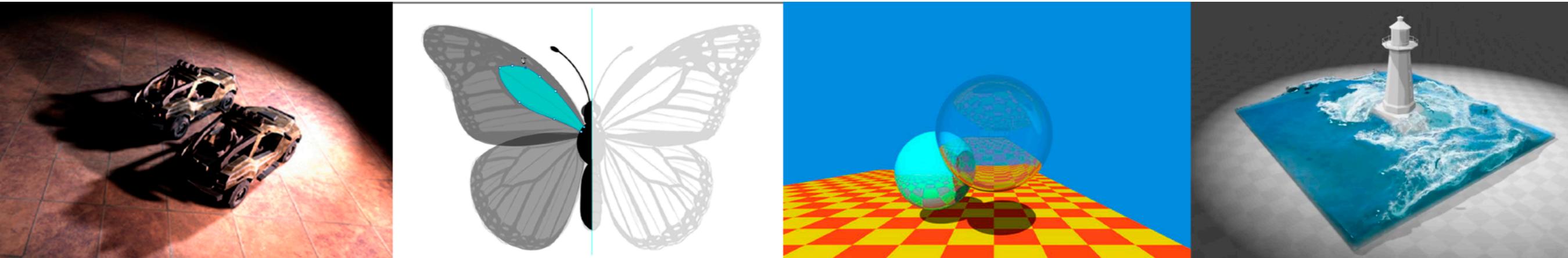


# Introduction to Computer Graphics

GAMES101, Lingqi Yan, UC Santa Barbara

## Lecture 8: Shading 2

(Shading, Pipeline and Texture Mapping)

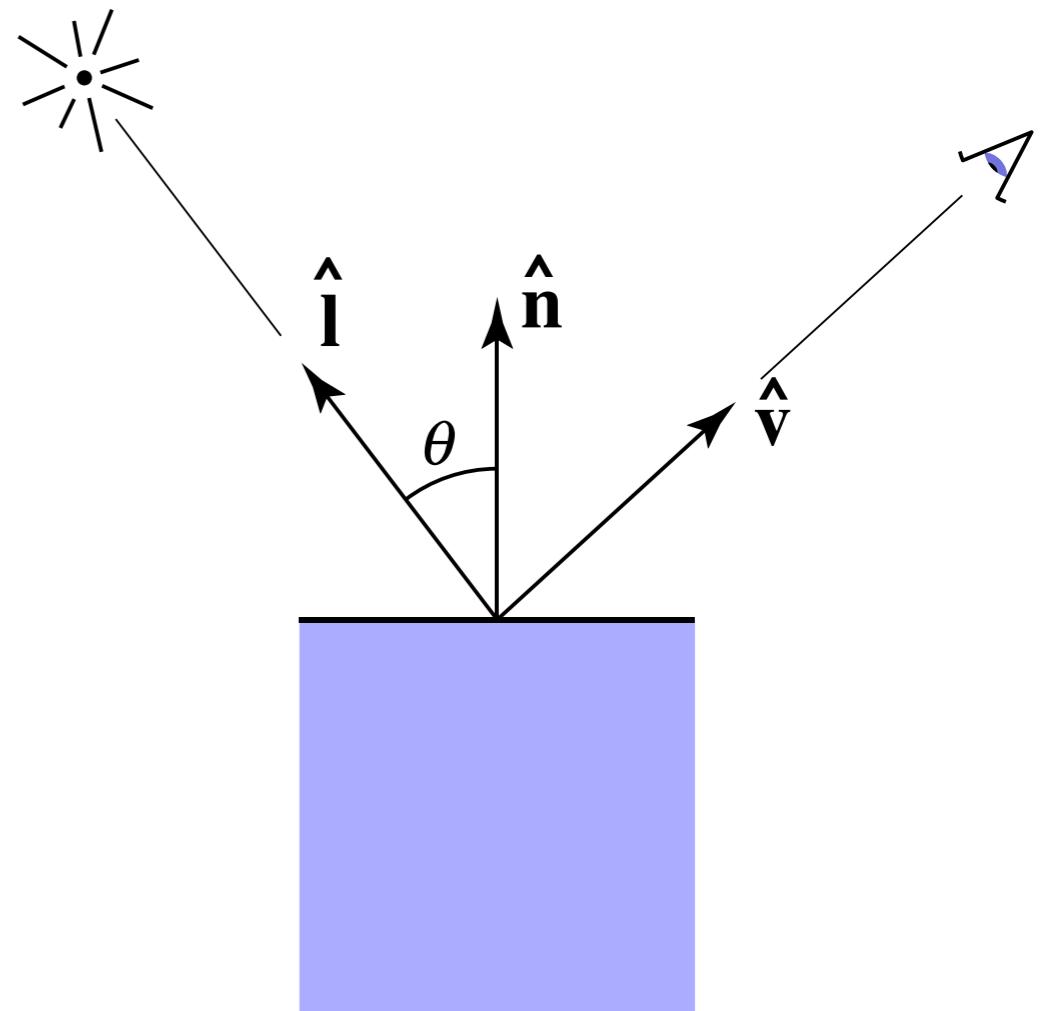


# Announcements

- Homework 2
  - 45 submissions so far
  - Upside down? No problem
  - Active discussions in the BBS, pretty good
- Next homework is for shading
- Today's topics
  - Easy, but a lot

# Last Lecture

- Shading 1
  - Blinn-Phong reflectance model
    - Diffuse
    - Specular
    - Ambient
  - At a **specific shading point**

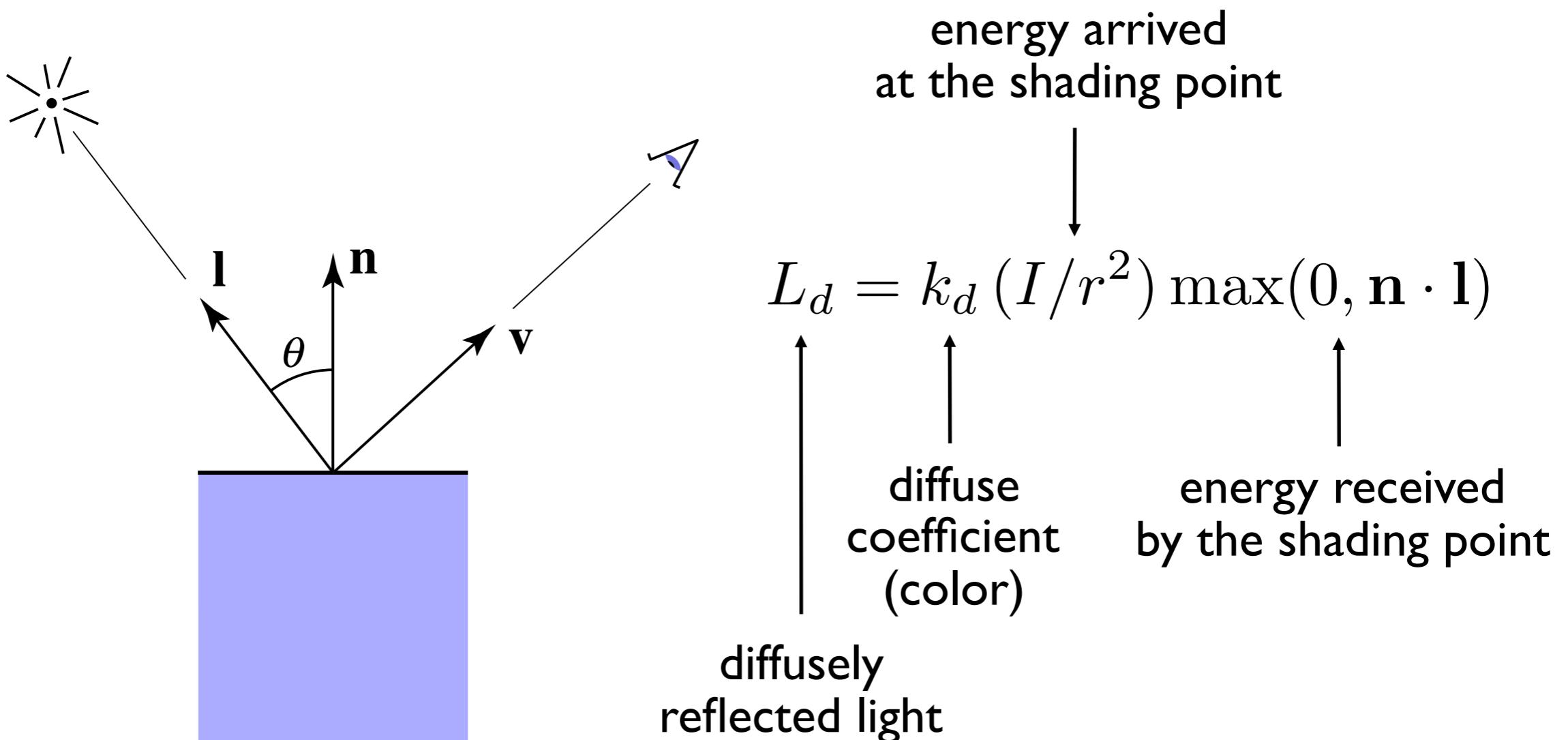


# Today

- Shading 2
  - Blinn-Phong reflectance model
    - Specular and ambient terms
  - Shading frequencies
  - Graphics pipeline
  - Texture mapping
  - Barycentric coordinates

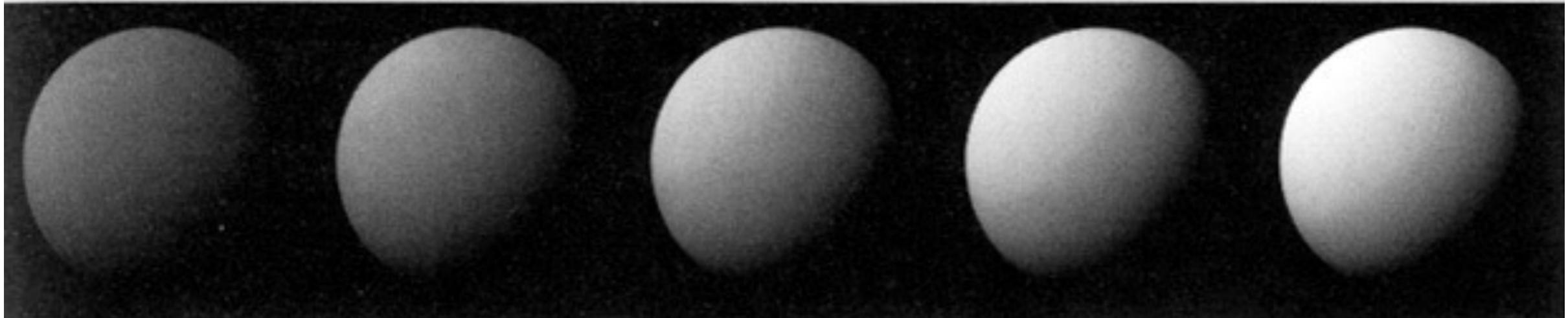
# Recap: Lambertian (Diffuse) Term

Shading **independent** of view direction



# Recap: Lambertian (Diffuse) Term

Produces diffuse appearance



$$k_d \longrightarrow$$

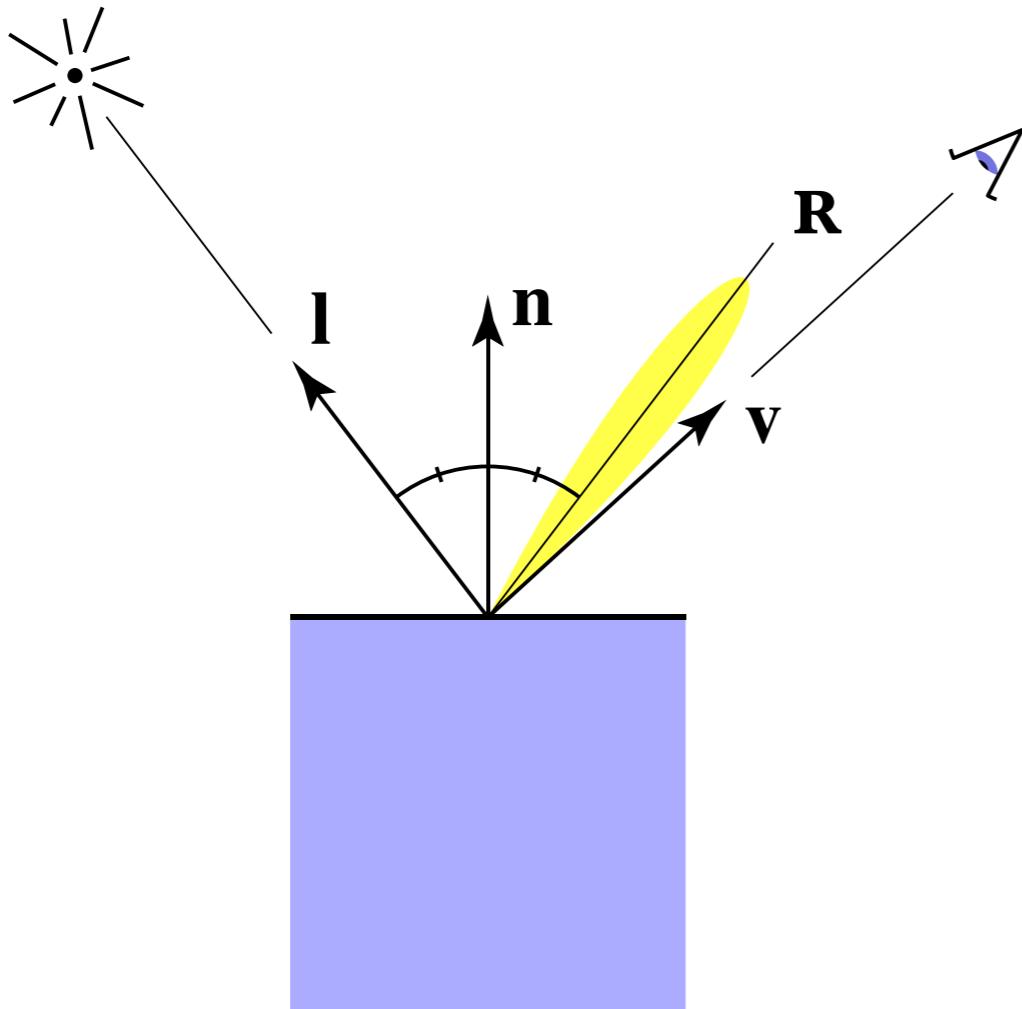
[Foley et al.]

高光项

# Specular Term (Blinn-Phong)

Intensity **depends** on view direction

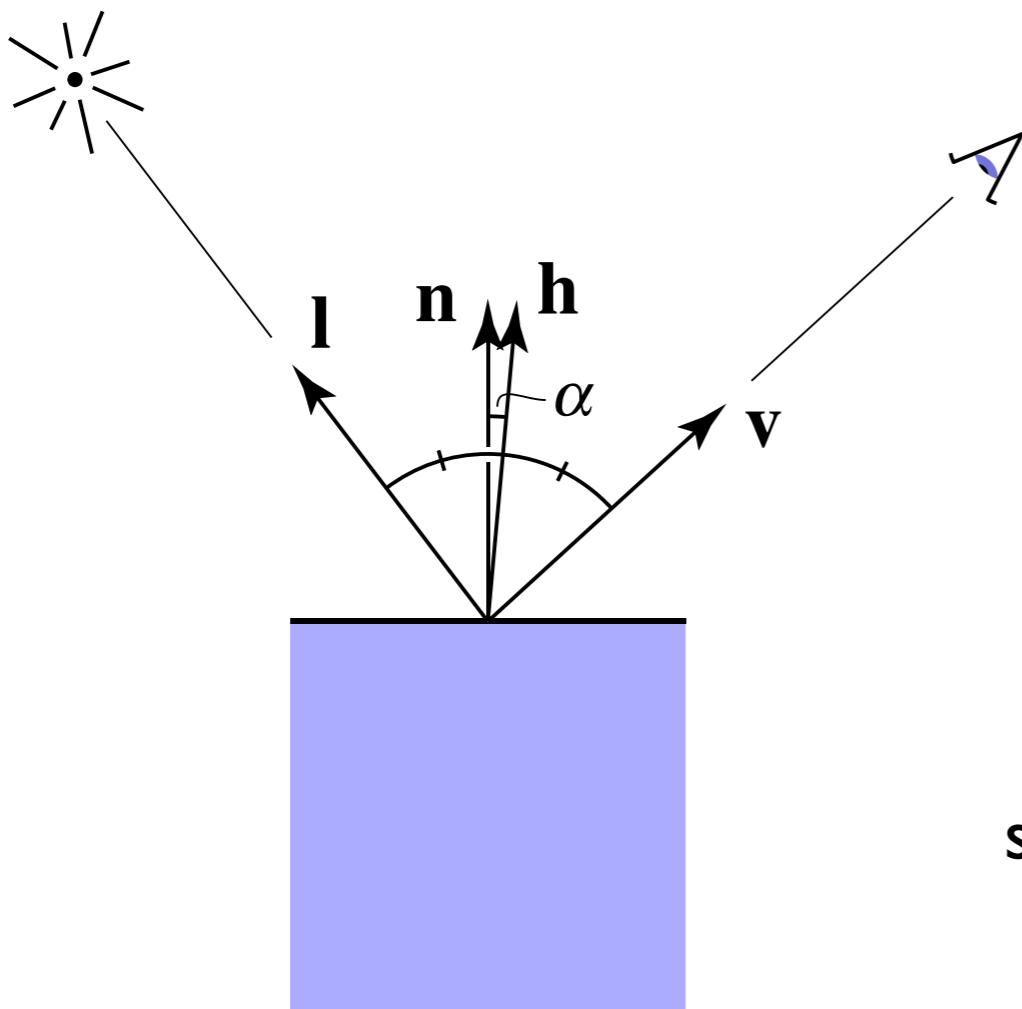
- Bright near mirror reflection direction



