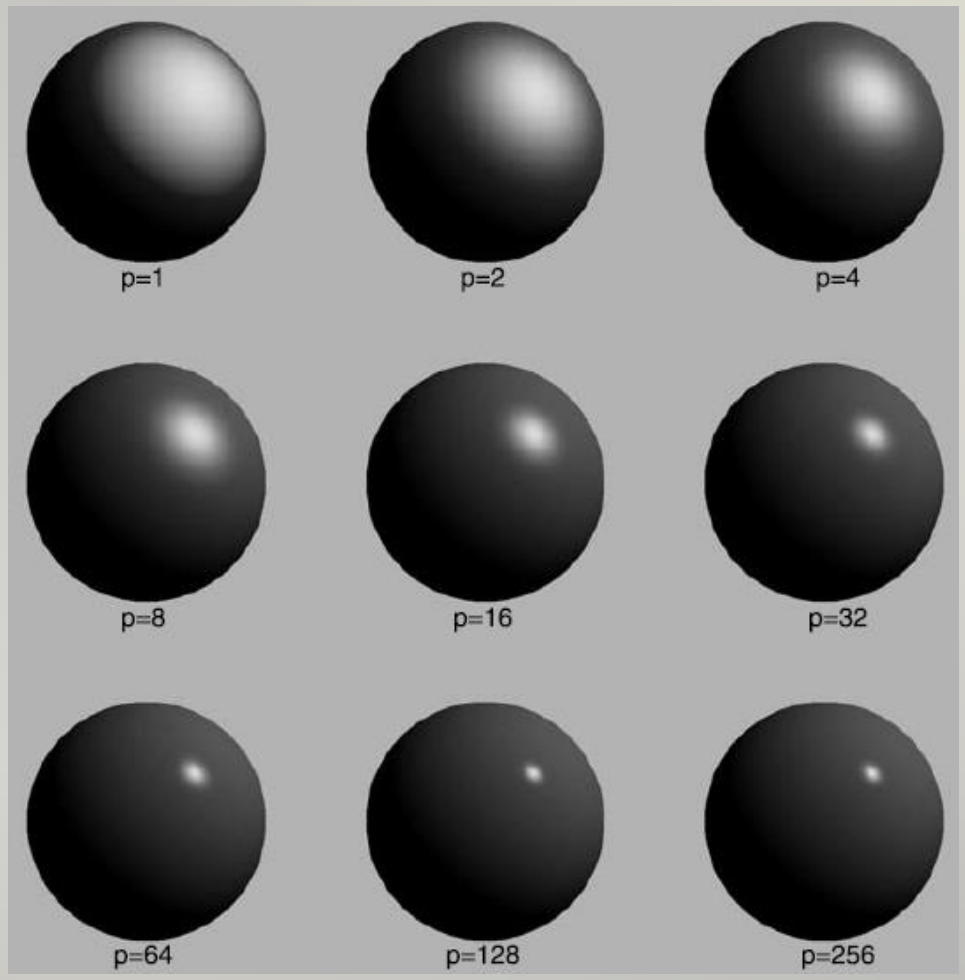
The Shininess Coefficient

- $\mathbf{r} \cdot \mathbf{o}$ alone will only depend on the angle θ_{ro}
- We can **NOT** control how the reflection is concentrated for certain material
- The term α will be able to let us control the shininess
  - Values of α between 100 and 200 correspond to metals
  - Values between 5 and 10 give surface that look like plastic

The Shininess Coefficient

- A larger α has a more concentrate region of highlight





Ambient Reflection

- The same thing as ambient light
- Can be considered as reflection of “background” light
- The ambient component in light model is only defined by the coefficients from the light and material

$$\mathbf{C_a} = k_a I_a$$

Adding up the Components

We can combine the 3 terms to form the Phong lighting model for a **single light source** and **single color component**:

$$I = \mathbf{C_d} + \mathbf{C_s} + \mathbf{C_a}$$

$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{r} \cdot \mathbf{o})^{\alpha} + k_a I_a$$

diffuse **specular** ambient

Here I computed is the intensity being reflected and seen by the observer

Light source Intensity

- The light model depends on a number of coefficients from
  - The light source intensity, and
  - The material property
- For light source intensity:
  - 9 coefficients counting 3 color channels (rgb)

Diffuse: I_{dr}, I_{dg}, I_{db},

Specular: I_{sr}, I_{sg}, I_{sb},

Ambient: I_{ar}, I_{ag}, I_{ab}

Material Properties

- For material properties, it matches those from light source intensity
  - 9 coefficients for 3 color channels

Diffuse: k_{dr}, k_{dg}, k_{db},

Specular: k_{sr}, k_{sg}, k_{sb},

Ambient: k_{ar}, k_{ag}, k_{ab}

  - Shininess coefficient α for specular term

Example

- We illustrate here the computation of the diffuse component with numbers
- Assume we have our normalized light intensity and material properties
  - $(I_{dr}, I_{dg}, I_{db}) = (1.0, 0.3, 0.0)$ | $(k_{dr}, k_{dg}, k_{db}) = (0.5, 1.0, 1.0)$
- And our lighting direction and normal
  - $\mathbf{l} = \langle 0.707, 0.707, 0.0 \rangle$ | $\mathbf{n} = \langle 0.0, 1.0, 0.0 \rangle$
  - So $\mathbf{l} \cdot \mathbf{n} = 0.0 + 0.707 + 0.0 = 0.707$
- $\mathbf{C_{dr}} = 0.5 * 1.0 * 0.707 = 0.3535$
- $\mathbf{C_{dg}} = 1.0 * 0.3 * 0.707 = 0.2121$
- $\mathbf{C_{db}} = 0.0 * 1.0 * 0.707 = 0.0$

$$\mathbf{C_d} = k_d I_d \; \mathbf{l} \cdot \mathbf{n}$$

Shading

- So far, we are considering the lighting model at a position
- Theoretically, we can compute reflected color on every positions on object
- However, how dense we need to apply the light model?
- On Each vertex? Each polygon ? Each surface? or Each pixel?

Shading

- In practice, there are 3 commonly used shading methods, and they are corresponding to different levels of applying the lighting model:
  - Flat Shading
  - Gouraud Shading
  - Phong Shading

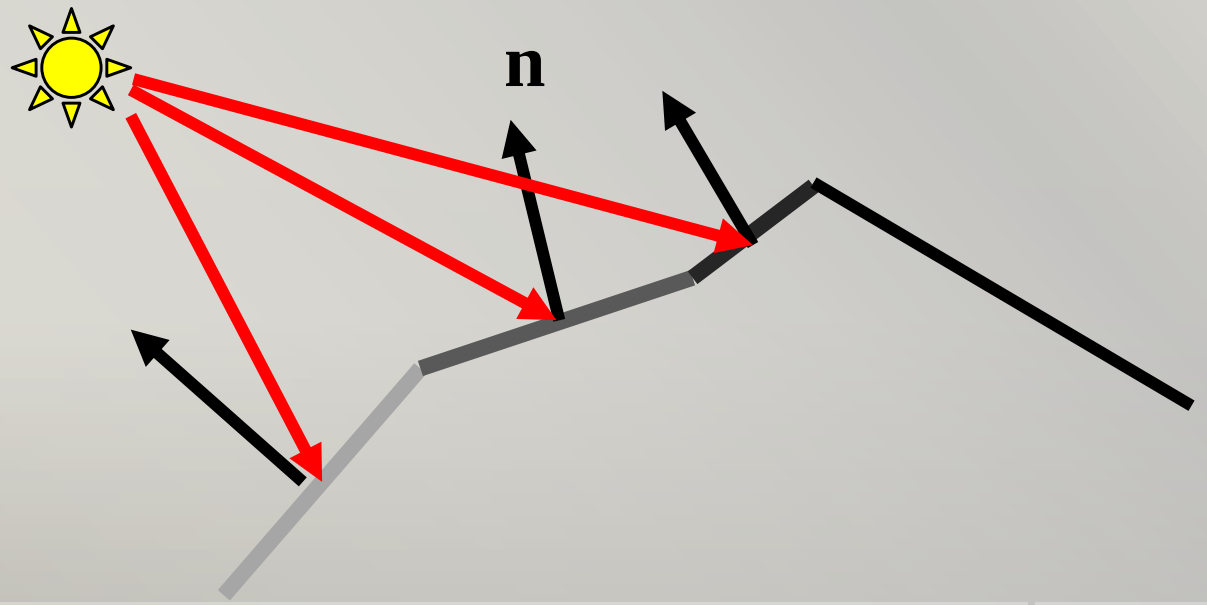
@Polygon

@Vertex

@Pixel (Fragment)

