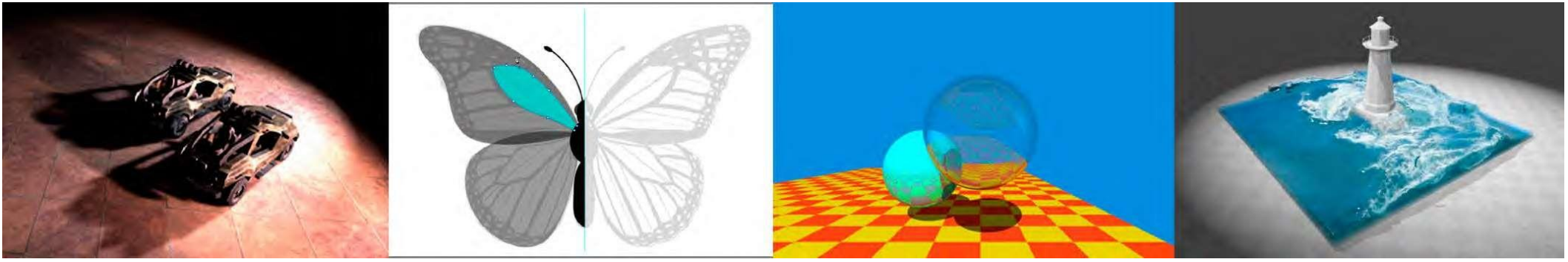
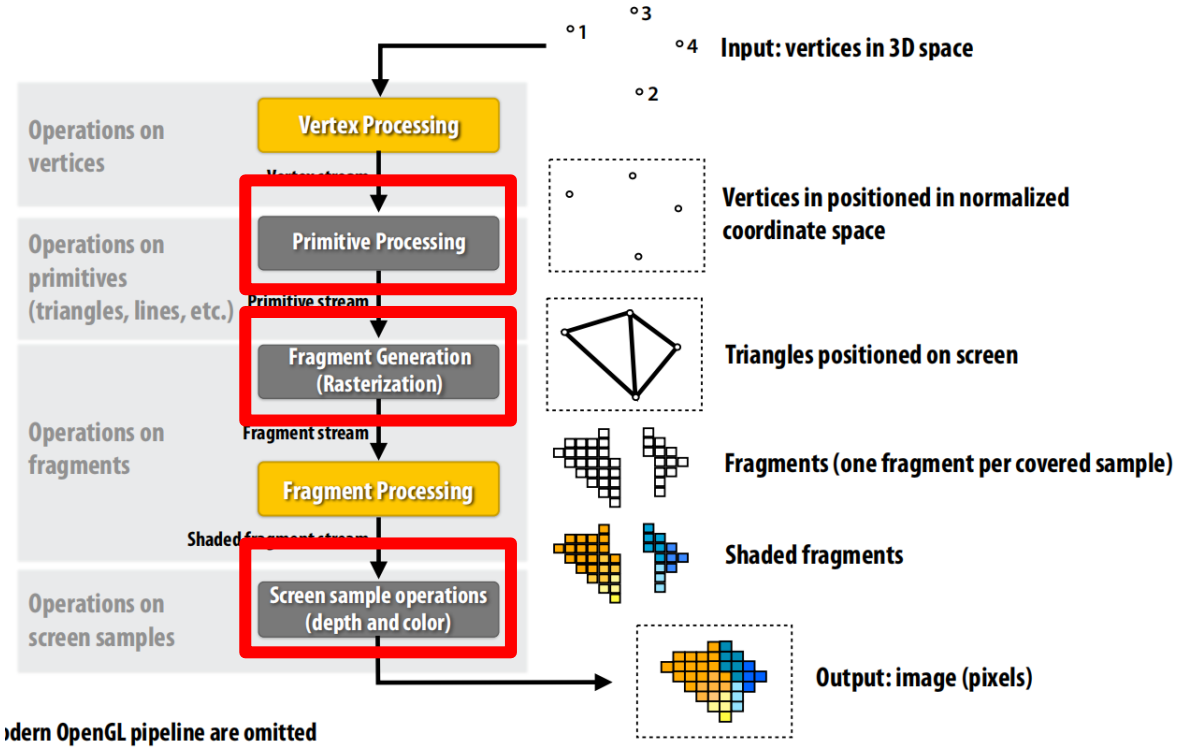
Computer Graphics

Shading 1 (Illumination, Shading and Graphics Pipeline)

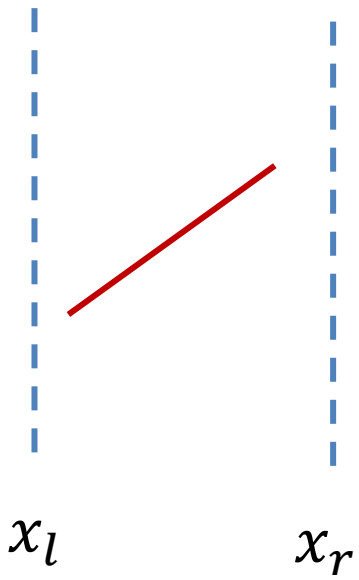


Last Lectures

- Clipping (裁剪)
- Scan Conversion of Line Segments
  - 直线段的扫描转化
- Scan Line Polygon Filling
  - 扫描线多边形填充
- Hidden Removal (消隐)



Liang-baskey algorithm



Which part of the line segment is visible?

$$x_l \overset{①}{\leq} x_1 + u\Delta x \overset{②}{\leq} x_r$$

$$\Delta x = x_2 - x_1,\ \Delta y = y_2 - y_1$$

Rewrite as

$$up_k \leq q_k$$

① $-u\Delta x \leq x_1 - x_l$ ① $\begin{cases} p_1 = -\Delta x \\ q_1 = x_1 - x_l \end{cases}$

② $u\Delta x \leq x_r - x_1$ ② $\begin{cases} p_2 = \Delta x \\ q_2 = x_r - x_1 \end{cases}$

1. 如果Δx=0，线段平行于X
   1. $q_k < 0$, **outside**
   2. $q_k \geq 0$, *inside*
2. 如果$\Delta x \neq 0$, 线段不平行
   - **Compute all $r_k = q_k / p_k$**
   - **u_1 is max(0, r_{in})**
   - **u_2 is min(1, r_{out})**
   - **If $u_1 > u_2$, outside the window**
   - **else, the part is from u_1 -> u_2**

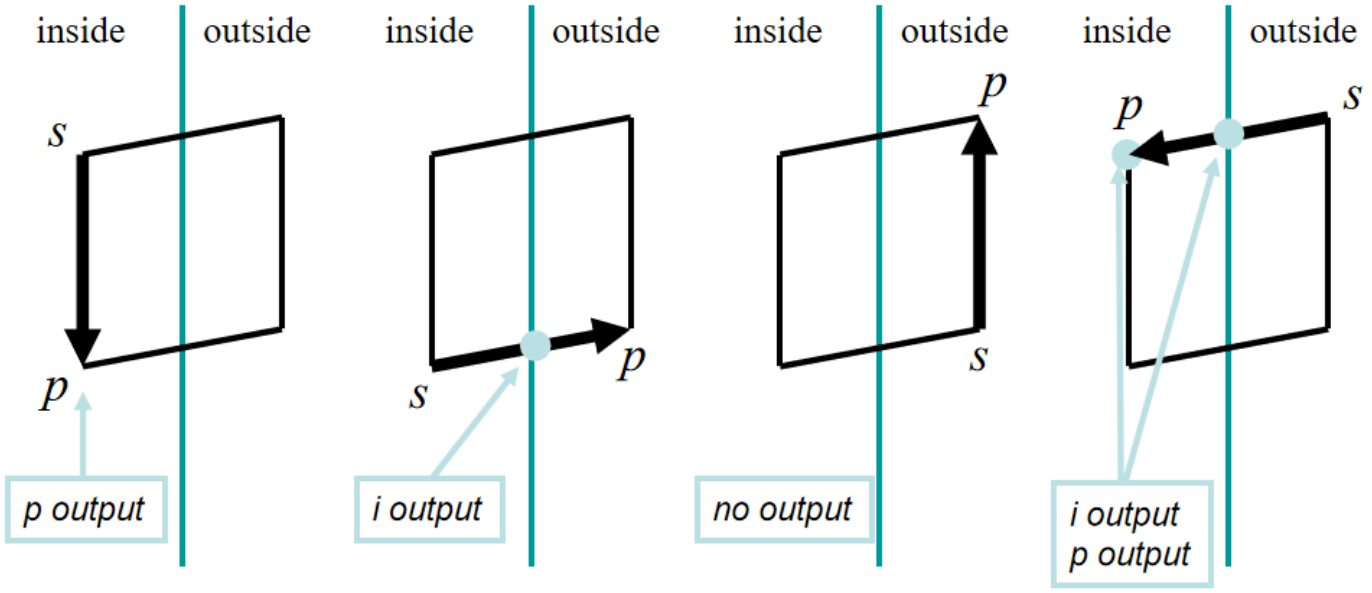
u_1 and u_2 defines the part of segment that is inside the window

Sutherland-hodgman algorithm

Let’s consider one step

The boundary divides the space into two parts: inside and outside

Consider a segment SP of the polygon, their relationship has 4 cases

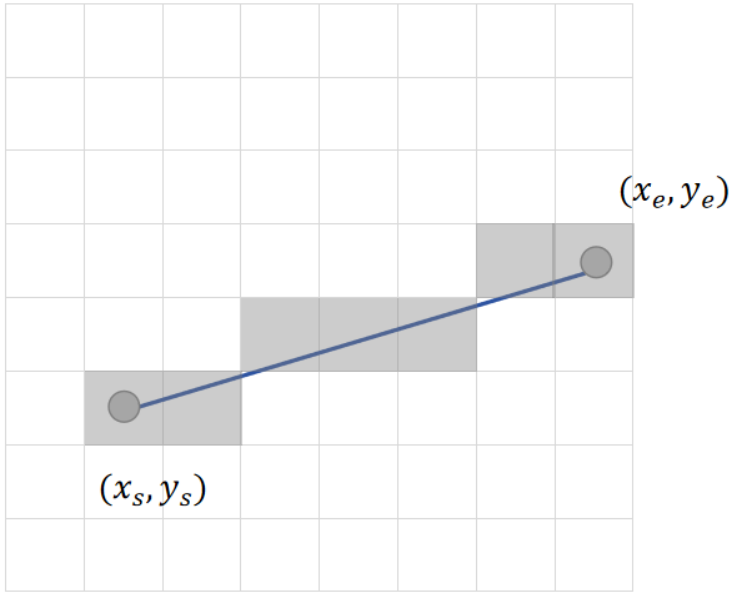


It outputs the points for the next steps.

How to draw a line segment?

DDA (Digital Differential Analyzer)

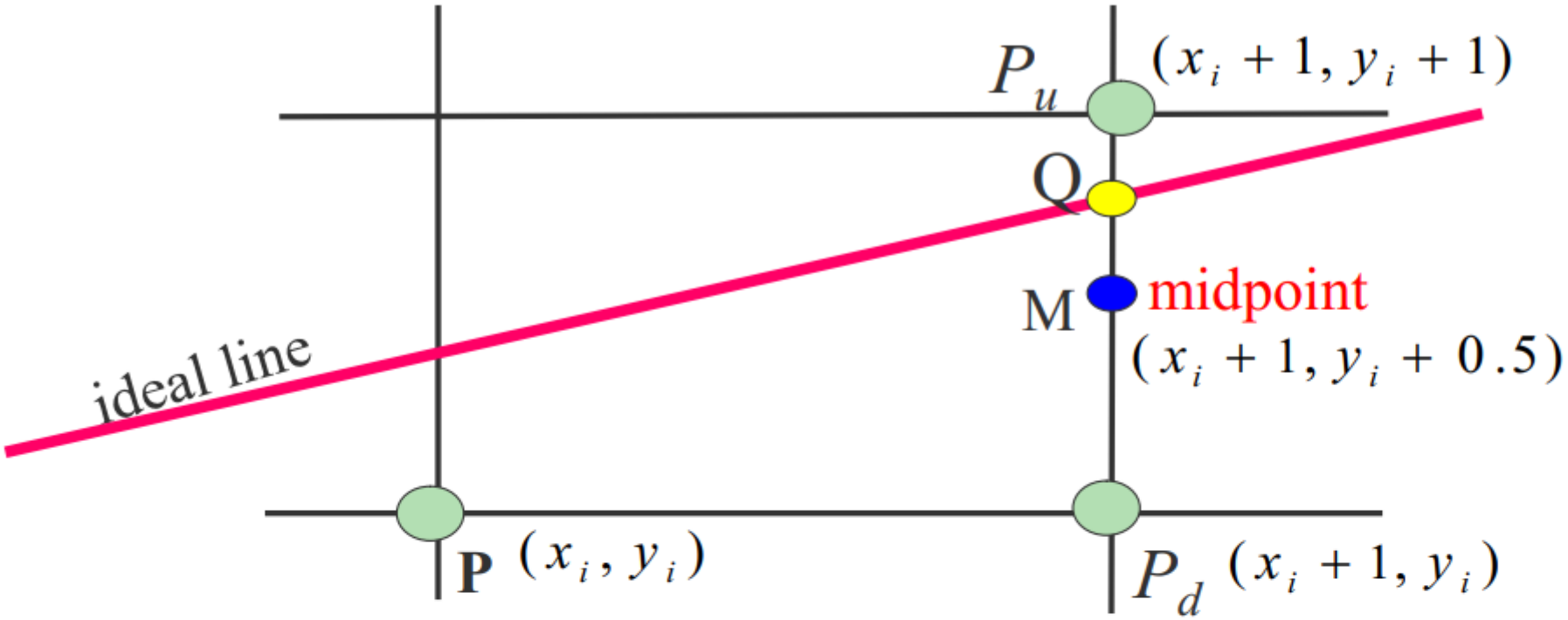
Light up all pixels intersected by the line
Digital Differential Analyzer (DDA, 数值微分法): find the pixels along x or y axis



Move along x axis

$$\Delta t = 1/\Delta x$$

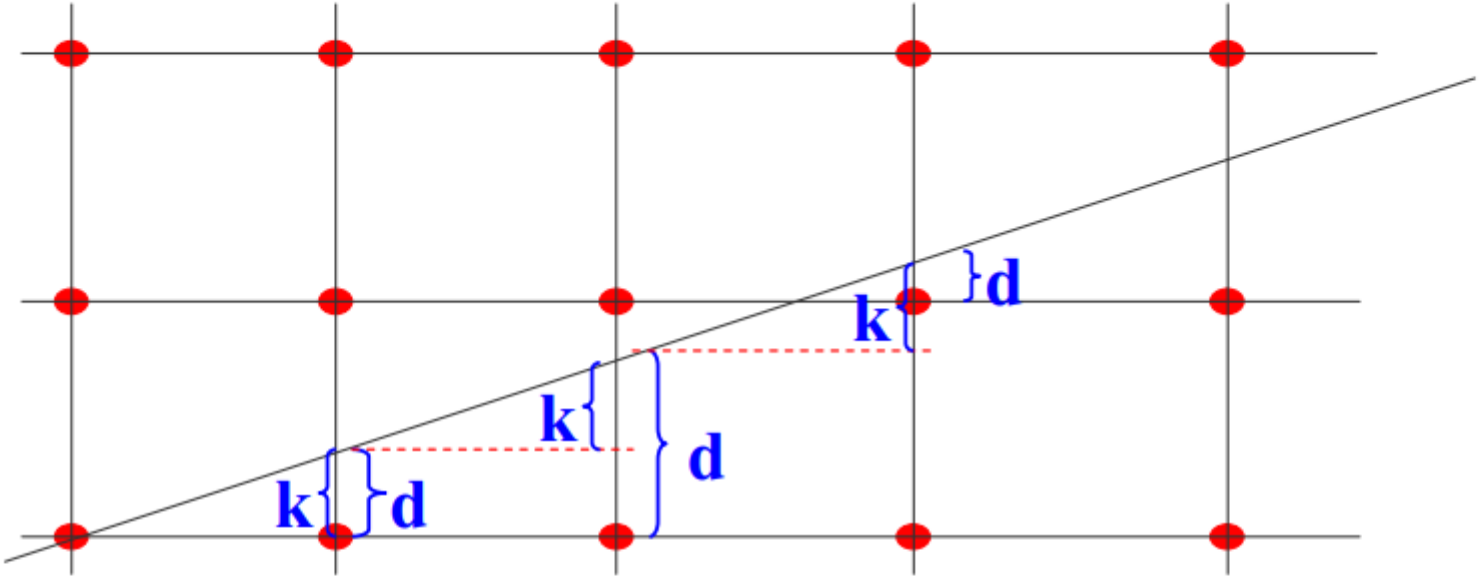
$$x = x_s + \Delta x \cdot \Delta t$$
$$y = y_s + \Delta y \cdot \Delta t$$



当M在Q的下方，则P_u离直线近，应为下一个象素点

当M在Q的上方，应取P_d 为下一点。

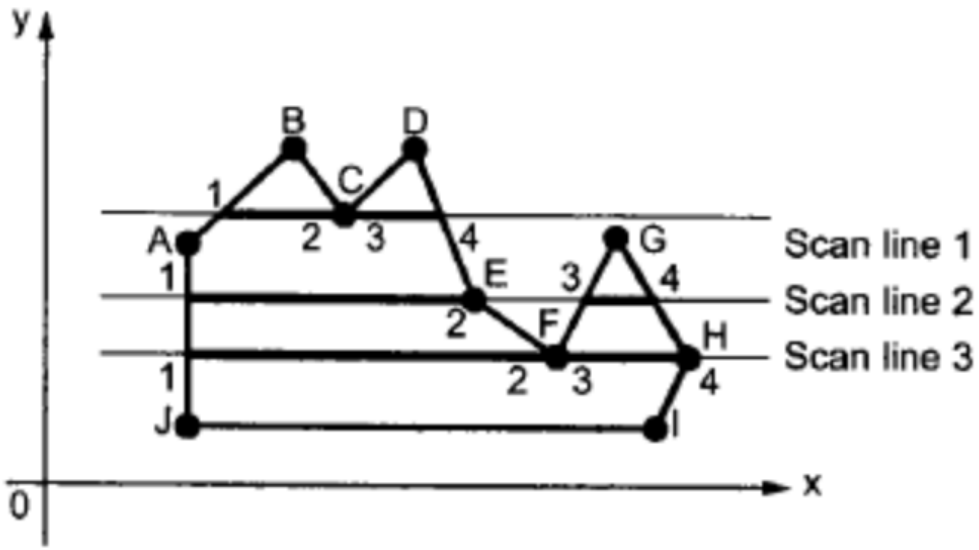
Bresenham算法的基本思想



该算法的思想是通过各行、各列像素中心构造一组虚拟网格线，按照直线起点到终点的顺序，计算直线与各垂直网格线的交点，然后根据误差项的符号确定该列象素中与此交点最近的象素。

Scanline fill algorithm

Scanline: scan along horizontal or vertical directions



For each scanline:

(1) Intersection. Compute the intersection between the scanline and polygon boundaries

(2) Sorting. Sort the intersections based on their x values.

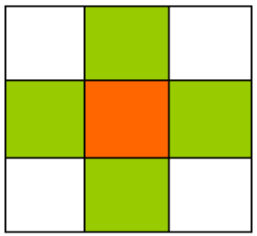
(3) Matching. Build points between <1st, 2nd>, <3rd, 4th>, ... intersections.

(4) Filling. Set the color for the pixels inside the paired interval.

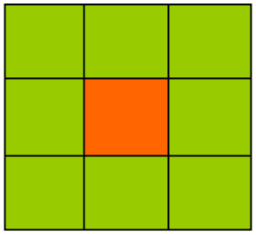
Easy to think, but low efficiency (intersection and sorting)

Flood fill algorithm （区域填充算法）

Suppose we already know one pixel inside the region



Four-neighbors

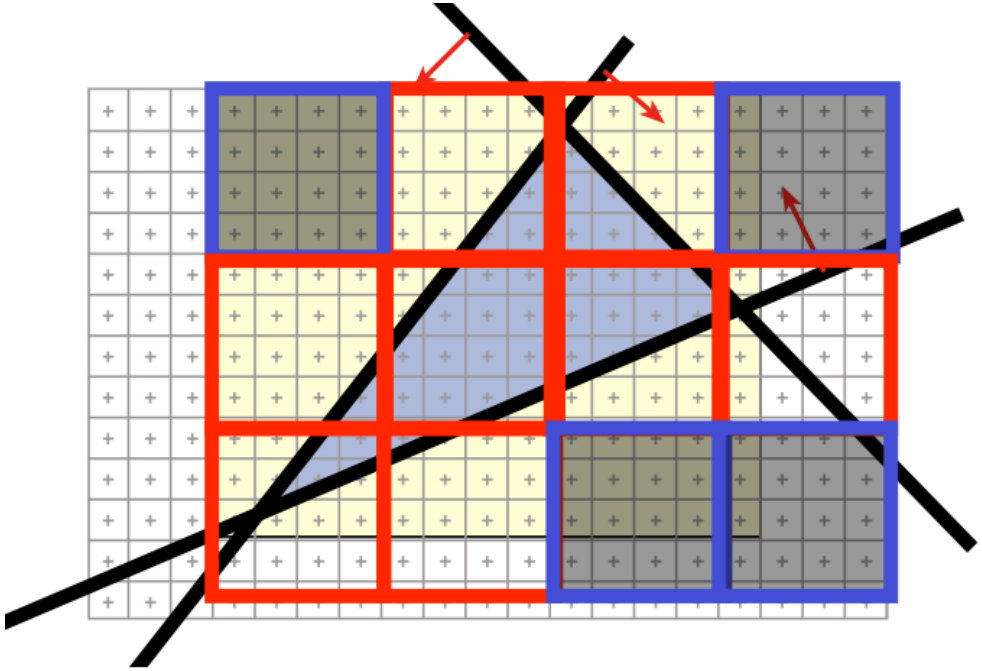


Eight-neighbors

```
void boundaryfill(int x,int y,int fill_color,int
boundary_color)
{
    if(getpixel(x,y)!=boundary_color &&
getpixel(x,y)!=fill_color)
    {
       putpixel(x,y,fill_color);
       boundaryfill(x+1,y,fill_color,boundary_color);
       boundaryfill(x-1,y, fill_color,boundary_color);
       boundaryfill(x,y+1, fill_color,boundary_color);
       boundaryfill(x,y-1, fill_color,boundary_color);
    }
}
```

Speed up the rasterizer

Axis-aligned bounding box
Incremental Edge Functions
Hierarchical Rasterization
    Conservatively test blocks before going to per-pixel level (can skip large blocks at once)



主要讲述的内容：

消隐的分类，如何消除隐藏线、隐藏面，主要介绍以下几个算法：

- Z缓冲区(Z-Buffer)算法
- 扫描线Z-buffer算法
- 区域子分割算法

核心思想

（1）增量思想：通过增量算法可以减少计算量

（2）编码思想

（3）符号判别—> 整数算法 尽可能的提高底层算法的效率，底层上提高效率才是真正解决问题

核心思想

（4）图形连贯性：利用连贯性可以大大减少计算量

（5）分而治之：把一个复杂对象进行分块，分到足够简单再进行处理

单选题 1分

设置

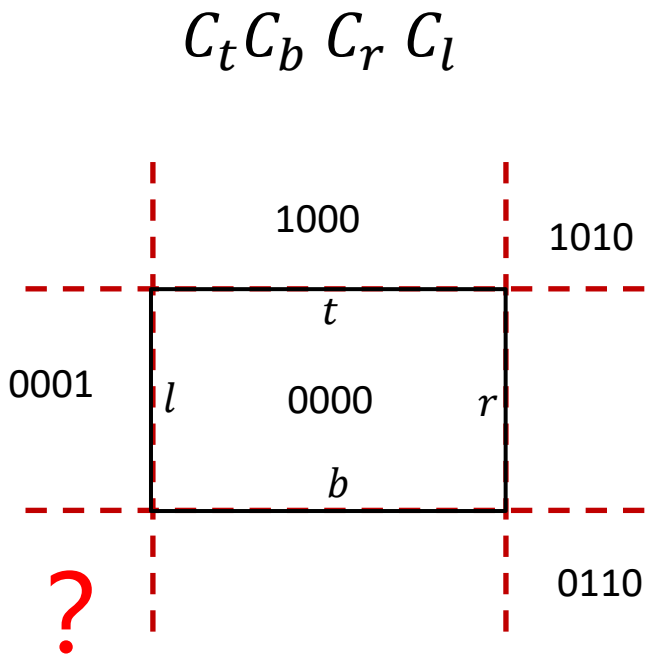
Sutherland-Cohen算法对屏幕空间进行编码，请问？处应该填写的编码是?

