Tutorial 2: Render 3D objects with OpenGL

Zhanrui Zhang

9/28/2022

Agenda

- Transformation in OpenGL
  - Homogenous coordinates
  - View transformation
  - Projection transformation
  - Solution of quiz1
- How to finish homework1: draw 3D object with OpenGL
  - Vertex shader
  - Fragment shader

Homogenous Coordinates

- 3D point = $(x, y, z, 1)^T$
- 3D vector = $(x, y, z, 0)^T$
- In general, a 3D point can be expressed by

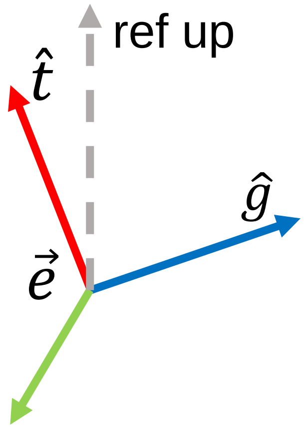
$$(kx, ky, kz, k)^T, k \neq 0$$

- Then we can use a 4x4 matrix for affine transformations

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

View Transformation

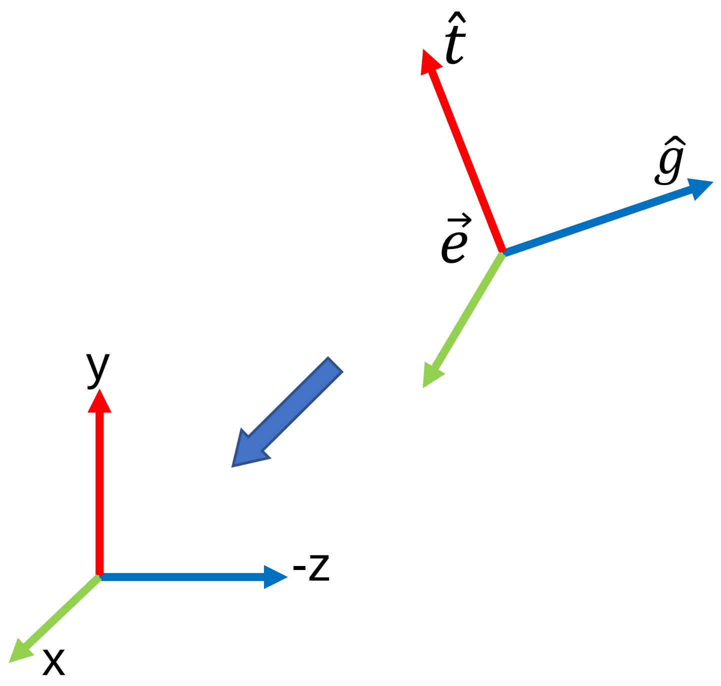
- How to define a camera in 3D space? (right handed)
- Position
- Look at / gaze direction
- Ref up direction
- Compute a orthonormal basis with cross product



e:相机所在位置
g:相机指向方向
ref up:不随相机转动改变

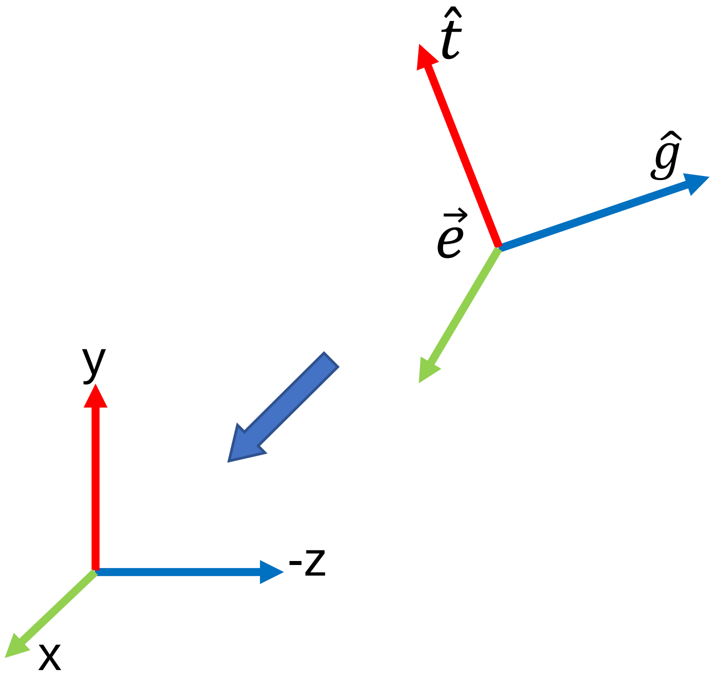
View Transformation

- Transform camera to:
  - position: origin point
  - up at y axis
  - look along –z axis
- And transform all objects along with the camera.
- Translate camera to origin point first, then rotate it such that $\hat{t}$ aligns to y and $\hat{g}$ aligns to -z.



View Transformation

- Translate camera to origin point
- $T_{view} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$



View Transformation

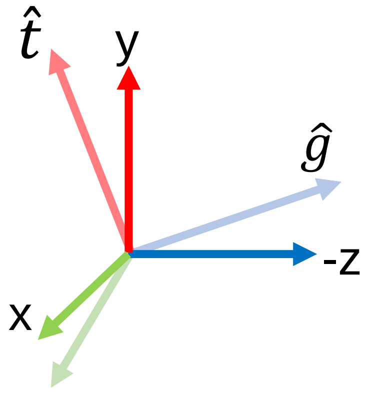
- Rotate $\hat{t}$ to y, rotate $\hat{g}$ to –z, rotate $\hat{g} \times \hat{t}$ to x
- This is difficult… So we consider inverse rotation.