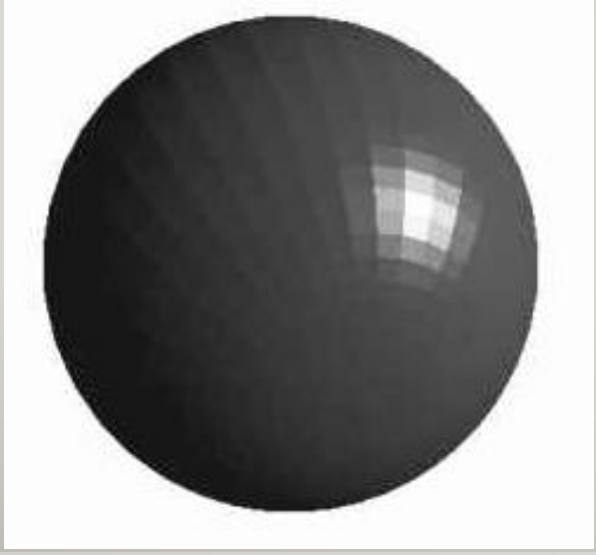
Flat Shading

- The whole polygon will be shaded with a single color
- As each polygon has a different normal, the shade will look different for different polygons in an object
  - Shades are computed by Phong light model



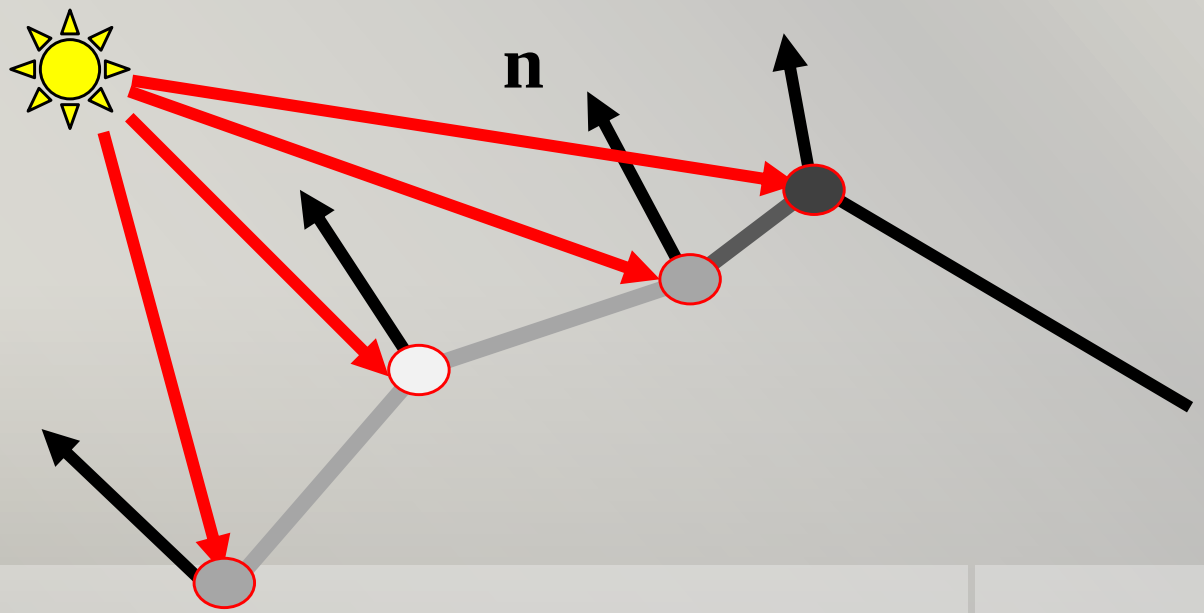
Flat Shading

- Usually, the shading will not look smooth enough
- It requires less computation
  - depends on number of polygons, which is relatively smaller
- With a densely divided surface, the result may look better. But it becomes more computational intensive



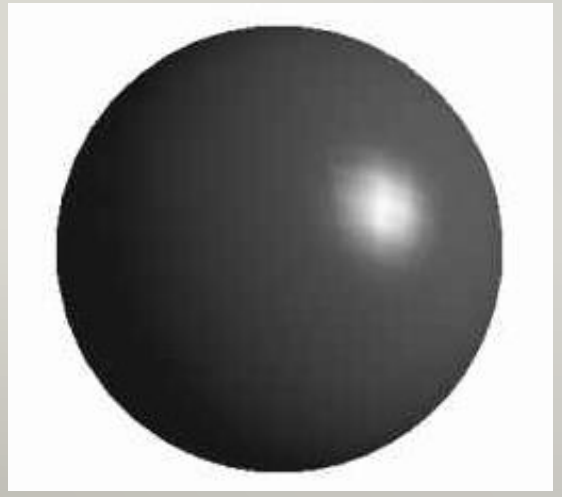
Gouraud Shading

- Lighting model are computed as each vertex of the polygon
  - Find average normal at each vertex (vertex normals)
  - Apply modified Phong model at each vertex
- **Interpolate** vertex shades across each polygon



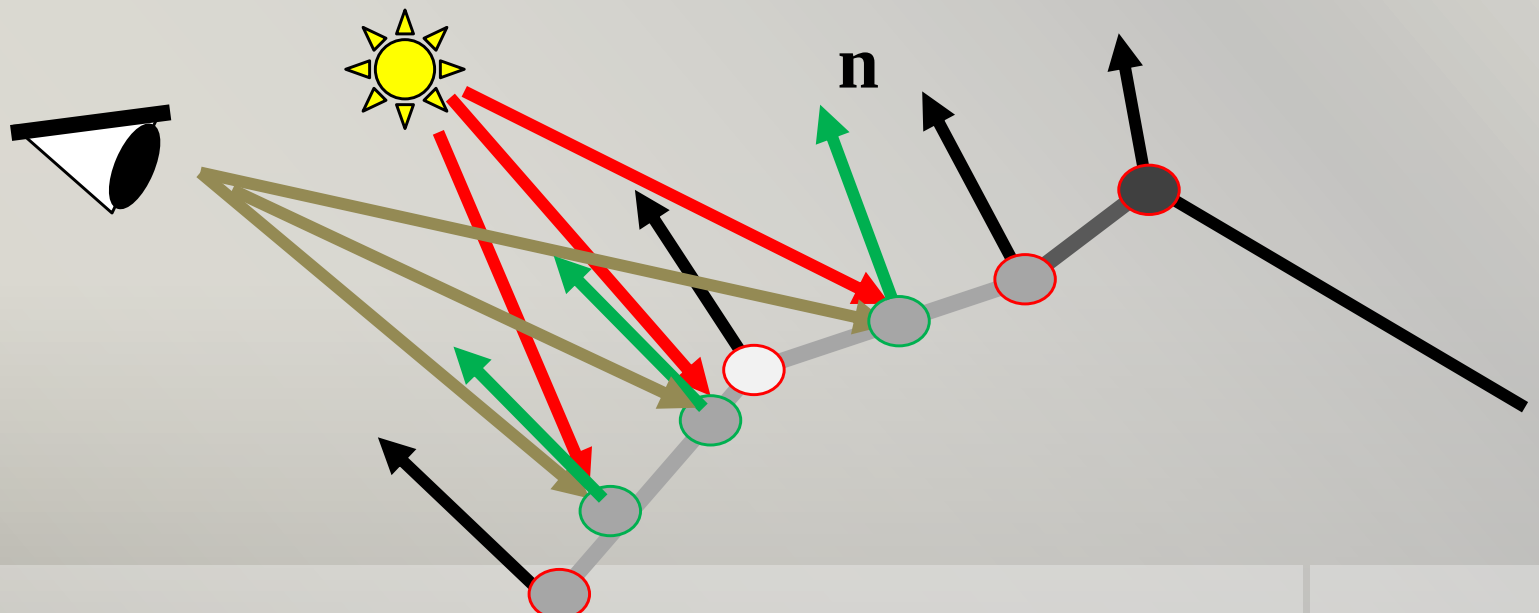
Gouraud Shading

- Usually looks smooth
- It applies the light model on each vertex
  - a bit more intensive than flat shade
- Moreover, interpolation is required
  - relatively less intensive than light model computation



Phong Shading

- Lighting model are computed at each pixel
  - Find vertex normals
  - Interpolate vertex normals across edges and then across polygon
  - Apply Phong model at each pixel (fragment)



Phong Shading

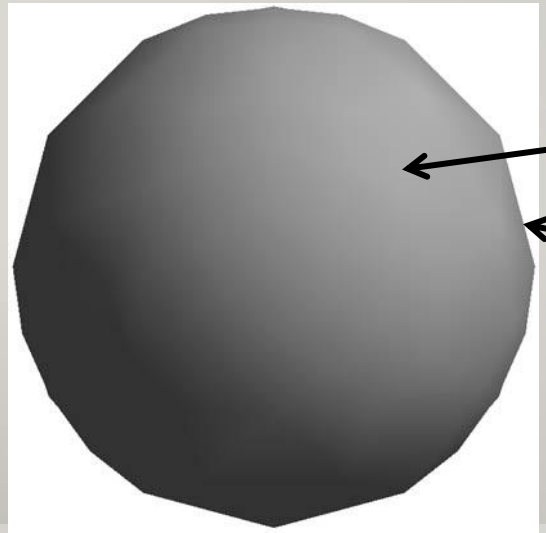
- Looks smooth and most precise
- The evaluation of normal at each viewable position from the camera is also computational intensive
- The computation of light model is proportional to the number of pixels (fragments)
- The best quality but the most expensive in computation