A 1010

B 0101

C 0100

D 1010



提交

单选题 1分

设置

Sutherland-Hodgman 算法中 空白 框 对应的顺序应当是?

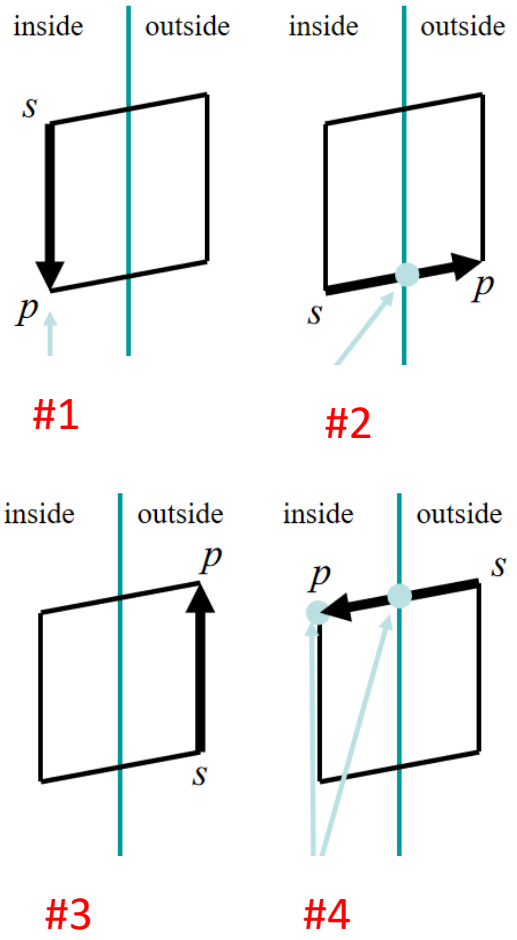
A 2,3,1,4

B 1,4,2,3

C 3,2,1,4

D 4,1,3,2

1, i output
2, i and p output
3, no output
4, p output



提交

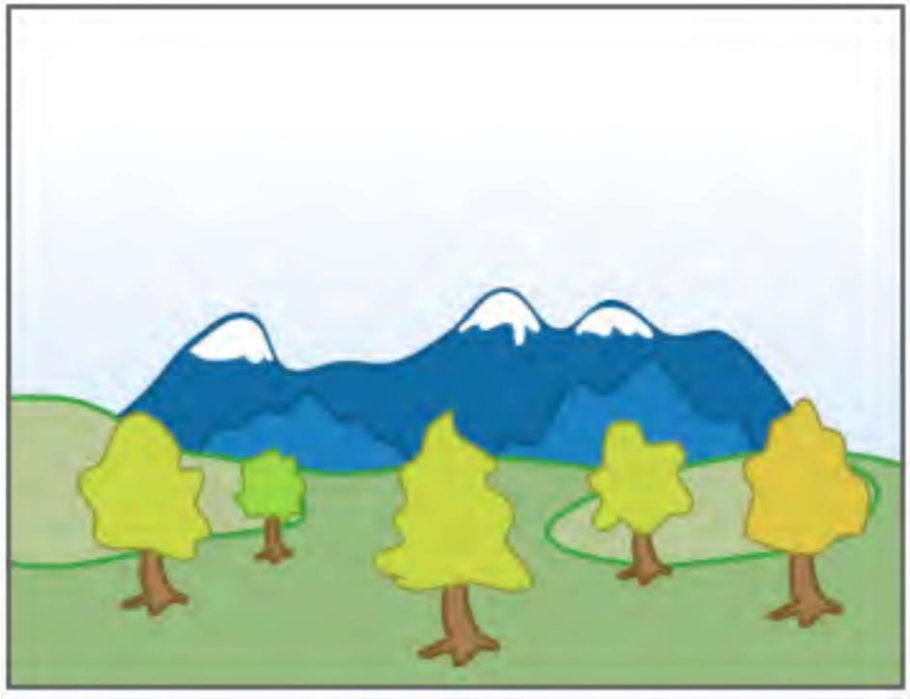
Today

- Visibility / occlusion
  - Z-buffering
- Shading
  - Blinn-Phong reflectance model
  - At a specific shading point
  - Shading frequencies
  - Graphics pipeline
  - Texture mapping
  - Barycentric coordinates

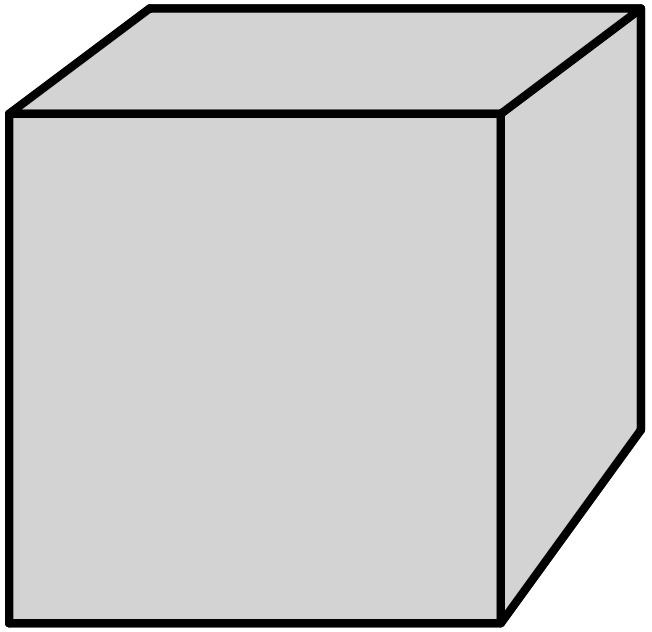
Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer



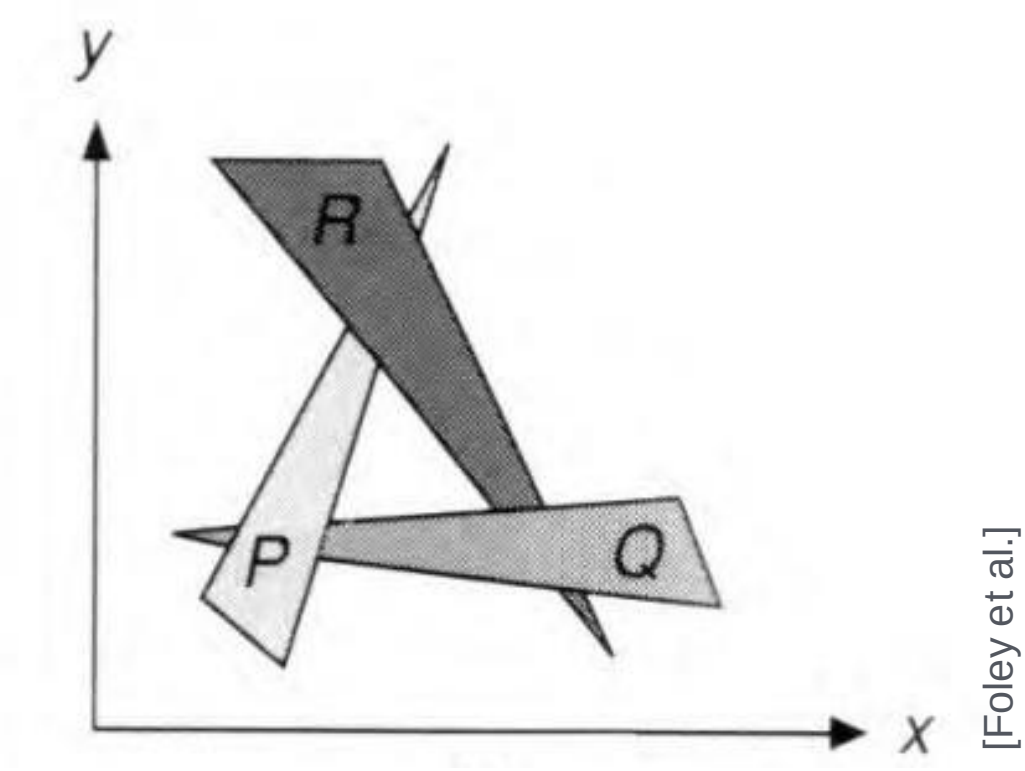
[Wikipedia]



Painter's Algorithm

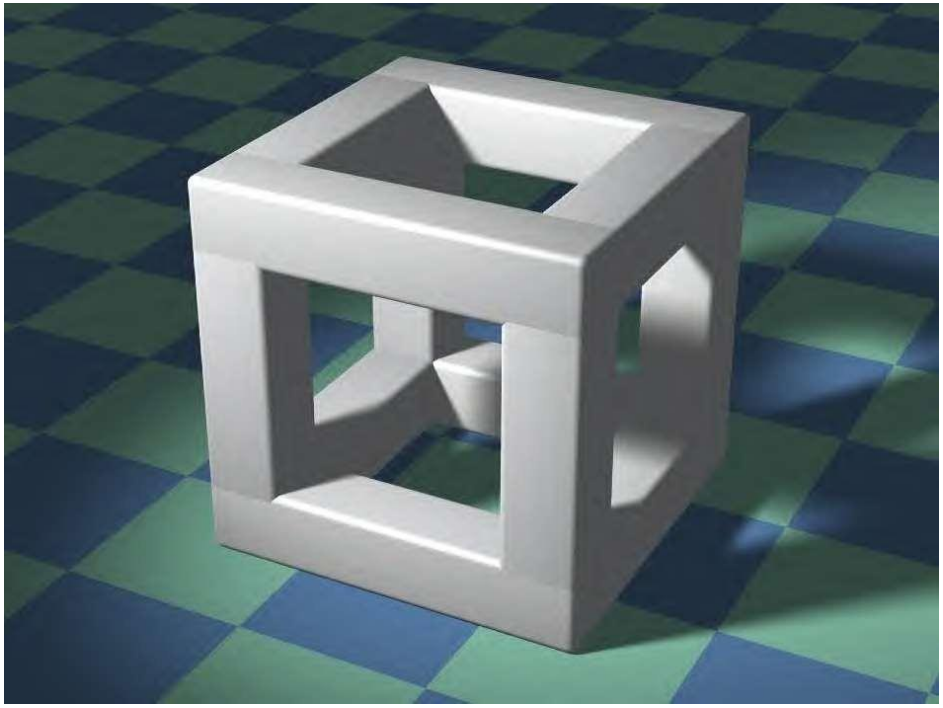
Requires sorting in depth (O(n log n) for n triangles)

Can have unresolvable depth order

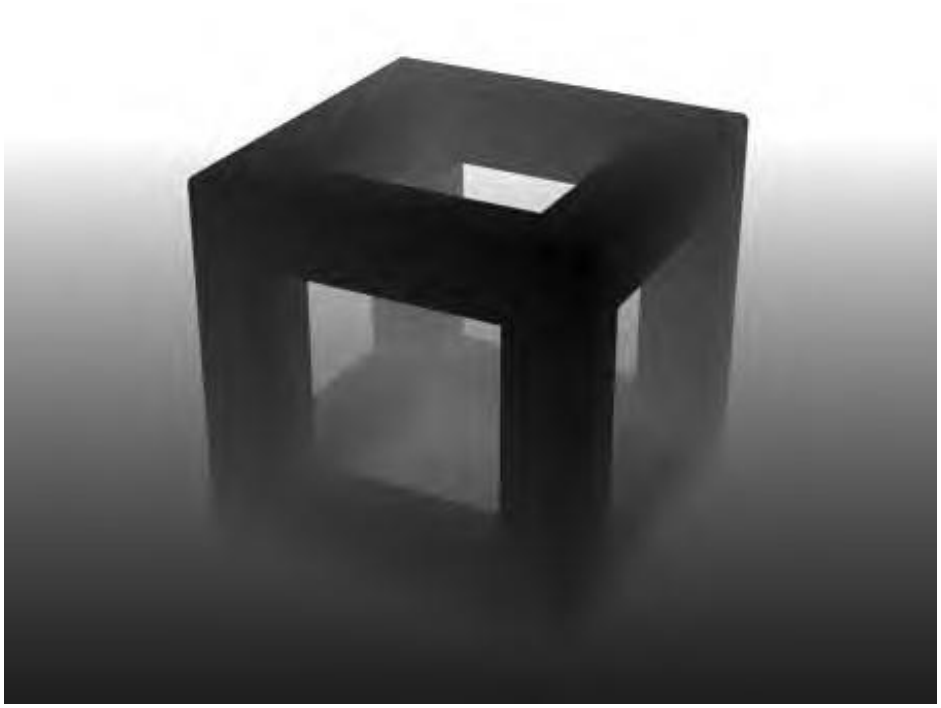


[Foley et al.]

Z-Buffer Example



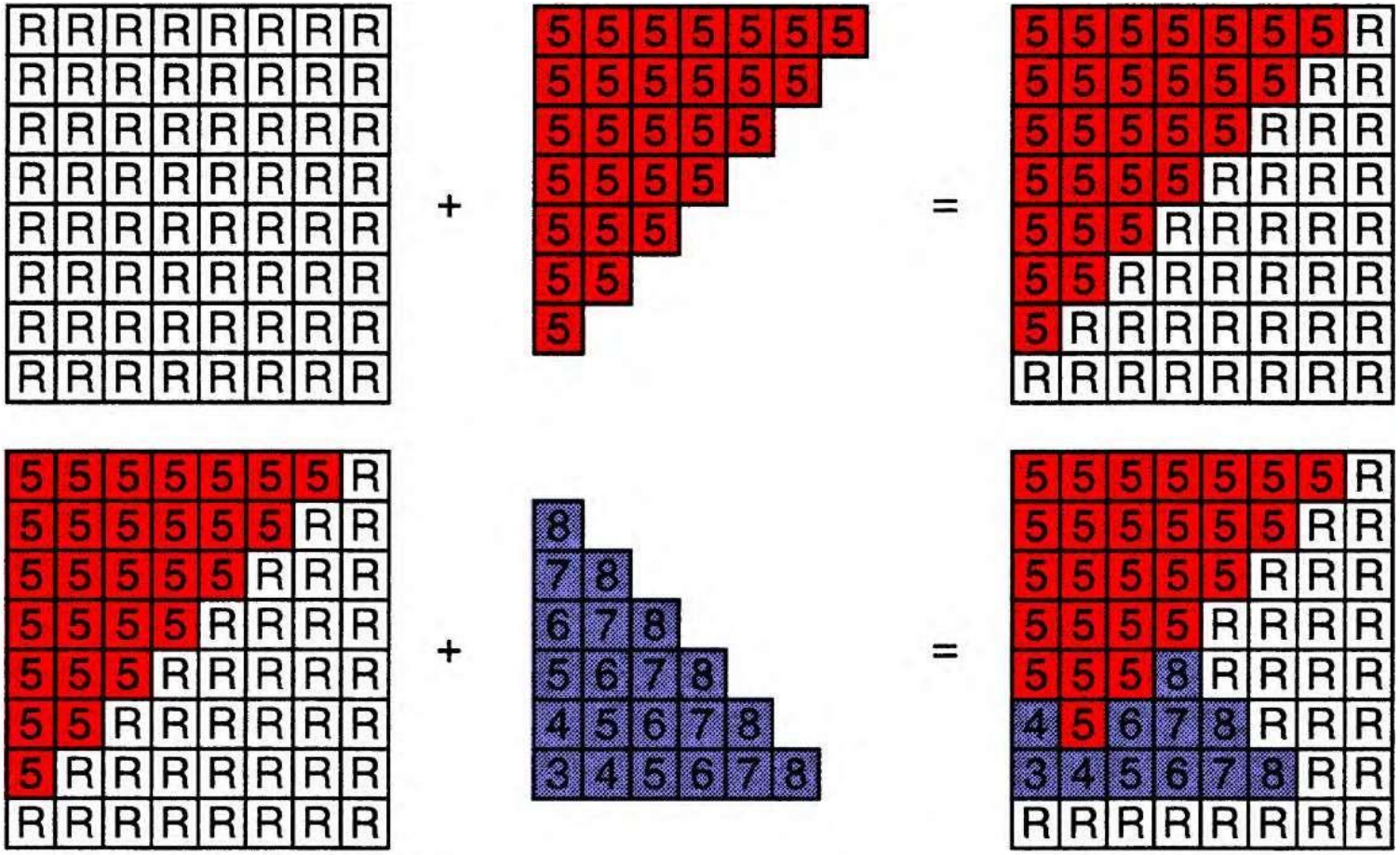
Rendering



Depth / Zbuffer

Image source: Dominic Alves, flickr.

Z-Buffer Algorithm



Z-Buffer Complexity

Complexity

- O(n) for n triangles (assuming constant coverage)
- How is it possible to sort n triangles in linear time?

Drawing triangles in different orders?

Most important visibility algorithm

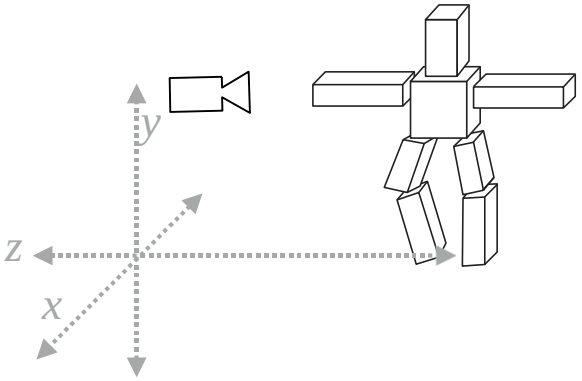
- Implemented in hardware for all GPUs

Questions?

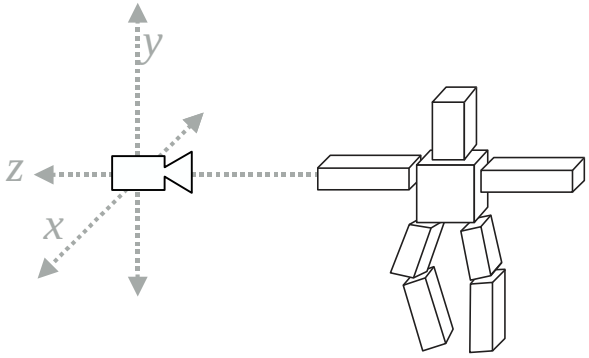
Today

- Visibility / occlusion
  - Z-buffering
- Shading
  - Illumination & Shading
  - Graphics Pipeline

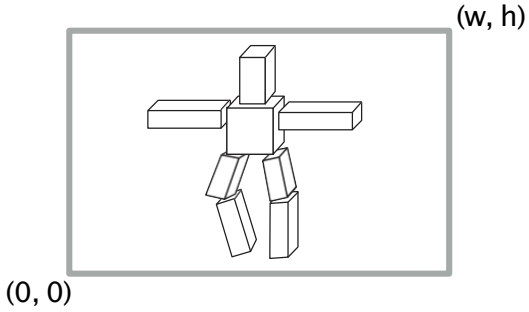
What We've Covered So Far



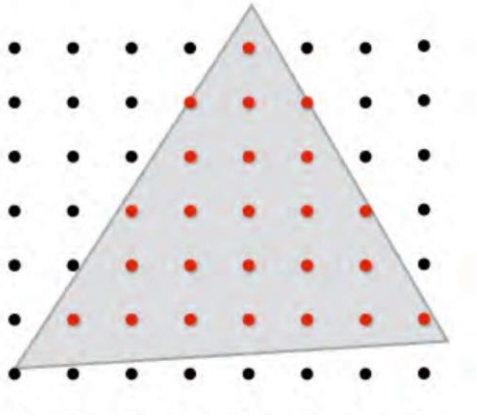
Position objects and the camera in the world



Compute position of objects relative to the camera

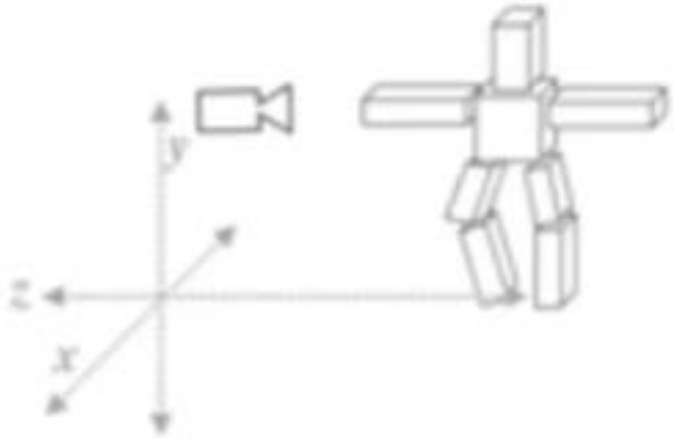


Project objects onto the screen

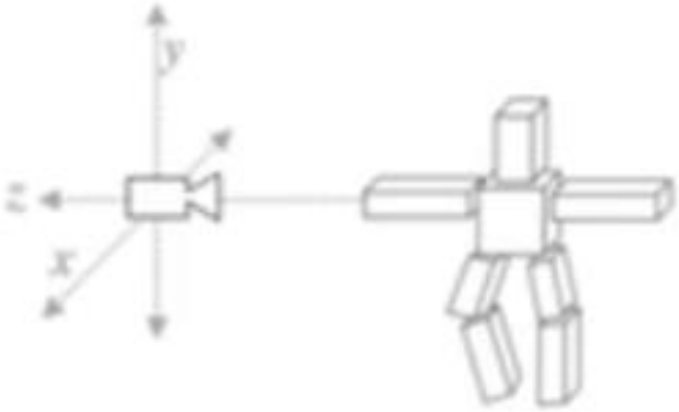


Sample triangle coverage

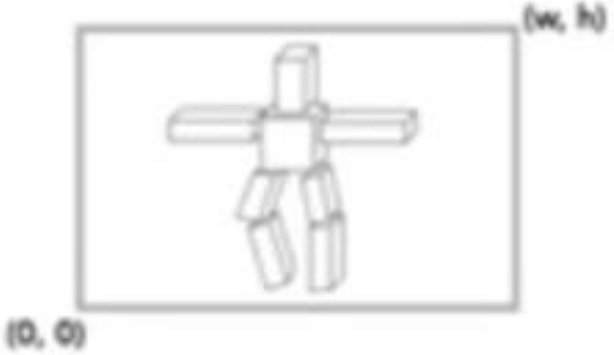
Rotating Cubes (Now You Can Do)



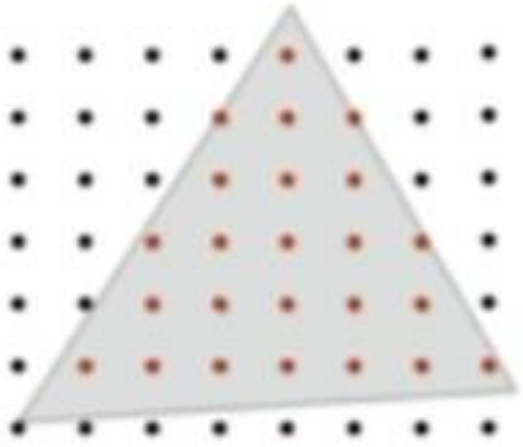
Position objects and the camera in the world



Compute position of objects relative to the camera



Project objects onto the screen



Sample triangle coverage

What Else Are We Missing?