深色旋转到浅色

- $$R_{view}^{-1} = \begin{array}{c} \begin{array}{cccc} \text{right} & \text{top} & \text{view/gaze} & \end{array} \\ \begin{bmatrix} x_{\hat{g}\times\hat{t}} & x_{\hat{t}} & x_{-\hat{g}} & 0 \\ y_{\hat{g}\times\hat{t}} & y_{\hat{t}} & y_{-\hat{g}} & 0 \\ z_{\hat{g}\times\hat{t}} & z_{\hat{t}} & z_{-\hat{g}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

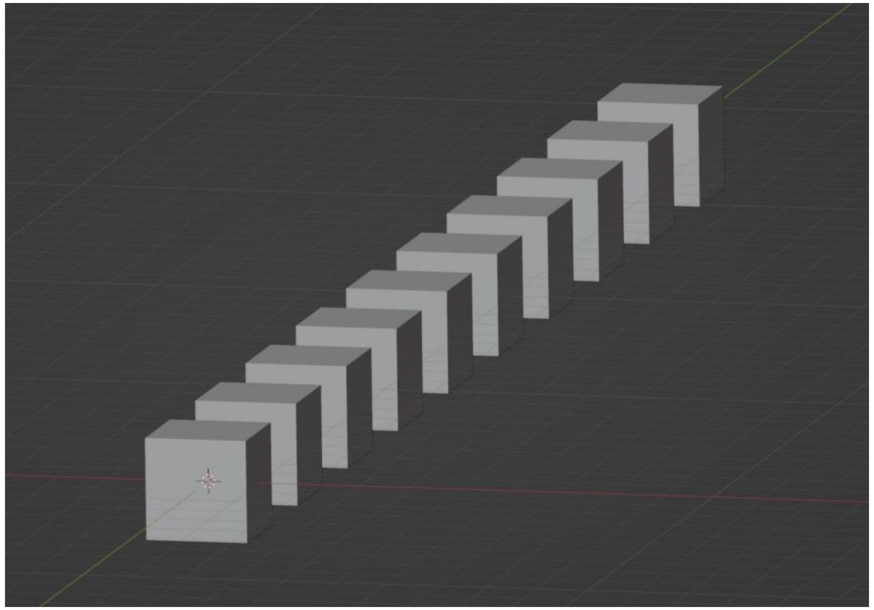
三个向量都单位化

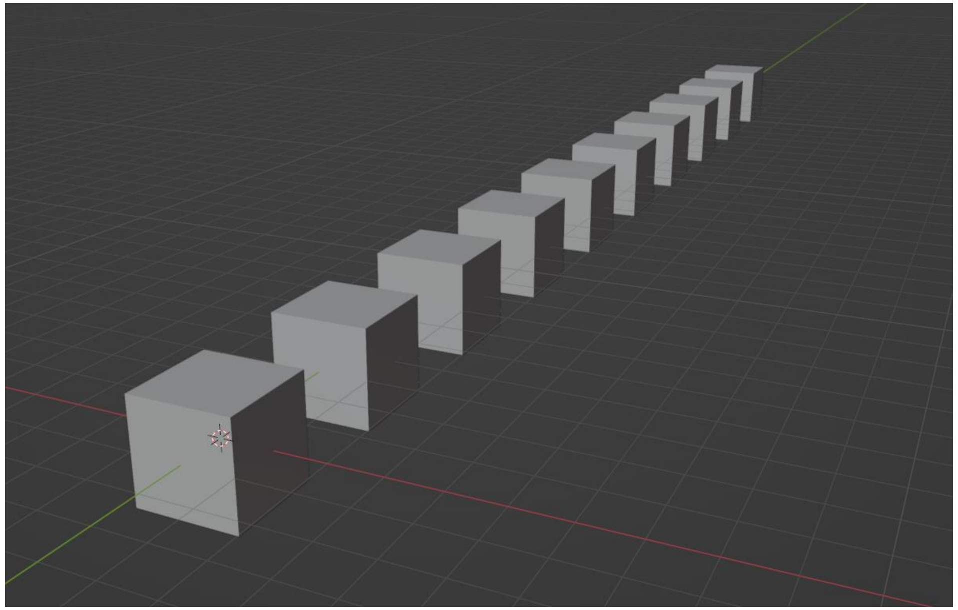
- $R_{view} = \left(R_{view}^{-1}\right)^T$ Why? Orthogonal matrix