Comparison of Different Shadings

	Flat	Gouraud	Phong
Compute Lighting	Each polygon	Each vertex	Each Pixel
Interpolation	None	Yes	None
Speed	Very Fast	Fast	Slow
Quality	Low	Good	Best

Smooth Shading v.s. Silhouette

- The smooth shading only applies to the coloring on the surface
- But NOT the smoothness of silhouette (outline shape)
  - It will relate to the level of subdivision of the surface



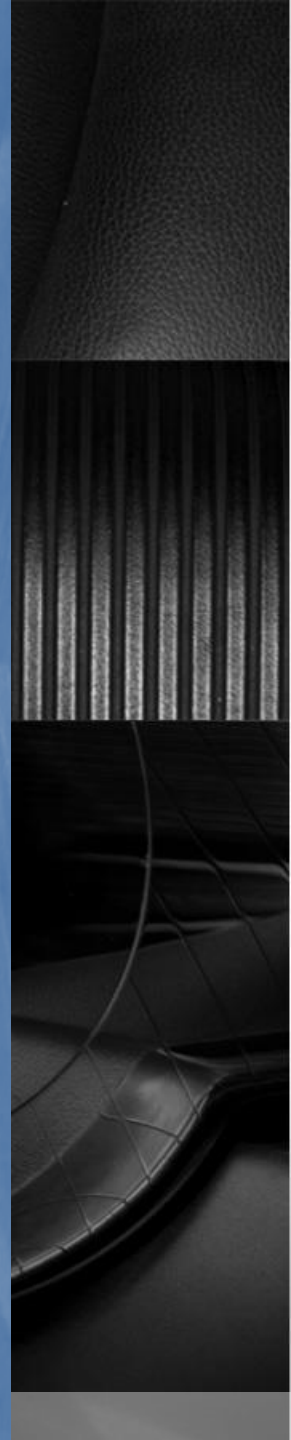
Advanced Lighting Effects

- Caustics
- Refraction
- Environmental Mapping
- Shadow

Summary

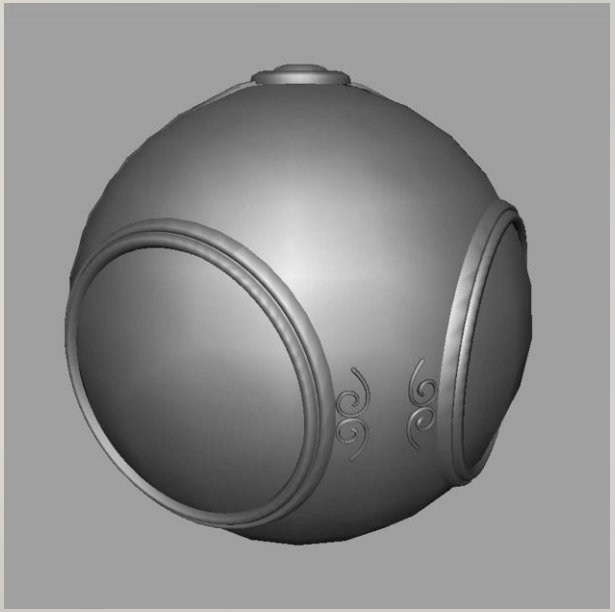
- We studied the interaction between light and surface in real world
- Single reflection is commonly used in realtime rendering
- Lighting model contains mainly 3 components
  - Diffuse, Specular and Ambient
- Commonly used shading methods include
  - Flat, Gouraud, and Phong shading

Texture Mapping and Antialiasing



Why Using Texture?

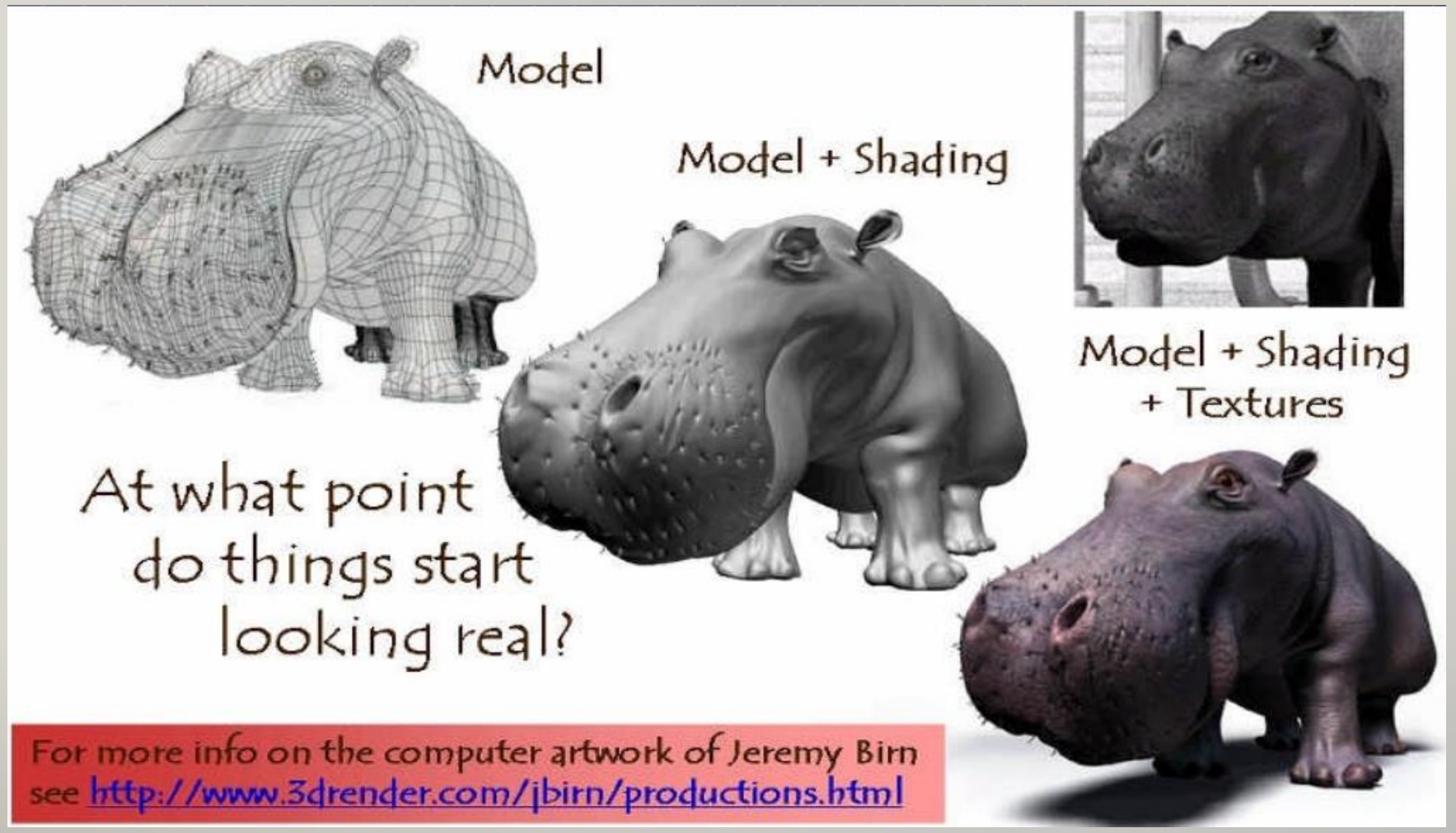
- You can add more details on your geometric model of the objects



geometric model

texture mapped

Why Using Texture?