# Specular Term (Blinn-Phong)

$\mathbf{v}$  close to mirror direction  $\Leftrightarrow$  **half vector** near normal  
半程向量

- Measure “near” by dot product of unit vectors



$$\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l})$$

(半程向量)

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

$$L_s = k_s (I/r^2) \max(0, \cos \alpha)^p$$

$\uparrow$

$$= k_s (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

$\uparrow$

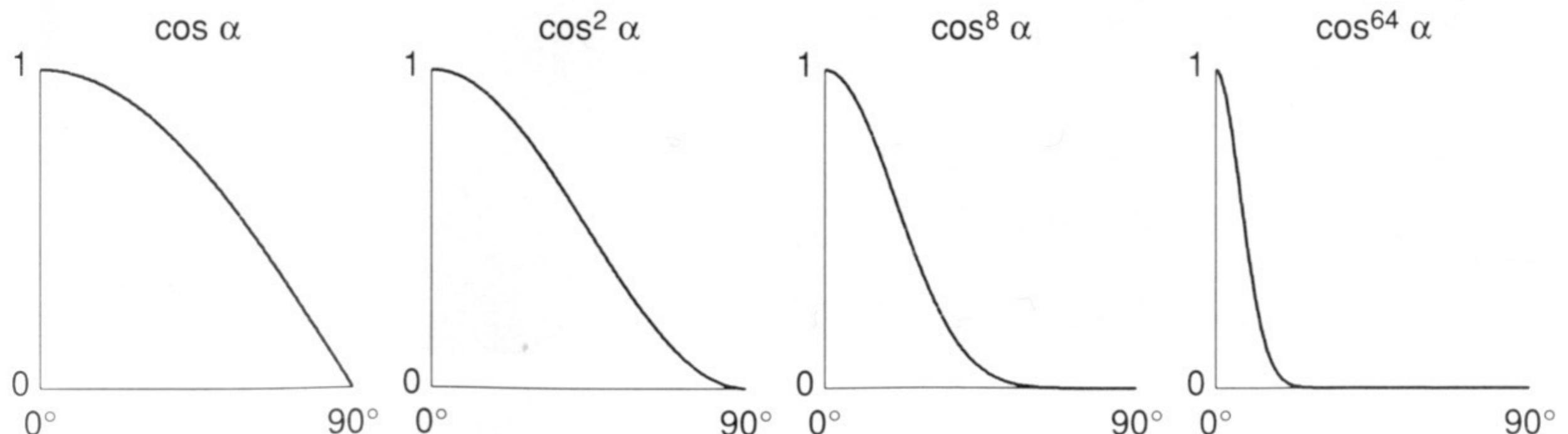
specularly reflected light      specular coefficient

半程向量 好算

# Cosine Power Plots

Increasing p narrows the reflection lobe

α容忍度太高

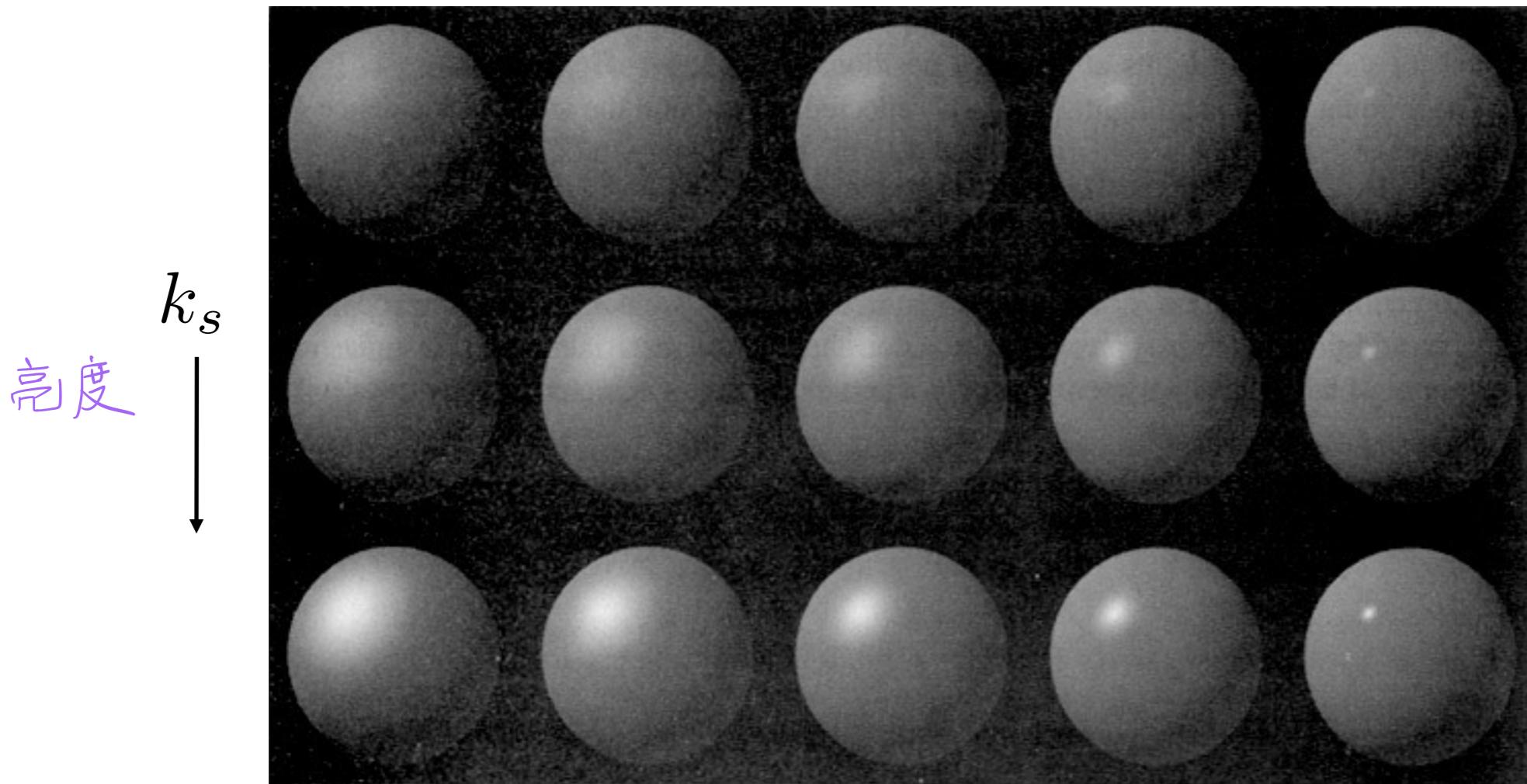


[Foley et al.]

# Specular Term (Blinn-Phong)

Blinn-Phong

$$L_s = k_s (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$



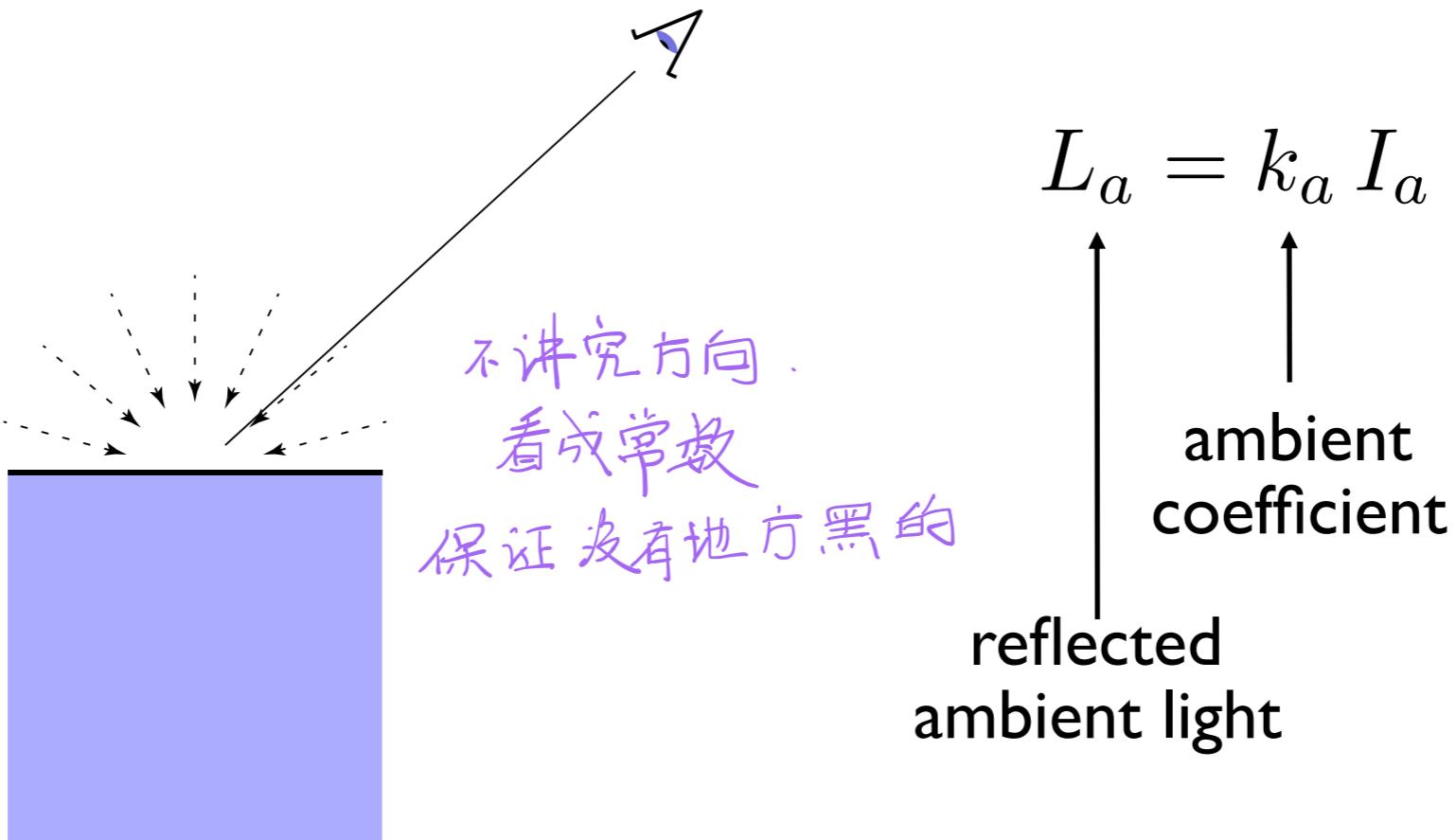
Note: showing  
 $L_d + L_s$  together

$p \longrightarrow$  高光大小

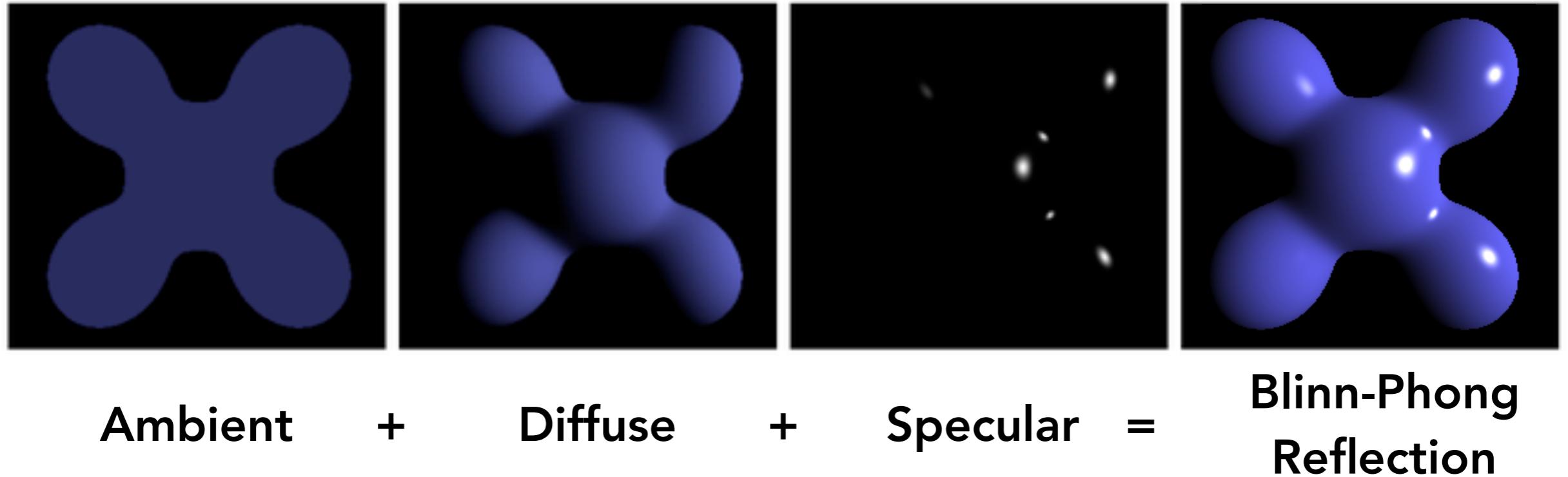
# Ambient Term

Shading that does not depend on anything

- Add constant color to account for disregarded illumination and fill in black shadows
- This is approximate / fake!



# Blinn-Phong Reflection Model



$$L = L_a + L_d + L_s$$

$$= k_a I_a + k_d (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

# Questions?

# Shading Frequencies

# Shading Frequencies

What caused the shading difference?

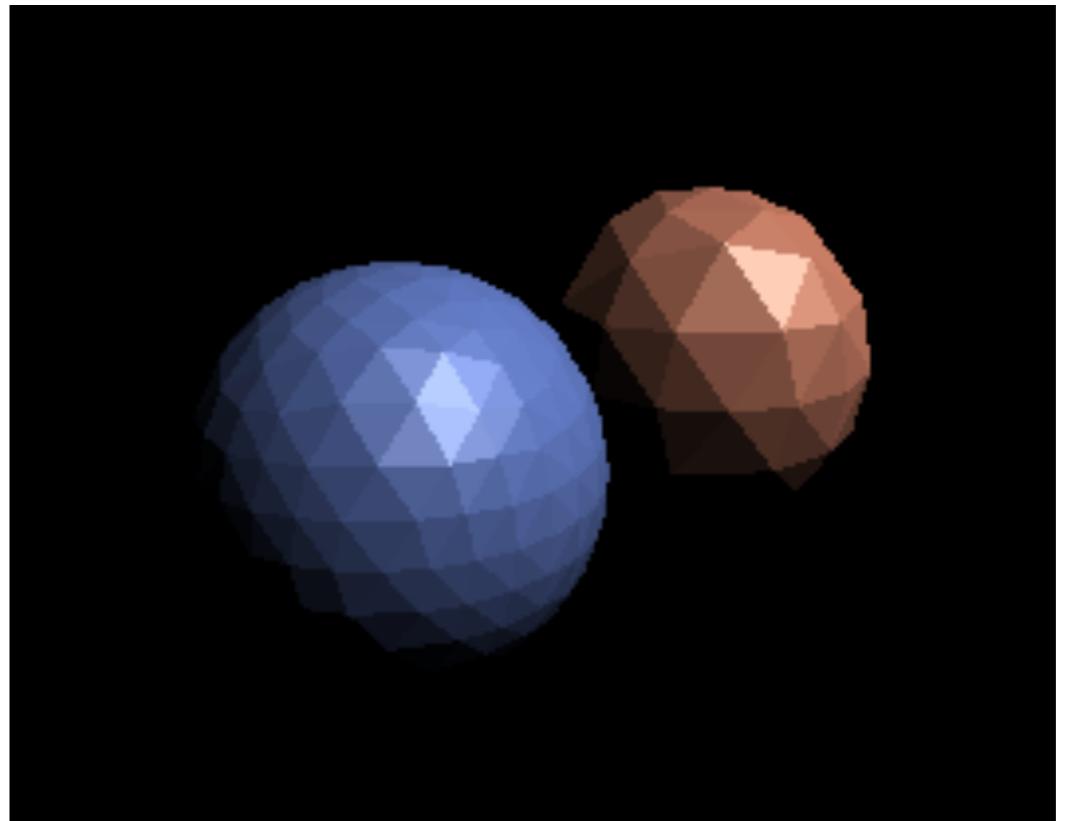


# Shade each triangle (flat shading)

每个平面  
算法  
shading .