Projection Transformation

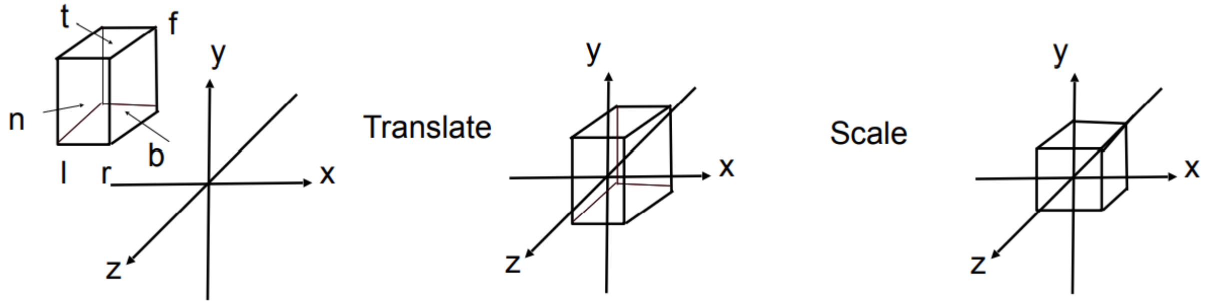
- Orthogonal v.s. Perspective





Orthographic Projection

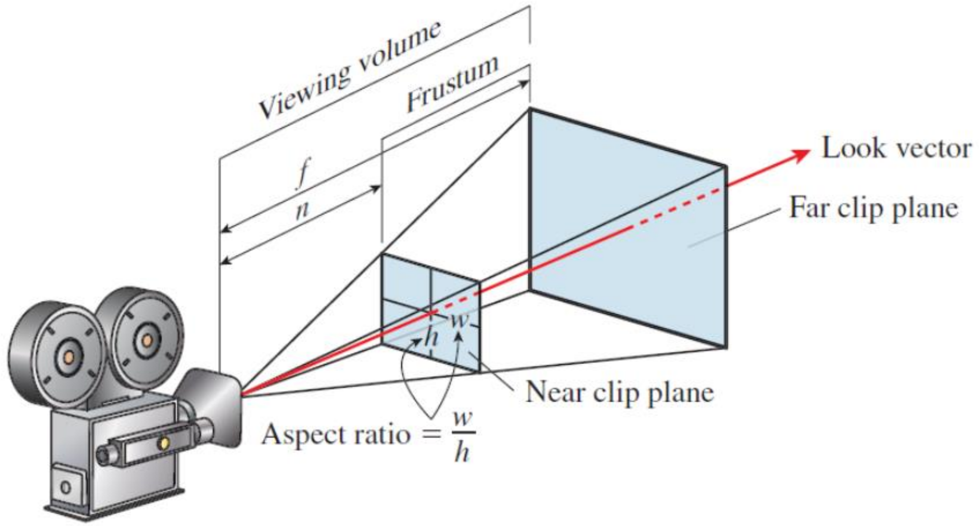
- Goal: map a cuboid $[l, r] \times [b, t] \times [-n, -f]$ to canonical cube $[-1, 1]^3$
- Translate center to origin first, then scale.

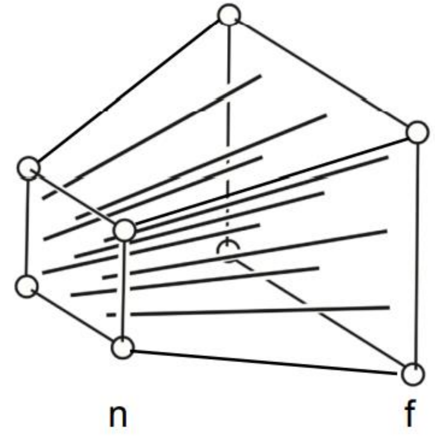


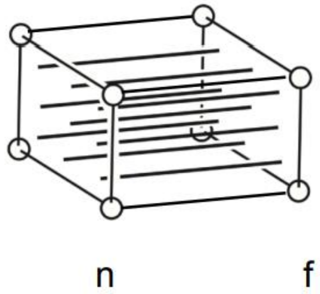
$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note!
OpenGL uses right-handed cartesian coordinates.
But NDC is left-handed cartesian coordinates.
So we need to flip z

Perspective Projection





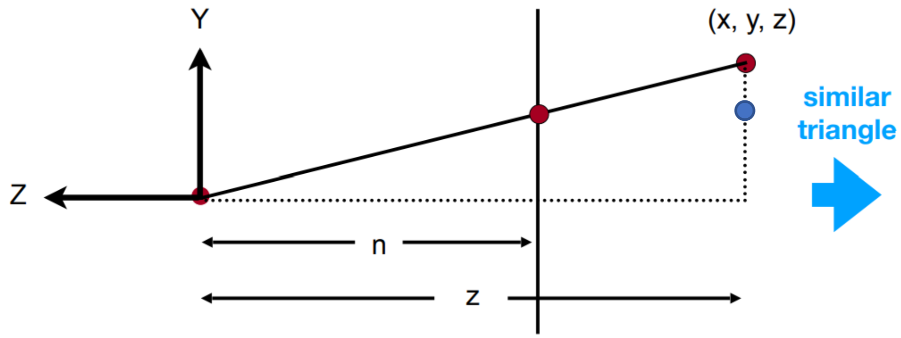


Perspective Projection

- $\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ 1 \end{bmatrix} = M_{persp} \begin{bmatrix} x_{view} \\ y_{view} \\ z_{view} \\ 1 \end{bmatrix}$
- $M_{persp} = M_{ortho} M_{persp \to ortho}$

$$M_{persp \to ortho} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ ? \\ -z \end{bmatrix}$$

$$M_{persp \to ortho} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



$$y' = \left|\frac{n}{z}\right| y \qquad x' = \left|\frac{n}{z}\right| x$$

Perspective Projection

- Figure out the third line of $M_{persp\to ortho}$.
- Any point on the near plane will not change.
- Any point's z coord on far plane will not change.