演示GPT

Credit: Bertrand Benoit. “Sweet Feast,” 2009. [Blender /VRay]

Shading

Shading: Definition

- In Merriam-Webster Dictionary

  shad·ing, ['ʃeɪdɪŋ], noun
  The darkening or coloring of an illustration or diagram with parallel lines or a block of color.

  使用平行线或色块使得插图或图案变暗或着色

- In this course

  The process of applying a material to an object.

A Simple Shading Model (Blinn-Phong Reflectance Model)

Perceptual Observations

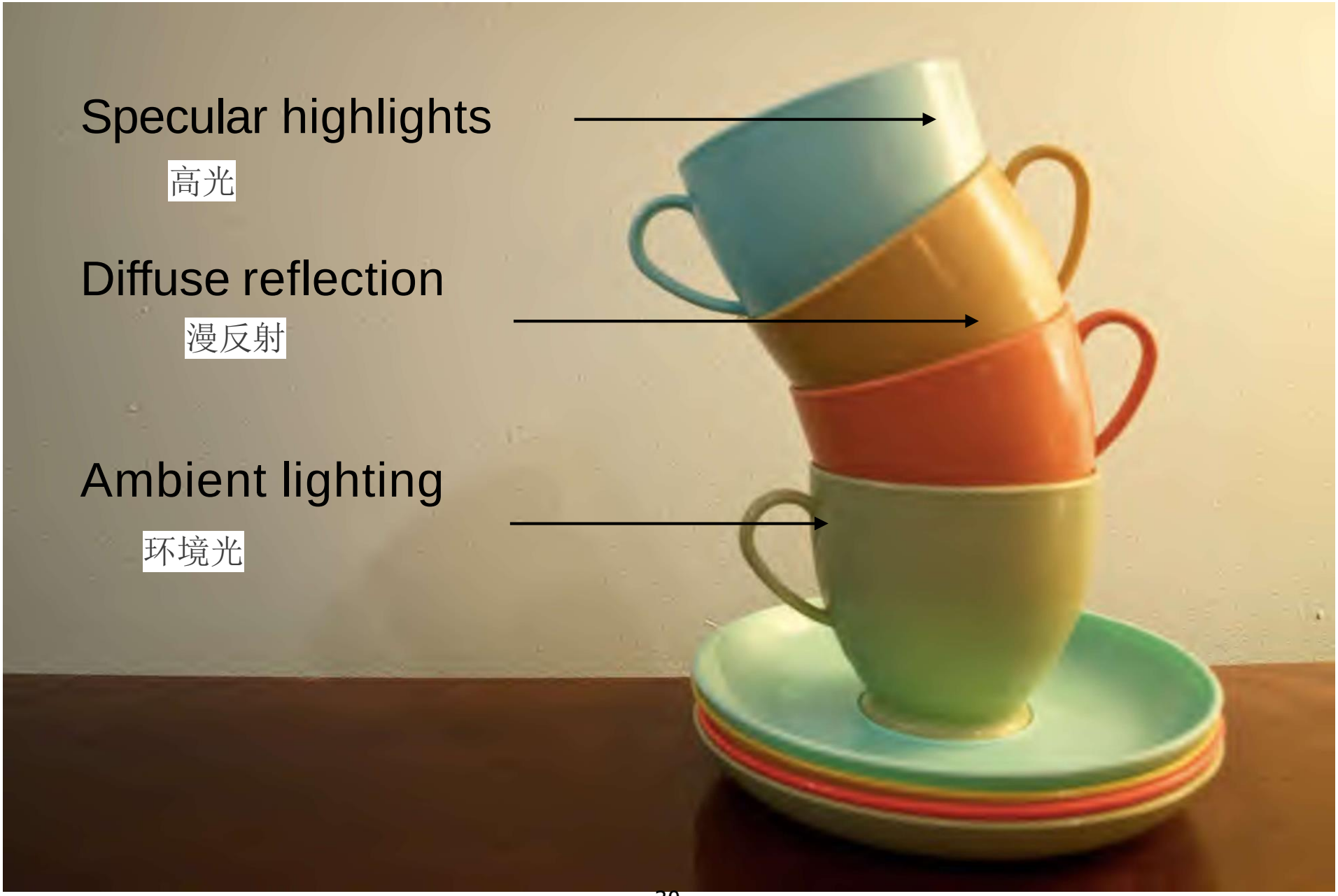


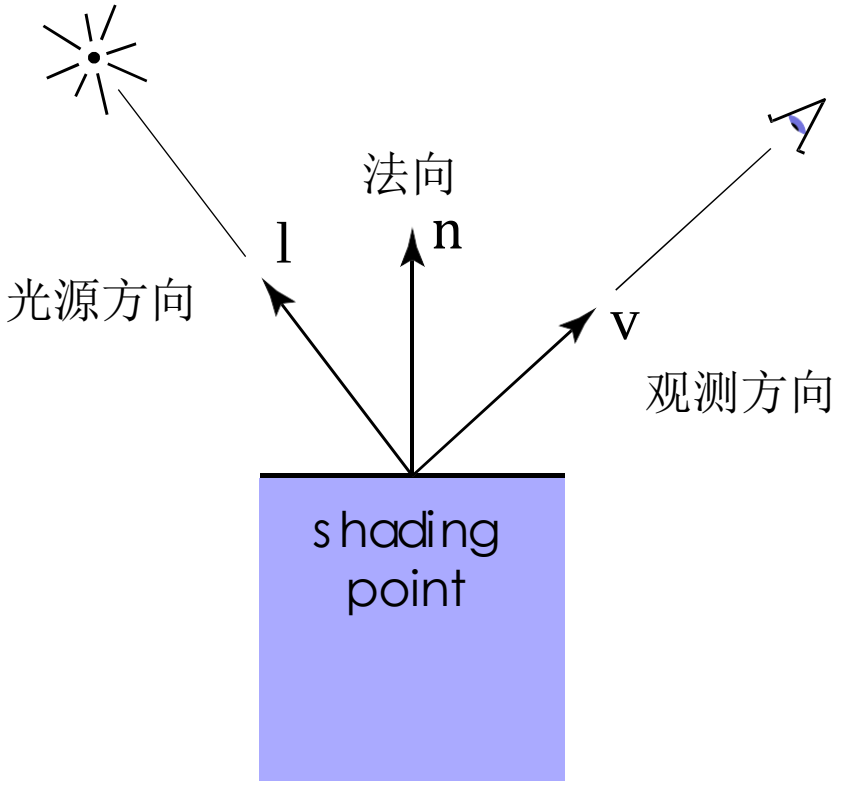
Photo credit: Jessica Andrews, flickr

Shading is Local

Compute light reflected toward camera
at a specific shading point

Inputs:

- Viewer direction, v
- Surface normal, n
- Light direction, l
  (for each of many lights)
- Surface parameters
  (color, shininess, …)

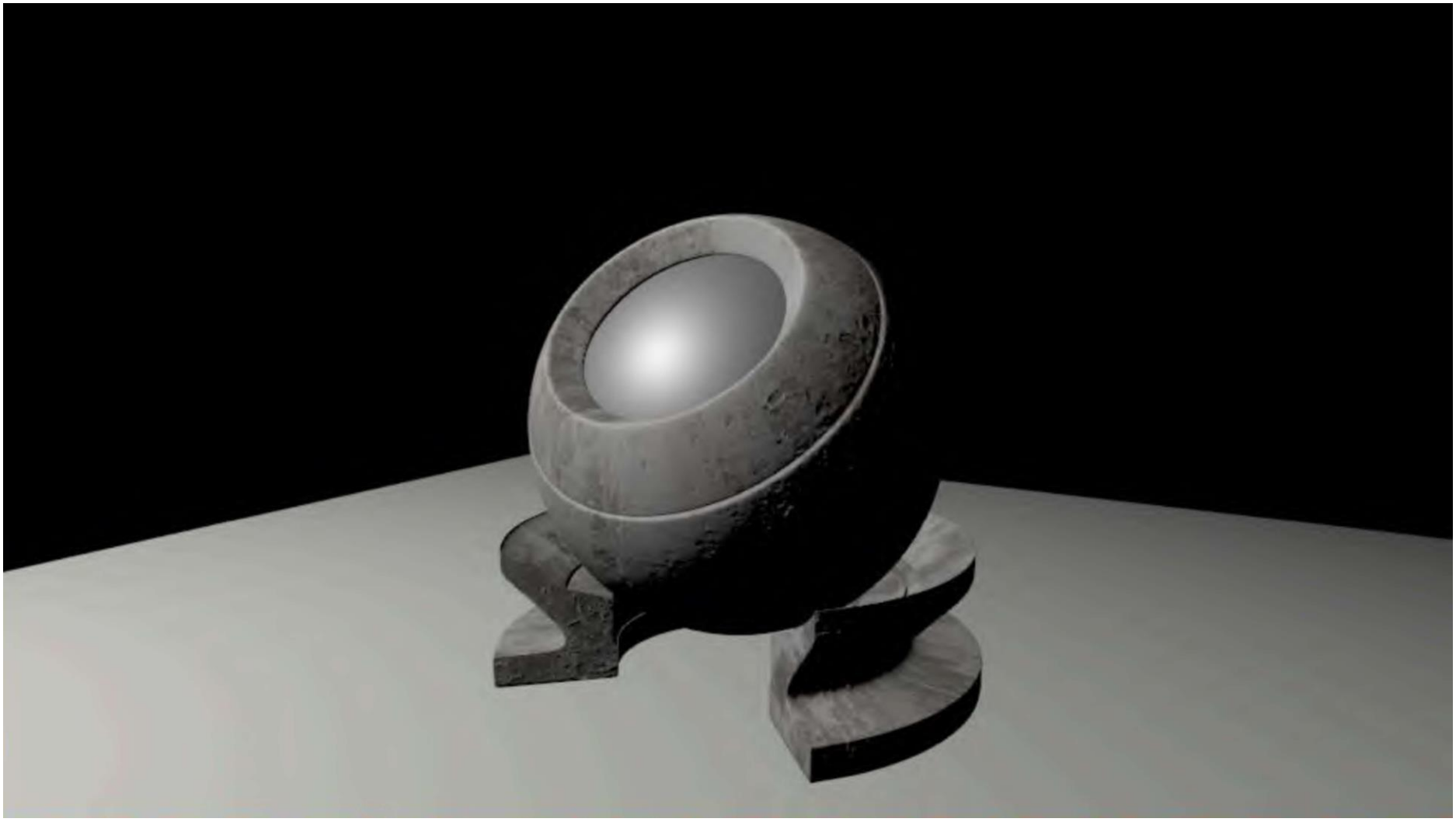


注意（note）：都是单位向量

Shading is Local

Why?

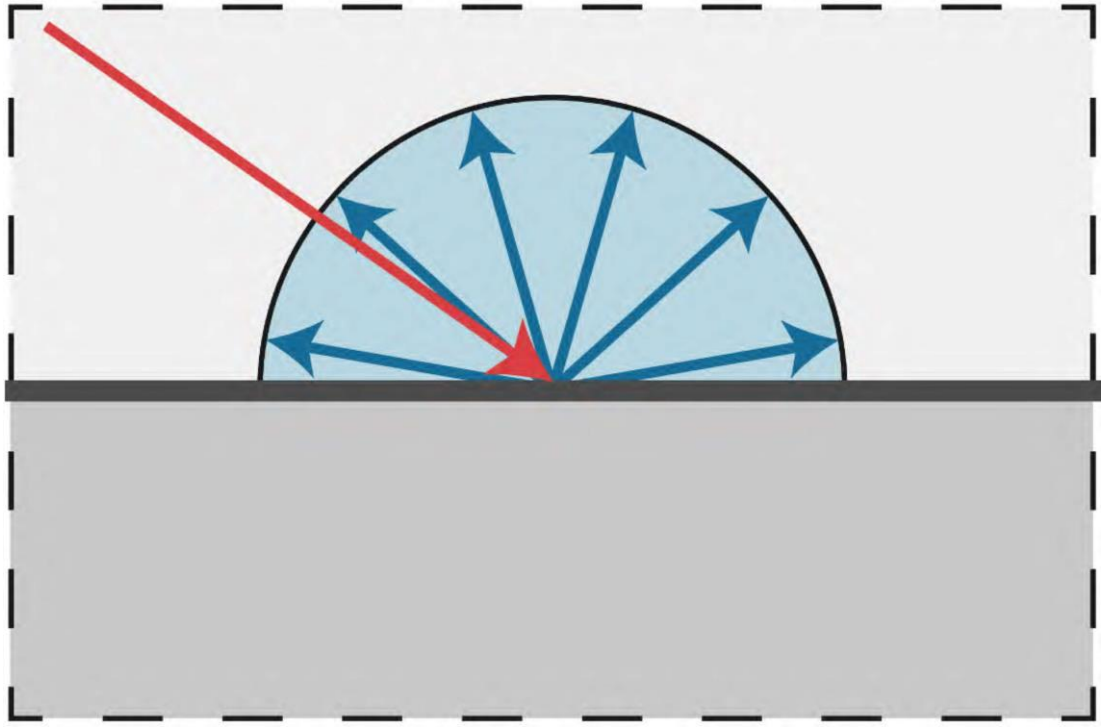
No shadows will be generated! (shading ≠ shadow)



Diffuse Reflection

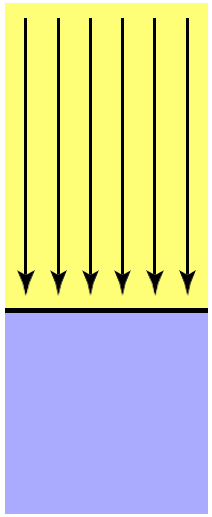
漫反射

- Light is scattered uniformly in all directions
  - Surface color is the same for all viewing directions

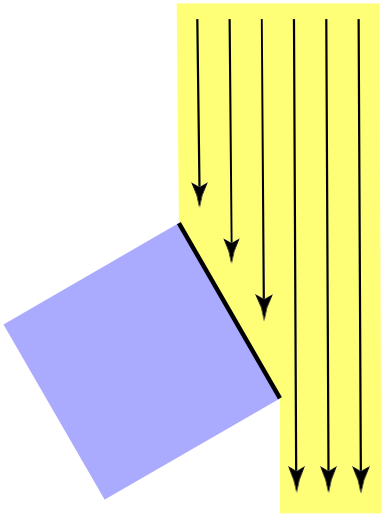


Diffuse Reflection

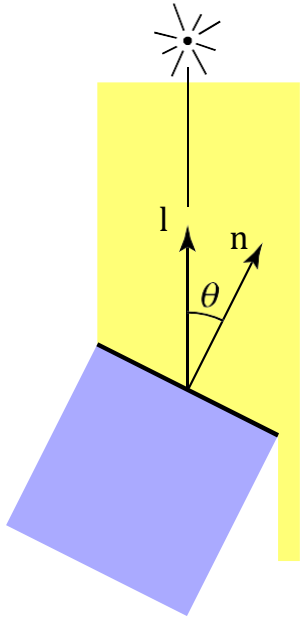
- But how much light (energy) is received?
  - Lambert's cosine law



Top face of cube receives a certain amount of light

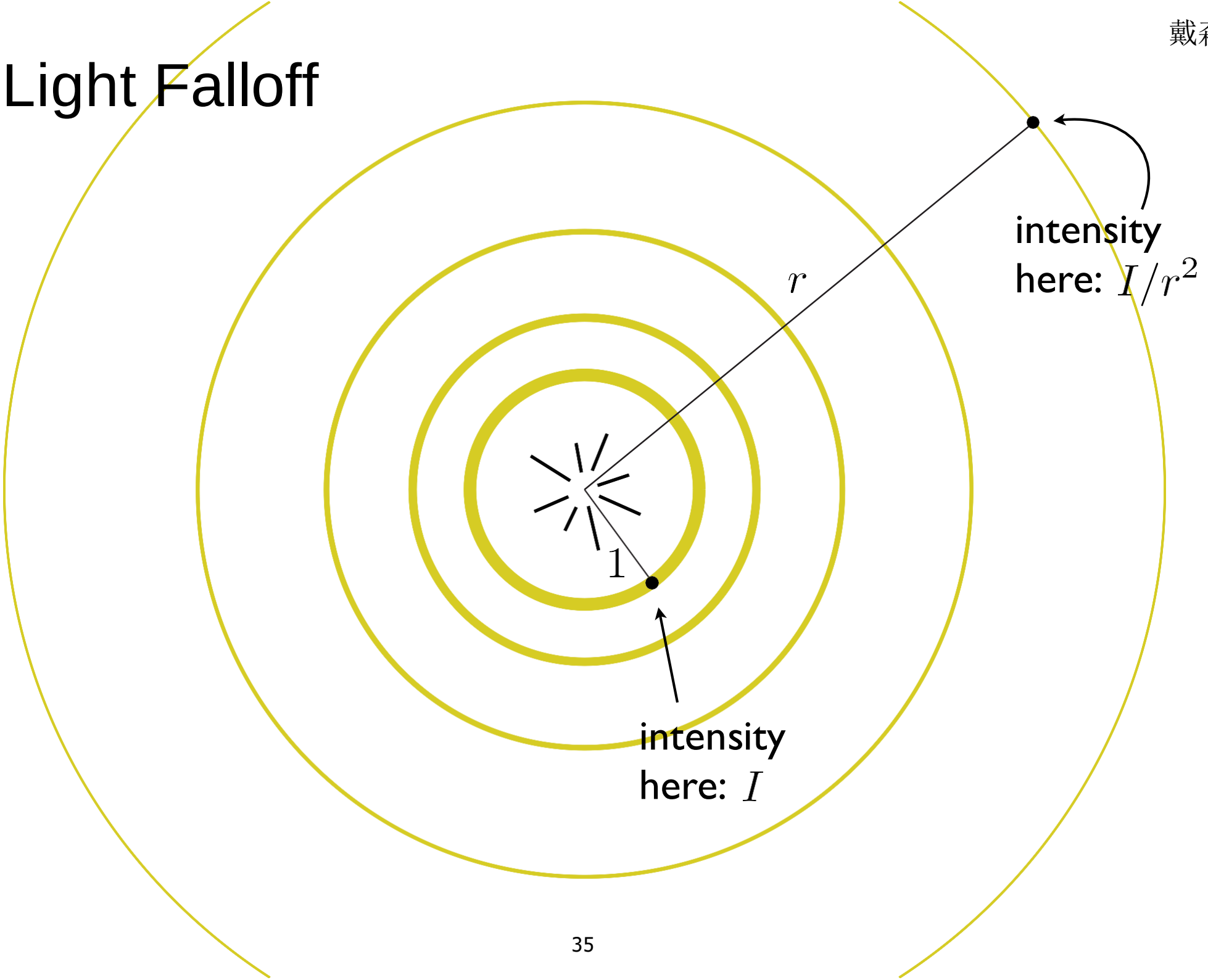


Top face of 60° rotated cube intercepts half the light



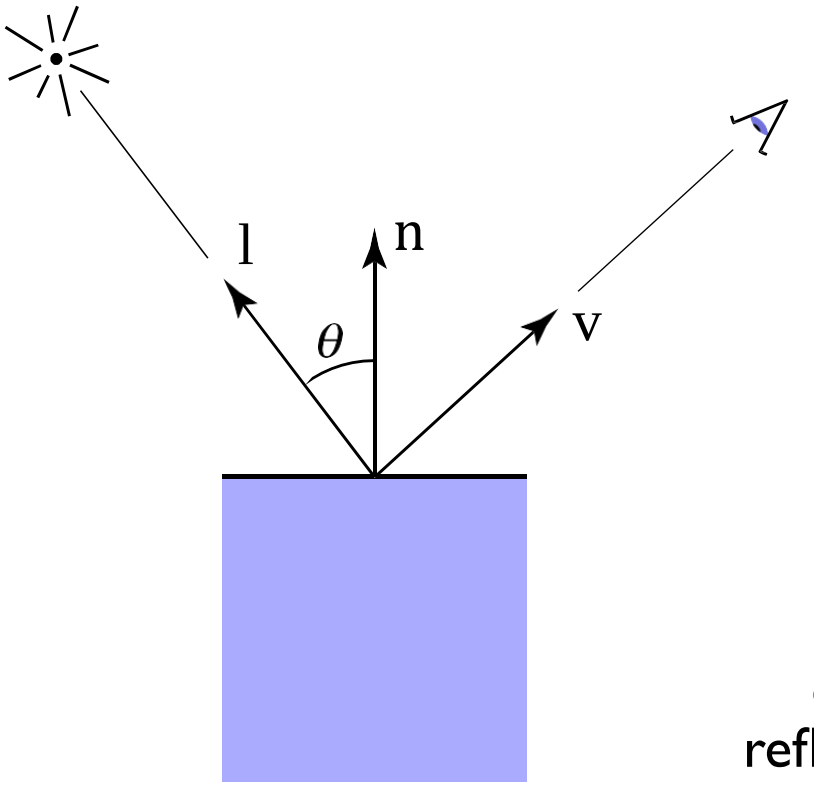
In general, light per unit area is proportional to $\cos \theta = l \cdot n$

Light Falloff



Lambertian (Diffuse) Shading

Shading independent of view direction



energy arrived
at the shading point

$$L_d = k_d \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l})$$

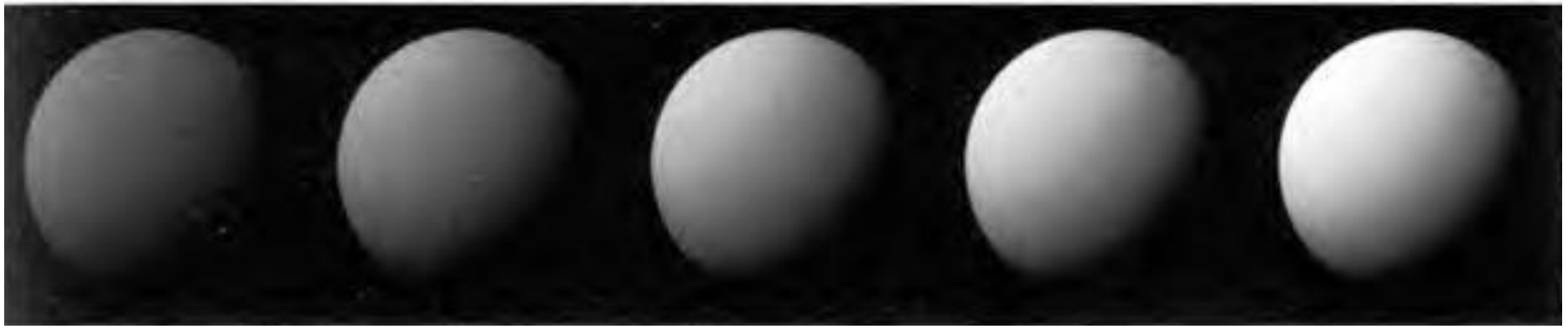
diffuse
coefficient
(color)

energy received
by the shading point

diffusely
reflected light

Lambertian (Diffuse) Shading

Produces diffuse appearance



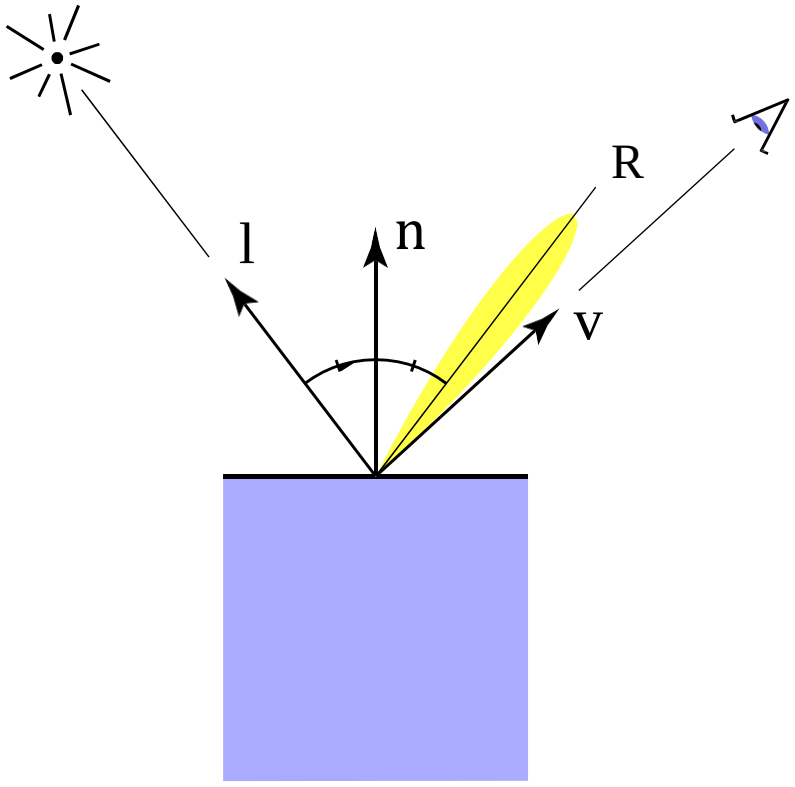
k_d ⟶

[Foley et al.]