## Flat shading

- Triangle face is flat — one normal vector
- Not good for smooth surfaces

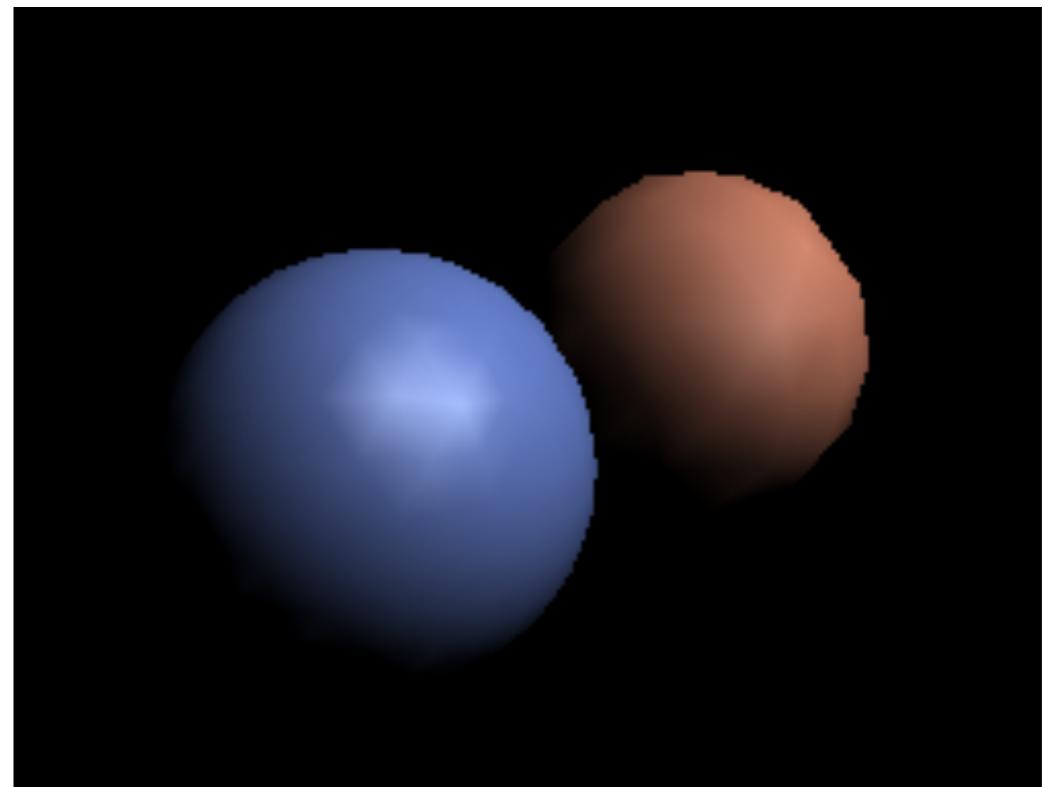


# Shade each vertex (Gouraud shading)

平面顶点计算法线，shading  
内部插值

## Gouraud shading

- Interpolate colors from vertices across triangle
- Each vertex has a normal vector (how?)

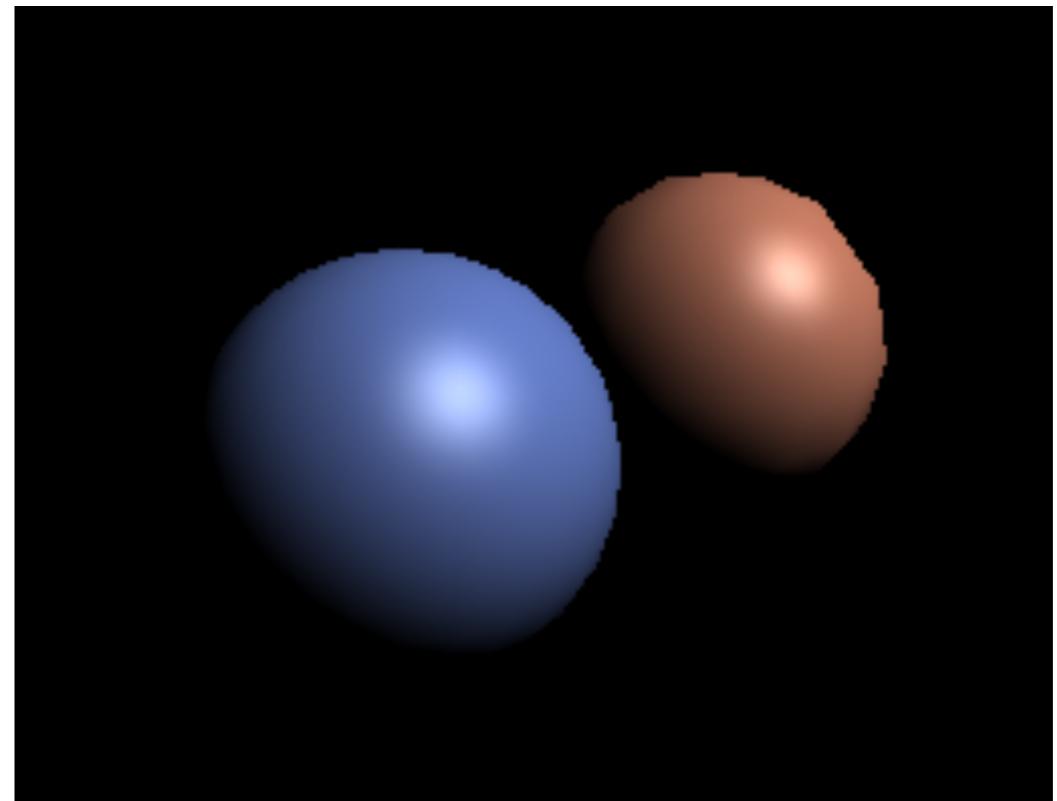


# Shade each pixel (Phong shading)

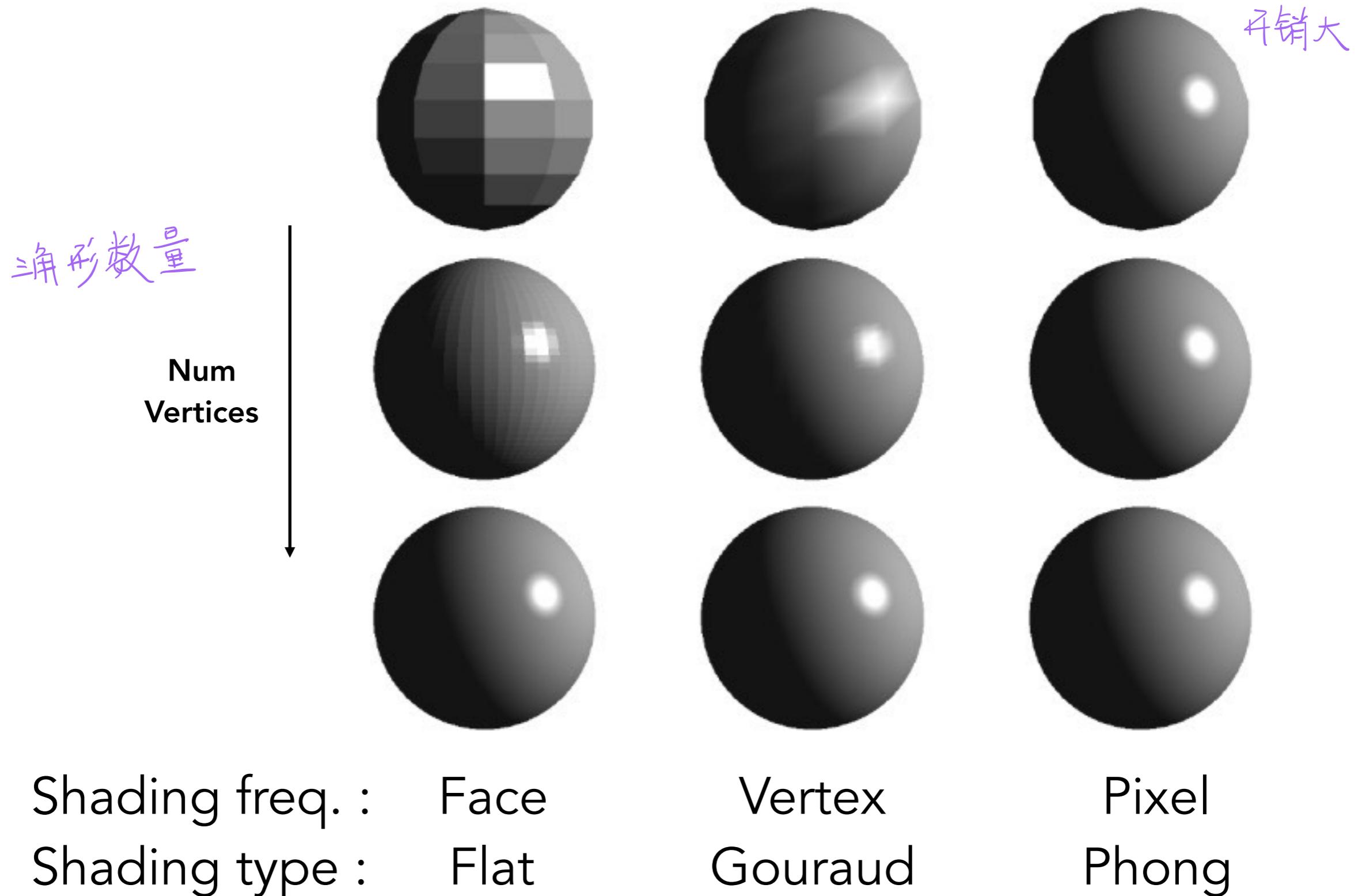
每个像素计算法线 shading .

## Phong shading

- Interpolate normal vectors across each triangle
- Compute full shading model at each pixel
- Not the **Blinn-Phong Reflectance Model**



# Shading Frequency: Face, Vertex or Pixel



# Defining Per-Vertex Normal Vectors

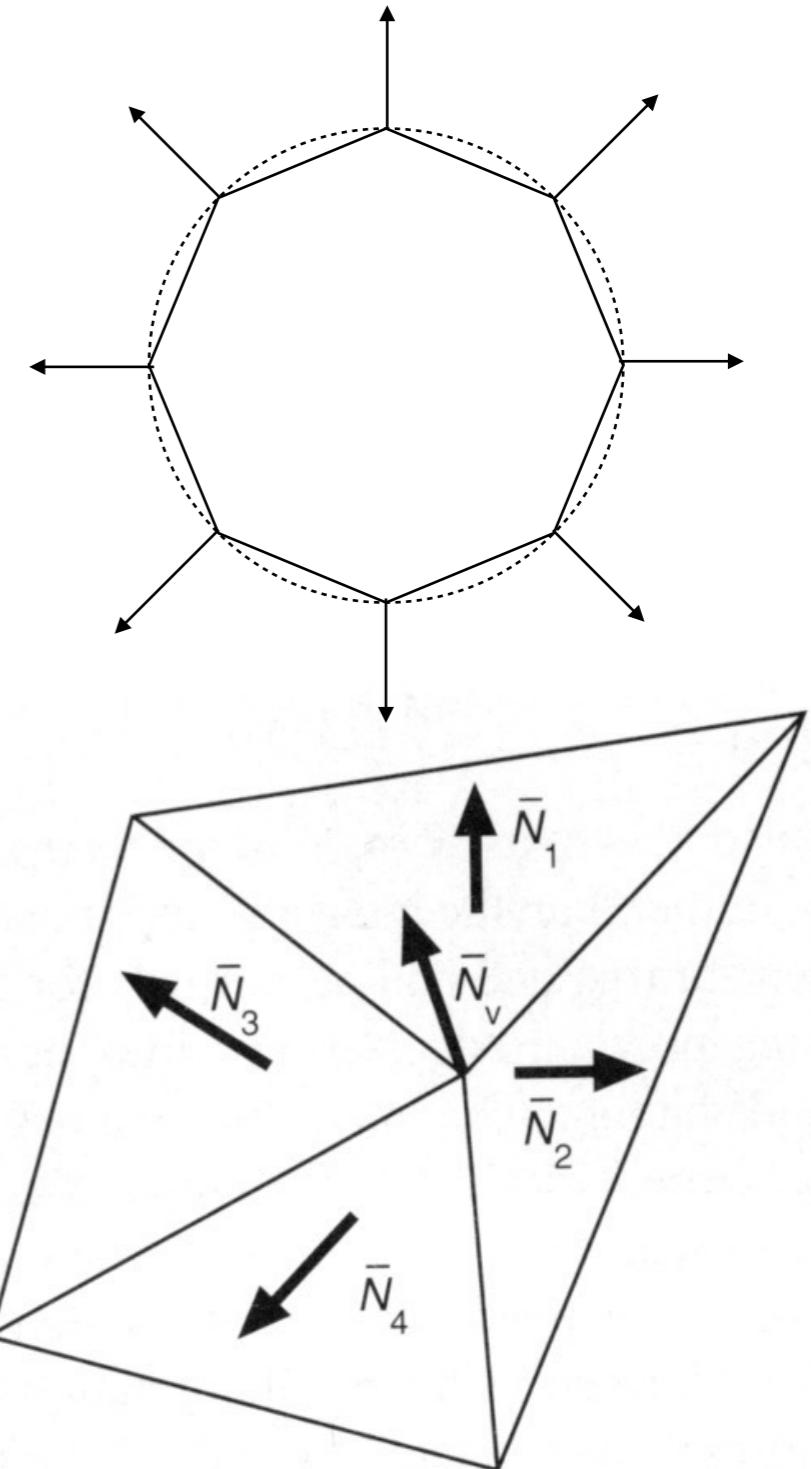
Best to get vertex normals from the underlying geometry

- e.g. consider a sphere

Otherwise have to infer vertex normals from triangle faces

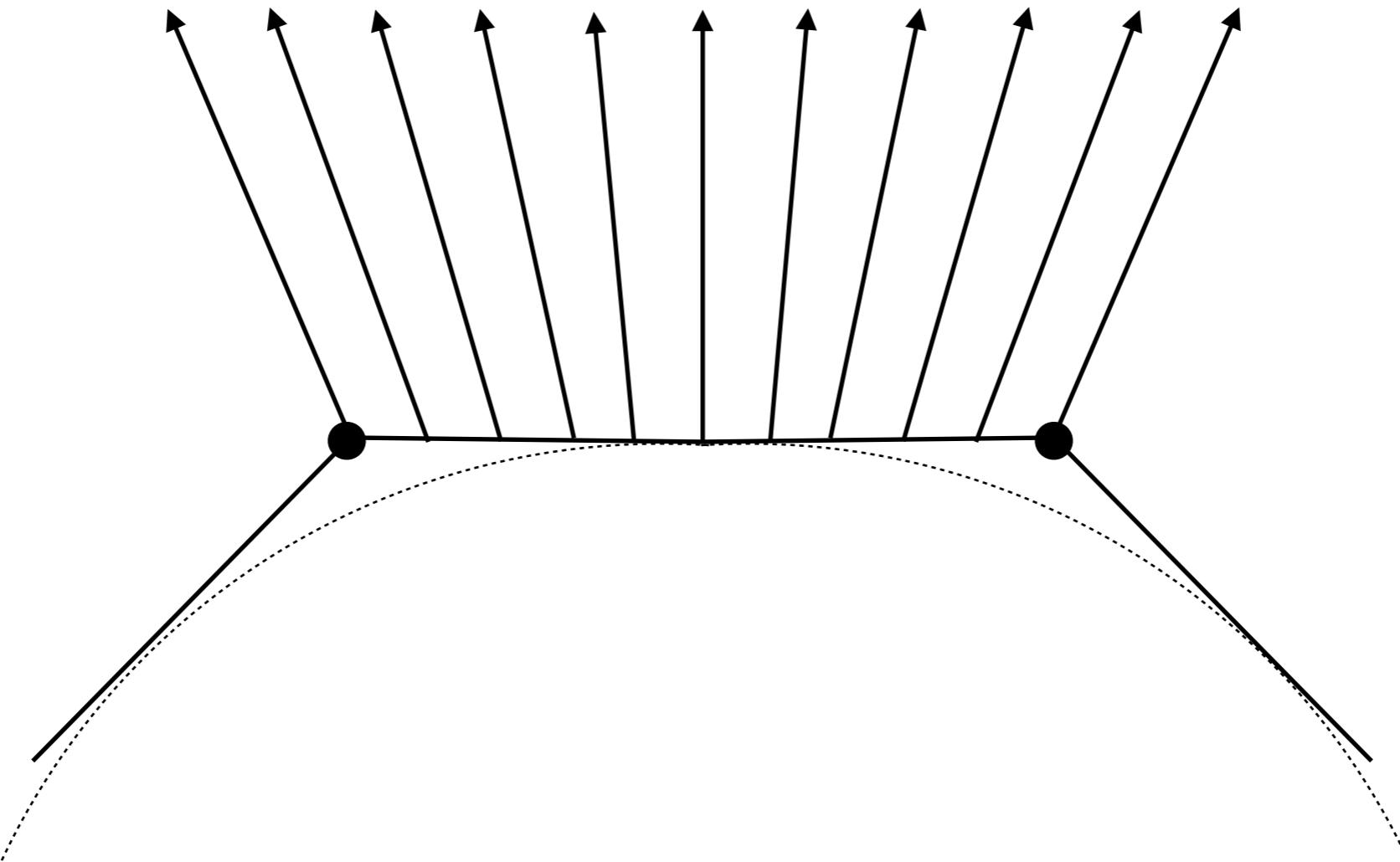
- Simple scheme: **average surrounding face normals**