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

For any point on near plane:

$$\begin{bmatrix} x \\ y \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ -n^2 \\ n \end{bmatrix}$$

So:

$$\begin{bmatrix} 0 & 0 & A & B \end{bmatrix} \begin{bmatrix} x \\ y \\ -n \\ 1 \end{bmatrix} = -n^2$$

For any point on near plane:

$$\begin{bmatrix} 0 \\ 0 \\ -f \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -n^2 \\ f \end{bmatrix}$$

So:

$$\begin{bmatrix} 0 & 0 & A & B \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -f \\ 1 \end{bmatrix} = -f^2$$

$$\begin{cases} -An + B = -n^2 \\ -Af + B = -f^2 \end{cases}$$

$$\begin{cases} A = n + f \\ B = nf \end{cases}$$

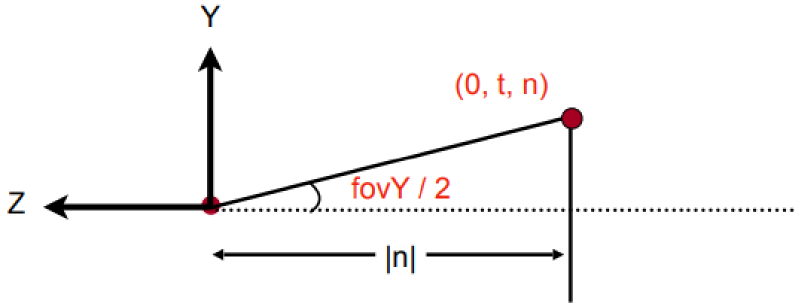
Perspective Projection

- $M_{persp\to ortho} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & nf \\ 0 & 0 & -1 & 0 \end{bmatrix}$
- So we can write final perspective matrix.

$$M_{persp} = M_{ortho} M_{persp\to ortho} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

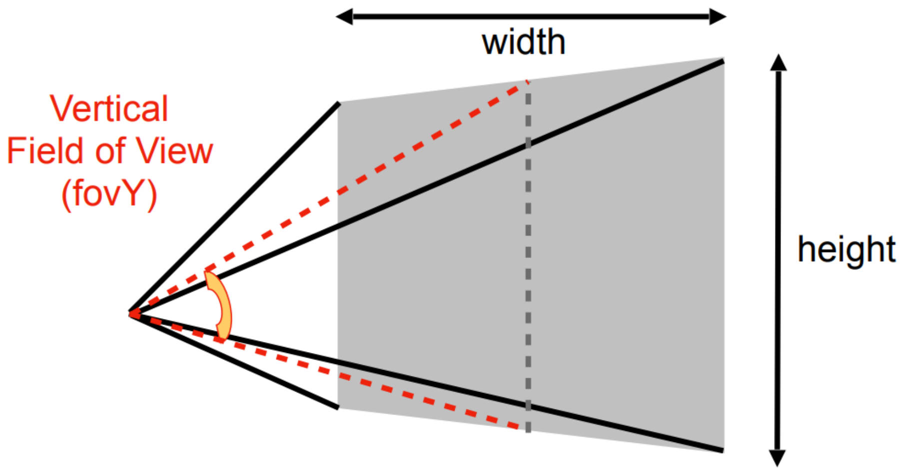
Perspective Projection

- How are l, r, t, b defined?
- Usually we assume symmetry. (i.e. l = -r, t = -b)
- $$\begin{cases} t = n \tan \dfrac{\text{fovY}}{2} \\ r = \text{asp} \times n \tan \dfrac{\text{fovY}}{2} \end{cases}$$



$$\tan \frac{fovY}{2} = \frac{t}{|n|}$$

$$aspect = \frac{r}{t}$$



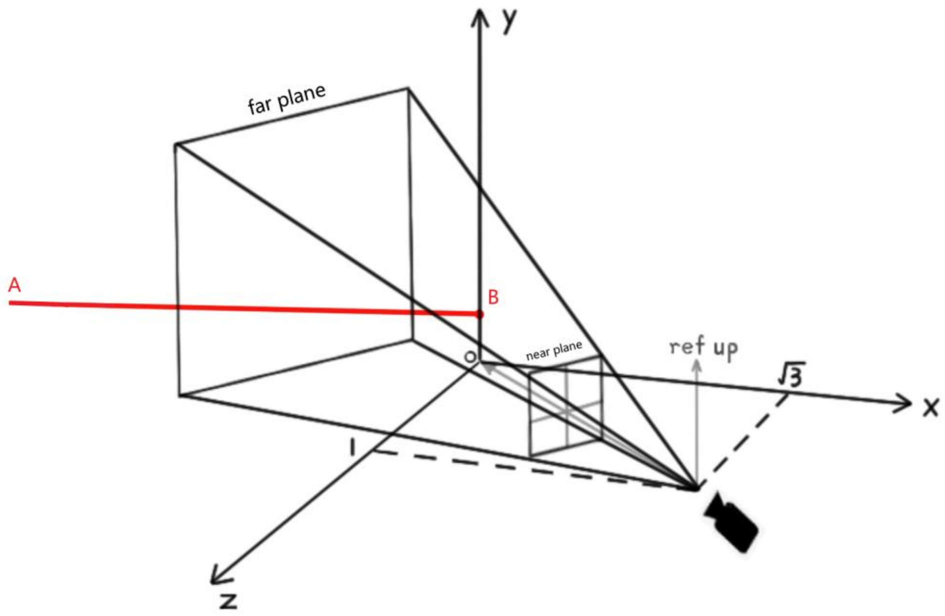
Aspect ratio = width / height

Viewport Tranformation

- Map all points in NDC space to screen, drop Z
- $[-1,\ 1]^2\ \rightarrow [0, width] \times [0, height]$

$$M_{viewport} = \begin{pmatrix} \frac{width}{2} & 0 & 0 & \frac{width}{2} \\ 0 & \frac{height}{2} & 0 & \frac{height}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Solution of Quiz.1