Specular Term （高光项） (Blinn-Phong)

Intensity depends on view direction

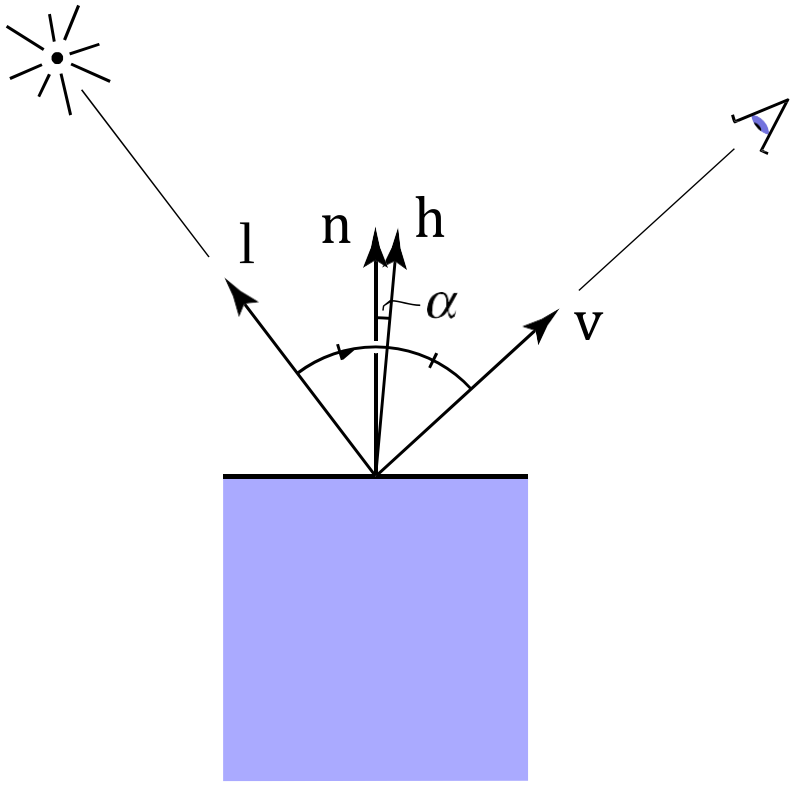
- Bright near mirror reflection direction



Specular Term (Blinn-Phong)

V close to mirror direction ⇔ half vector near normal

- Measure "near" by dot product of unit vectors



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

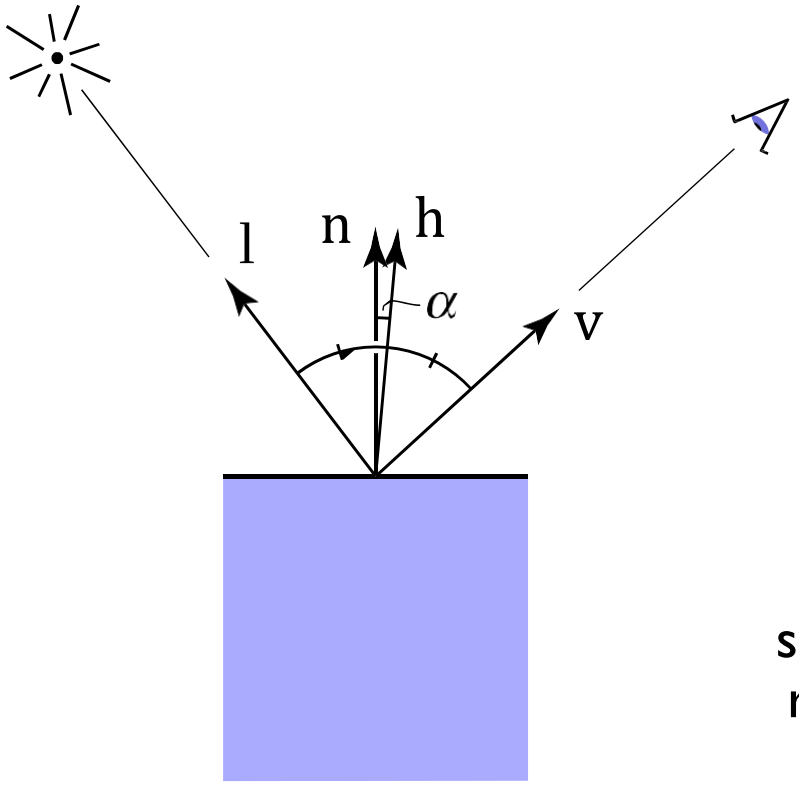
(半程向量)

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

Specular Term (Blinn-Phong)

V close to mirror direction ⇔ half vector near normal

- Measure "near" by dot product of unit vectors



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

(半程向量)

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

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

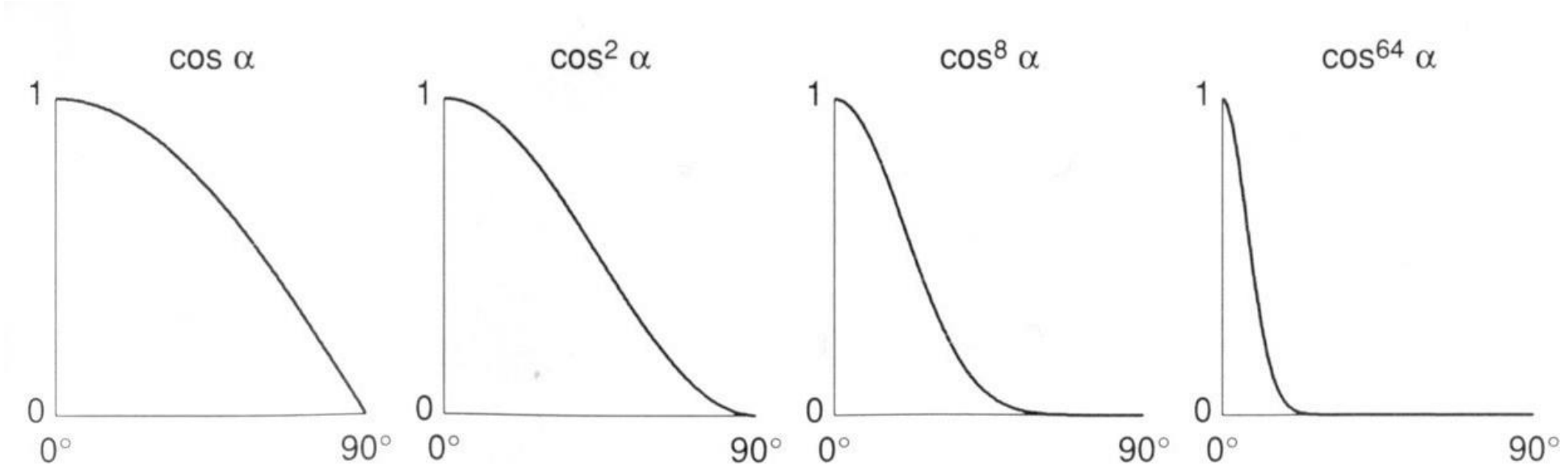
$$= k_s \left(I/r^2\right) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

specularly reflected light

specular coefficient

Cosine Power Plots

Increasing p narrows the reflection lobe



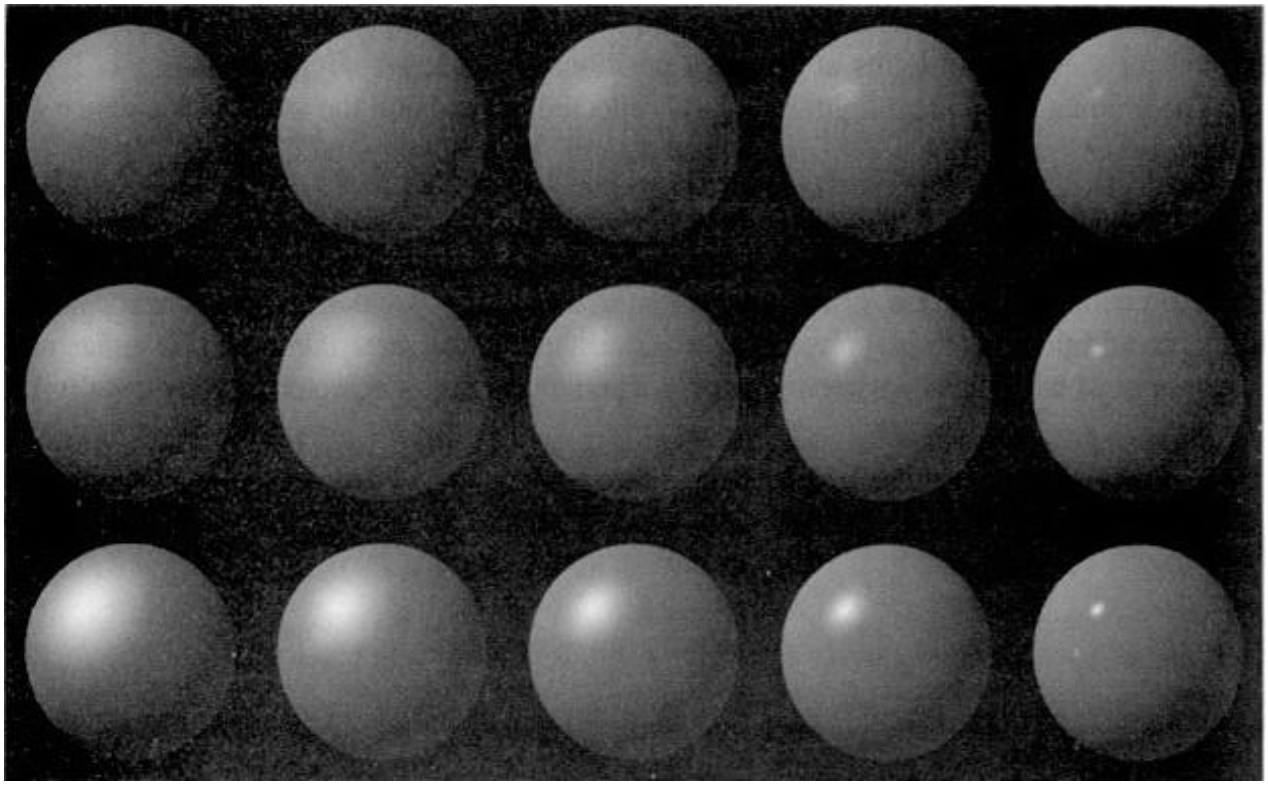
[Foley et al.]

通常用：100-200

Specular Term (Blinn-Phong)

Blinn-Phong

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



k_s

p

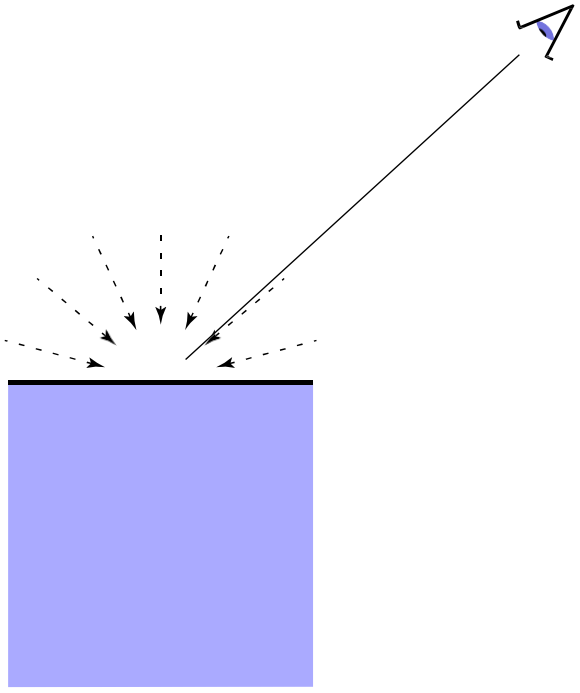
Note: showing
Ld + Ls together

[Foley et al.]

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!

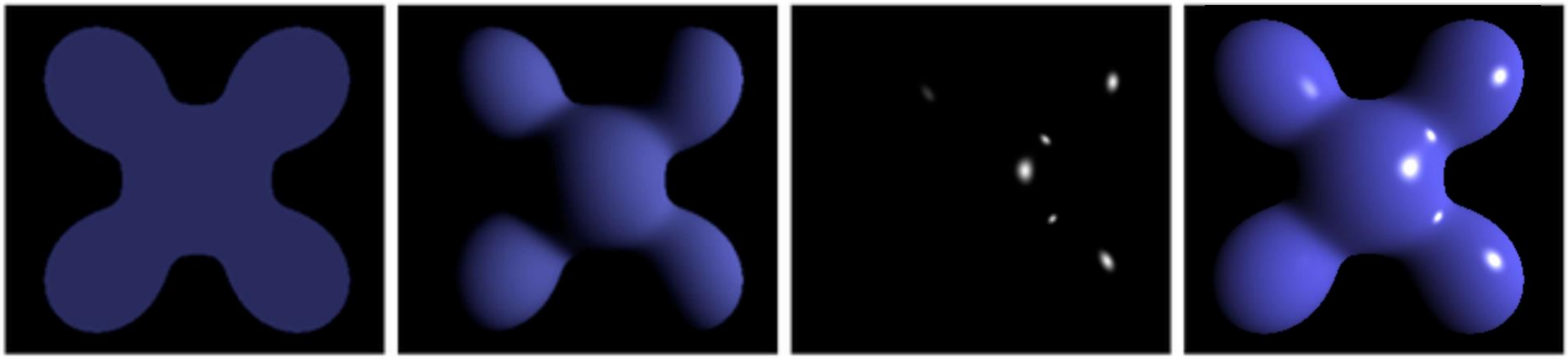


$$L_a = k_a I_a$$

ambient coefficient

reflected ambient light

Blinn-Phong Reflection Model



Ambient + Diffuse + Specular = Blinn-Phong Reflection

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

Shading Frequencies

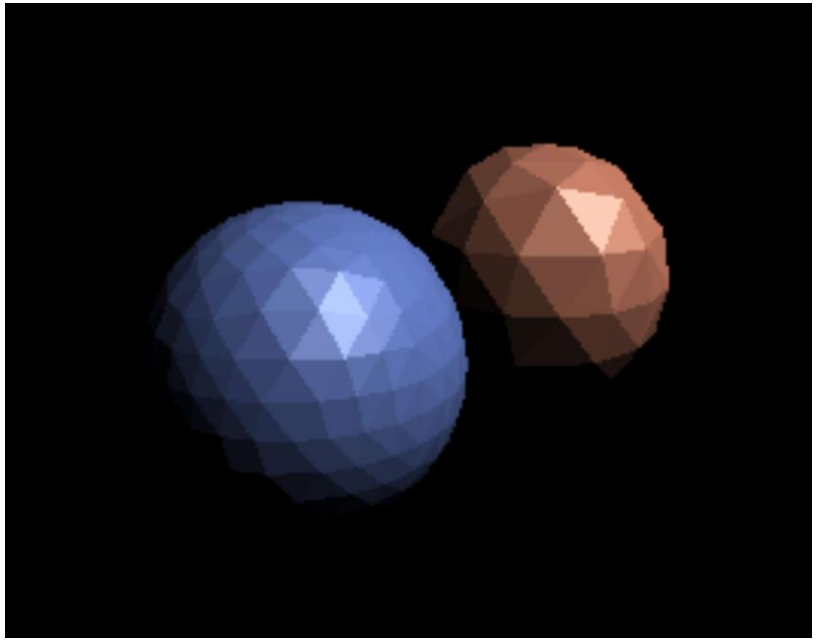
Shading Frequencies

What caused the shading difference?

Shade each triangle (flat shading)

Flat shading

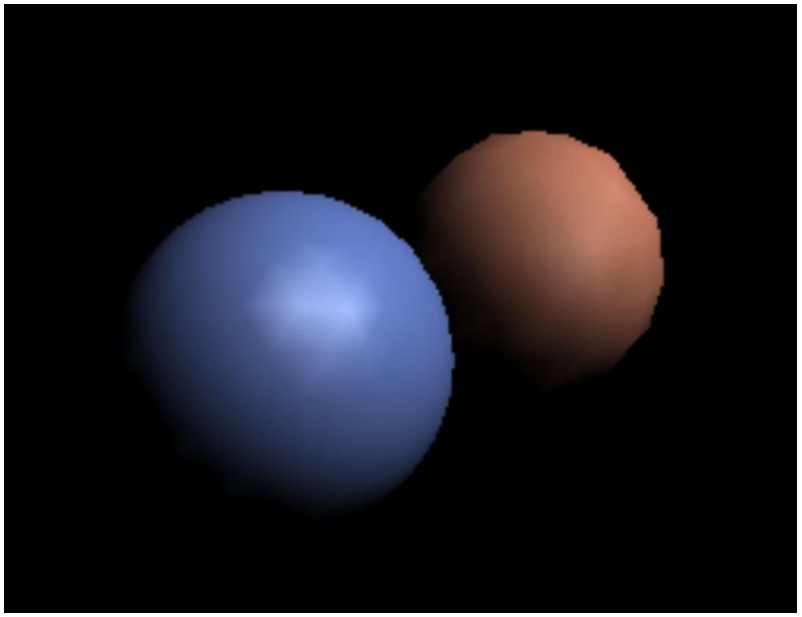
- Triangle face is flat — one normal vector
- Not good for smooth surfaces



Shade each vertex (Gouraud shading)

Gouraud shading

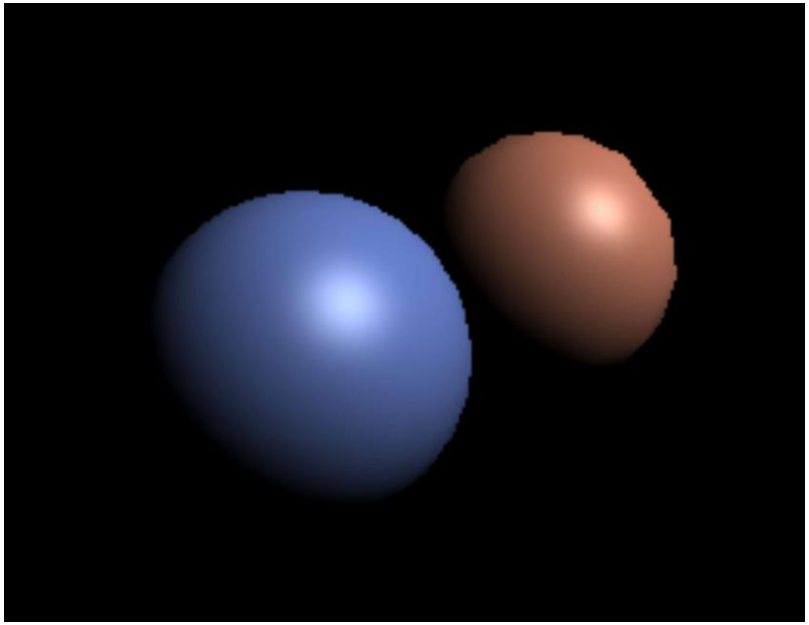
- Interpolate colors from vertices across triangle
- Each vertex has a normal vector (how?)