$$N_v = \frac{\sum_i N_i}{\|\sum_i N_i\|}$$



# Defining Per-Pixel Normal Vectors

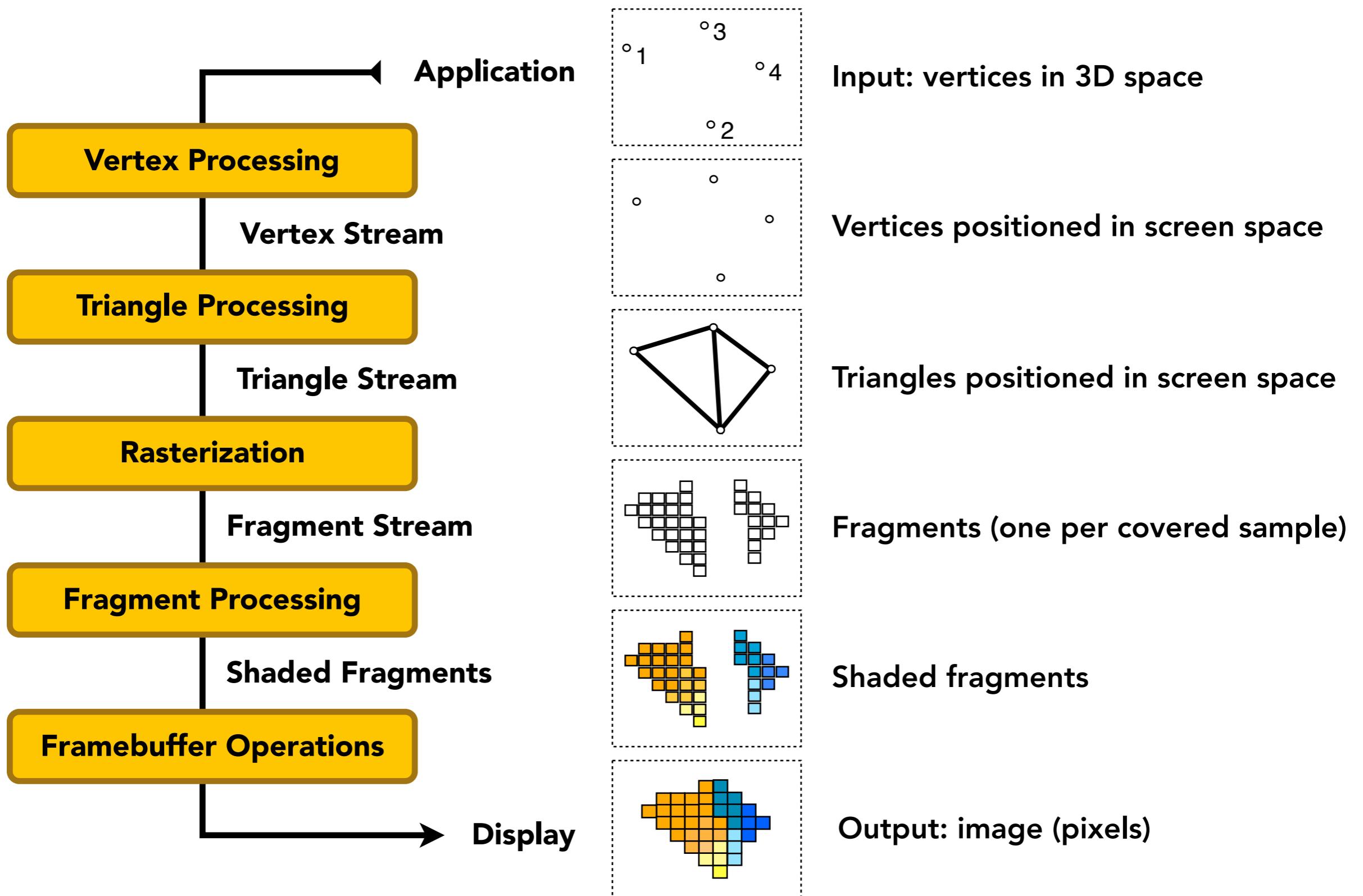
Barycentric interpolation (introducing soon) 使用重心坐标  
of vertex normals



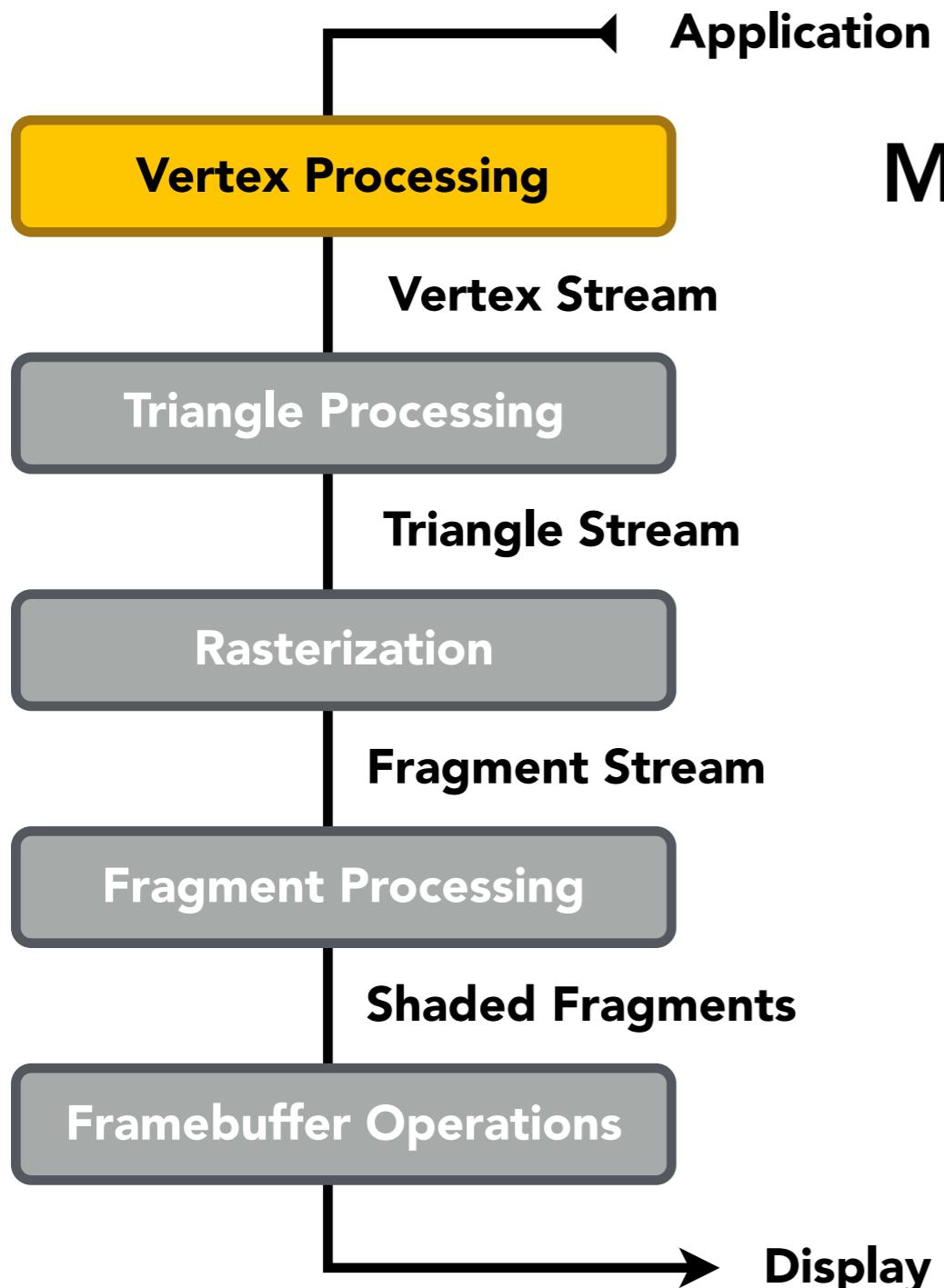
Don't forget to **normalize** the interpolated directions