1. Consider a line segment AB in world space. $A(-2\sqrt{3}, \frac{\sqrt{3}}{2}, 4)$, $B(0, \frac{\sqrt{3}}{3}, 0)$. A pin-hole camera is positioned at $(\sqrt{3}, 0, 1)$, looking at the origin point. The reference up vector of the camera is $(0, 1, 0)$. The vertical field of view (fovY) is 60 degrees and aspect ratio is 1. The distance from camera to near plane is $\sqrt{3}$, and distance from camera to far plane is $3\sqrt{3}$. Screen resolution is 800×800. The origin point of the screen is at its left-bottom corner.

   Please calculate the coordinate of two end points of the line segment that screen can display in screen space.

   Hint: You need to account for clipping.

   Solution:

   gaze g=(-sqrt3,0,1)=(-sqrt3/2,0,-1/2)
   top t=(0,1,0)
   right r=gxt=(1/2,0,-sqrt3/2)

   Mview=Rview*Tview

$$M_{view} = \begin{bmatrix} \frac{1}{2} & 0 & -\frac{\sqrt{3}}{2} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad M_{persp} = \begin{bmatrix} \sqrt{3} & 0 & 0 & 0 \\ 0 & \sqrt{3} & 0 & 0 \\ 0 & 0 & -2 & -3\sqrt{3} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

   In NDC space:

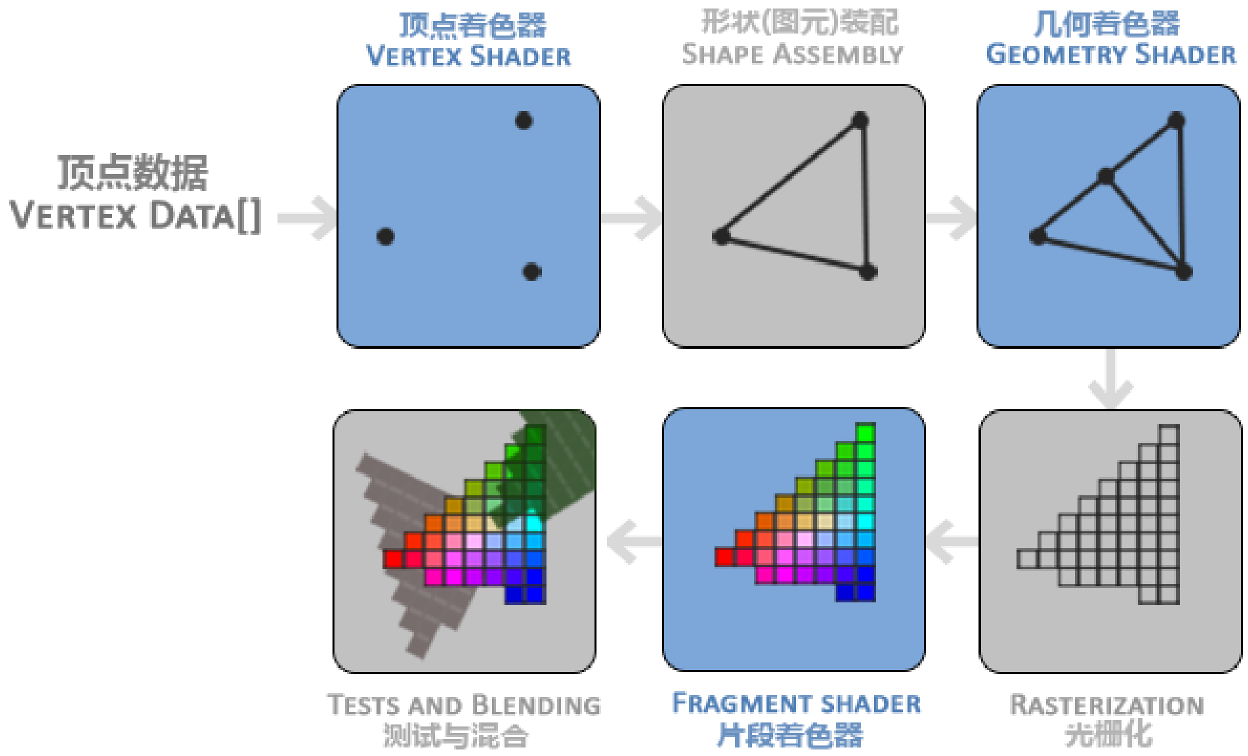
$$A' = M_{persp} M_{view} A = (-3, \frac{1}{2}, 2 - \sqrt{3}, 1)$$

$$B' = M_{persp} M_{view} B = (0, \frac{1}{2}, 2 - \frac{3\sqrt{3}}{2}, 1)$$

   A' is not in $[-1, 1]^3$, so the end point on screen should be $(-1, \frac{1}{2}, z)$ (no need to compute z's value). Convert $(-1, 0.5)$ and $(0, 0.5)$ to screen space. Two end points are $(0, 600)$ and $(400, 600)$ respectively.