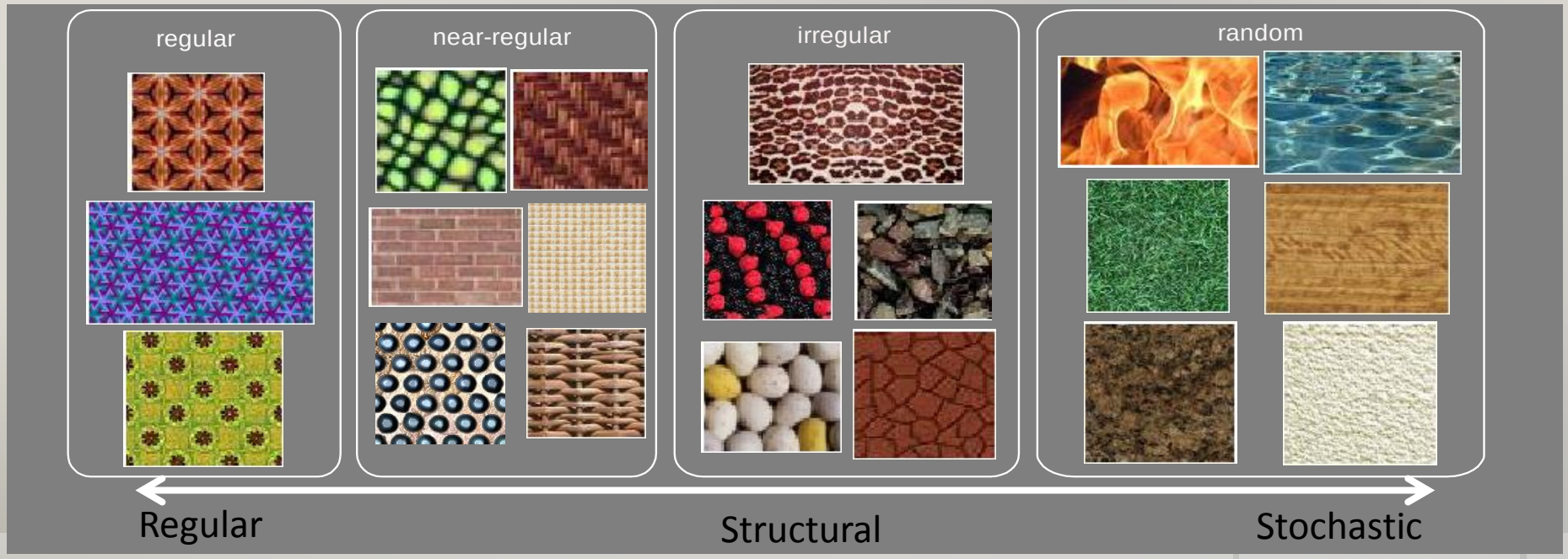


Why Using Texture?

- Use a fine subdivided object (i.e. more, smaller triangles) can add details to object surface. But …
  - Difficult to generate and assign the colors
  - Takes longer to render and more memory space
- Use texture is an effective but simple way to improve details
  - Can be prepared more easily
  - Stored once and reused for different objects
  - Easily compressed to reduce size
  - Rendered very quickly

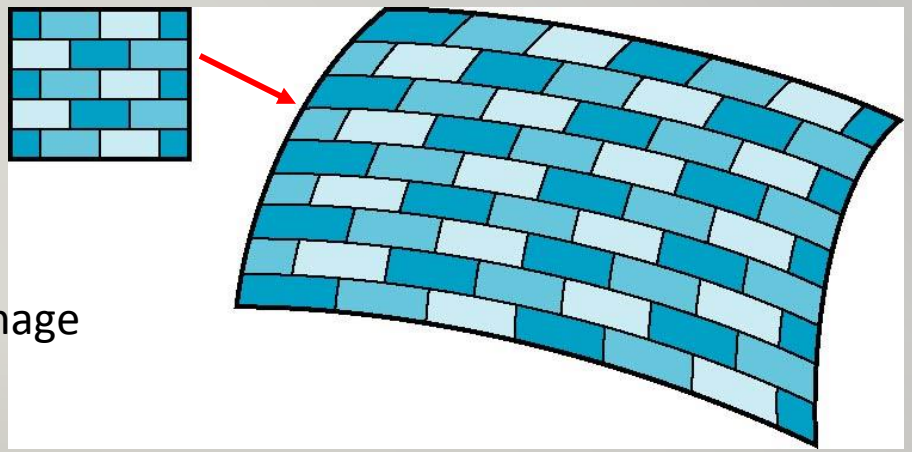
Texture is a 2D entity

- Textures are 2D images commonly used to define characteristics of a surface with following features
  - ✖ uniformity, density, coarseness, regularity, linearity, directionality, direction, frequency, and phase



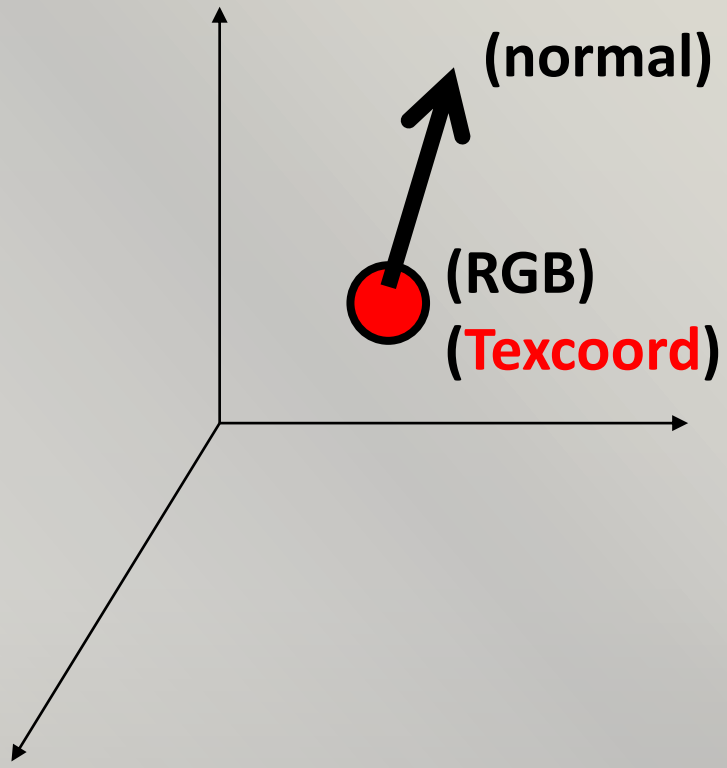
Texture Mapping

- The process of putting the 2D image in to the surface of a 3D object
  - Involved a correspondence (mapping) between coordinate systems



Vertices Data

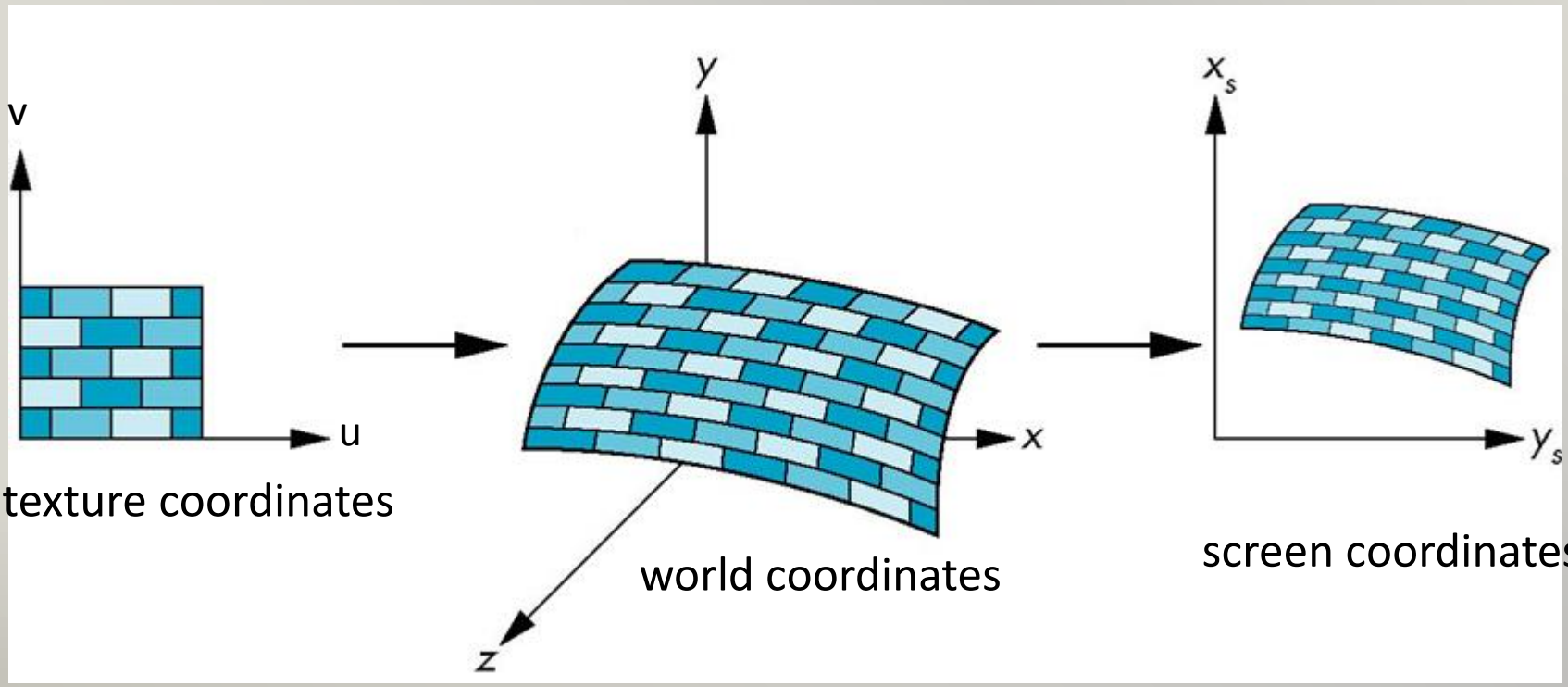
- Discussed in last lecture, associate with vertices are a number of properties:
  - Normal vector
  - Material / Color
  - **Texture coordinate**