# Graphics (**Real-time Rendering**) Pipeline

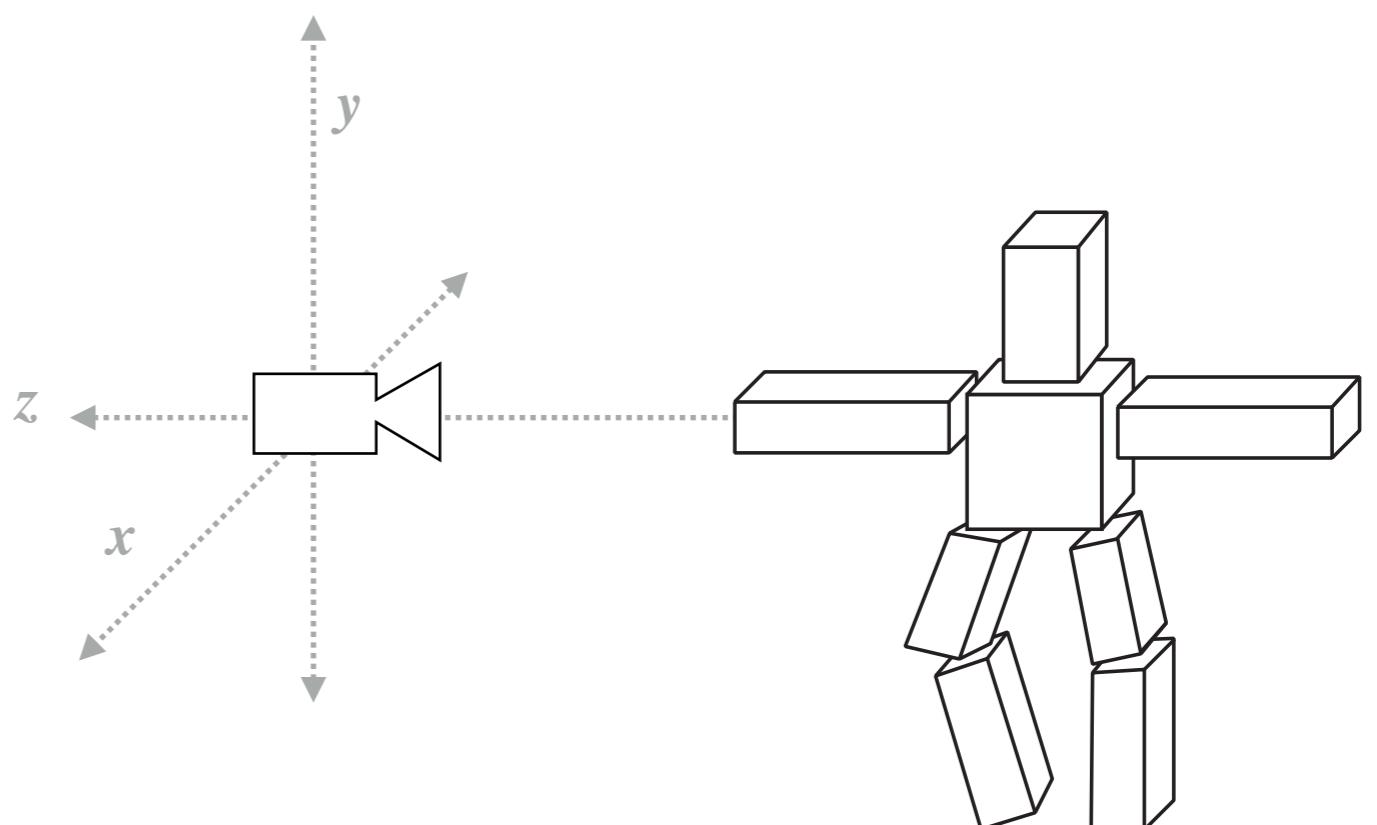
# Graphics Pipeline



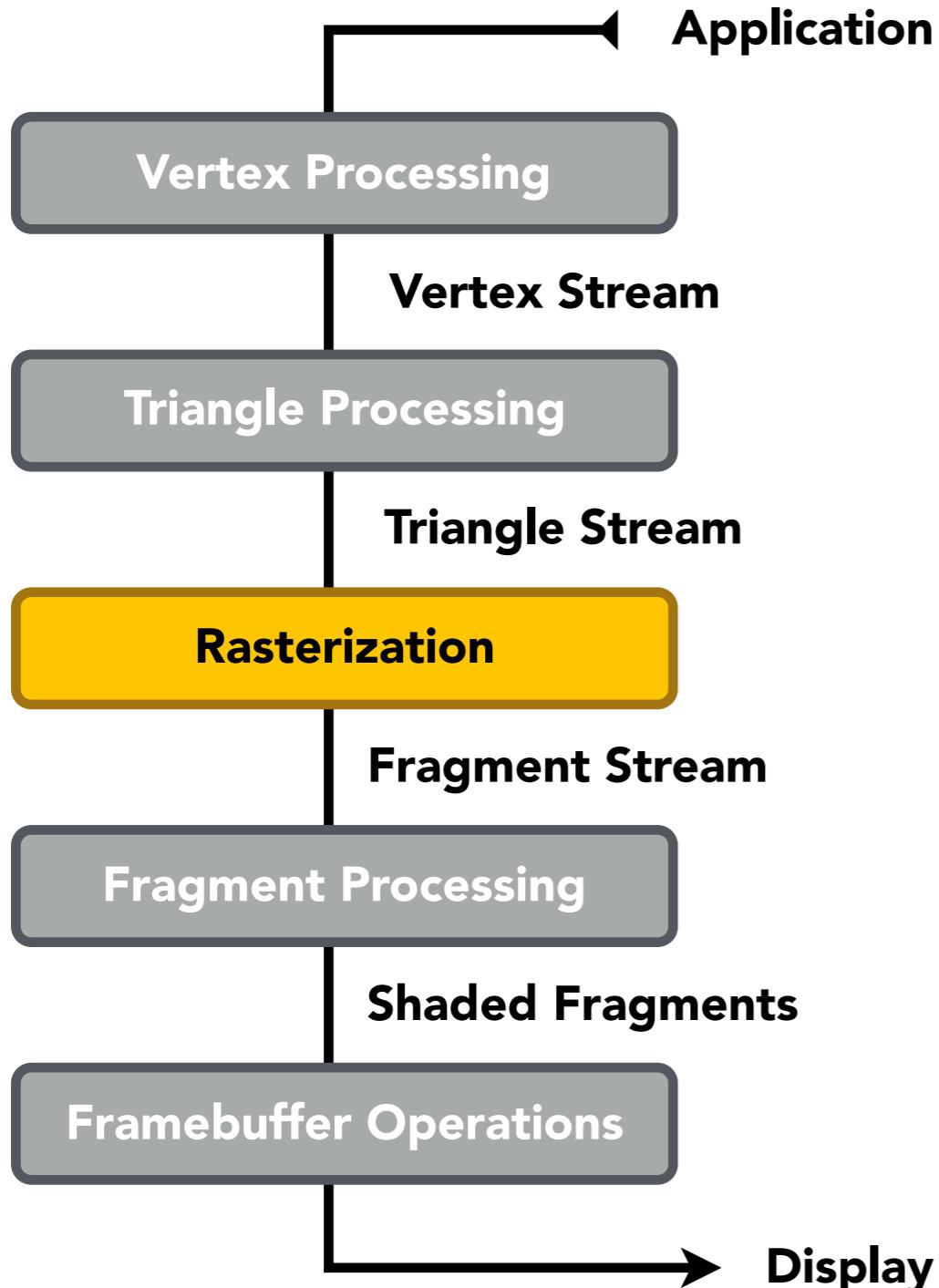
# Graphics Pipeline



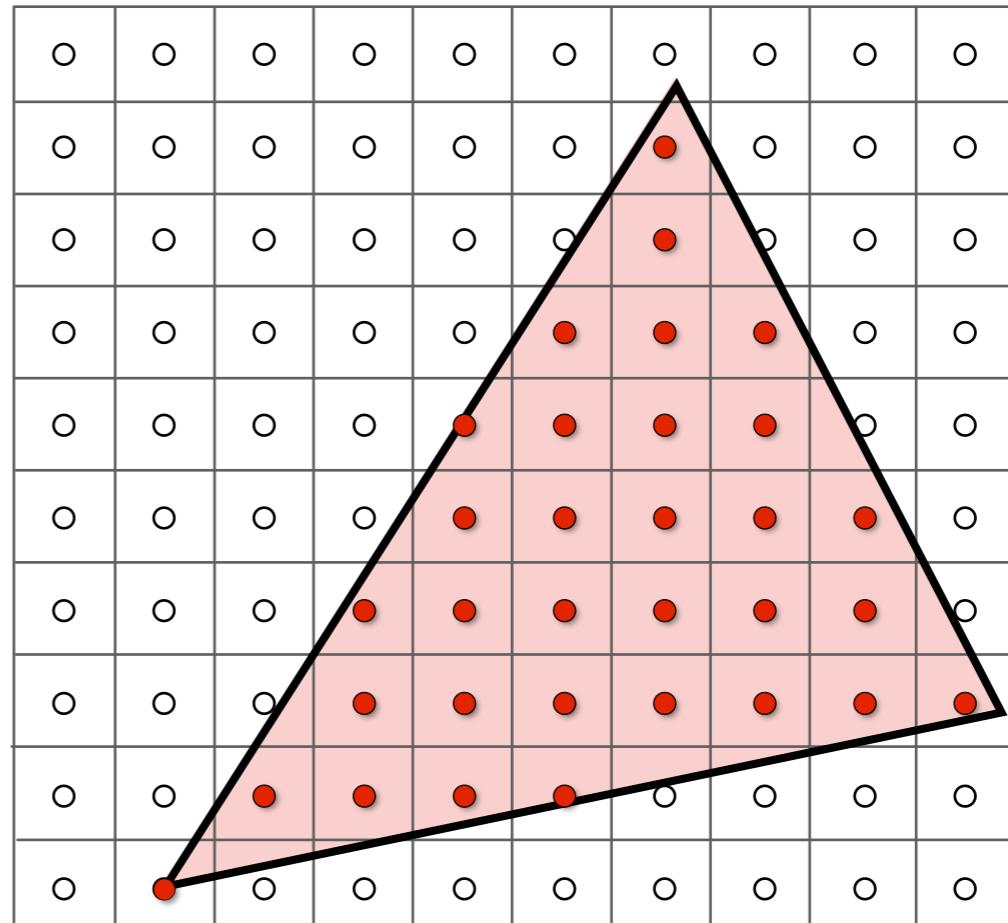
**Model, View, Projection transforms**



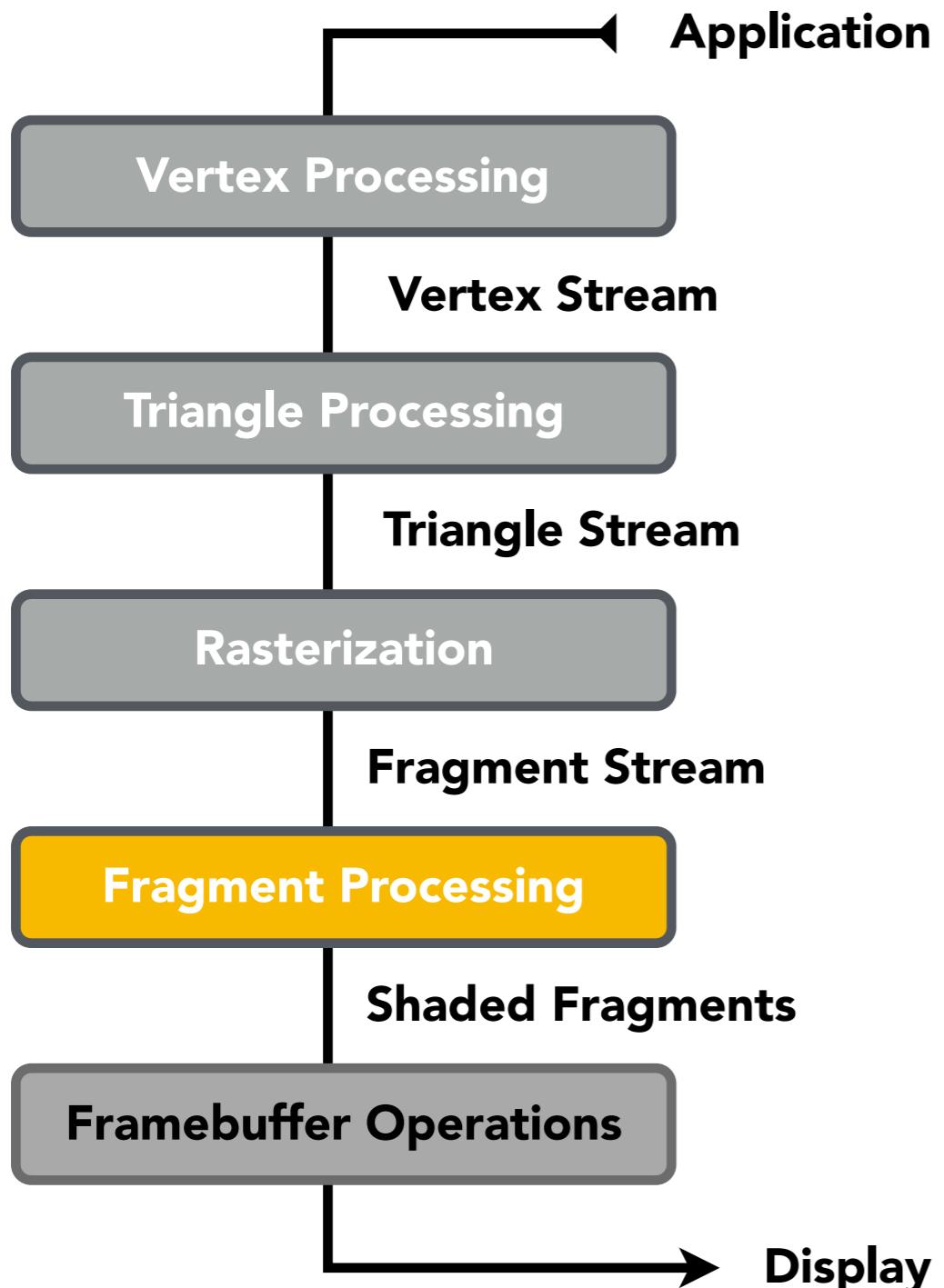
# Graphics Pipeline



## Sampling triangle coverage



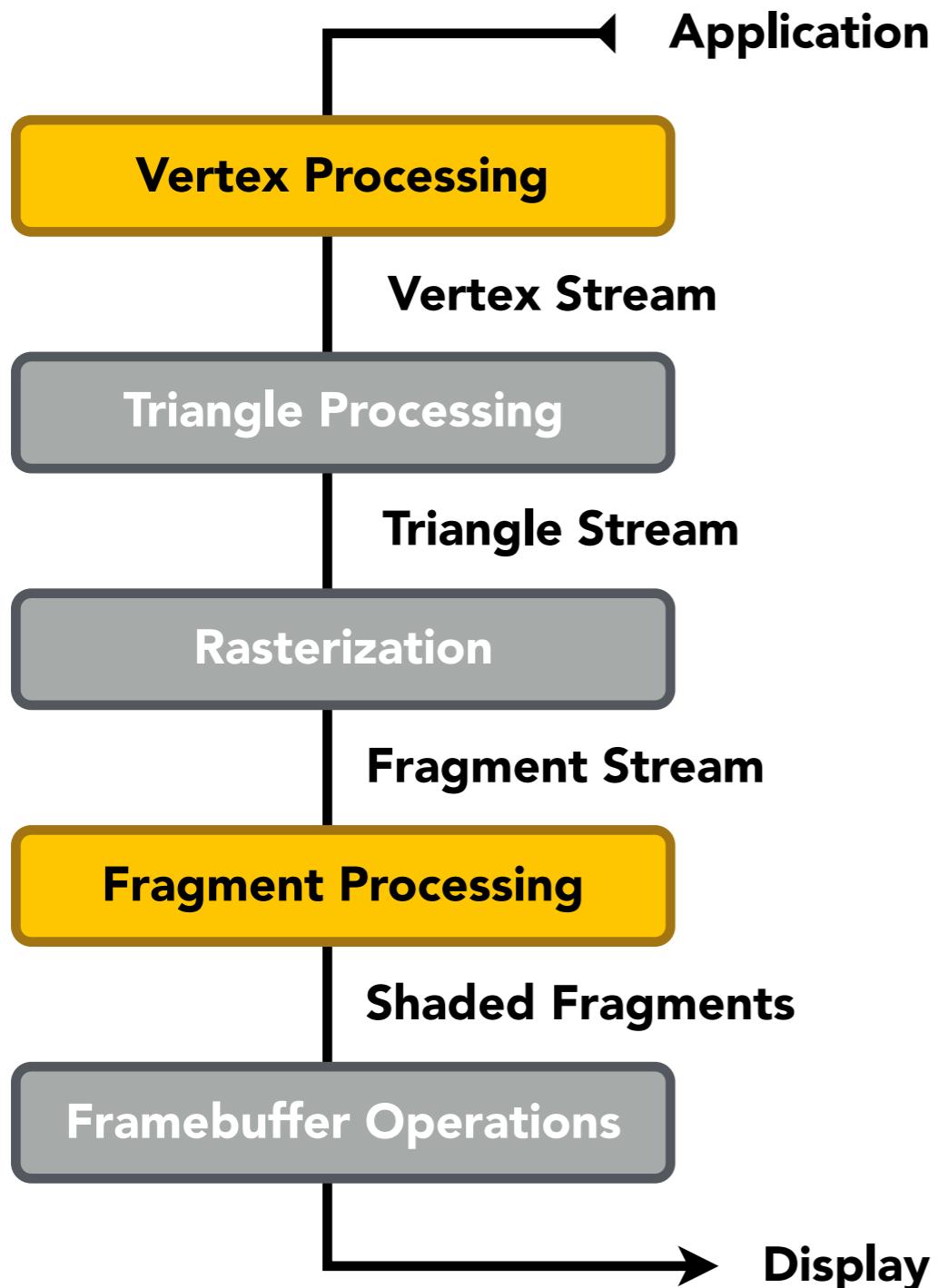
# Rasterization Pipeline



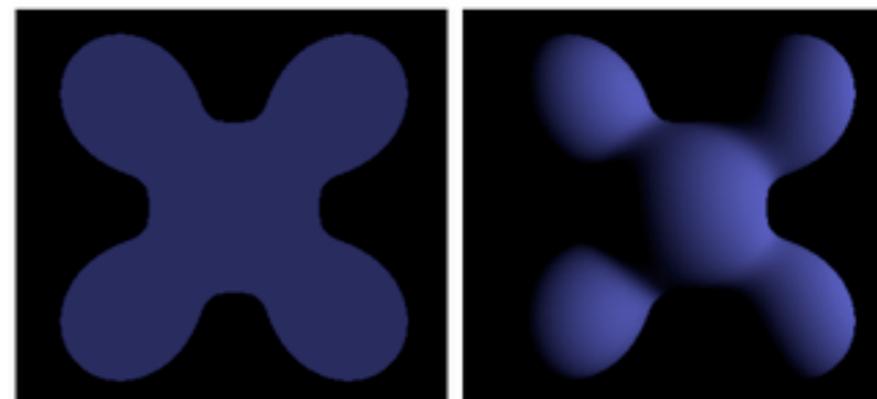
## Z-Buffer Visibility Tests



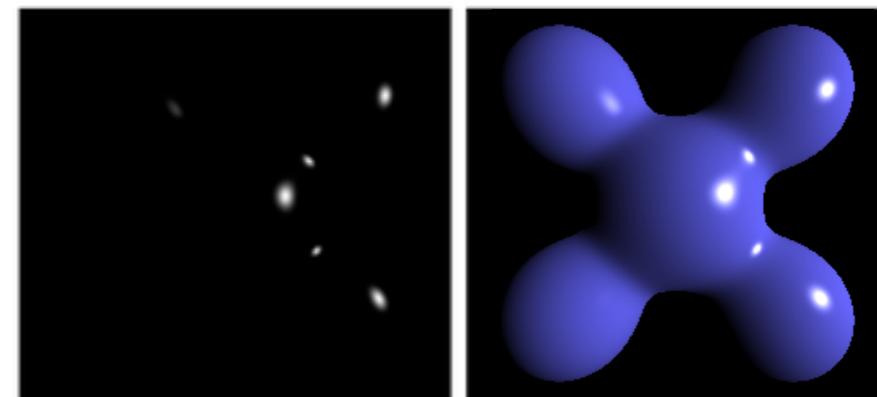
# Graphics Pipeline



**Shading**

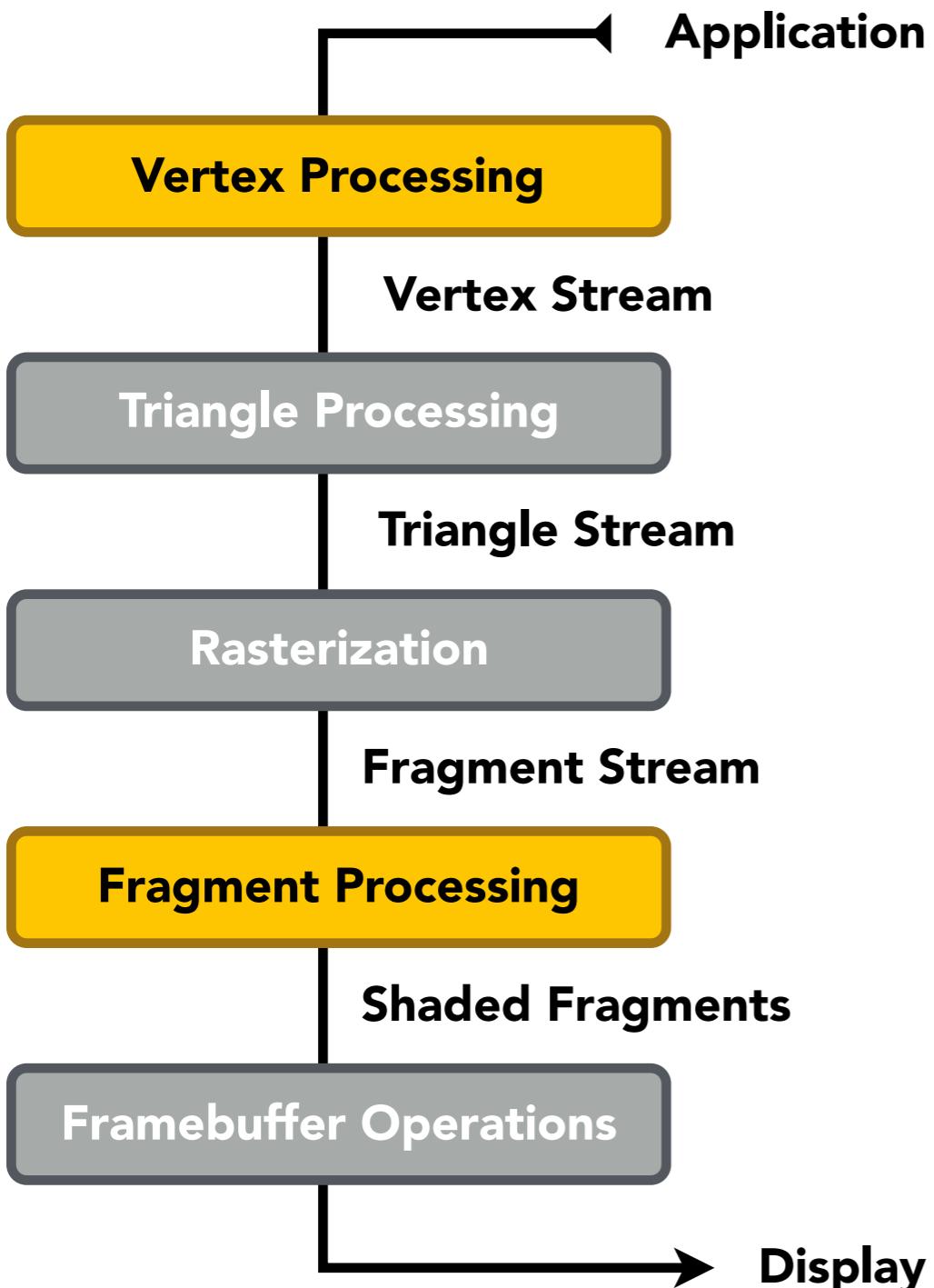


Ambient + Diffuse

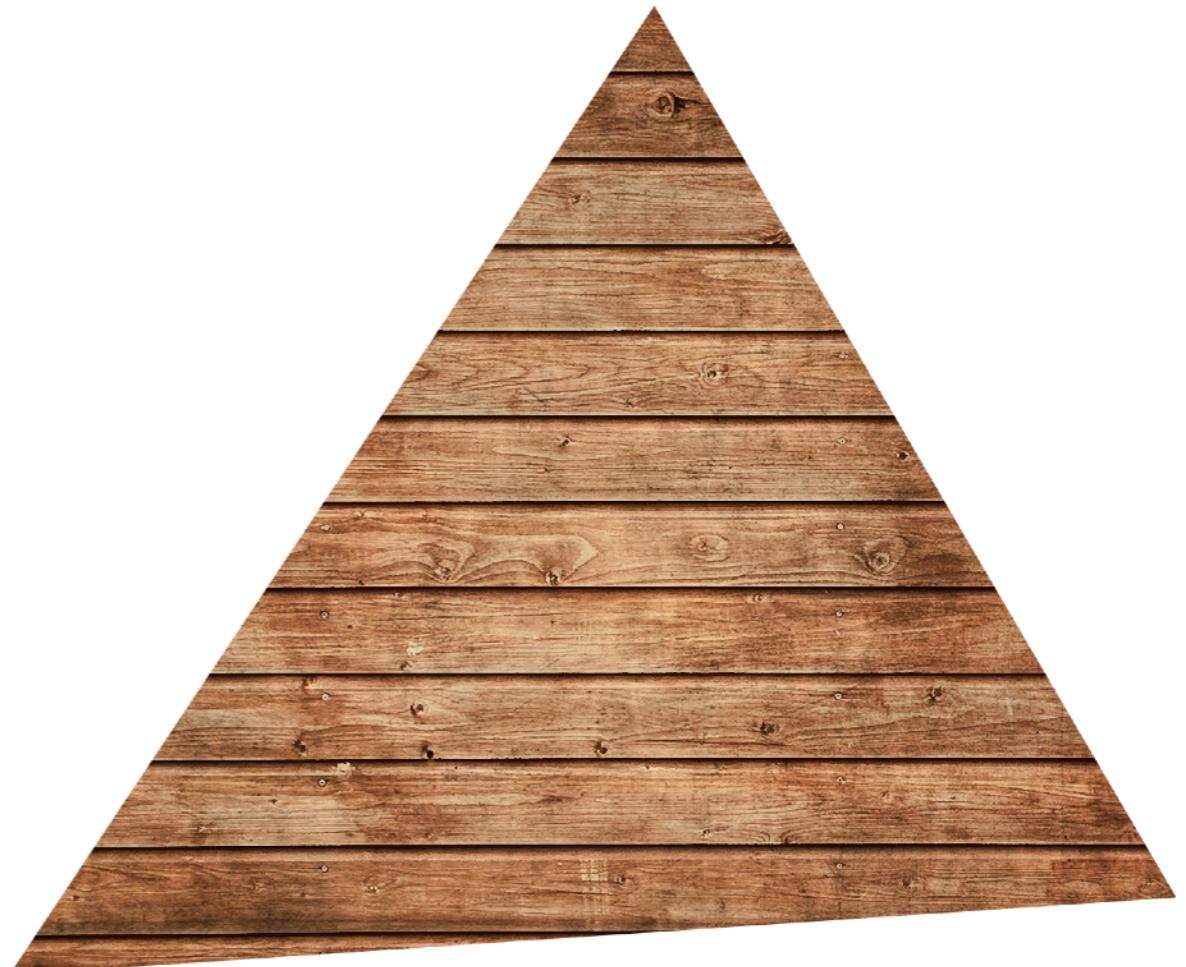


+ Specular = Blinn-Phong Reflectance Model

# Graphics Pipeline



**Texture mapping  
(introducing soon)**



# Shader Programs

只需写一个顶点或一个像素的 shader

- Program vertex and fragment processing stages
- Describe operation on a single vertex (or fragment)