Shader

A standard OpenGL rendering pipeline

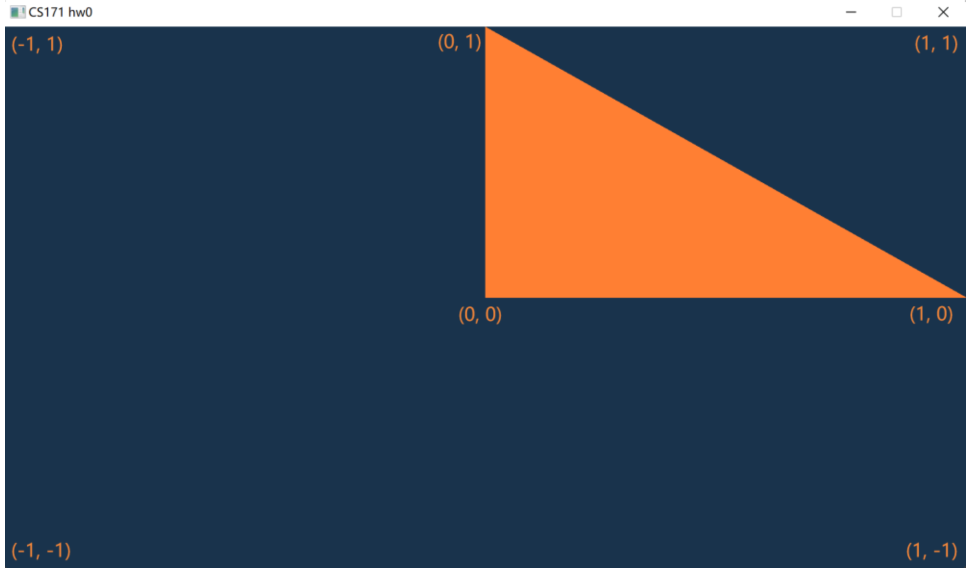


Shader

- How shader work?
- Here is the typical structure of a shader
- Written in GLSL (similar gramma as C)
- Usually put in file `my_shader.glsl`

```
#version version_number
in type in_variable_name;
in type in_variable_name;

out type out_variable_name;

uniform type uniform_name;

void main()
{
  // process input
  ...
  // output result to out variable
  out_variable_name = weird_stuff_we_processed;
}
```

Vertex Shader



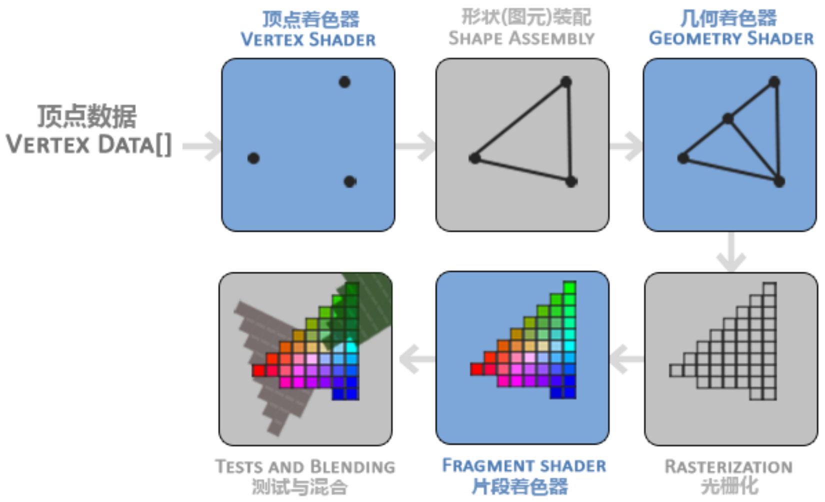
- Call back: in last tutorial…
  - we draw a right triangle.
  - Vertex coordinates are given in NDC space.
  - But now we only have coordinate in world space.

```
const float vertices[] = { 0, 0, 0, 1, 0, 0,  0, 1, 0};
```

```
// in vertex shader
#version 330 core
layout (location = 0) in vec3 pos;

void main() {
    gl_Position = vec4(pos, 1.0);
}
```

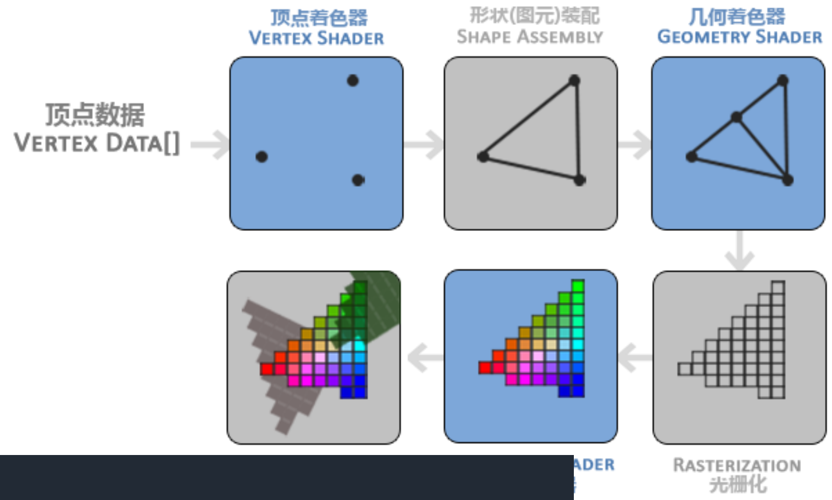
```
// in fragment shader
#version 330 core
out vec4 color;

void main() {
    color = vec4(1.0, 0.5, 0.2, 1.0);
}
```

Vertex Shader



- Transform 3D world coordinate to NDC coordinate
- Vertex Coordinate $= M_{proj}\, M_{modelview} P$
- Take vertex coordinate (and other info) from VBO with `layout`.
- Projection, model and view matrix are the same for all vertices, so use `uniform`.