Coordinate Systems / Space

- Texture space (UV)
  - Used to identify points in the image to be mapped
- Object space
  - Conceptually, where the mapping takes place
- Camera space and Screen space
  - Where the final image is really produced

Texture Mapping



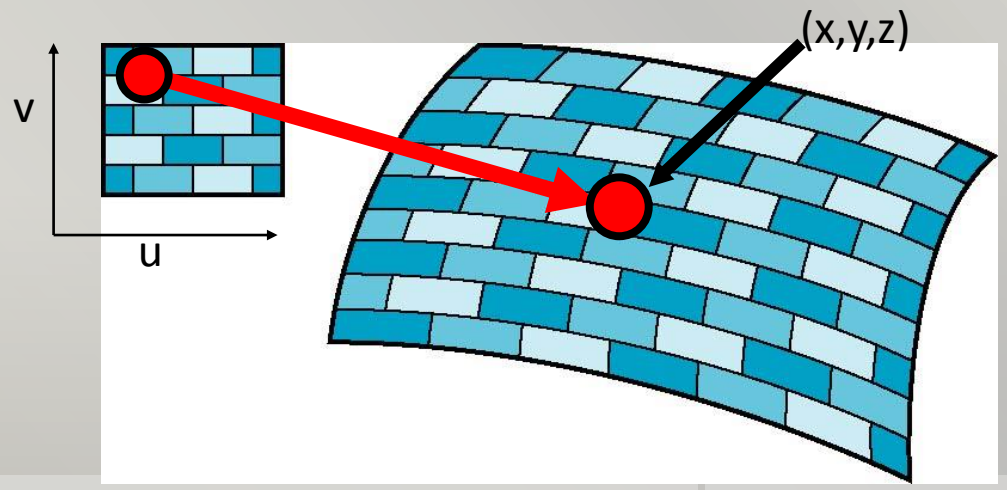
Mapping Functions

- Basic problem is how to define the mapping
- Consider mapping from texture coordinates to a point a surface
  - i.e. Given a pixel, we want to know to which point on an object it corresponds
  - More natural, similar to placing wallpaper on a wall
- We need 3 functions

$$x = fx(u,v)$$

$$y = fy(u,v)$$

$$z = fz(u,v)$$



Mapping Functions

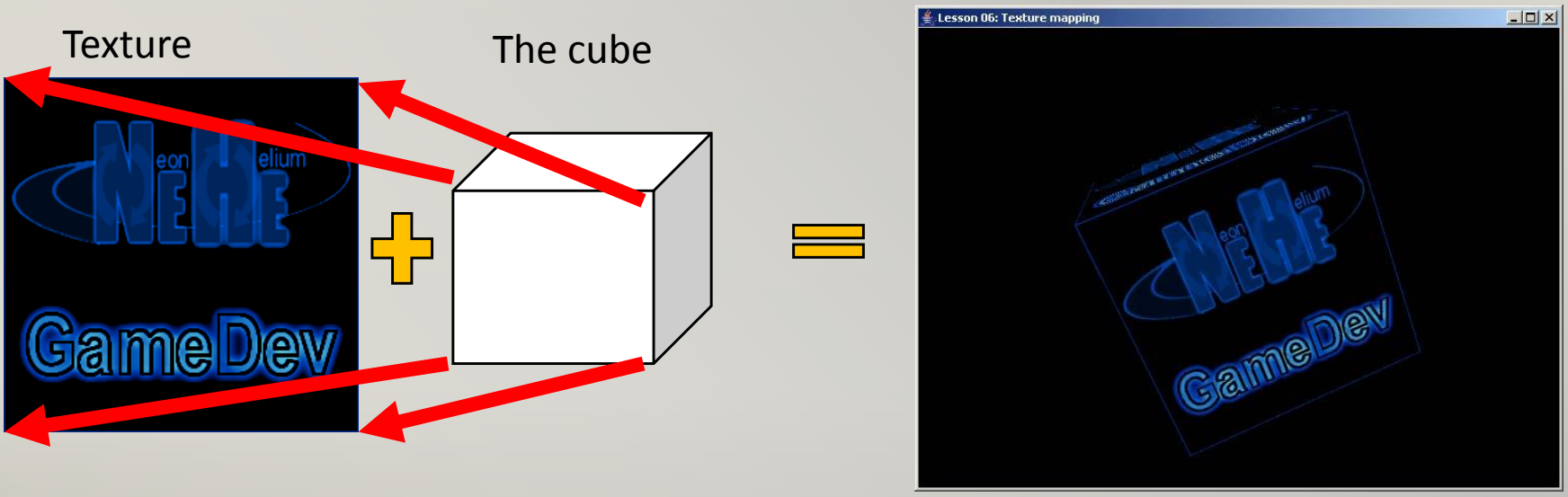
- Remember that we really want to define the texture coordinate for every vertex
  - i.e. Given a point on an object, we want to know to which point in the texture it corresponds
- So, commonly we need a map of the form

  u = fu(x,y,z)

  v = fv(x,y,z)
- It can be tedious and difficult for many cases, especially for **complex object surface**

Simple Rectangular Mapping

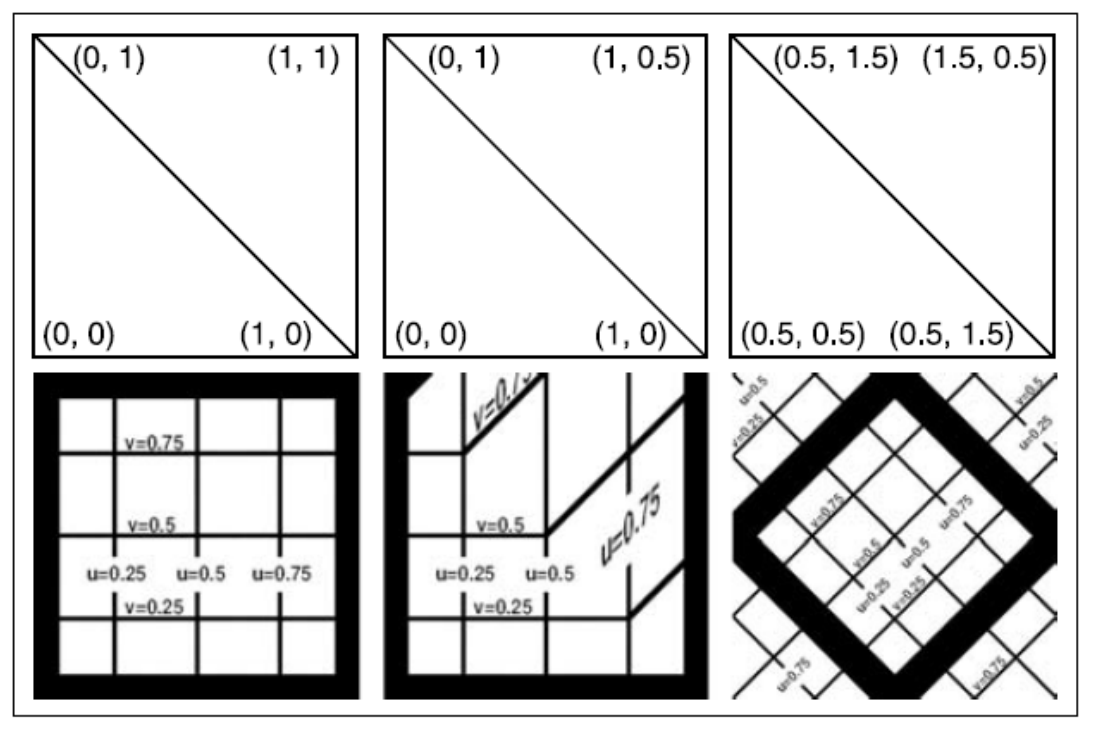
- A most simple case is to map the texture on a rectangle or cube
  - Because they are both planar entities
  - Simply match the corners



For more info, please refers to http://www.java-tips.org/other-api-tips/jogl/texture-mapping-nehe-tutorial-jogl-port.html

Simple Rectangular Mapping

- View from a certain face of the cube (e.g. z=0), the mapping happens like:
- xy(0,1) ➡ uv(0,1) ; xy(0,0) ➡ uv(0,0) ….