Example GLSL fragment shader program

```
全局变量
uniform sampler2D myTexture;
uniform vec3 lightDir;    光照方向

varying vec2 uv;
varying vec3 norm;      法线

void diffuseShader()
{
    vec3 kd;    漫反射系数
    kd = texture2d(myTexture, uv);
    kd *= clamp(dot(-lightDir, norm), 0.0, 1.0);  保证在(0,1)
    gl_FragColor = vec4(kd, 1.0);
}
```

- Shader function executes once per fragment.
- Outputs color of surface at the current fragment's screen sample position.
- This shader performs a texture lookup to obtain the surface's material color at this point, then performs a diffuse lighting calculation.

# Shader Programs

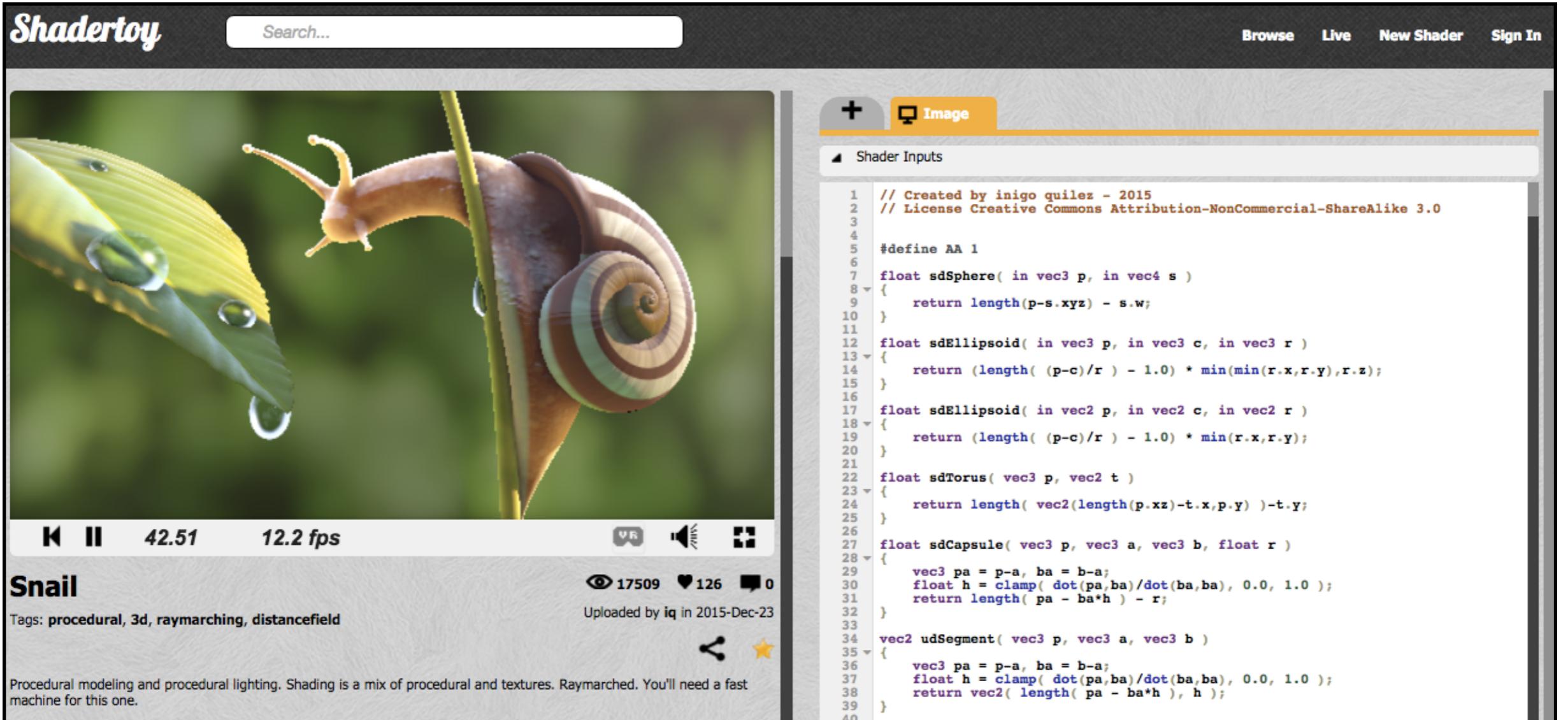
- Program vertex and fragment processing stages
- Describe operation on a single vertex (or fragment)

Example GLSL fragment shader program

```
uniform sampler2D myTexture;      // program parameter
uniform vec3 lightDir;           // program parameter
varying vec2 uv;                // per fragment value (interp. by rasterizer)
varying vec3 norm;               // per fragment value (interp. by rasterizer)

void diffuseShader()
{
    vec3 kd;
    kd = texture2d(myTexture, uv);          // material color from texture
    kd *= clamp(dot(-lightDir, norm), 0.0, 1.0); // Lambertian shading model
    gl_FragColor = vec4(kd, 1.0);           // output fragment color
}
```

# Snail Shader Program



The screenshot shows the Shadertoy interface. On the left, there is a preview window displaying a close-up of a snail crawling on a green leaf with water droplets. Below the preview, the title "Snail" is displayed, along with its tags: "procedural, 3d, raymarching, distancefield". The video player shows a play button, a timestamp of "42.51", and a frame rate of "12.2 fps". To the right of the preview, there are icons for volume, brightness, and full screen. Below these icons, the number of views (17509), likes (126), and comments (0) are shown, along with the upload date ("Uploaded by iq in 2015-Dec-23"). At the bottom of the preview area, a descriptive text reads: "Procedural modeling and procedural lighting. Shading is a mix of procedural and textures. Raymarched. You'll need a fast machine for this one." On the right side of the interface, the "Image" tab is selected. Under "Shader Inputs", the following GLSL code is displayed:

```
// Created by inigo quilez - 2015
// License Creative Commons Attribution-NonCommercial-ShareAlike 3.0

#define AA 1

float sdSphere( in vec3 p, in vec4 s )
{
    return length(p-s.xyz) - s.w;
}

float sdEllipsoid( in vec3 p, in vec3 c, in vec3 r )
{
    return (length( (p-c)/r ) - 1.0) * min(min(r.x,r.y),r.z);
}

float sdEllipsoid( in vec2 p, in vec2 c, in vec2 r )
{
    return (length( (p-c)/r ) - 1.0) * min(r.x,r.y);
}

float sdTorus( vec3 p, vec2 t )
{
    return length( vec2(length(p.xz)-t.x,p.y) )-t.y;
}

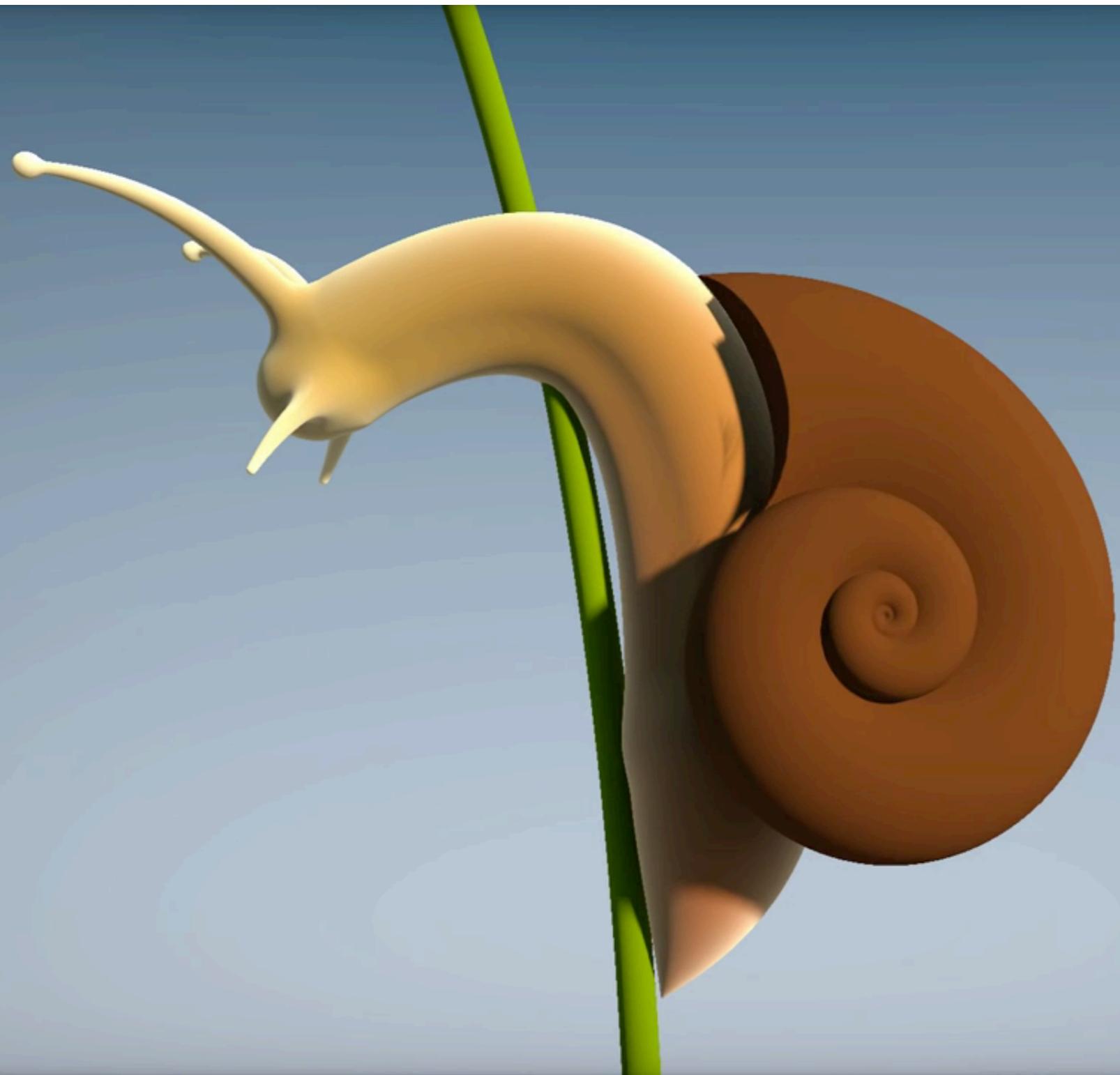
float sdCapsule( vec3 p, vec3 a, vec3 b, float r )
{
    vec3 pa = p-a, ba = b-a;
    float h = clamp( dot(pa,ba)/dot(ba,ba), 0.0, 1.0 );
    return length( pa - ba*h ) - r;
}

vec2 udSegment( vec3 p, vec3 a, vec3 b )
{
    vec3 pa = p-a, ba = b-a;
    float h = clamp( dot(pa,ba)/dot(ba,ba), 0.0, 1.0 );
    return vec2( length( pa - ba*h ), h );
}
```

## Inigo Quilez

Procedurally modeled, 800 line shader.  
<http://shadertoy.com/view/ld3Gz2>

# Snail Shader Program



Inigo Quilez, <https://youtu.be/XuSnLbB1j6E>

# Goal: Highly Complex 3D Scenes in Realtime

- 100's of thousands to millions of triangles in a scene
- Complex vertex and fragment shader computations
- High resolution (2-4 megapixel + supersampling)
- 30-60 frames per second (even higher for VR)



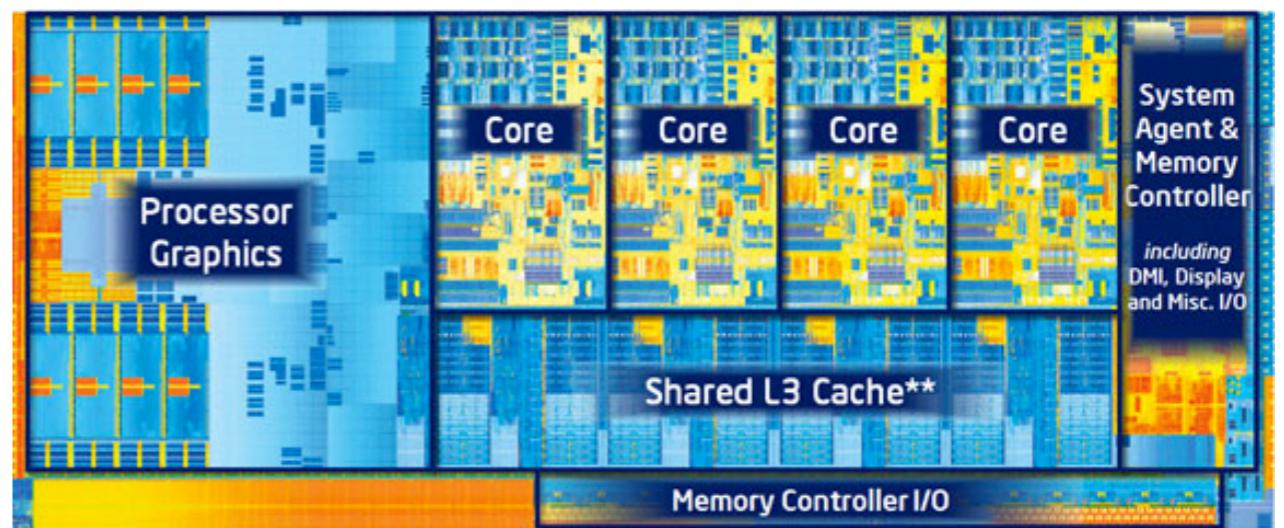
Unreal Engine Kite Demo (Epic Games 2015)