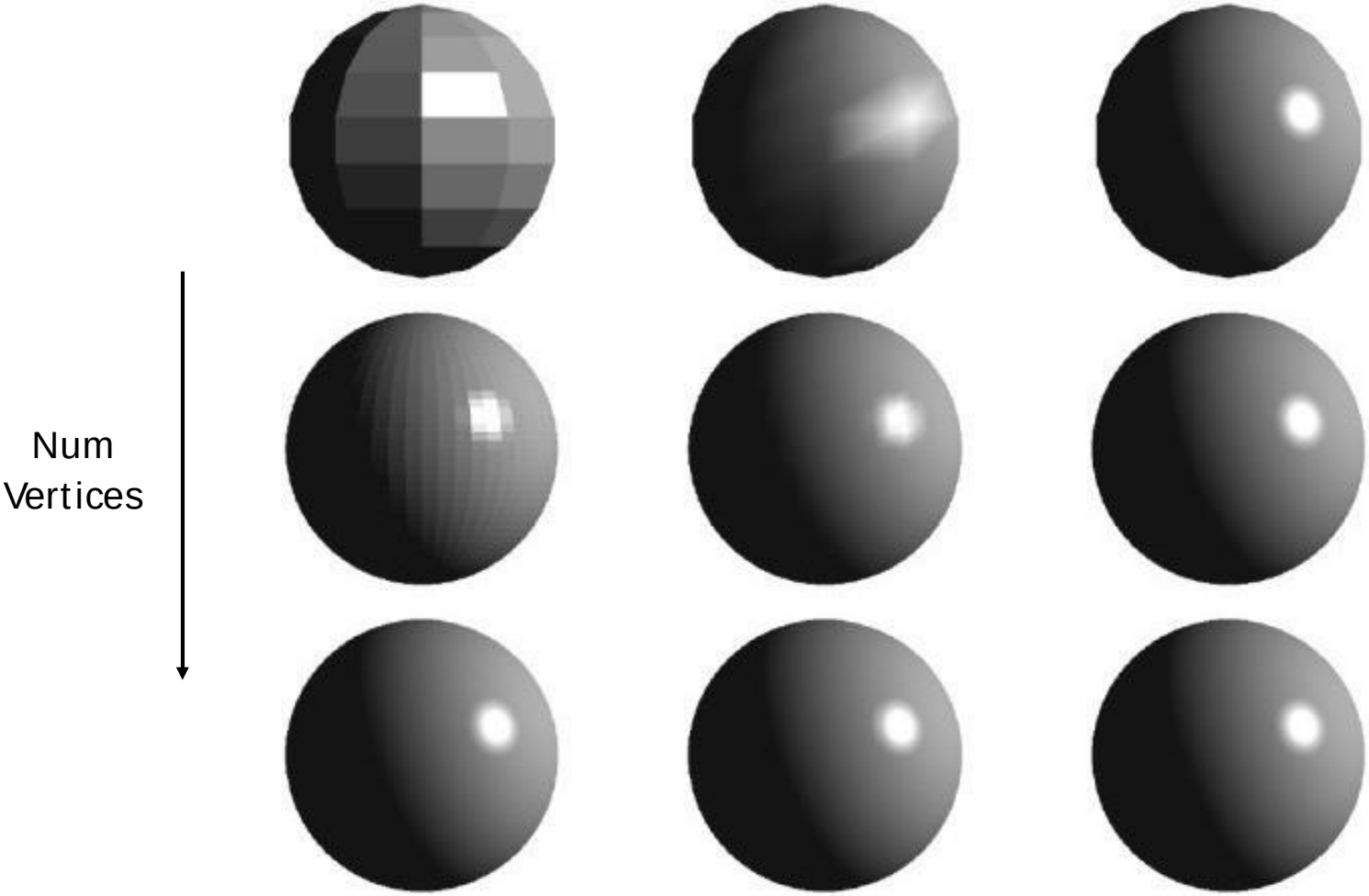
Shade each pixel (Phong shading)

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



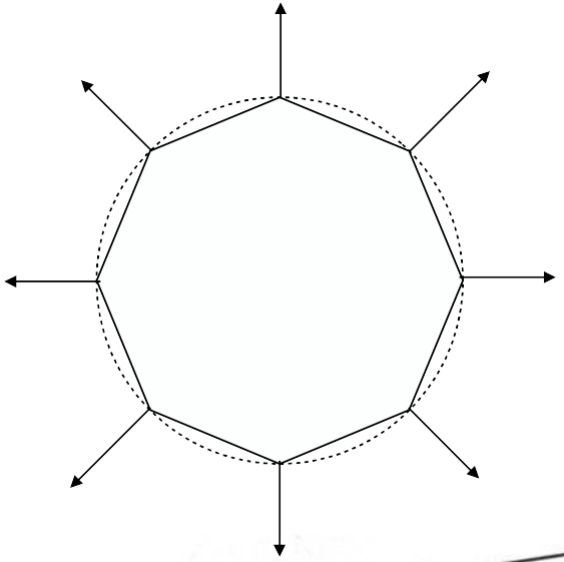
Shading freq. :	Face	Vertex	Pixel
Shading type :	Flat	Gouraud	Phong

Image credit: Happyman, http://cg2010studio.com/

Defining Per-Vertex Normal Vectors

Best to get vertex normals from the underlying geometry

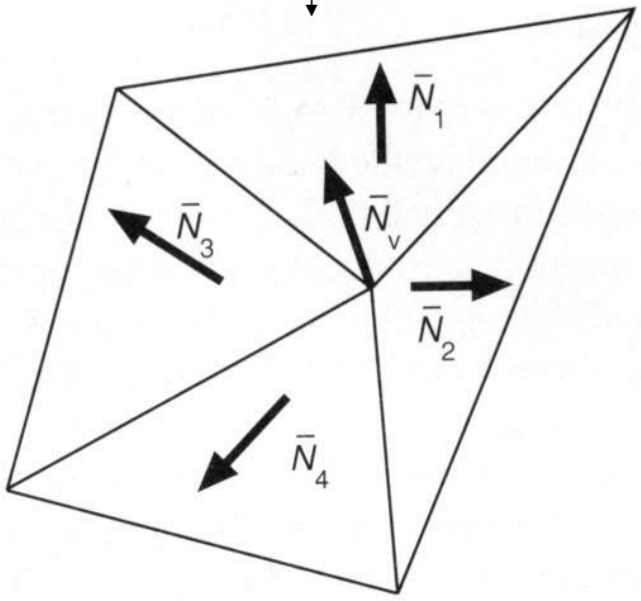
- e.g. consider a sphere



Otherwise have to infer vertex normals from triangle faces

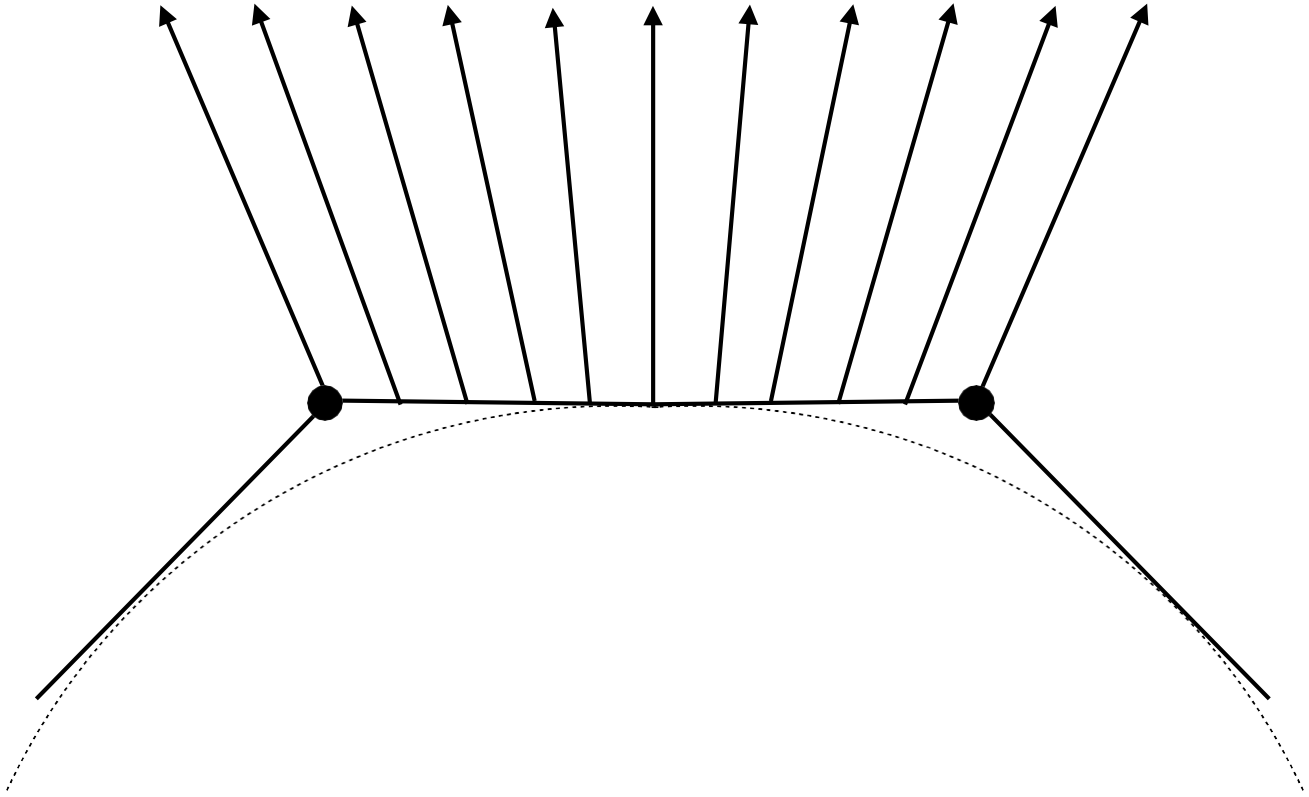
- Simple scheme: average surrounding face normals