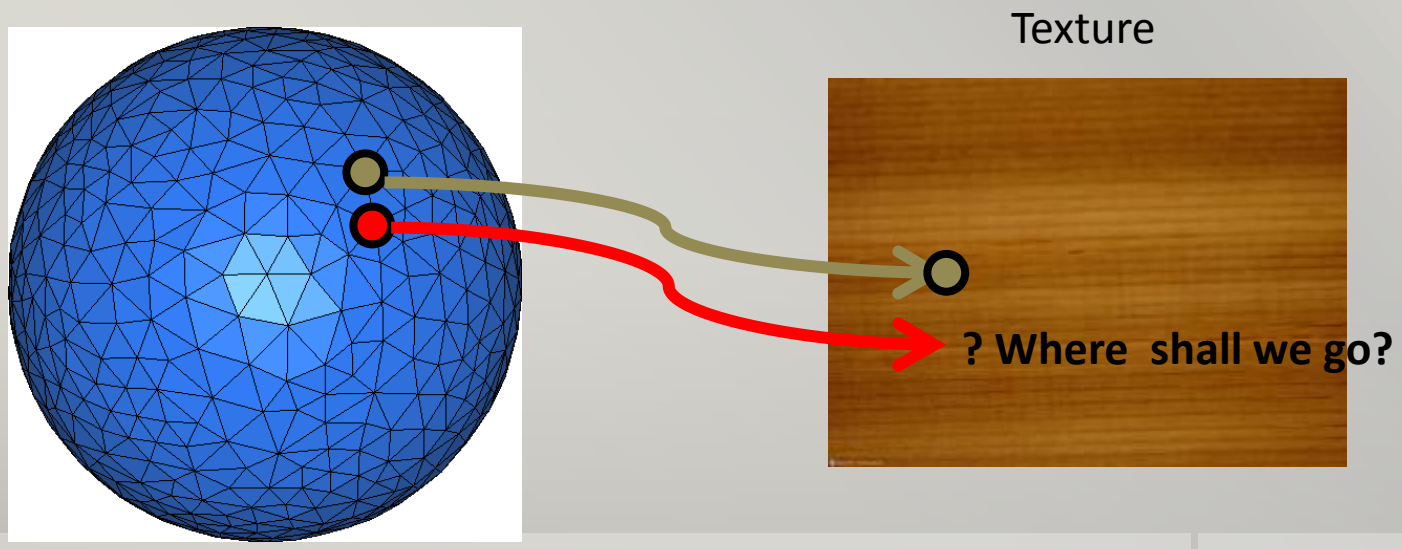


Texture coordinates assigned on 4 corners

How it looks like after mapping

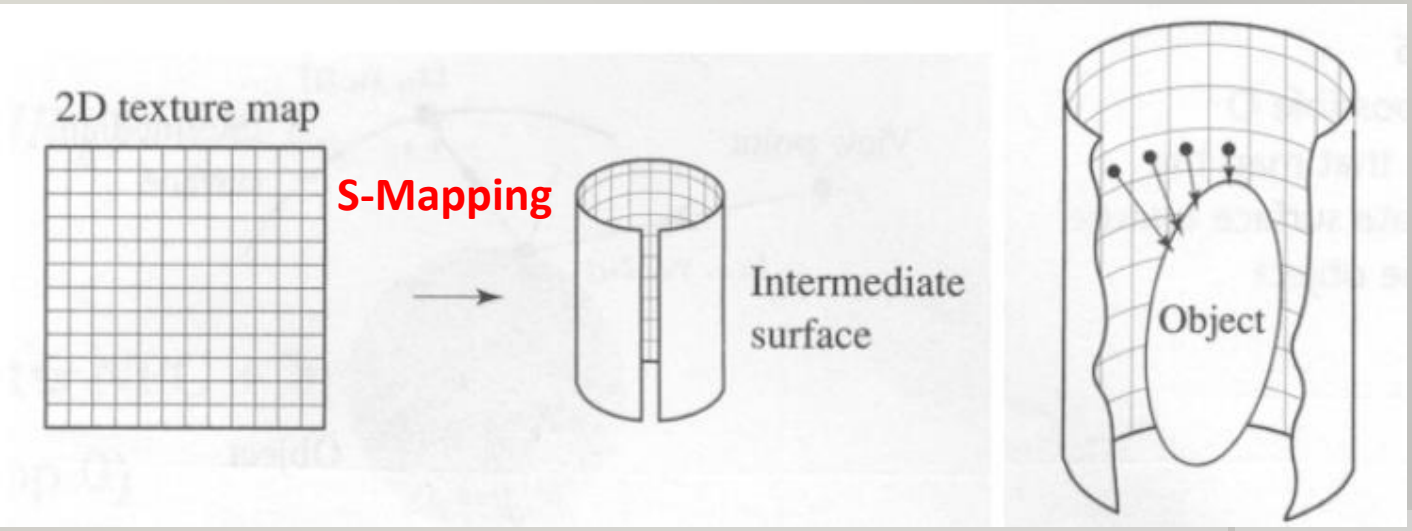
Two-stage mapping

- However, it may become non-trivial to map on a complex shape
  - It may be easy to assign the first coordinate, however, it comes to problem how to assign the nearby ones



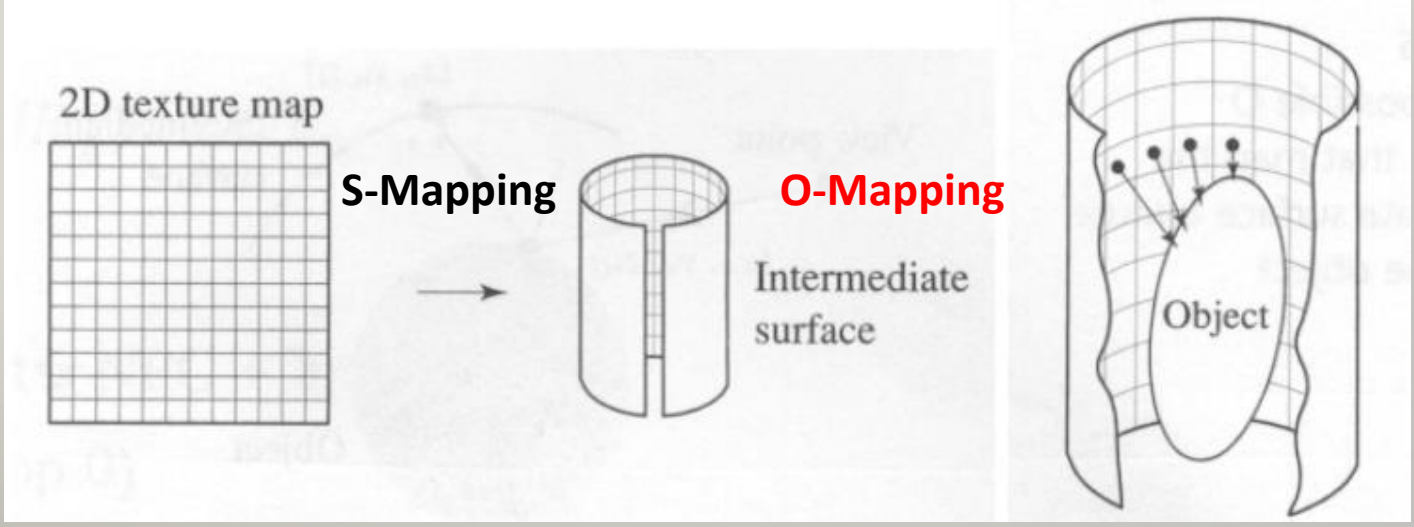
Two-stage mapping

- One solution to the mapping problem is using a **two-stage mapping**
- First map the texture to a simple proxy surface
  - S-Mapping
  - Example: map to cylinder, sphere, and plane, etc.



Two-stage mapping: O-Mapping

- The second step is to map from proxy object to actual object
  - O-Mapping
  - Usually the actual object has a more complex surface than the proxy object