# Graphics Pipeline Implementation: GPUs

**Specialized processors for executing graphics pipeline computations**

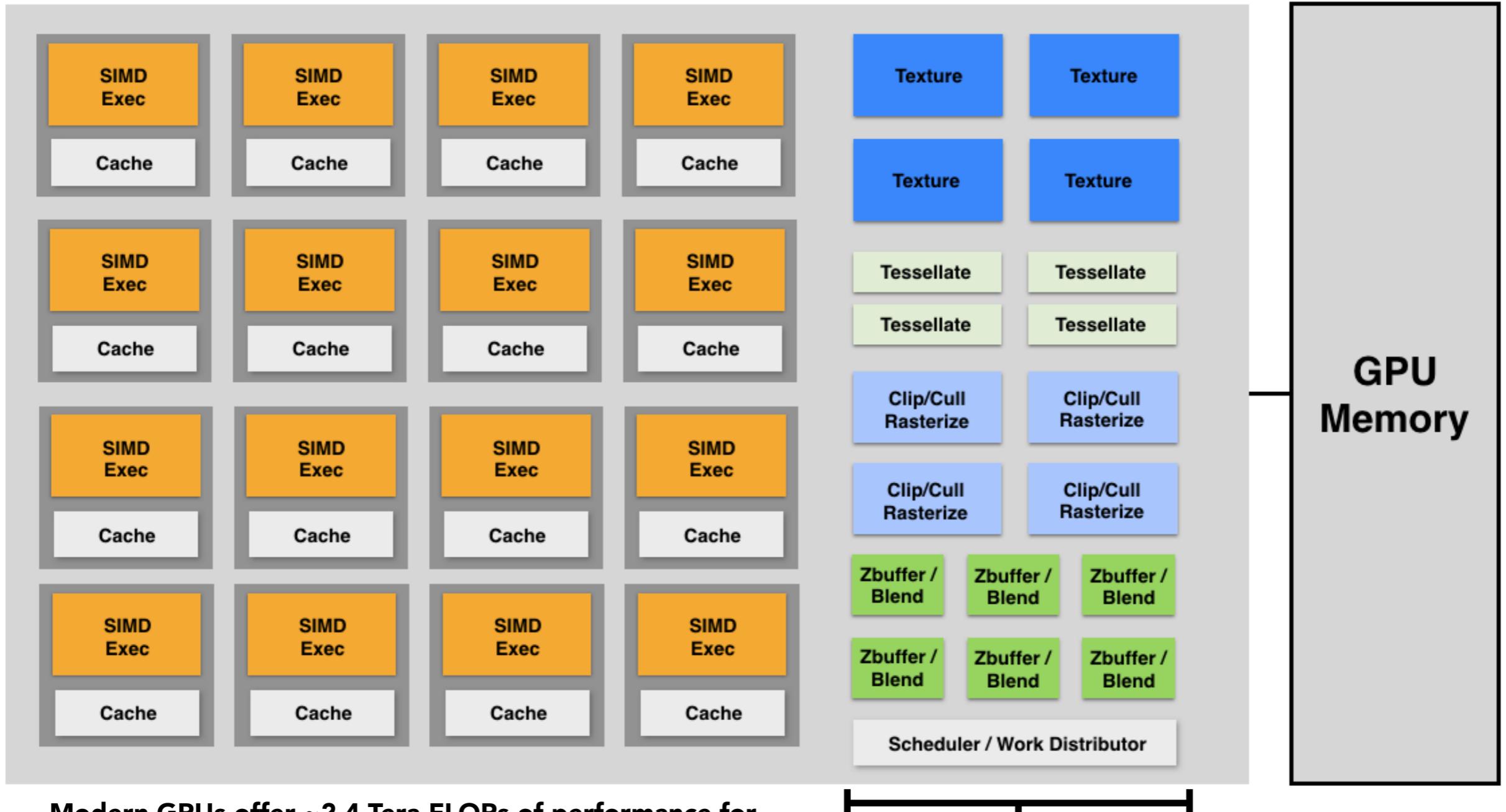


**Discrete GPU Card**  
**(NVIDIA GeForce Titan X)**



**Integrated GPU:**  
**(Part of Intel CPU die)**

# GPU: Heterogeneous, Multi-Core Processor



Modern GPUs offer ~2-4 Tera-FLOPs of performance for executing vertex and fragment shader programs

Tera-Op's of fixed-function compute capability over here

# Texture Mapping

# Different Colors at Different Places?

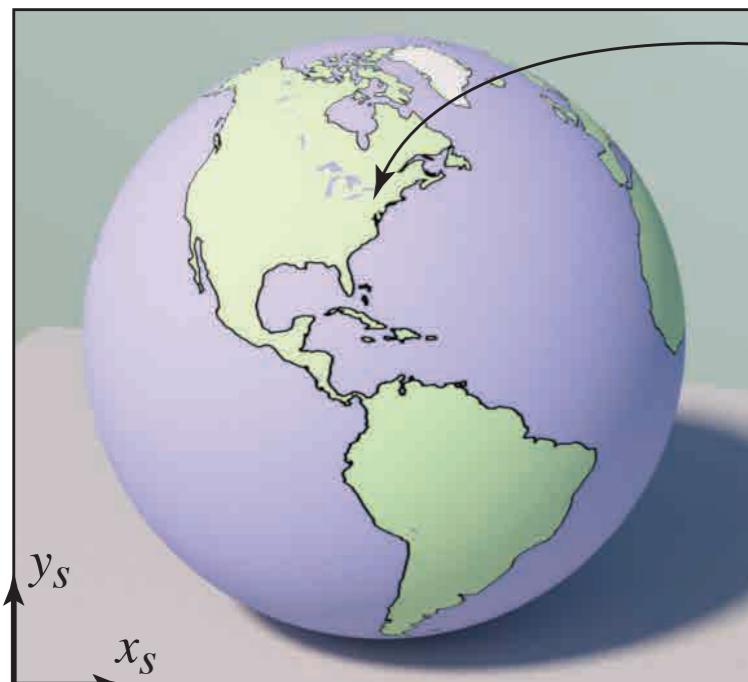


# Surfaces are 2D

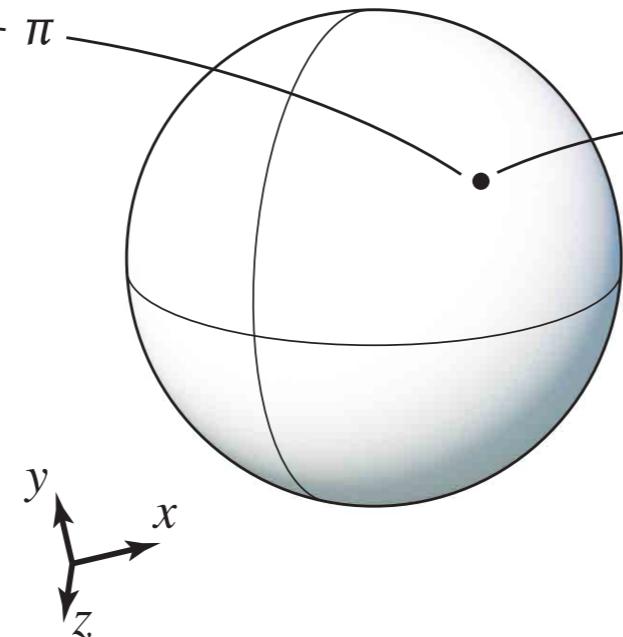
Surface lives in 3D world space

三维物体表面二维

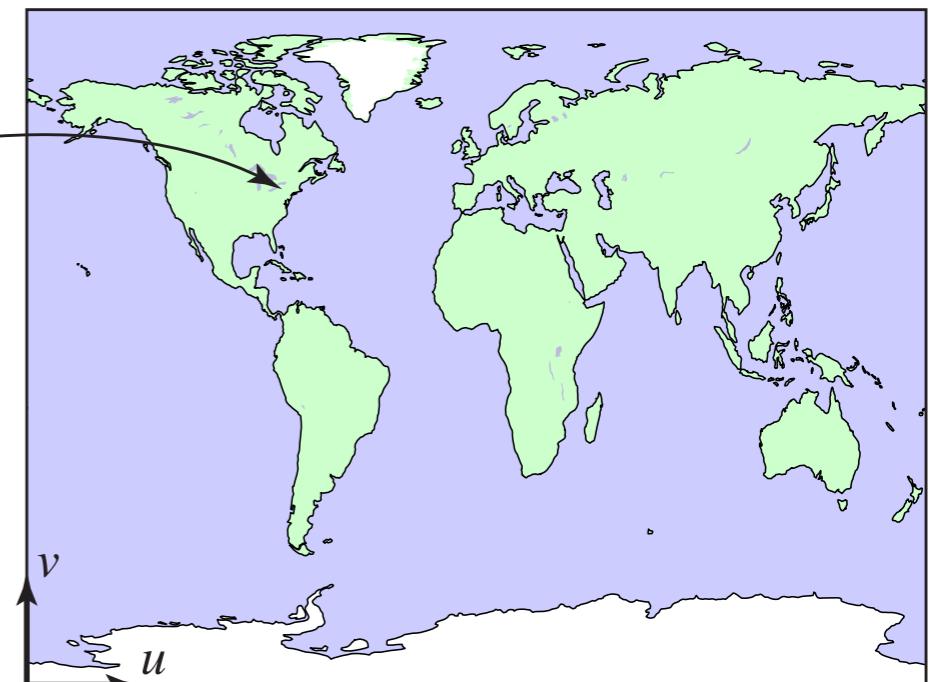
Every 3D surface point also has a place where it goes in the 2D image (**texture**).



Screen space



World space



Texture space

# Texture Applied to Surface

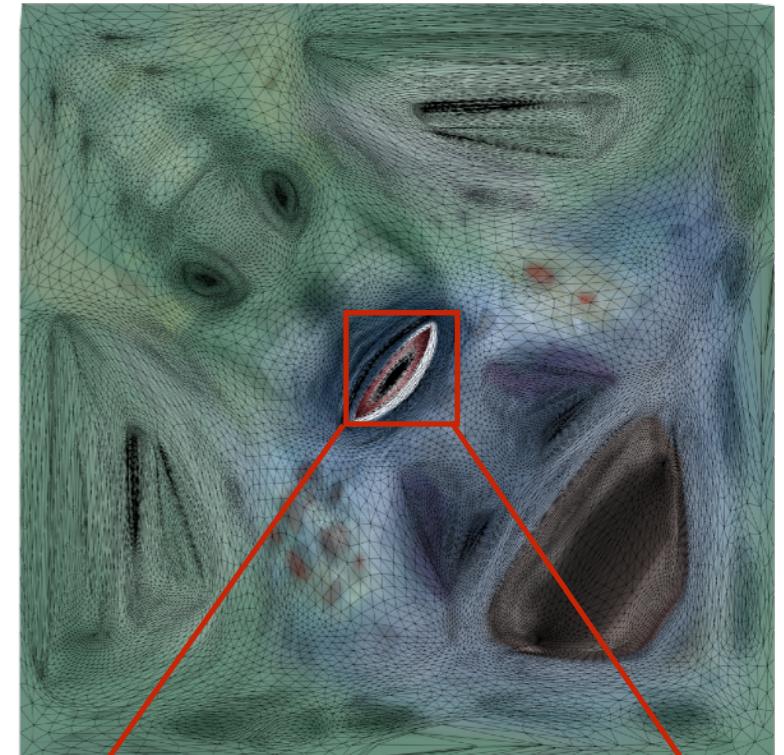
Rendering without texture



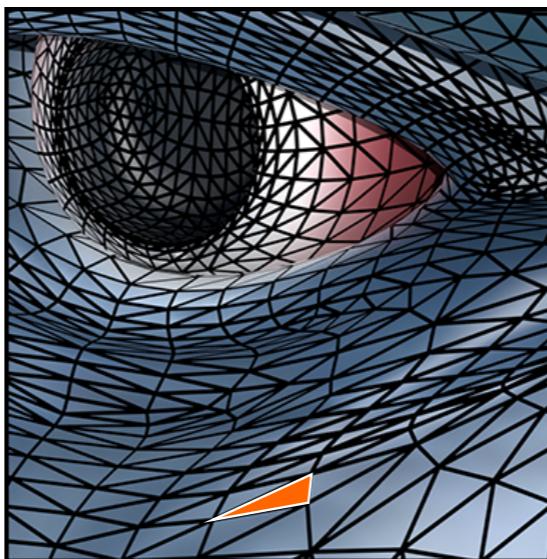
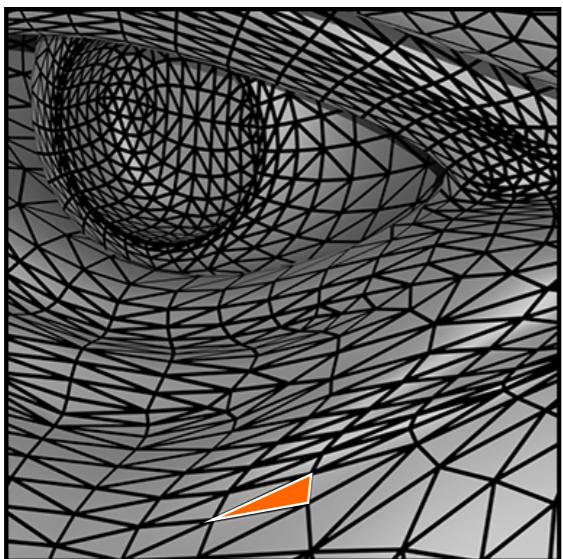
Rendering with texture



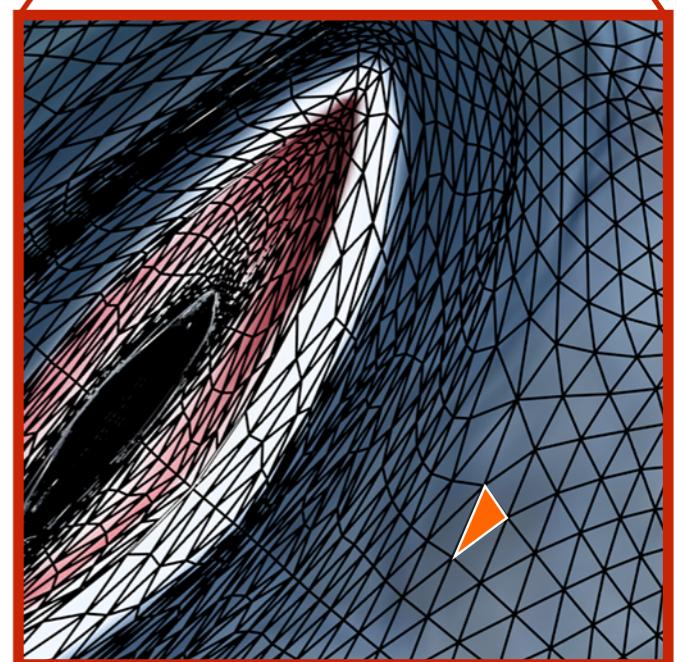
Texture



Zoom



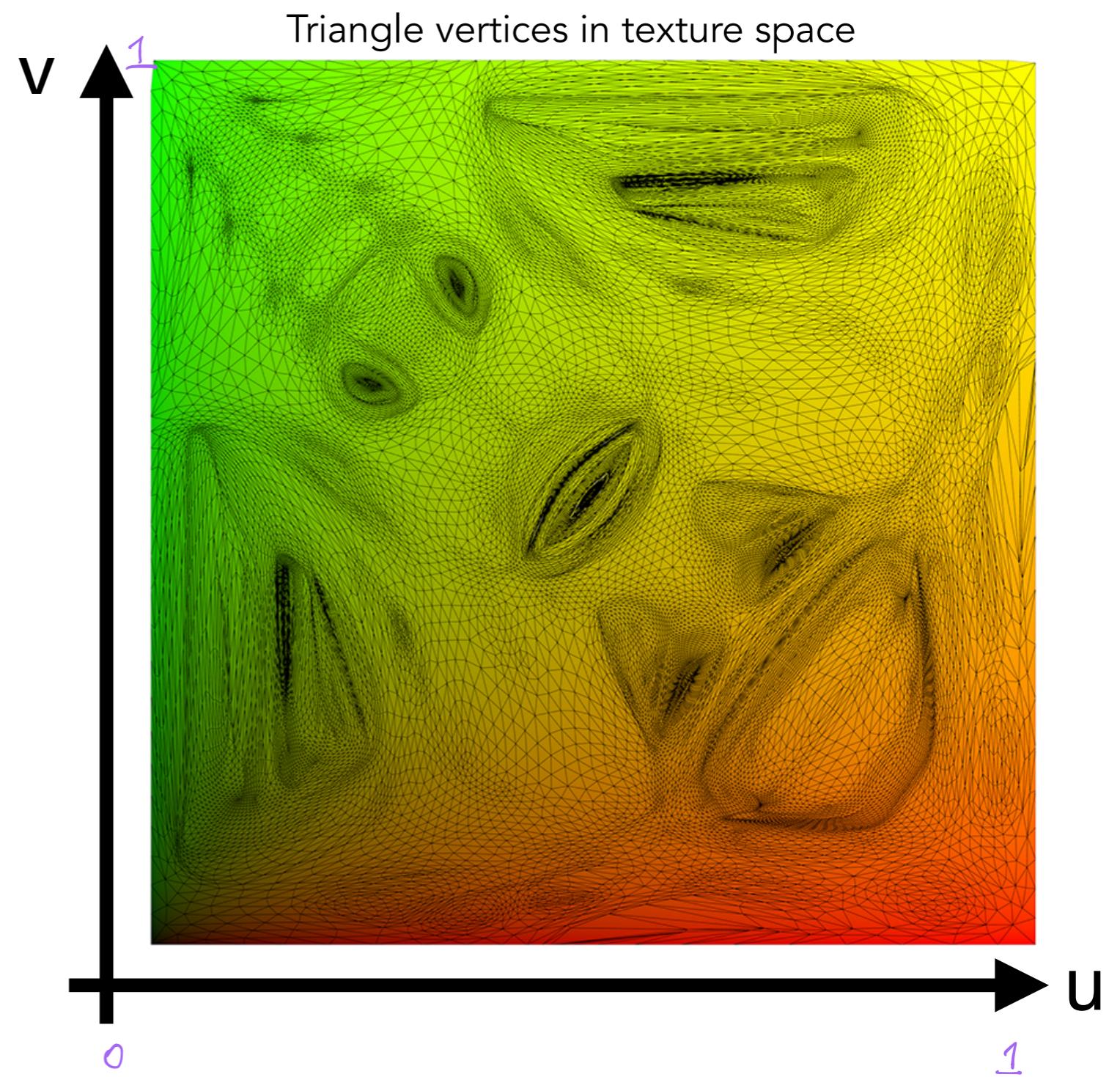
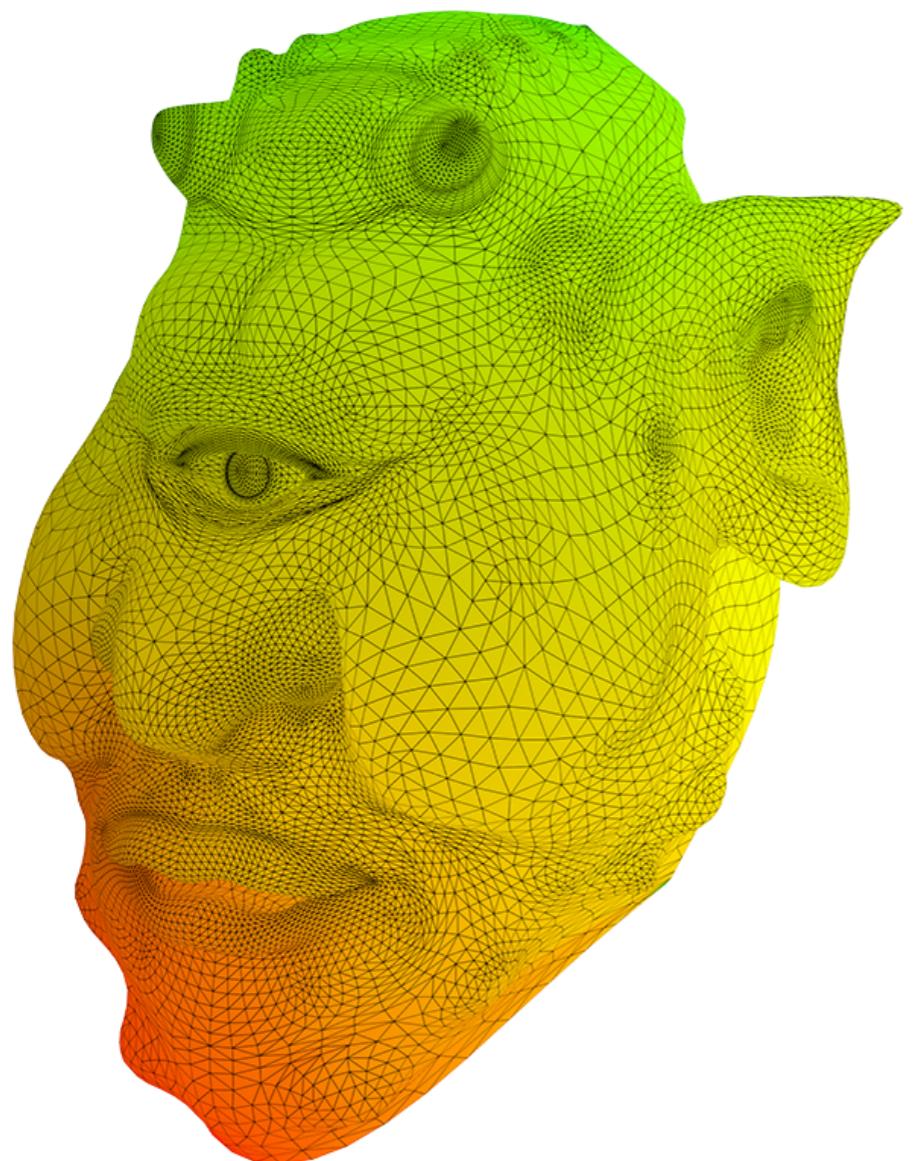
Each triangle “copies” a piece of the texture image to the surface.