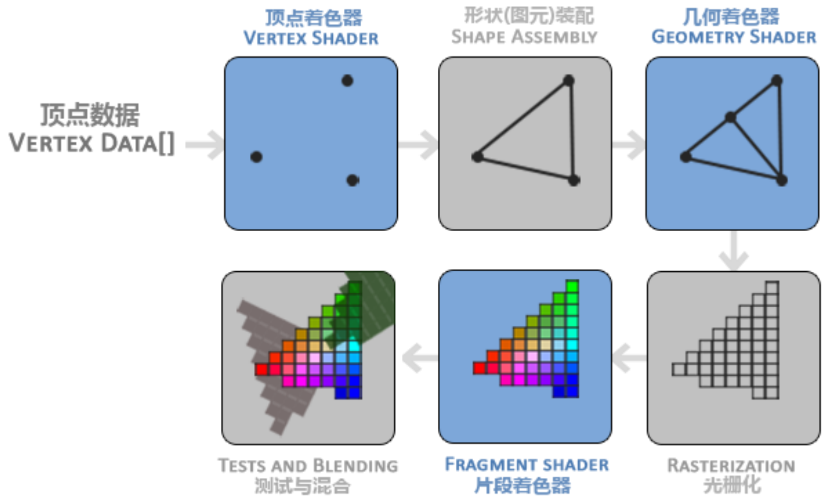
Vertex Shader - Layout




```
// in vertex shader
layout (location = 0) in vec3 pos;
layout (location = 1) in vec3 normal;
```

```
// in headers and .cpp files
struct Vertex {
  vec3 position;
  vec3 normal;
};

// vertex attribute: position
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void *)0);
glEnableVertexAttribArray(0);
// vertex attribute: normal
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex),
                                    (void *)(sizeof(glm::vec3)));
glEnableVertexAttribArray(1);
```

Vertex Shader - Uniform

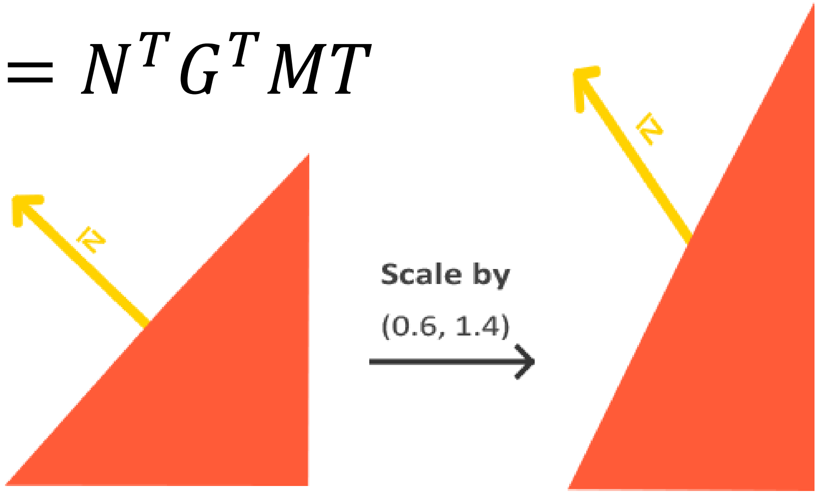



```
// in vertex shader
uniform mat4 model;
uniform mat4 view;
uniform mat4 proj;
```

```
// in .cpp files
// generate perspective projection matrix.
glm::mat4 proj = glm::perspective(glm::radians(fov), aspect_ratio, near, far);
// define a identify model matrix for Stanford bunny.
glm::mat4 model_bunny;

int modelLoc = glGetUniformLocation(ourShader.ID, "model");
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model_bunny));
... // same for View Matrix and Projection Matrix

// We have provide a class Shader, and you only need to call Shader::setMat4().
shader.setMat4("model", model_bunny);
```

Vertex Shader – Normal Transformation

- Let normal vector be N and tangent vector be T.
- Suppose we calculate lighting in view space…
- After model view transformation M, T became T'.
- After transformation, correct normal should be N'. Let correct normal transformation matrix be G.
- $N' \cdot T' = (GN) \cdot (MT) = (GN)^T(MT) = N^T G^T MT$
- $N' \cdot T'$ and $N \cdot T$ should both be 0.
- So $G^T M = I, G = (M^{-1})^T$