S-Mapping: Cylindrical Mapping

- The mapping is relatively simple, like wrapping a paper around a cylinder

  $x = r \cos(2\pi u)$

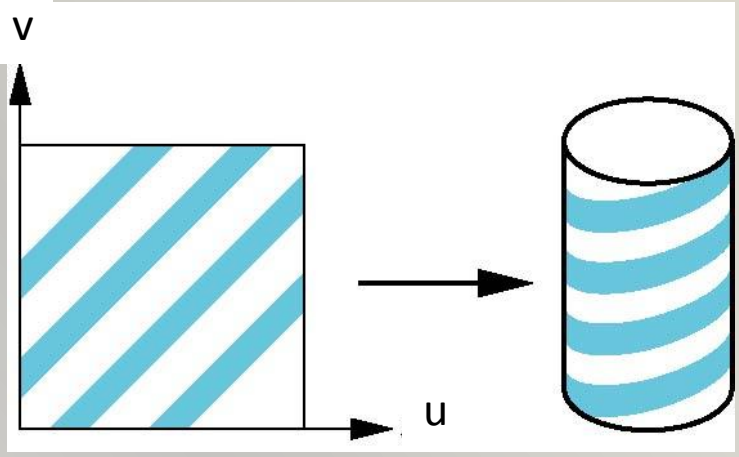
  $y = hv$

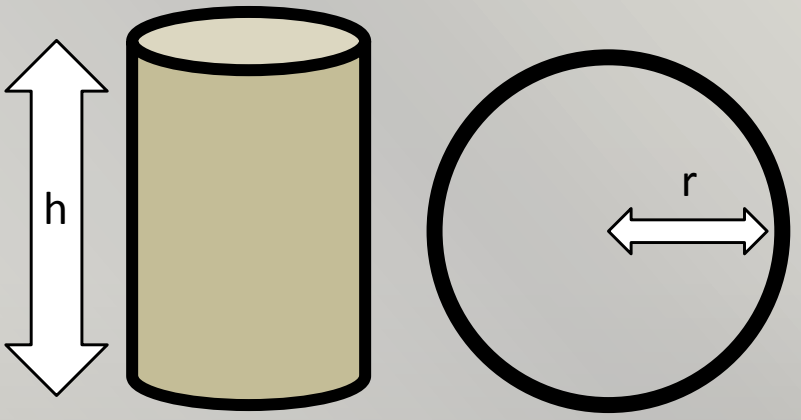
  $z = r \sin(2\pi u)$

- The inverse map will be

  $u = \arccos(x/r)(2\pi u)$

  $v = y$





Top view

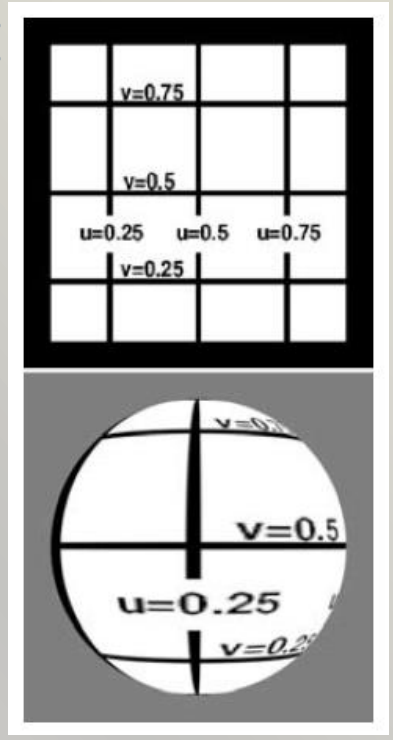
S-Mapping: Spherical Map

- In a similar manner to the cylinder but have to decide where to put the distortion
- Spheres are used in environmental ma

$$x = r \cos 2\pi u$$

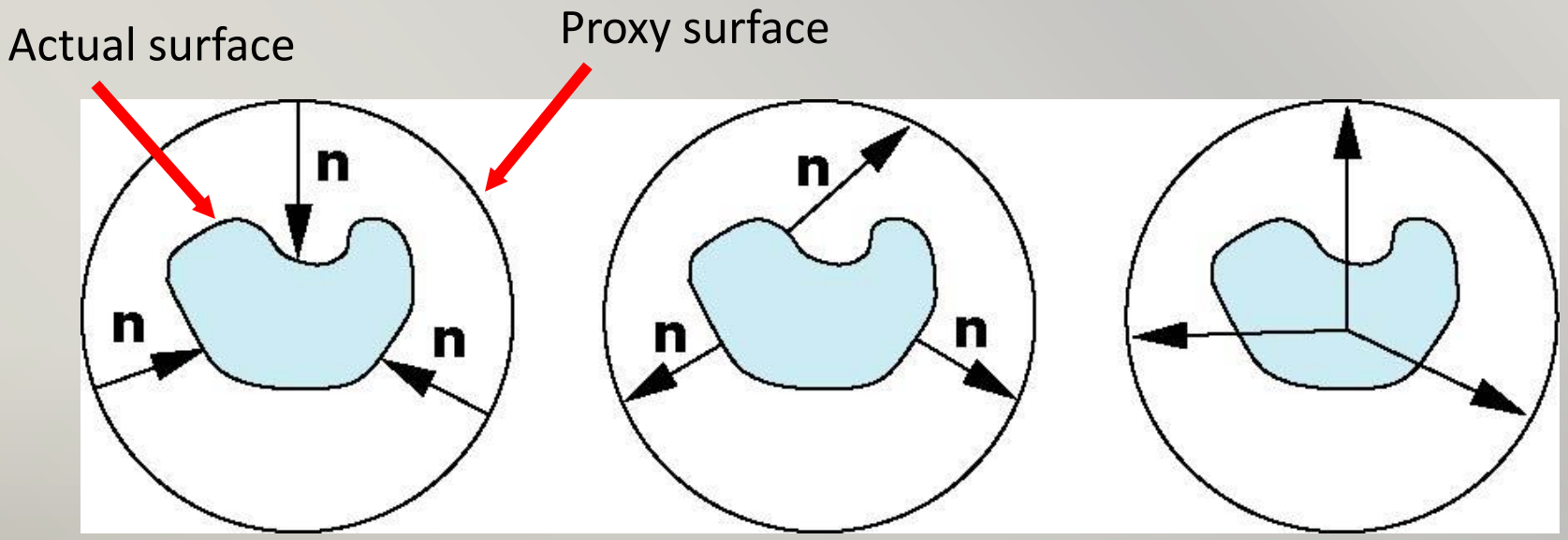
$$y = r \sin 2\pi u \cos 2\pi v$$

$$z = r \sin 2\pi u \sin 2\pi v$$



O-Mapping

- Three common ways to assign coordinates from proxy to actual surface
  - Normals from proxy to actual
  - Normals from actual to proxy
  - Vectors from center of proxy



Two-stage mapping

- However, no matter which method to use, the o-mapping step is computational heavy and time consuming
  - Requires many ray-object interception computations
  - Proportional to the number of vertices
- In summary
  - S-mapping: maps from texture space to a simple intermediate surface, such as a cylinder or sphere

    T(u,v) -> T'(x', y', z')
  - O-mapping: maps the 3D texture pattern onto the 3D object surface

    T'(x',y',z') -> O(x,y,z)

Sample Results

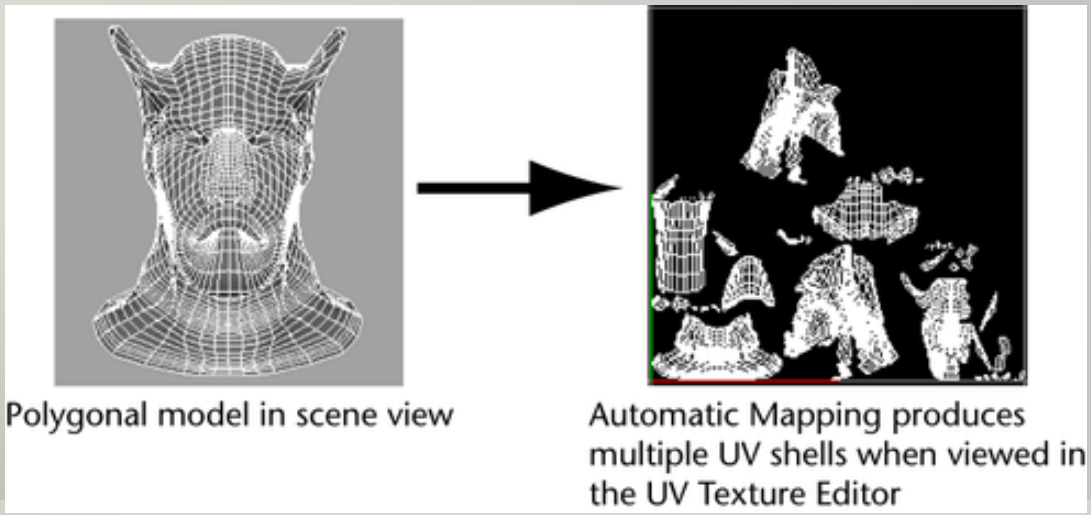
S-Map : Cylindrical

S-Map : Spherical

S-Map : Plane

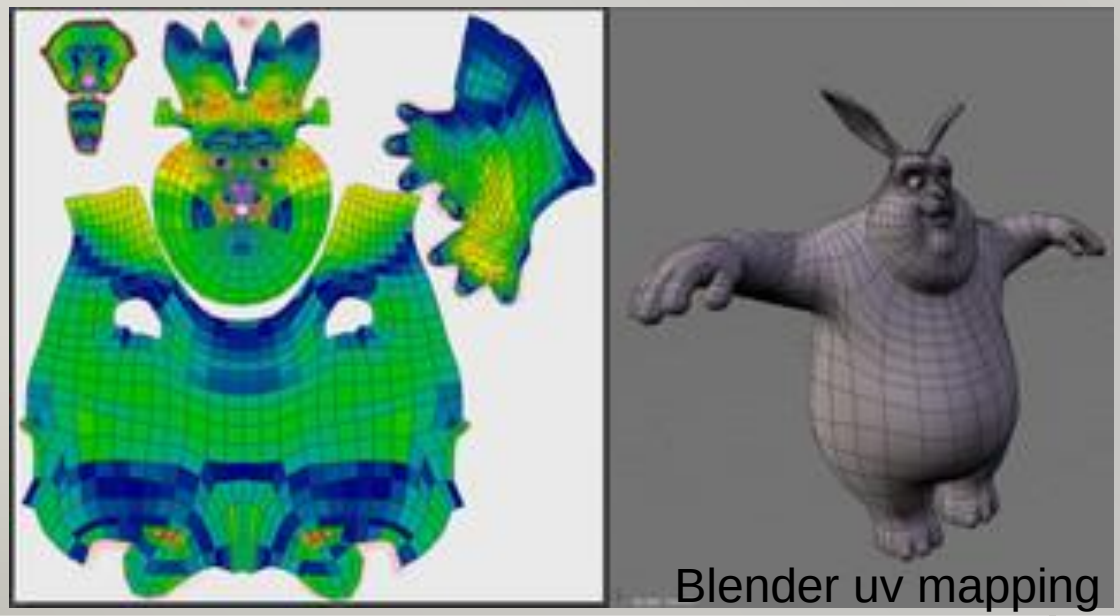
UV mapping

- A major drawback of two-stage mapping is
  - lack of flexibility
  - May introduce strong distortion
  - The texture assignment is sometimes unexpected
- Many existing software remain the work of texture assignment to the user