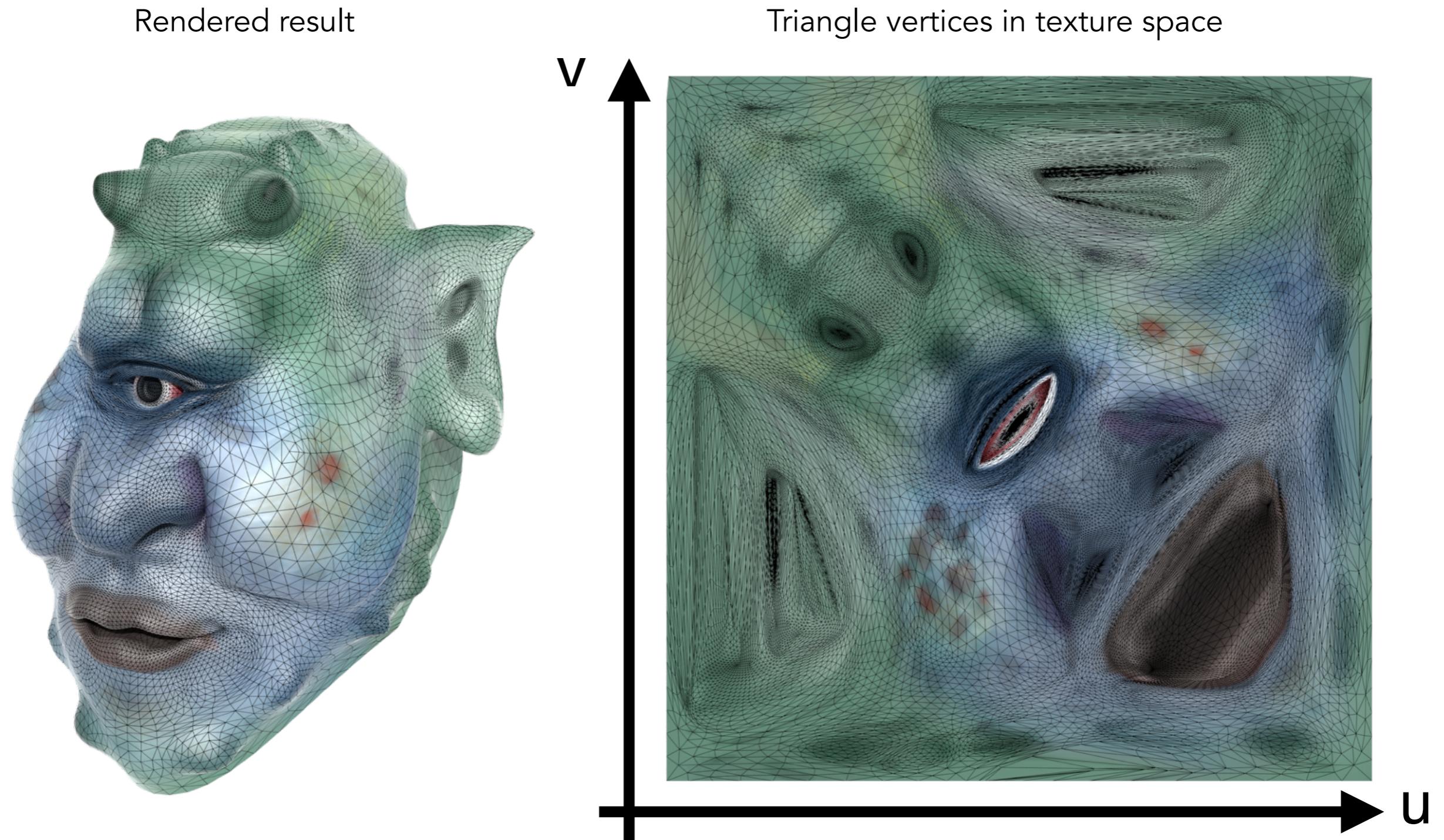
# Visualization of Texture Coordinates

Each triangle vertex is assigned a texture coordinate  $(u,v)$

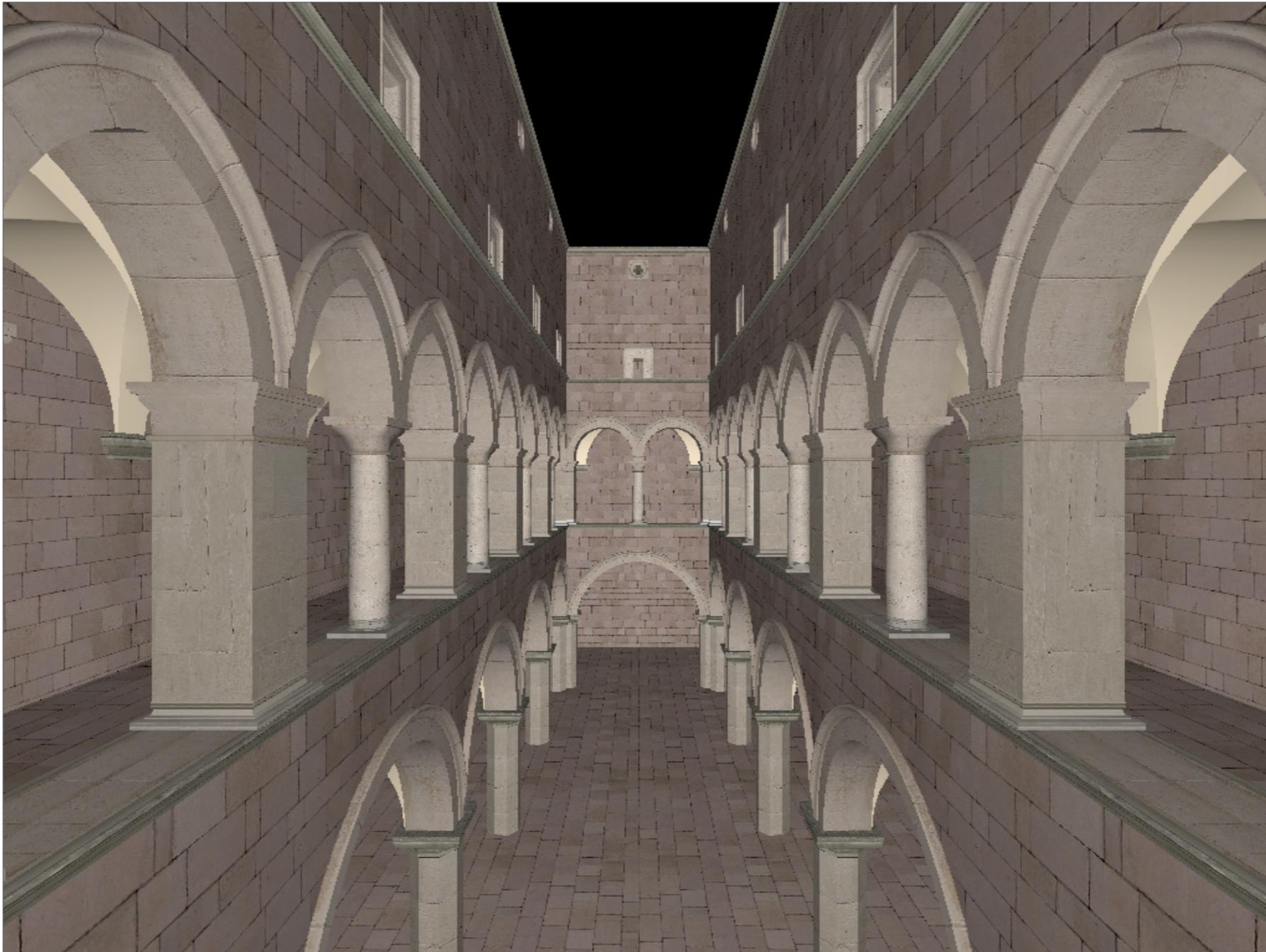
Visualization of texture coordinates



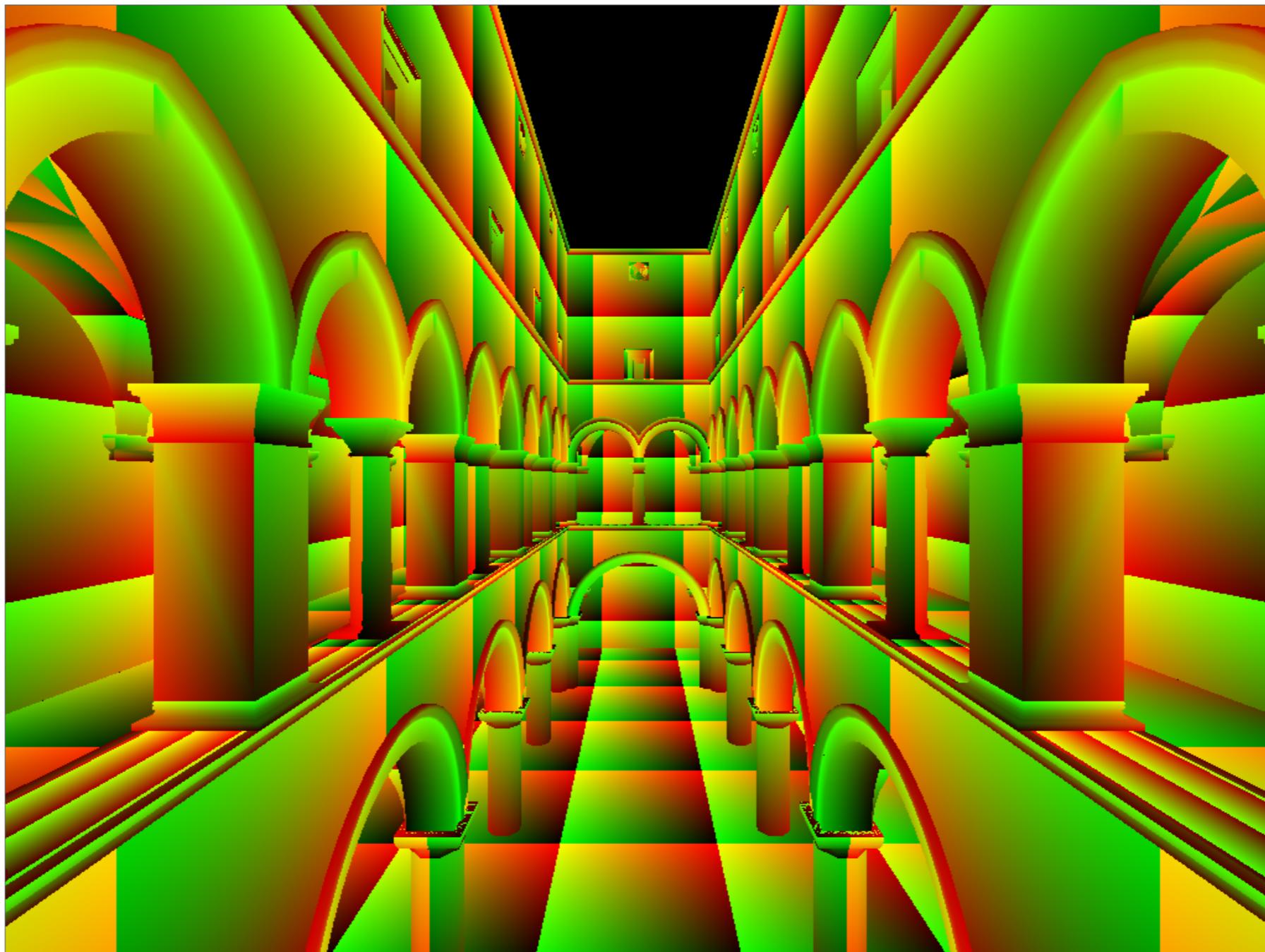
# Texture Applied to Surface



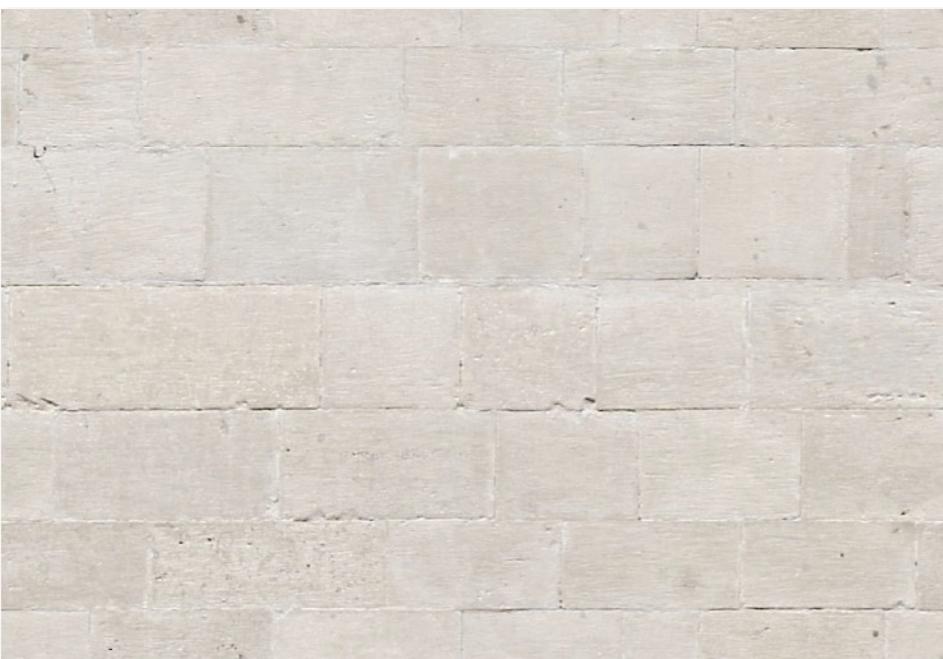
# Textures applied to surfaces



# Visualization of texture coordinates



# Textures can be used multiple times!



example textures  
used / **tiled**

# Thank you!

(And thank Prof. Ravi Ramamoorthi and Prof. Ren Ng for many of the slides!)