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



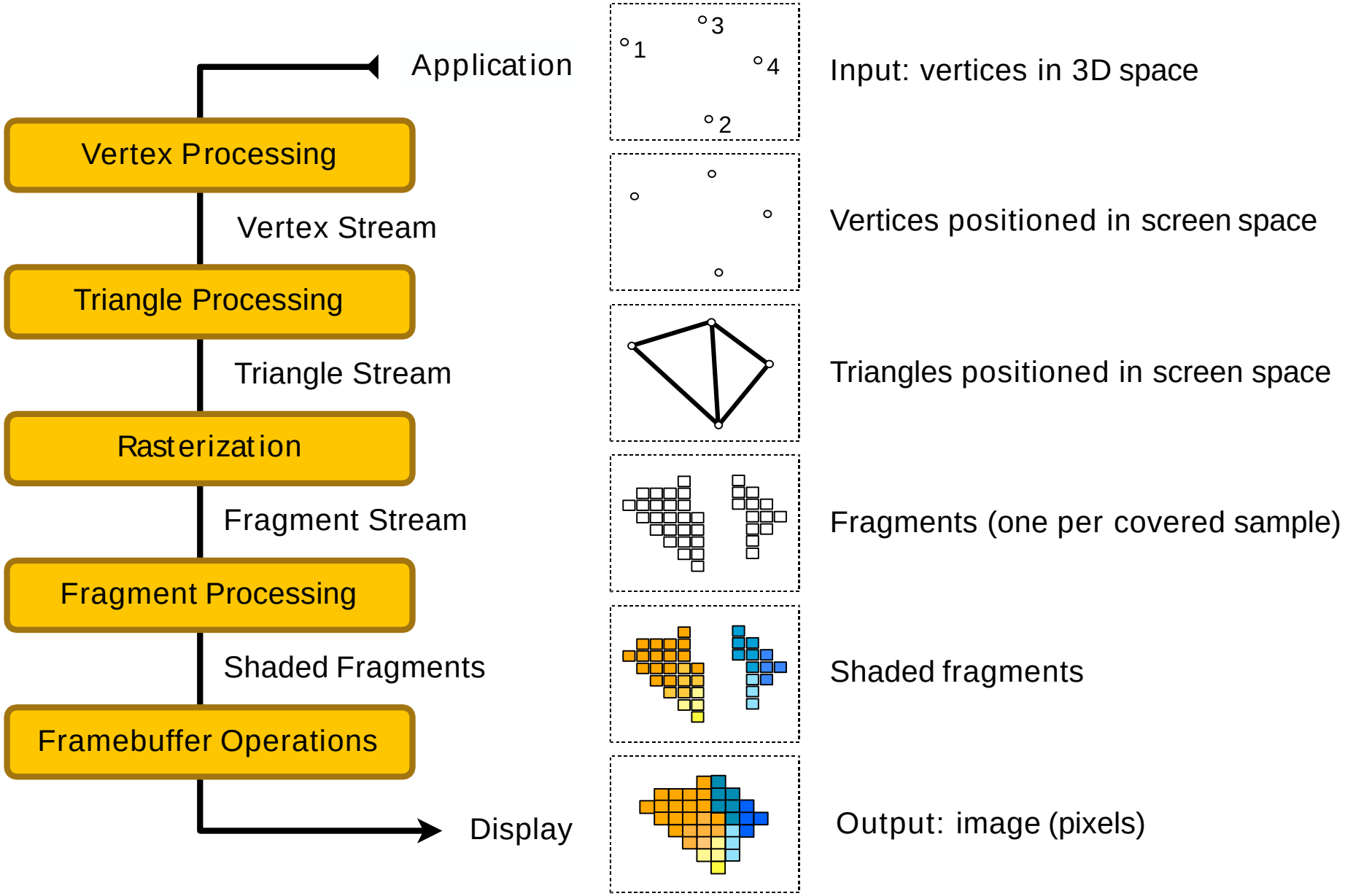
Defining Per-Pixel Normal Vectors

Barycentric interpolation (introducing soon)
of vertex normals

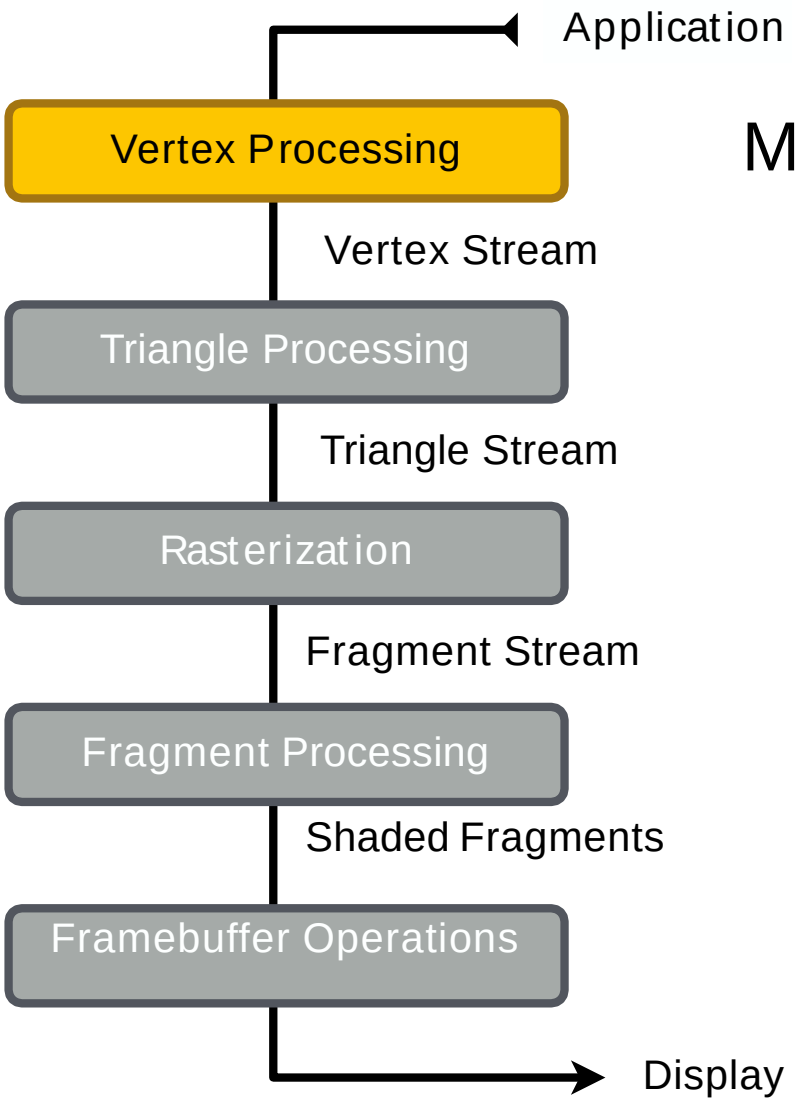


Don’t forget to normalize the interpolated directions

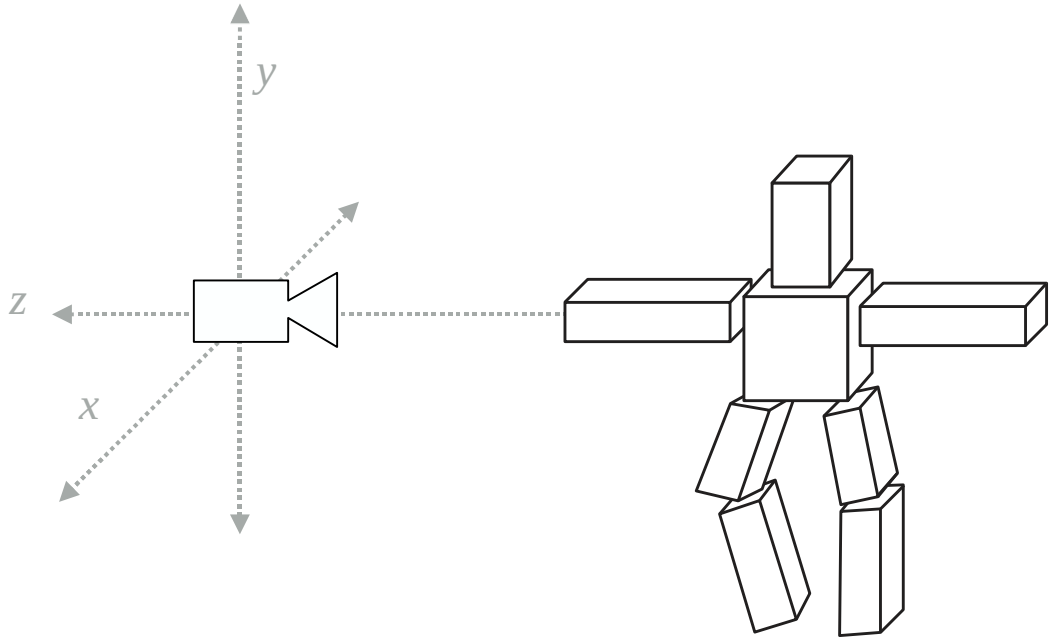
Graphics Pipeline



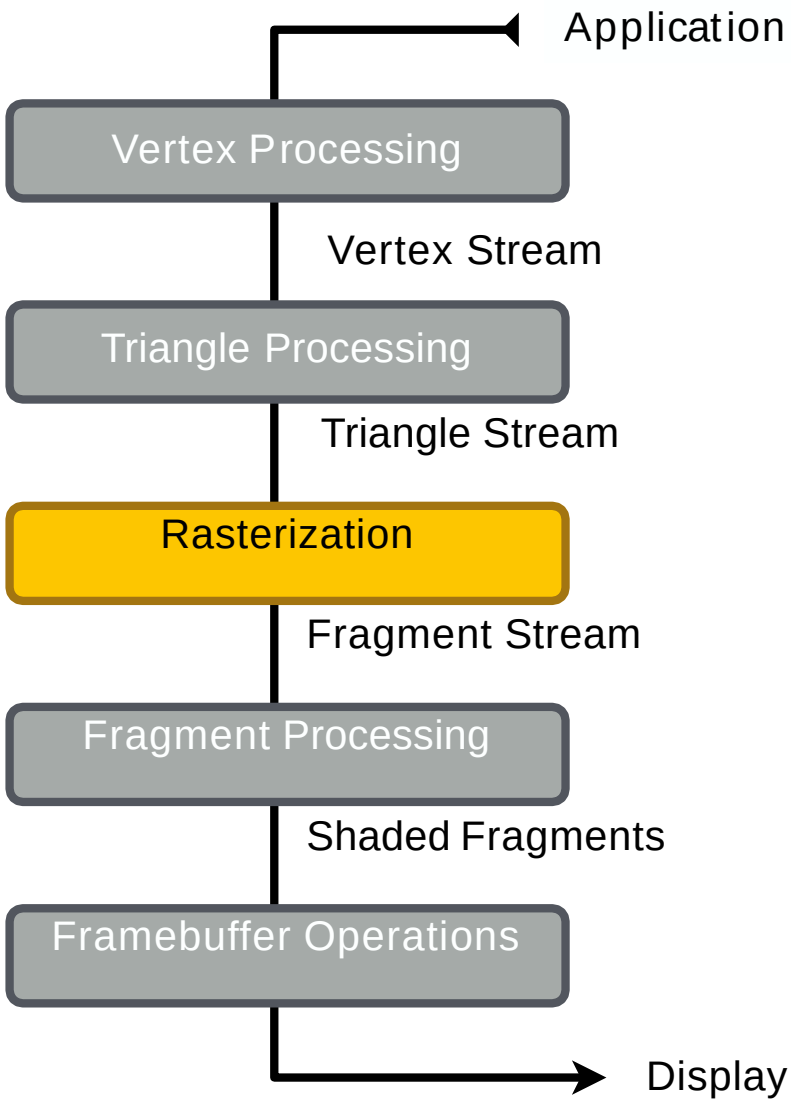
Graphics Pipeline



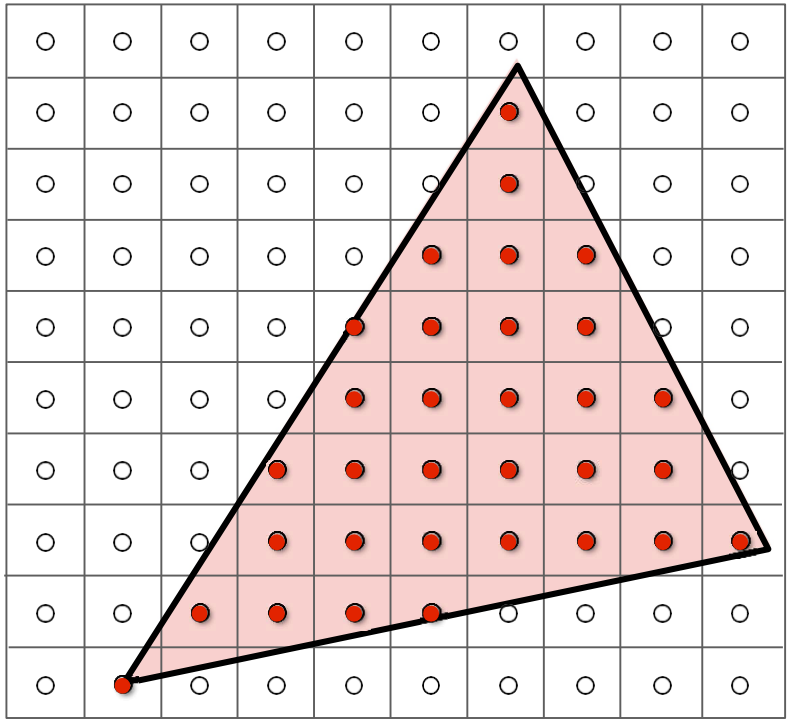
Model, View, Projection transforms



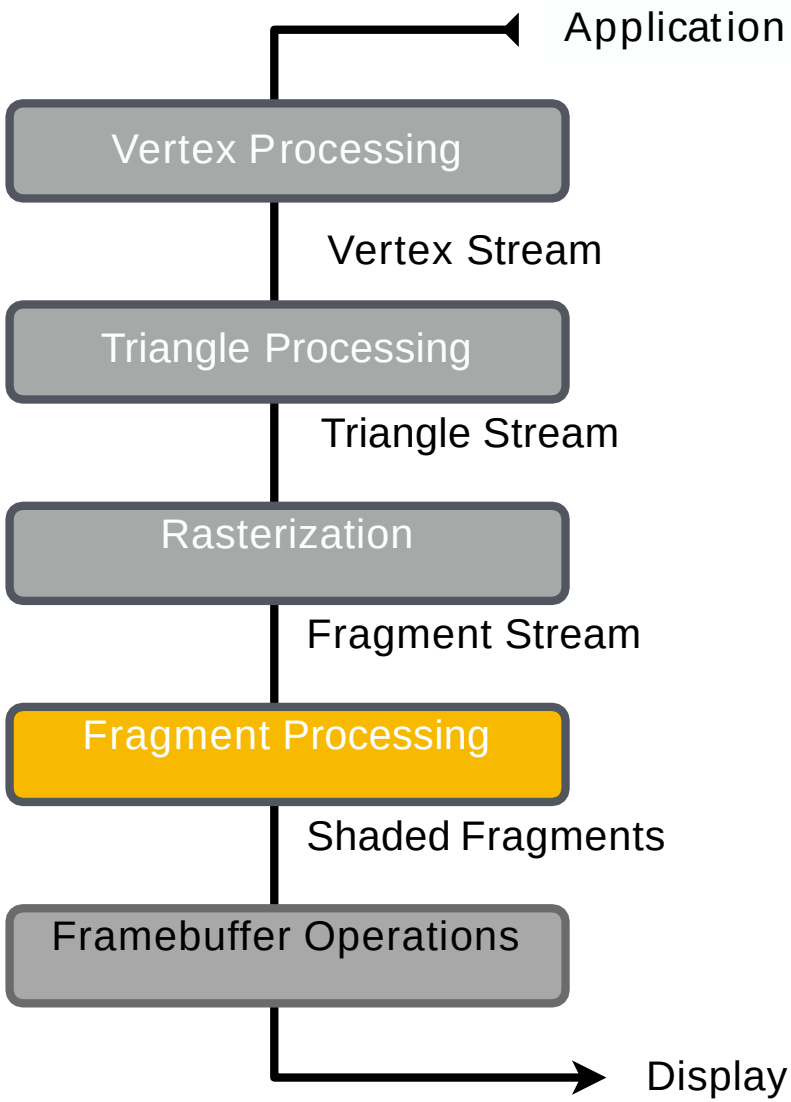
Graphics Pipeline



Sampling triangle coverage

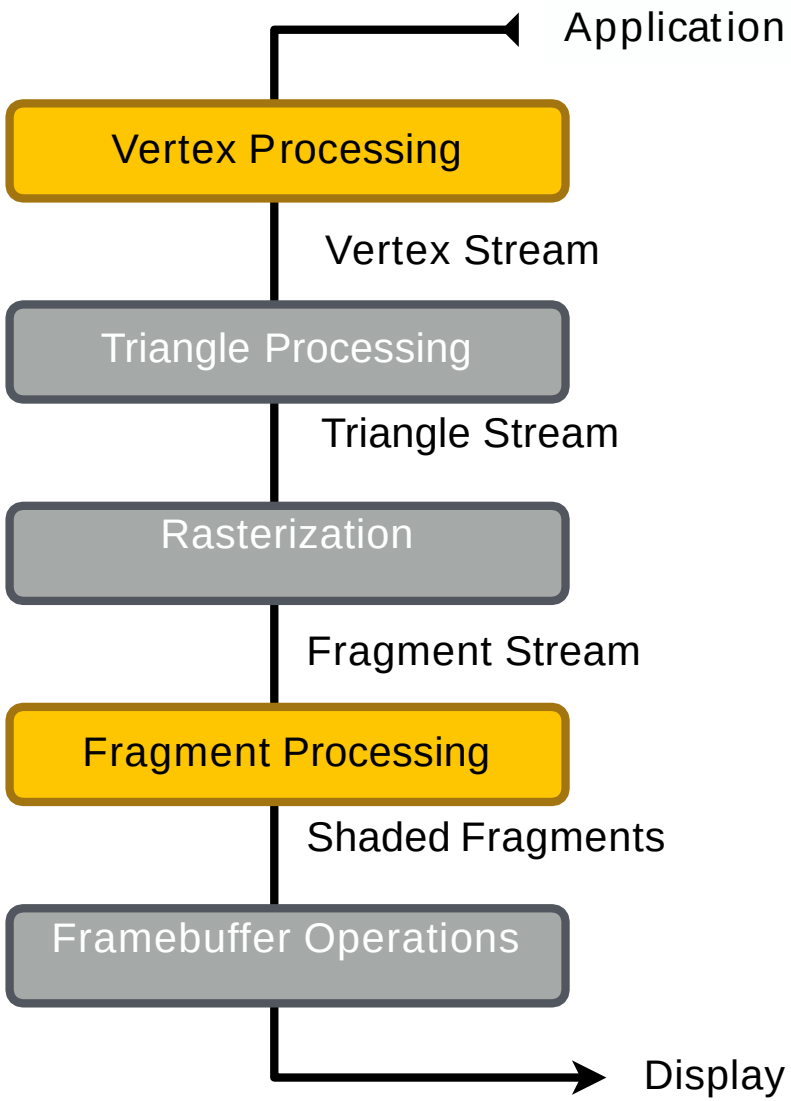


Rasterization Pipeline

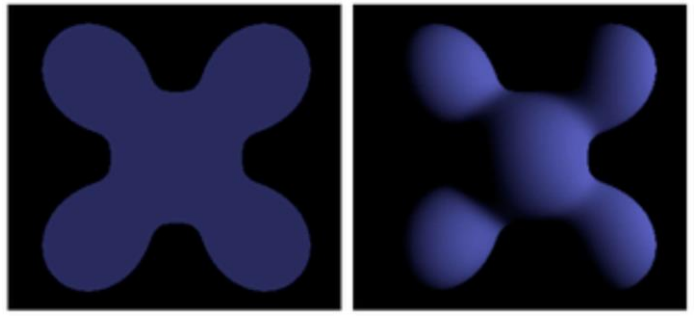


Z-Buffer Visibility Tests

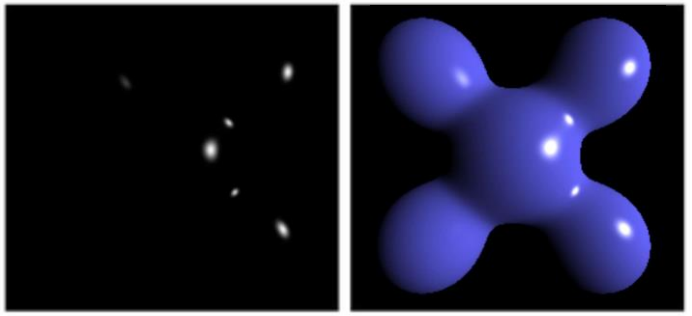
Graphics Pipeline



Shading

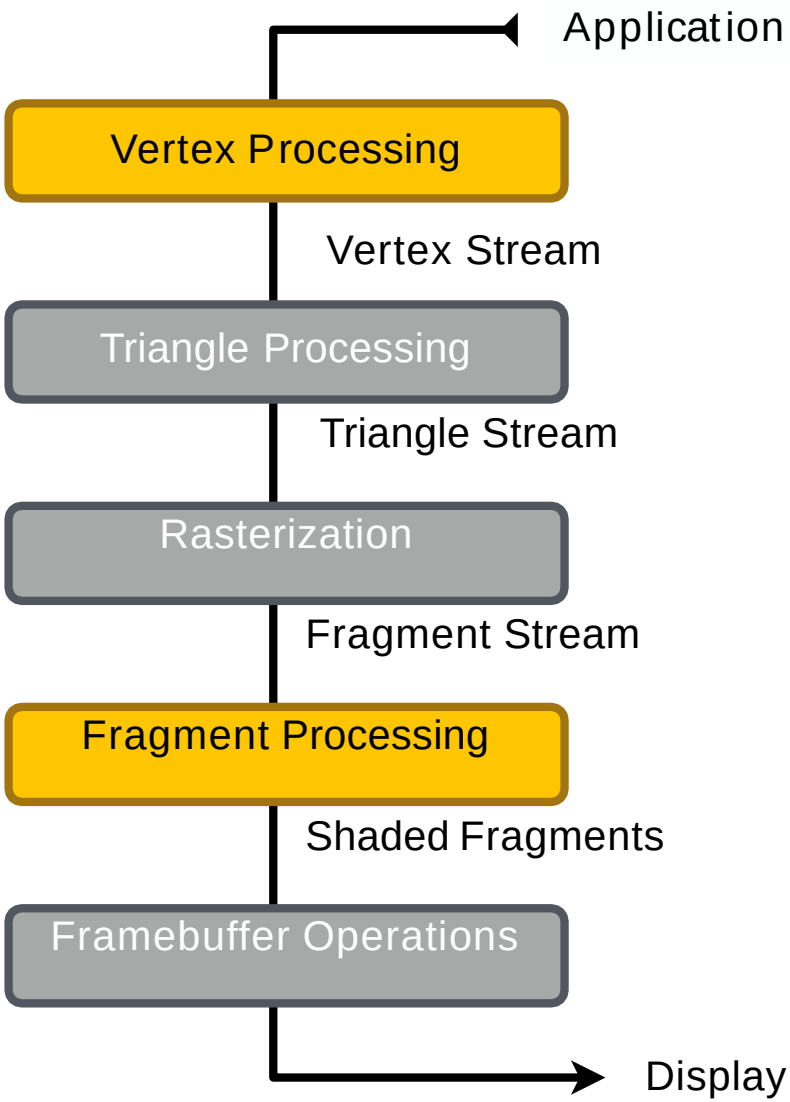


Ambient + Diffuse



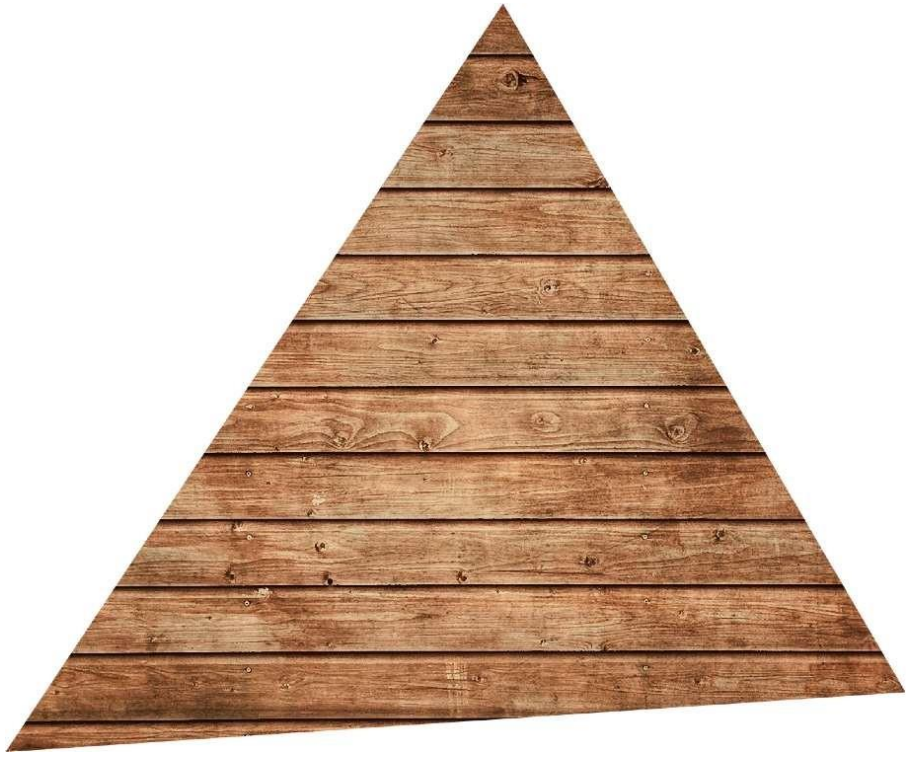
+ Specular = Blinn-Phong Reflectance Model

Graphics Pipeline



Texture mapping

(introducing soon)



Shader Programs

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

Example GLSL fragment shader program

```
uniform sampler2D myTexture;
uniform vec3 lightDir;
varying vec2 uv;
varying vec3 norm;

void diffuseShader()
{
  vec3 kd;
  kd = texture2d(myTexture, uv);
  kd *= clamp(dot(–lightDir, norm), 0.0, 1.0);
  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)

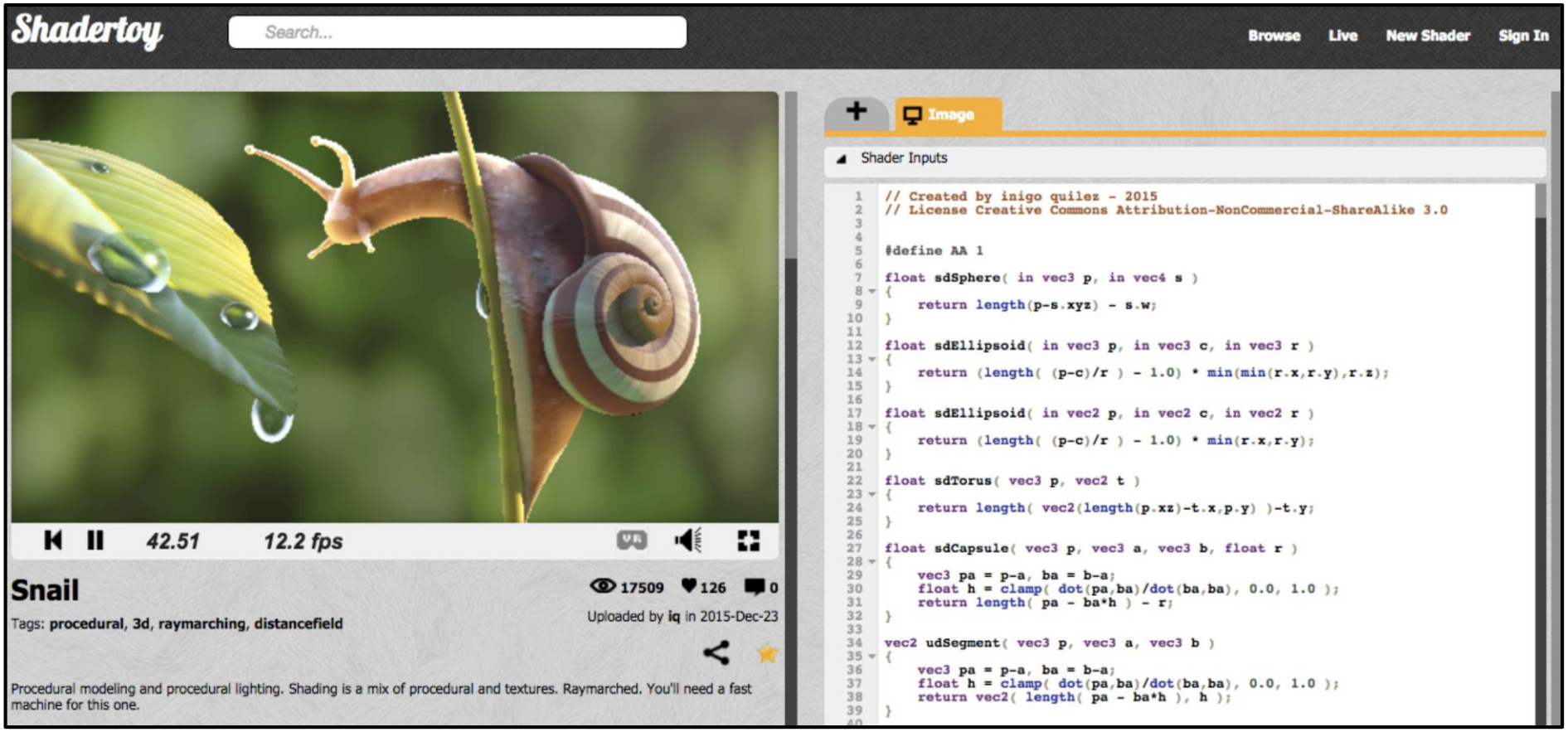
Example GLSL fragment shader program

```
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

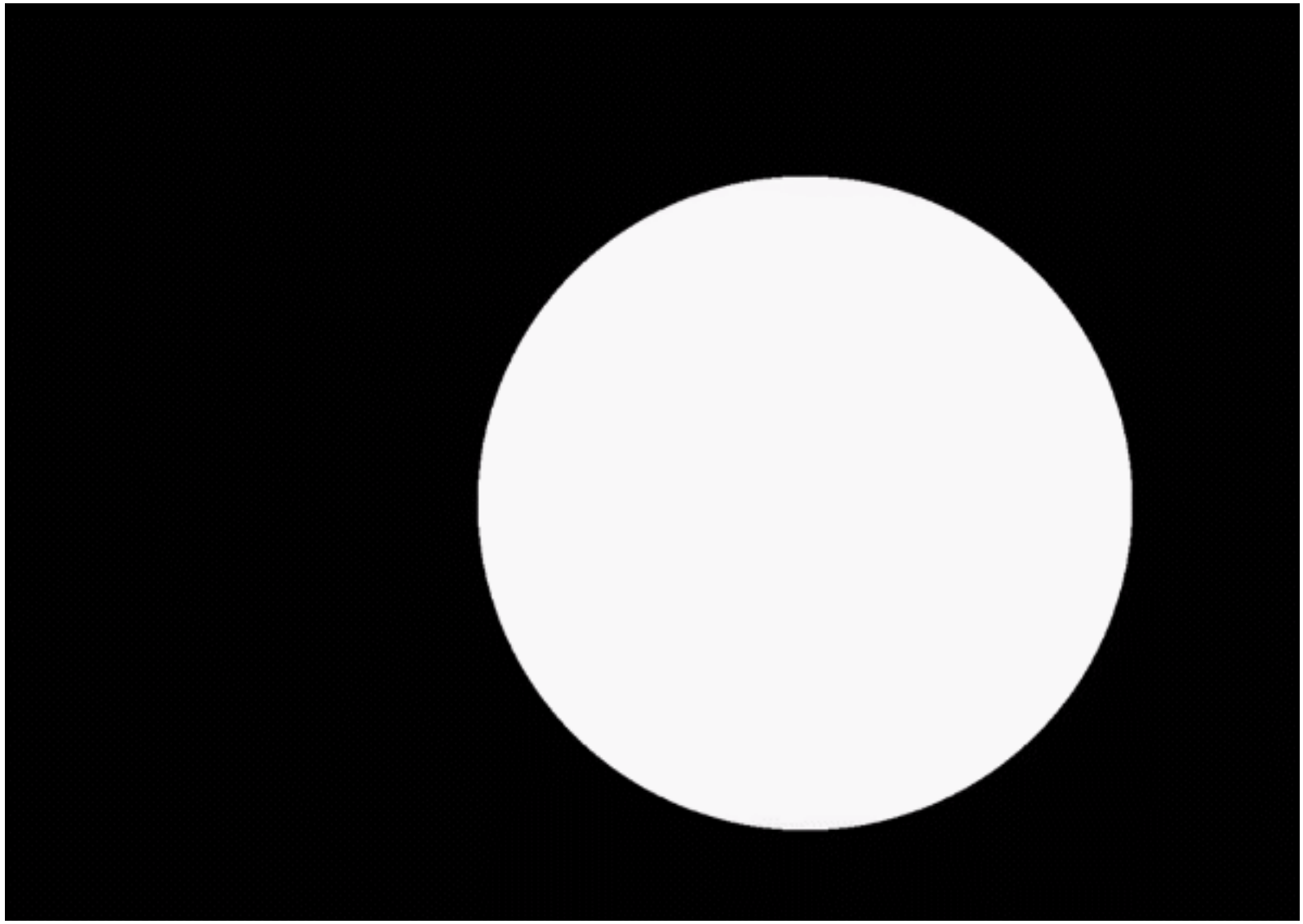
演示



Inigo Quilez

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

Snail Shader Program



https://www.alanzucconi.com/2016/07/01/signed-distance-functions/#google_vignette

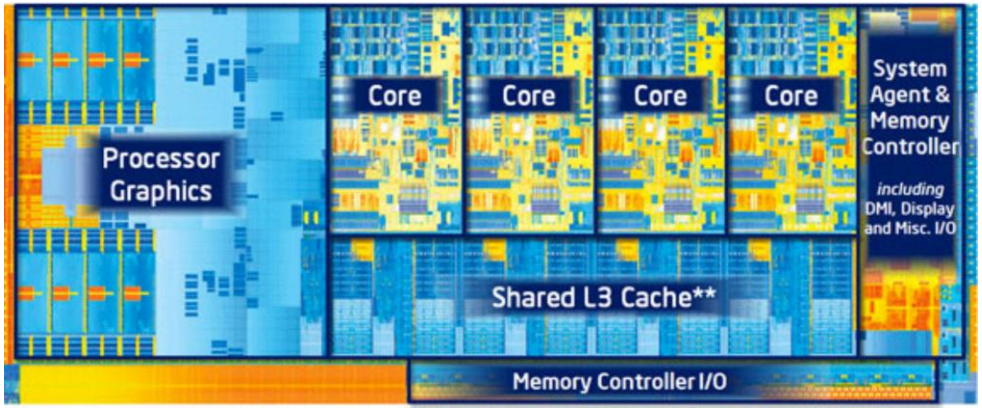
Goal: Highly Complex 3D Scenes in Realtime

Unreal Engine Kite Demo (Epic Games 2015)

Graphics Pipeline Implementation: GPUs

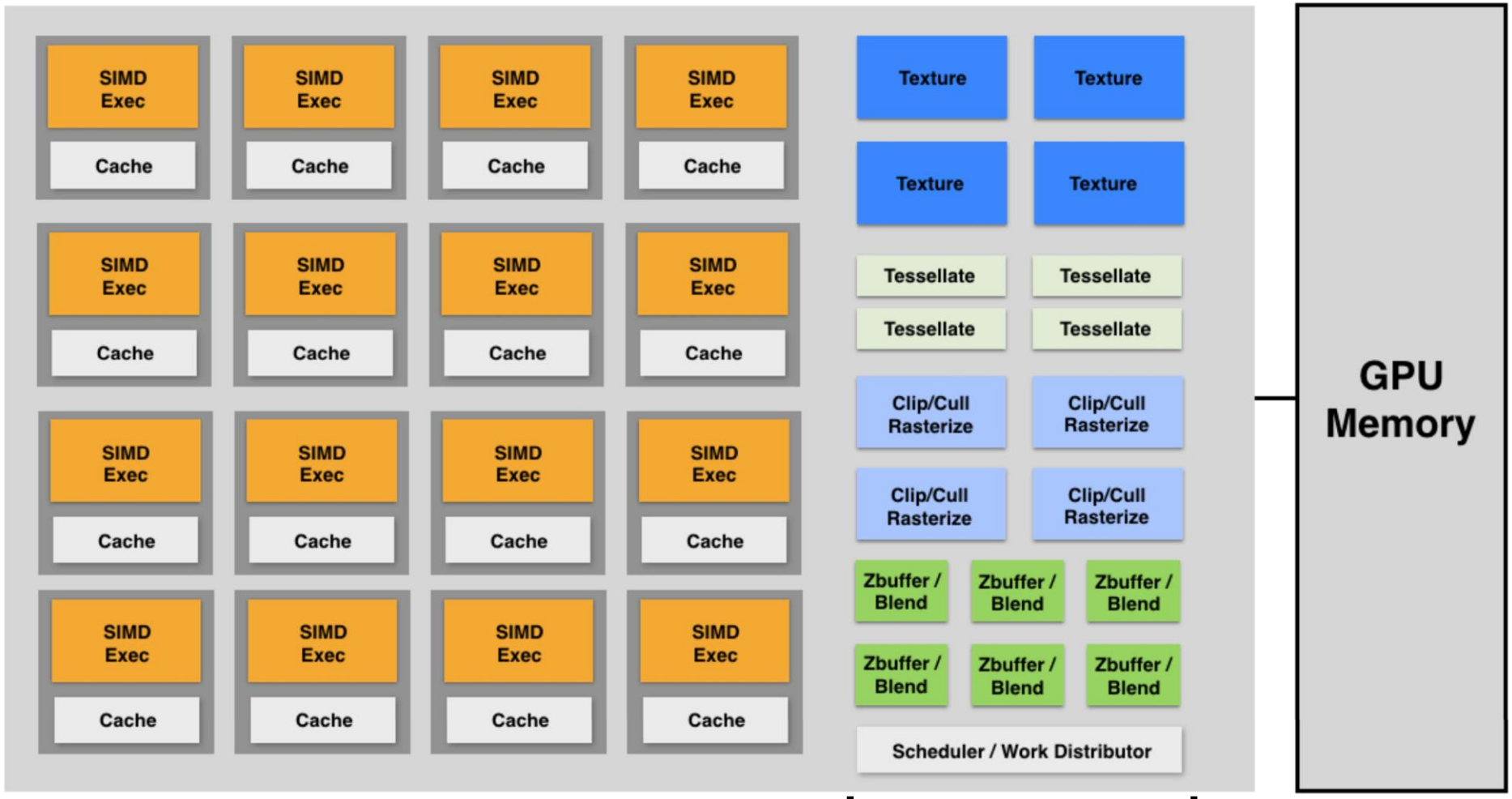
Specialized processors for executing graphics pipeline computations

Discrete GPU Card
(NVIDIA GeForce Titan X)



Integrated GPU:
(Part of Intel CPU die)

GPU: Heterogeneous, Multi-Core Procesor



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

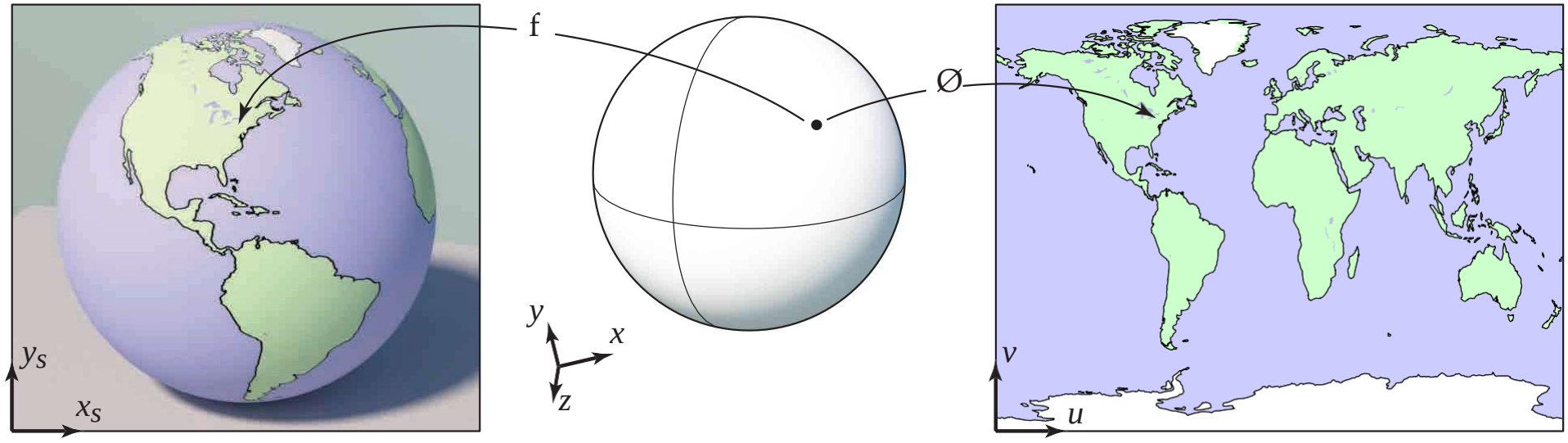
Texture Mapping

Different Colors at Different Places?