UV mapping

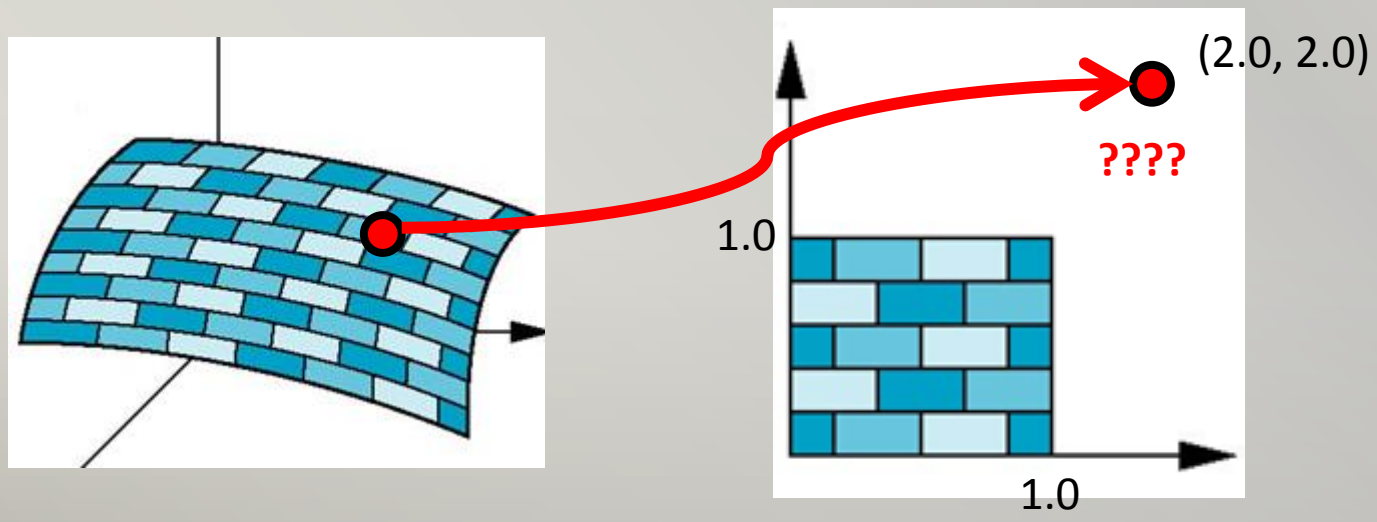
- Surfaces are unfolded and placed on a 2D texture
- Many discontinuous patches may appear
  - Much tedious to edit



Blender uv mapping

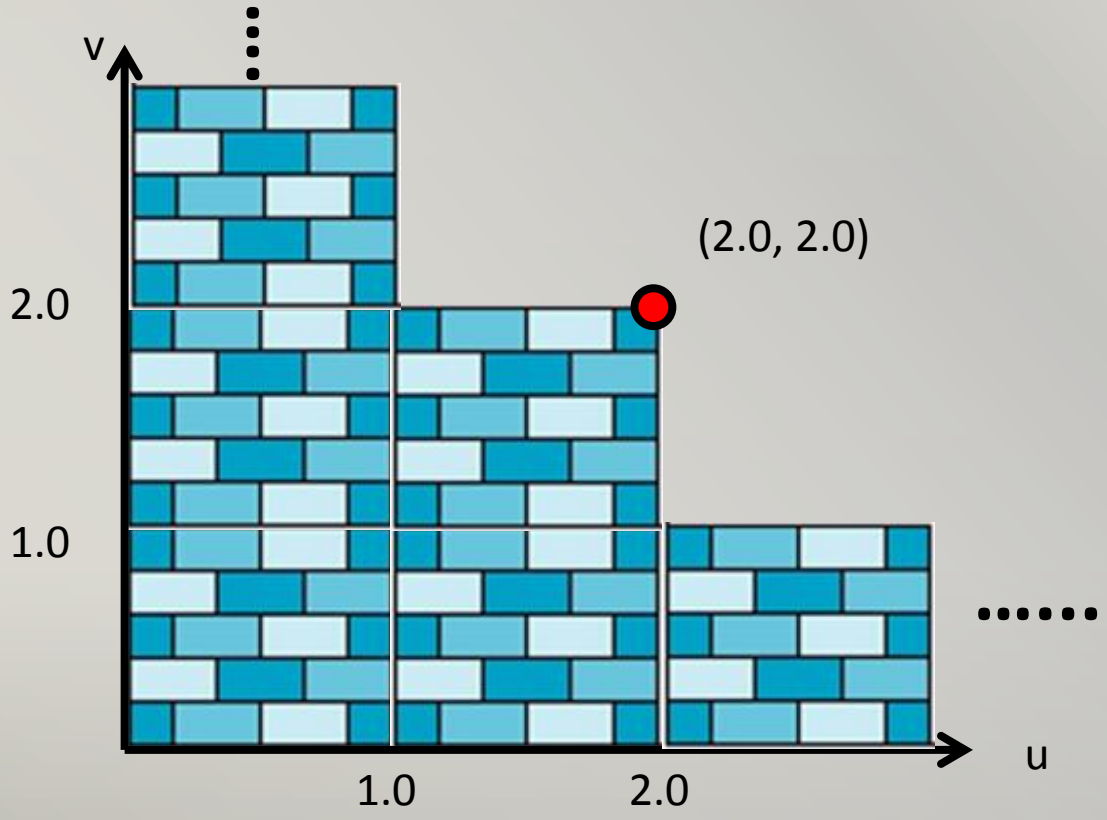
Texture Coordinate

- Consider the case when we assign a texture coordinate out of the range of u,v (usually (0.0,1.0))
  - E.g. uv(2.0, 2.0)
- What will you expect to happen ?



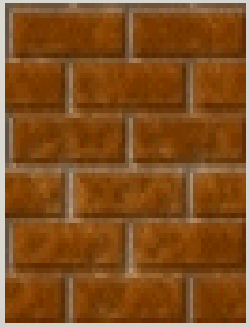
Texture Coordinate - Tiling

- The most commonly used way is tiling
  - Like repeating the texture infinitely

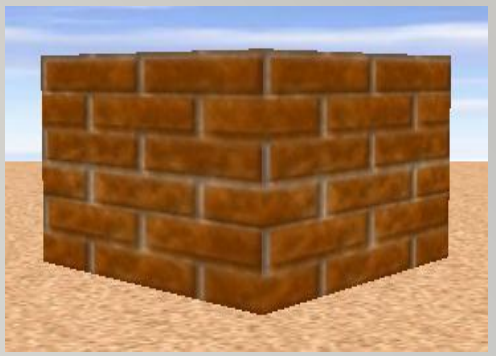


Texture Coordinate - Tiling

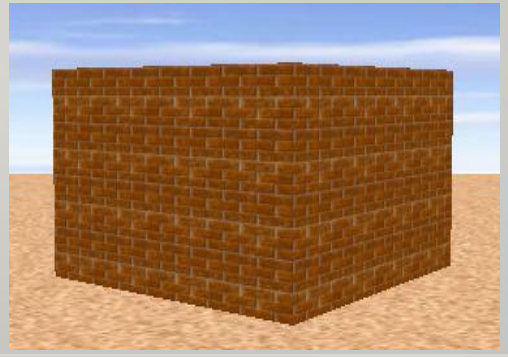
- A typical example is the brick pattern
- We can use a small texture and “tile” it across the wall
- Tiling allows you to scale repetitive textures to make texture elements just the right size.



Texture



Without Tiling

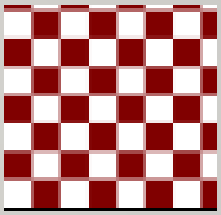


With Tiling

Texture Coordinate– Clamping

- Another alternative to tiling is clamping
  - Stop repeating
  - Simply do nothing
- In OpenGL, it will repeat the last row or column of pixels in the texture when out of the (0-1) texture coordinates

Texture



http://www.it.hiof.no/~borres

Summary

- Texture mapping is commonly used way to add details to object
- We have studied the methods to assign texture coordinates to objects
  - Direct UV mapping
  - Two-stage mapping