Surfaces are 2D

Surface lives in 3D world space

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



Screen space

World space

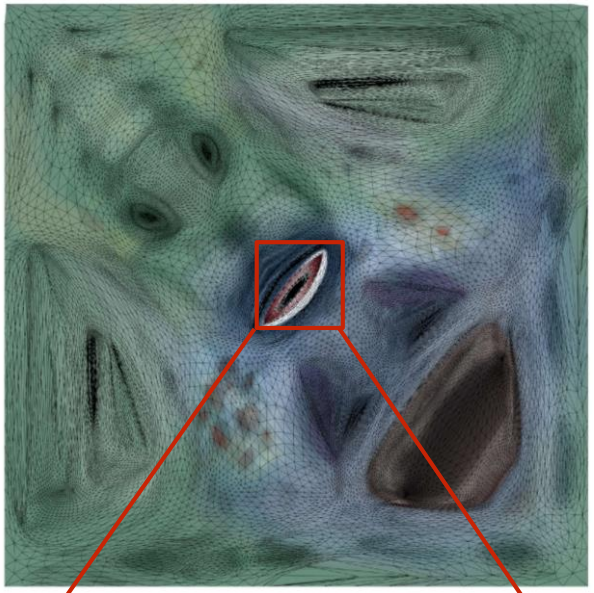
Texture space

Texture Applied to Surface

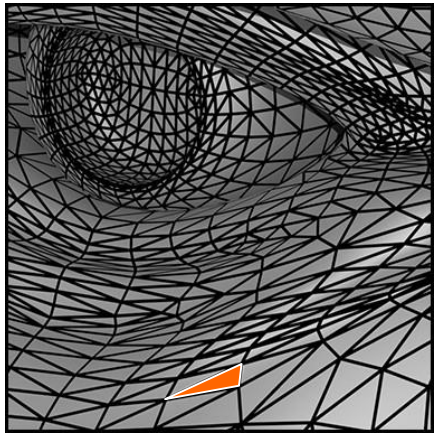
Rendering without texture

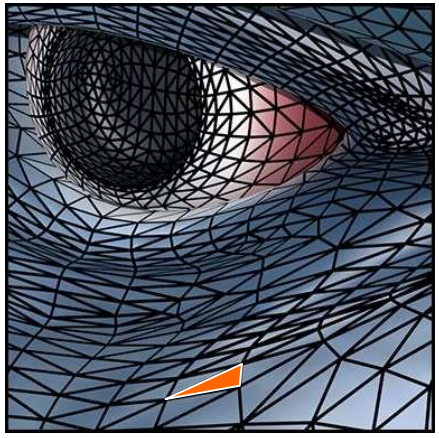
Rendering with texture

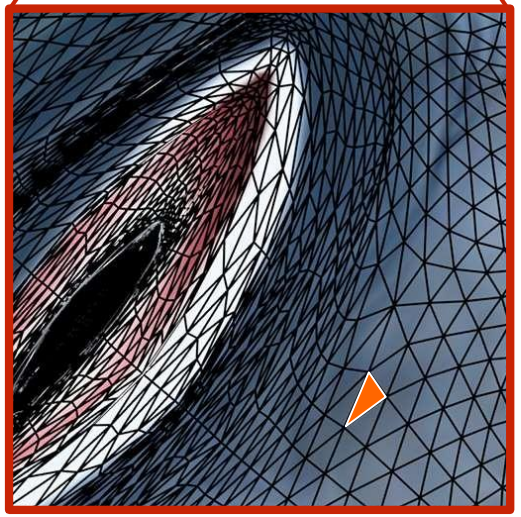
Texture



Zoom





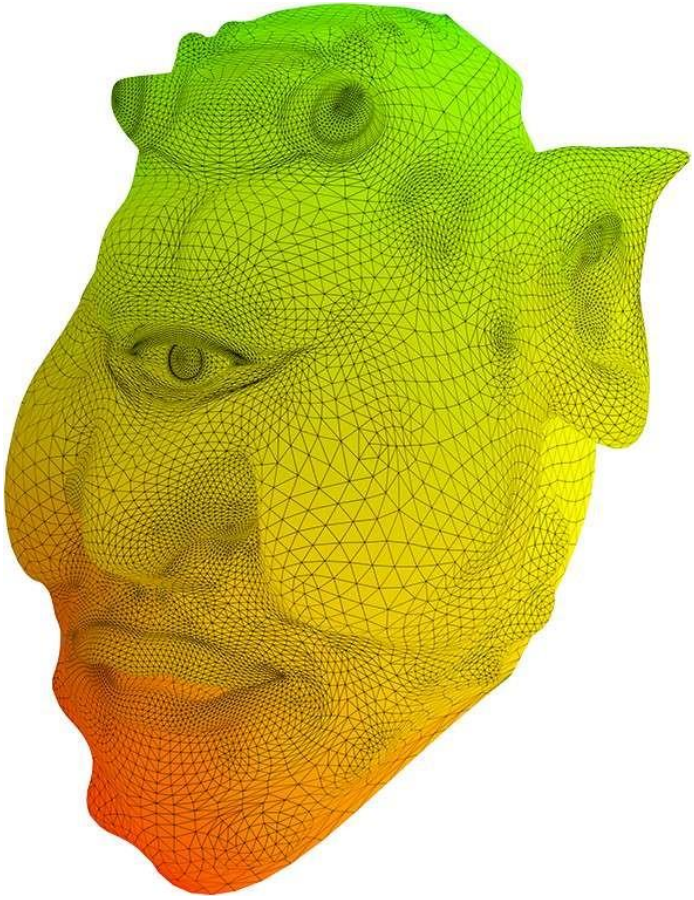


Each triangle “copies” a piece of the texture image to the surface.

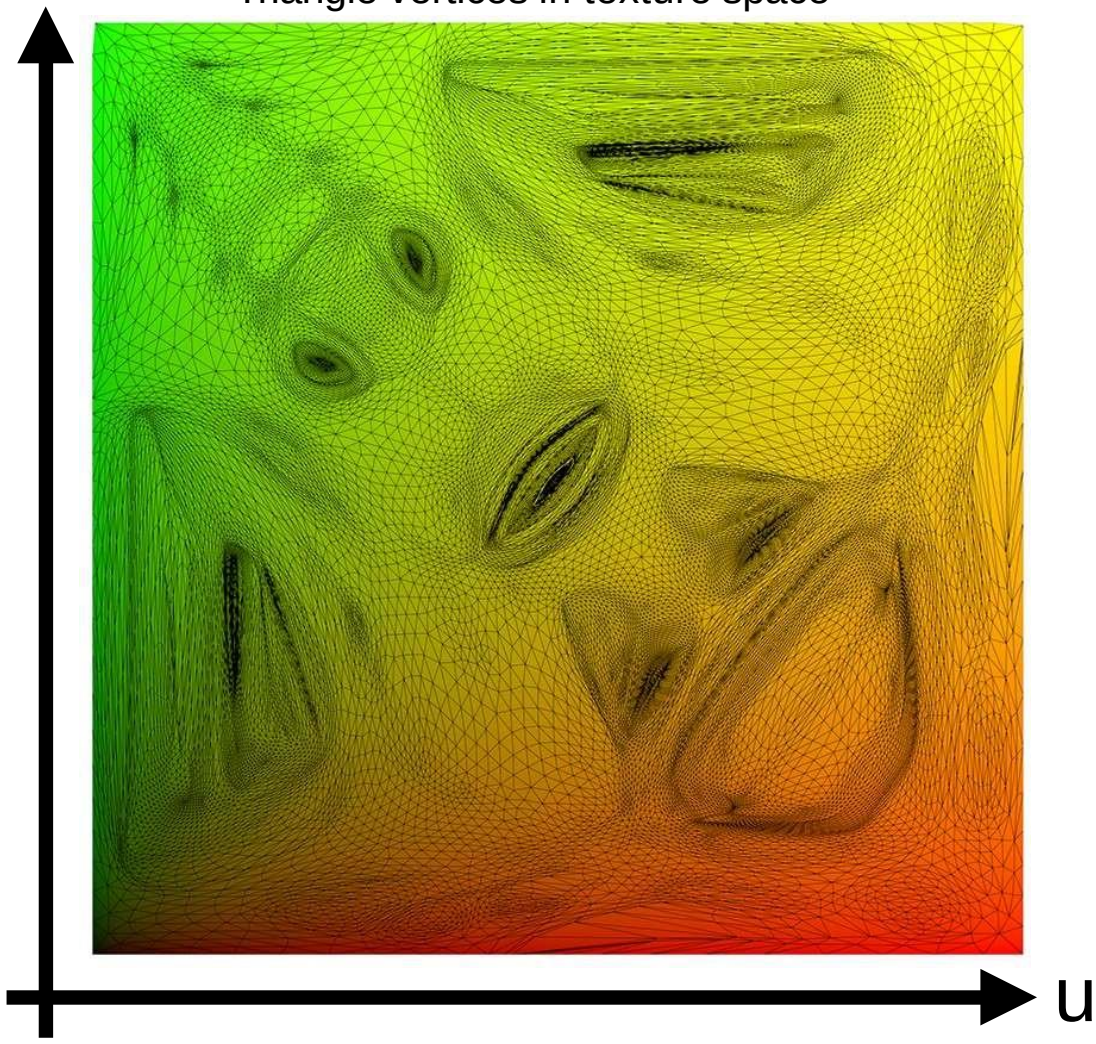
Visualization of Texture Coordinates

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

Visualization of texture coordinates

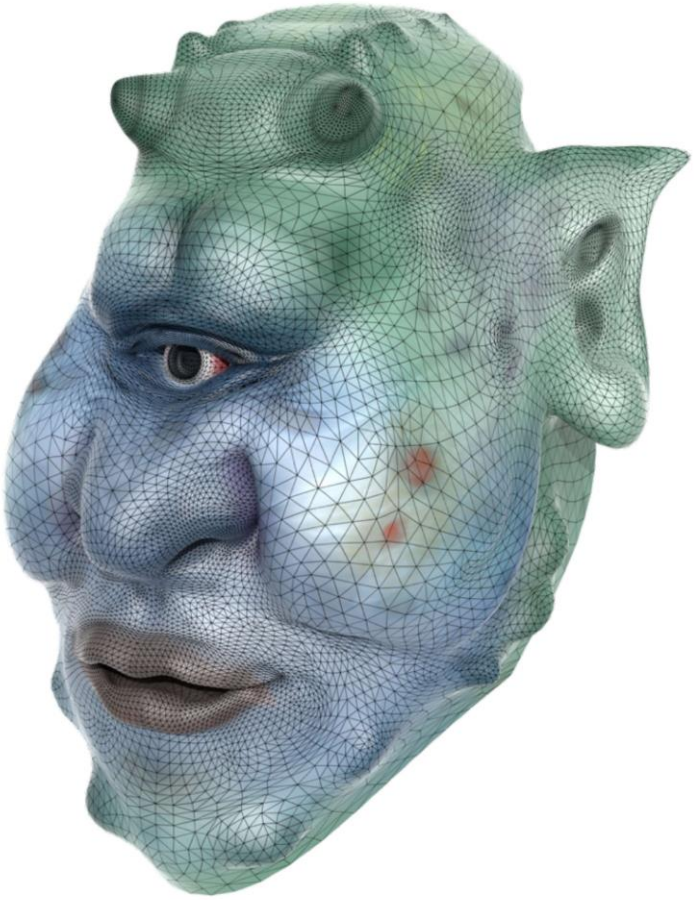


Triangle vertices in texture space

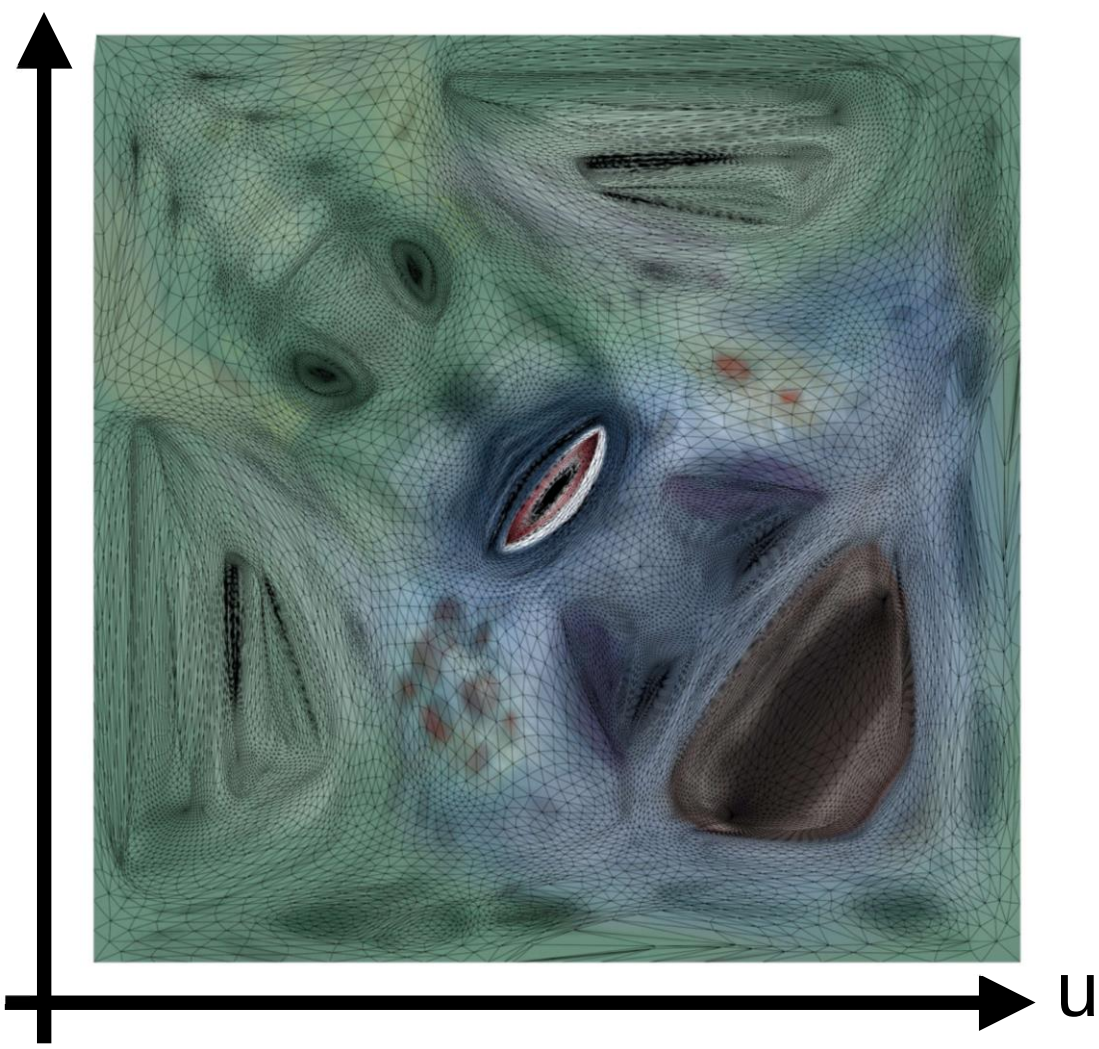


Texture Applied to Surface

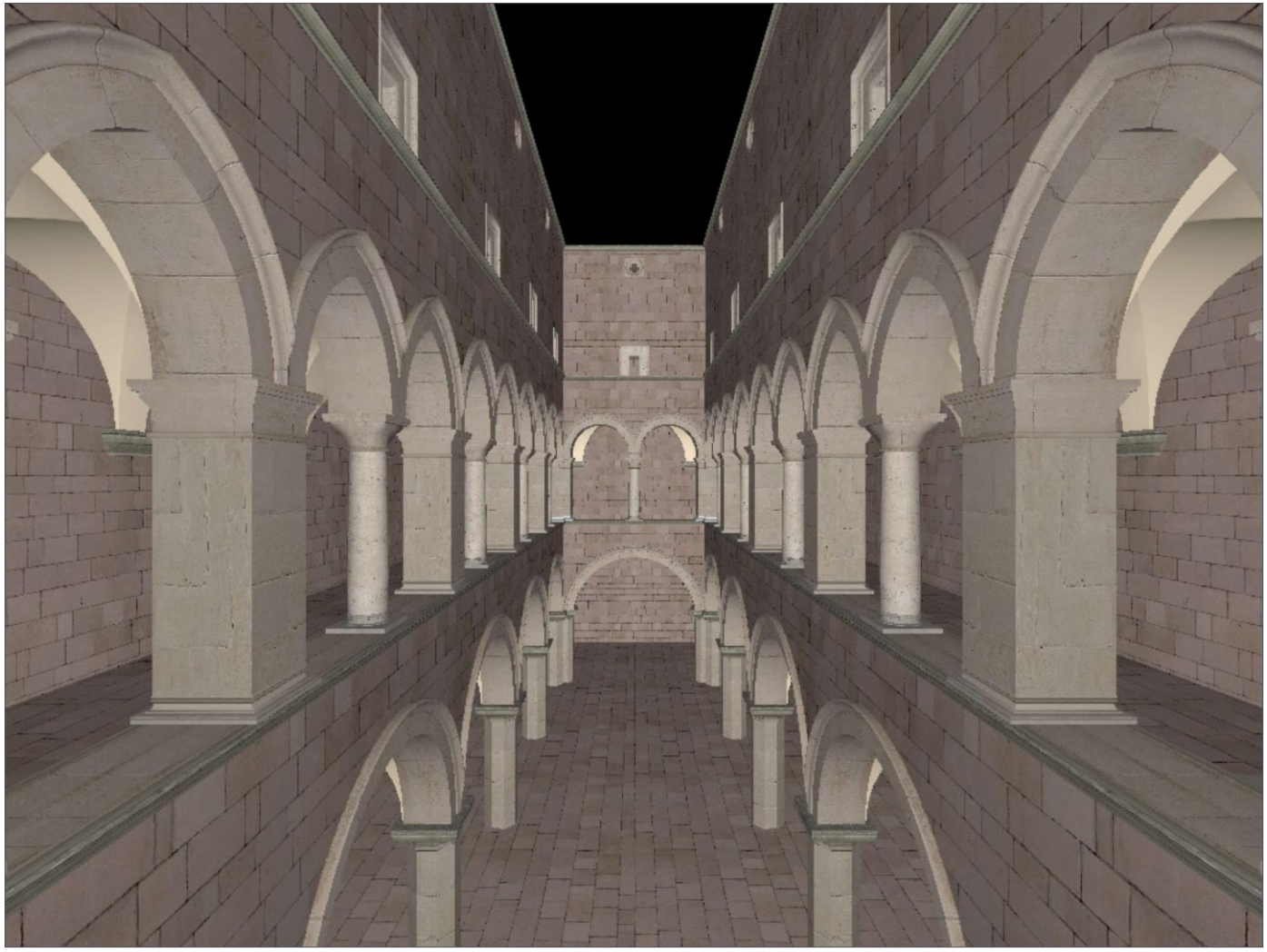
Rendered result



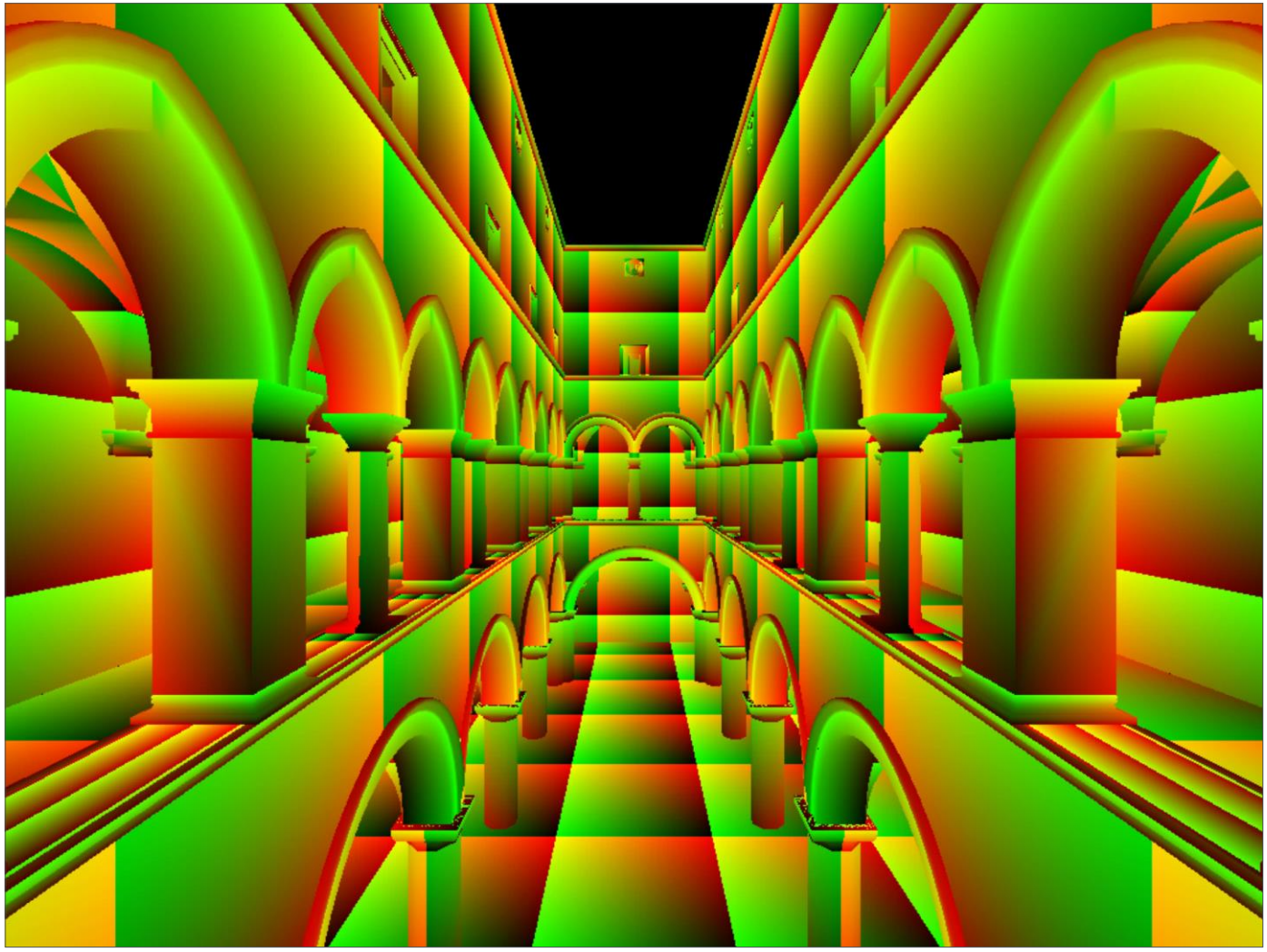
Triangle vertices in texture space



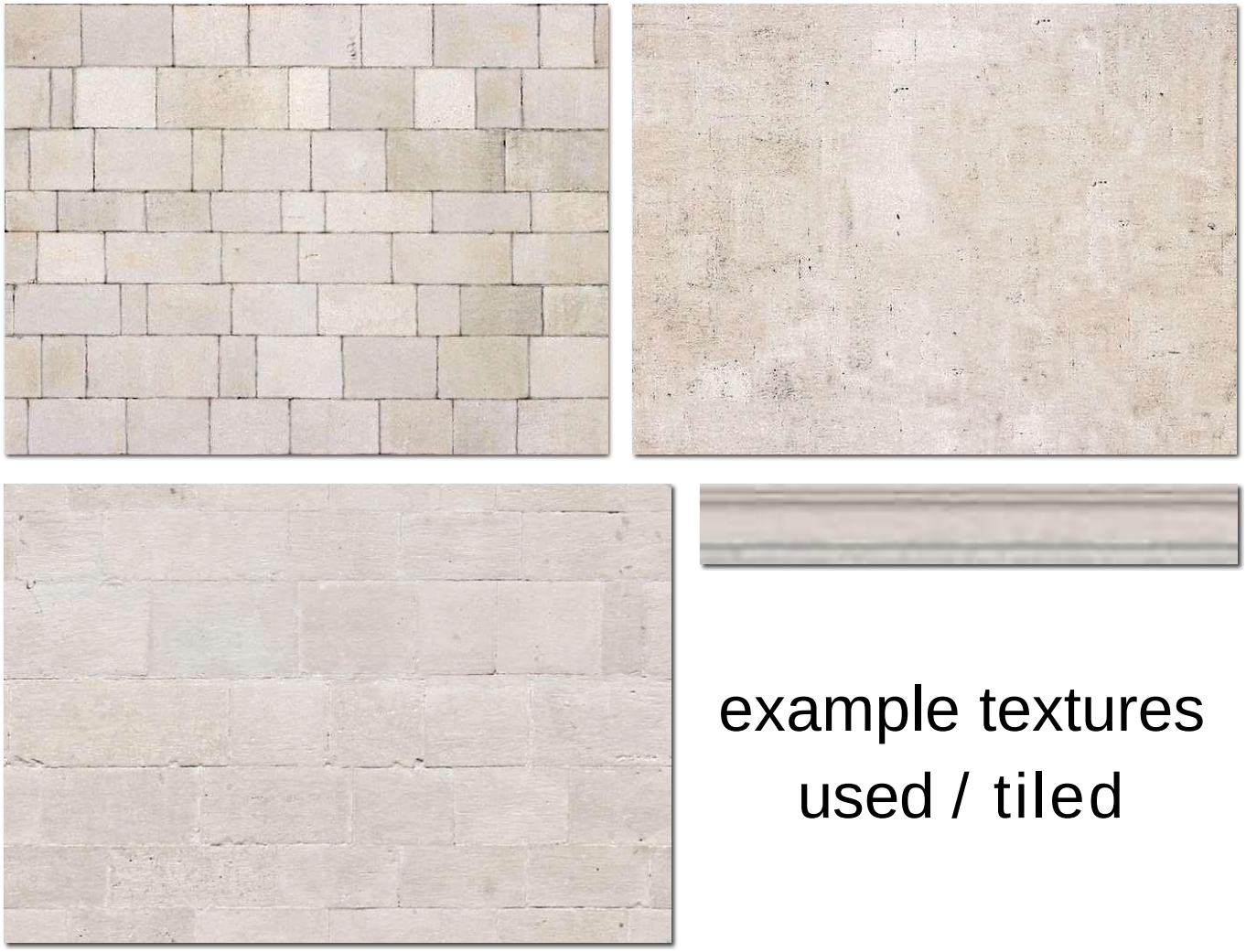
Textures applied to surfaces



Visualization of texture coordinates



Textures can be used multiple times!



example textures
used / tiled

Interpolation Across Triangles: Barycentric Coordinates

(重心坐标)

Interpolation Across Triangles

Why do we want to interpolate?

- Specify values at vertices
- Obtain smoothly varying values across triangles

What do we want to interpolate?

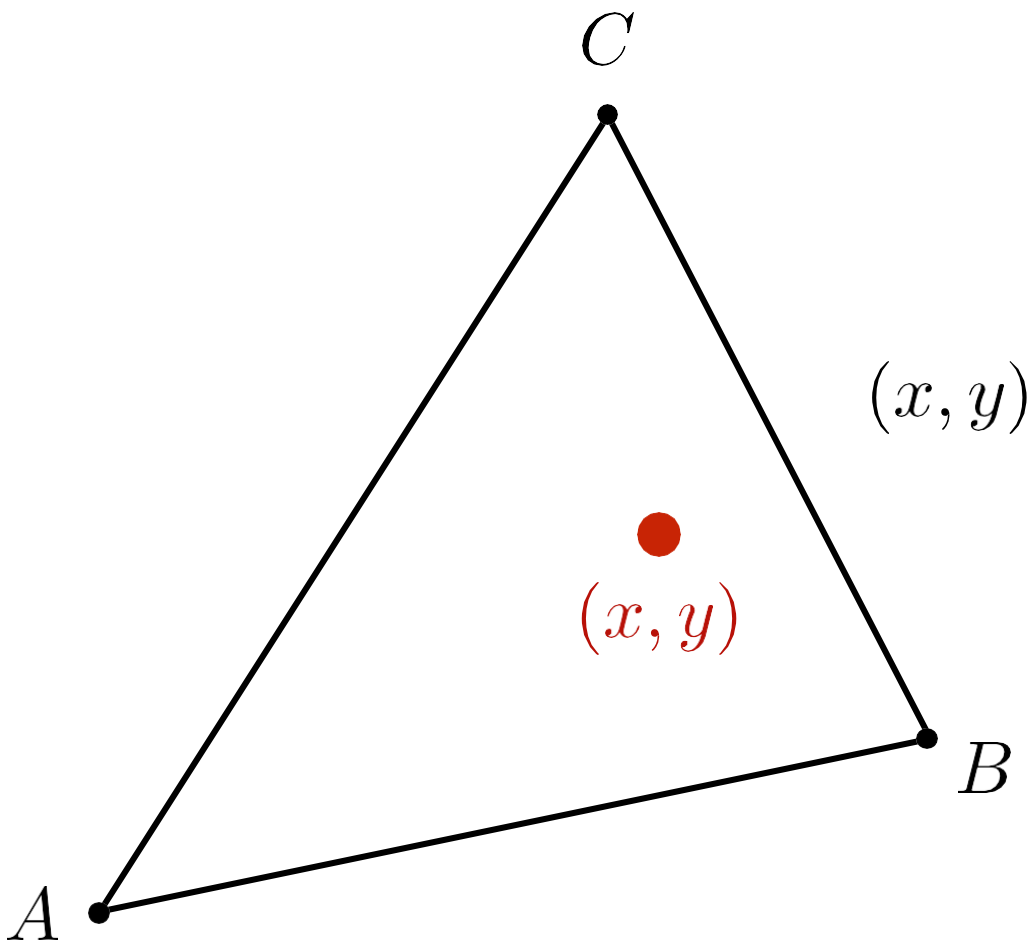
- Texture coordinates, colors, normal vectors, …

How do we interpolate?

- Barycentric coordinates

Barycentric Coordinates

A coordinate system for triangles (α, β, γ)



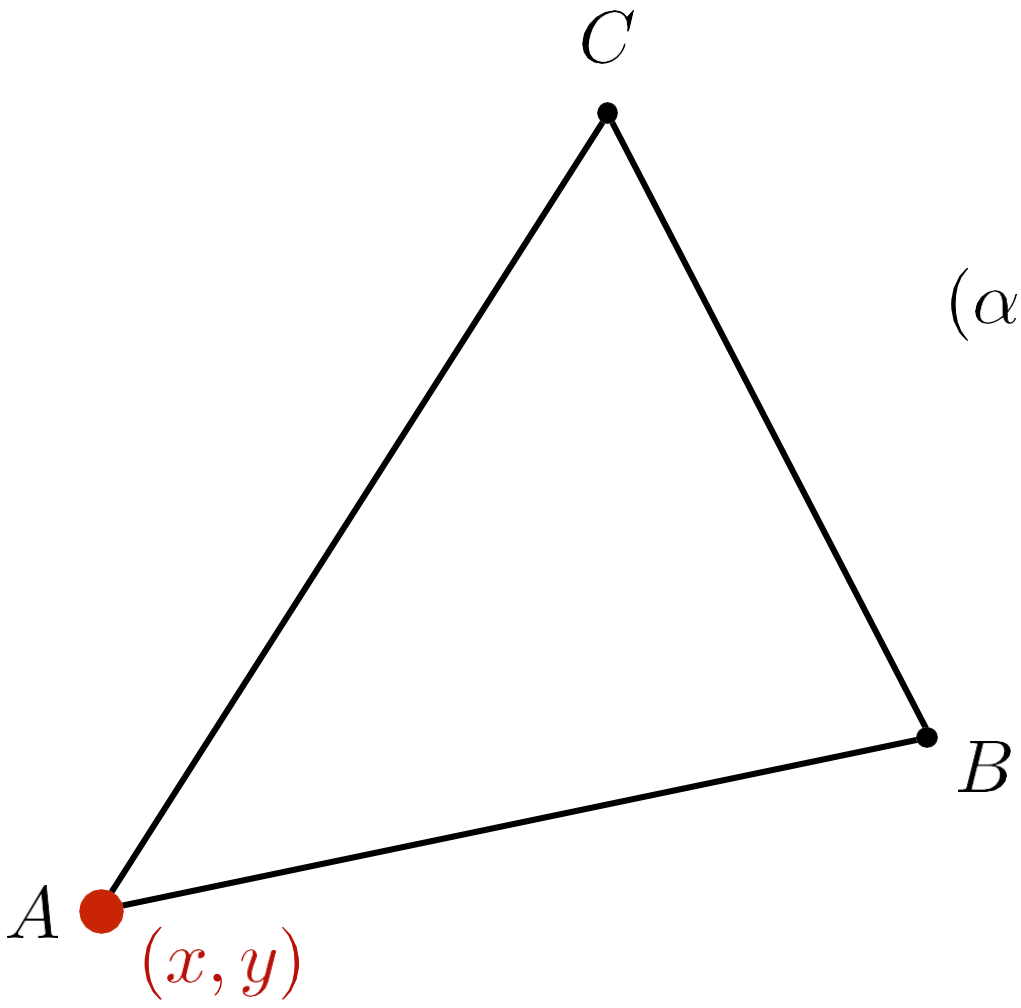
$$(x, y) = \alpha\, A + \beta\, B + \gamma\, C$$

$$\alpha + \beta + \gamma = 1$$

Inside the triangle if all three coordinates are non-negative

Barycentric Coordinates

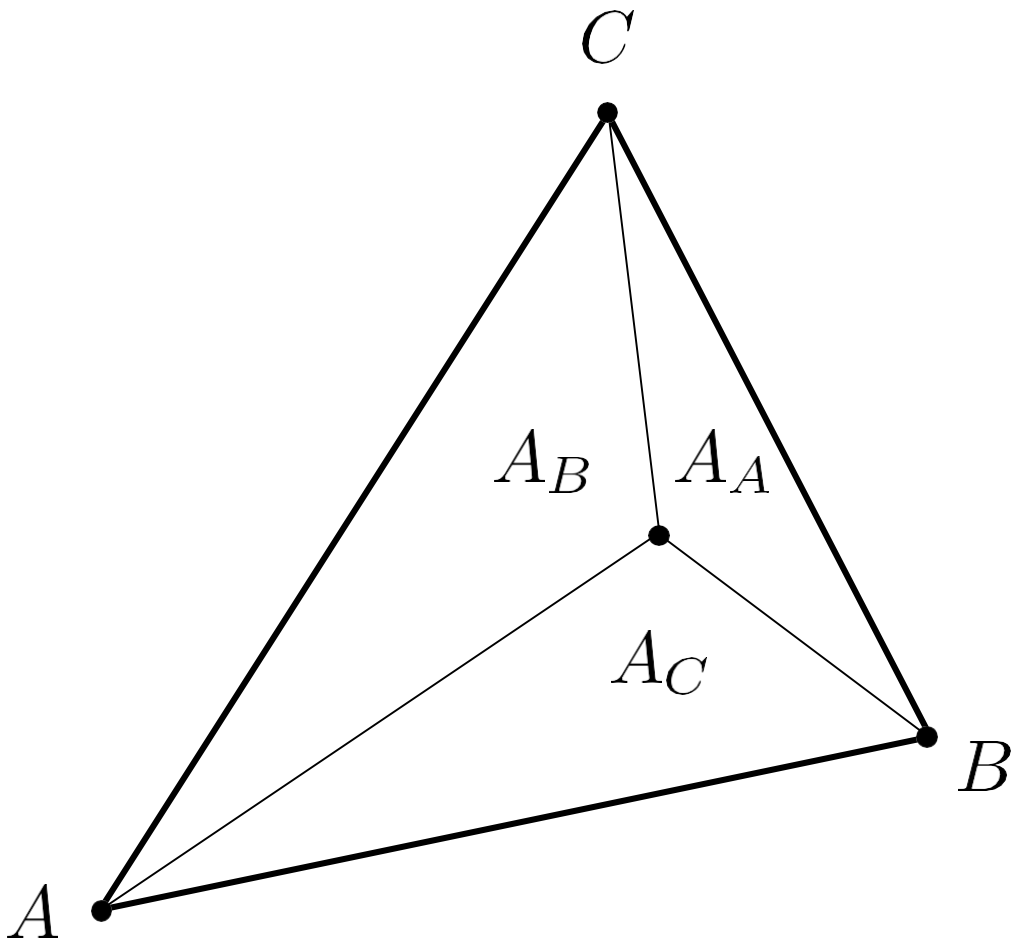
What's the barycentric coordinate of A?



$$(\alpha, \beta, \gamma) = (1, 0, 0)$$
$$(x, y) = \alpha A + \beta B + \gamma C$$
$$= A$$

Barycentric Coordinates

Geometric viewpoint — proportional areas



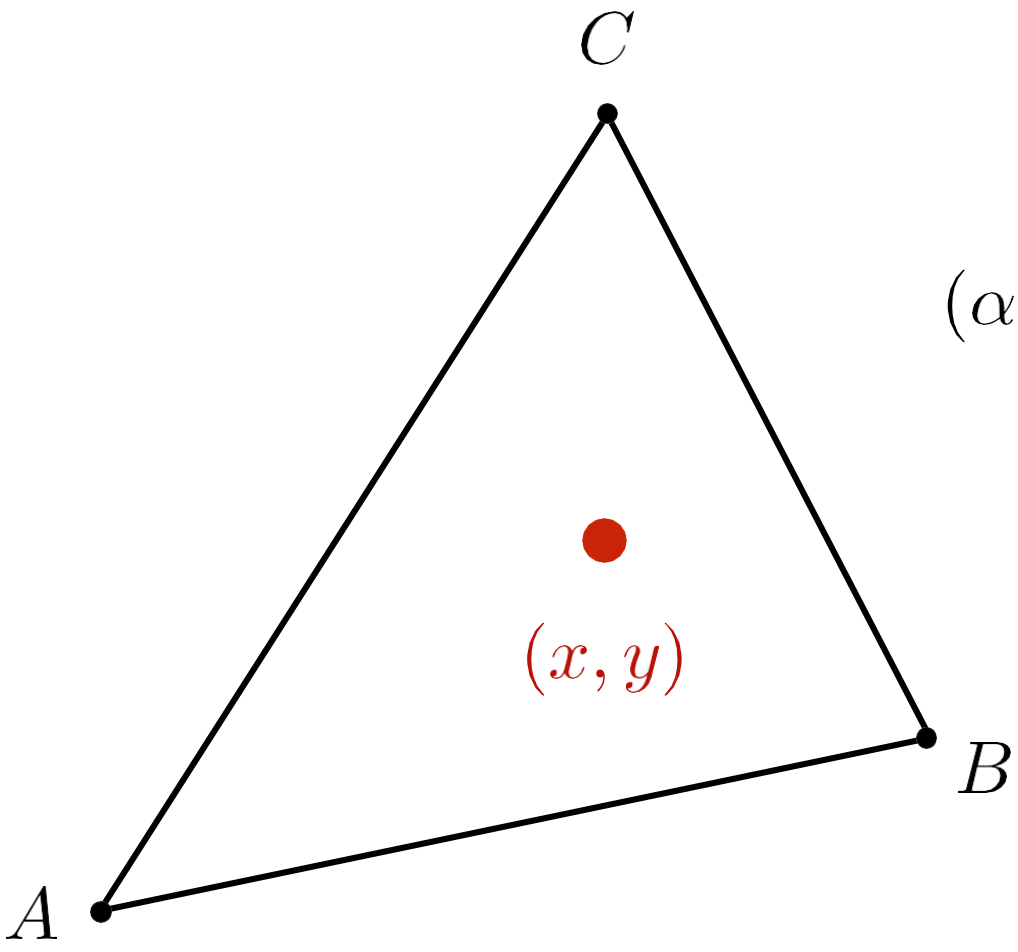
$$\alpha = \frac{A_A}{A_A + A_B + A_C}$$

$$\beta = \frac{A_B}{A_A + A_B + A_C}$$

$$\gamma = \frac{A_C}{A_A + A_B + A_C}$$

Barycentric Coordinates

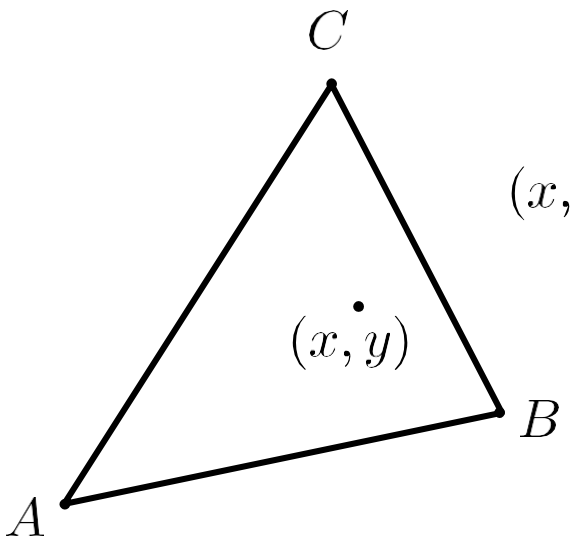
What's the barycentric coordinate of the centroid?



$$(\alpha, \beta, \gamma) = \left(\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}\right)$$

$$(x, y) = \tfrac{1}{3}\,A + \tfrac{1}{3}\,B + \tfrac{1}{3}\,C$$

Barycentric Coordinates: Formulas



$$(x, y) = \alpha\, A + \beta\, B + \gamma\, C$$
$$\alpha + \beta + \gamma = 1$$

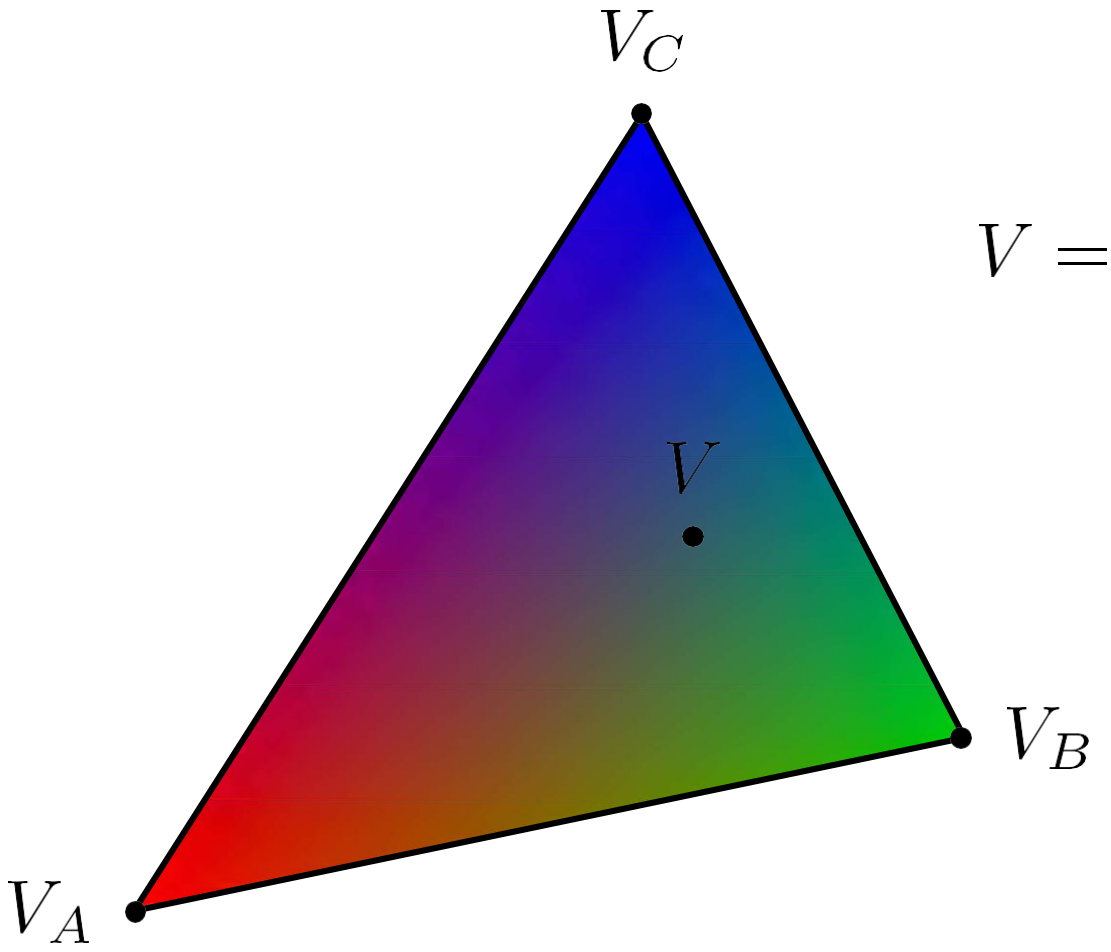
$$\alpha = \frac{-(x - x_B)(y_C - y_B) + (y - y_B)(x_C - x_B)}{-(x_A - x_B)(y_C - y_B) + (y_A - y_B)(x_C - x_B)}$$

$$\beta = \frac{-(x - x_C)(y_A - y_C) + (y - y_C)(x_A - x_C)}{-(x_B - x_C)(y_A - y_C) + (y_B - y_C)(x_A - x_C)}$$

$$\gamma = 1 - \alpha - \beta$$

Using Barycentric Coordinates

Linearly interpolate values at vertices



$$V = \alpha V_A + \beta V_B + \gamma V_C$$

V_A, V_B, V_C can be positions, texture coordinates, color, normal, depth, material attributes…

However, barycentric coordinates are not invariant under projection!

Thank you!