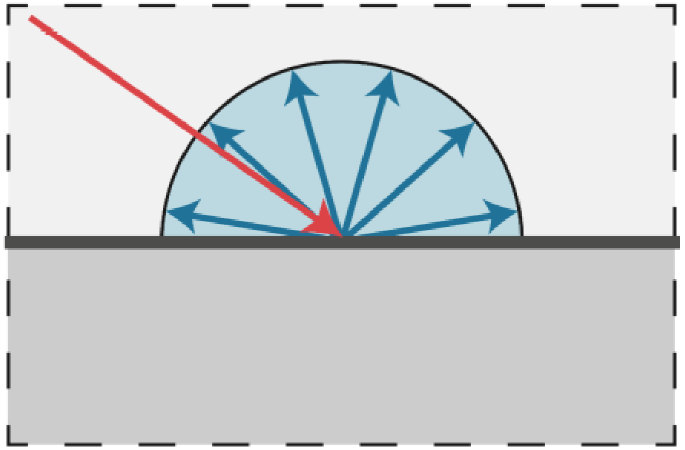


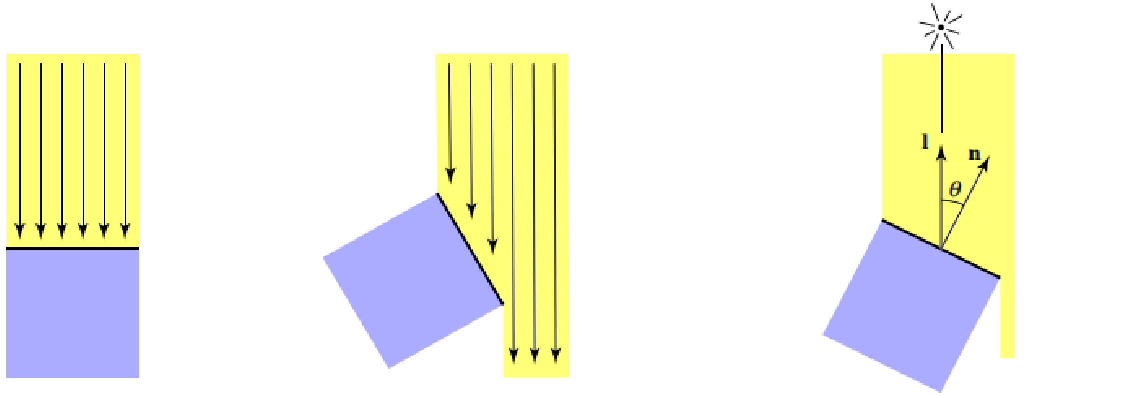
Vertex Shader

```glsl
#version 330 core
layout (location = 0) in vec3 pos;
layout (location = 1) in vec3 normal;

uniform mat4 model;
uniform mat4 view;
uniform mat4 proj;

out vec3 frag_normal;

void main() {
    gl_Position = proj * view * model * vec4(pos, 1.0);
    frag_normal
        = vec3((transpose(inverse(view * model))) * vec4(normal, 0.0));
}
```

Fragment Shader – Blinn-Phong Model

- Diffuse
  - color is the same for all viewing directions.
  - decided by angle between surface normal and light direction





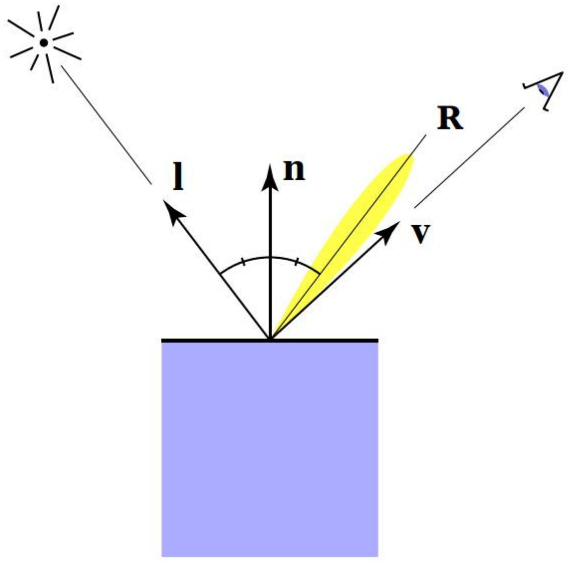
Top face of cube receives a certain amount of light

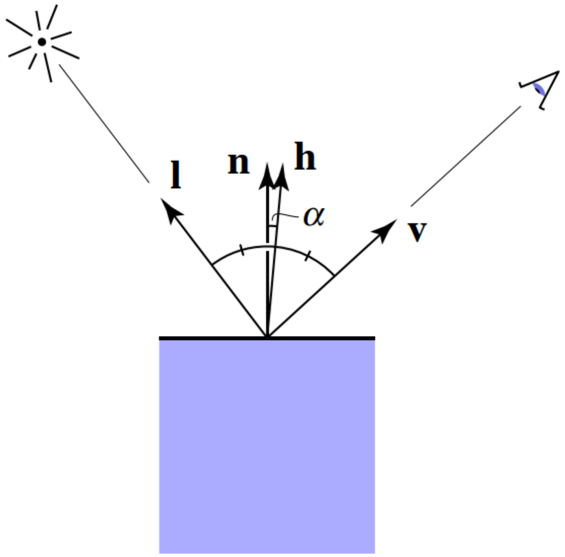
Top face of 60° rotated cube intercepts half the light

In general, light per unit area is proportional to cos θ = l • n

Fragment Shader – Blinn-Phong Model

- Specular
  - brighter near mirror reflection direction
  - V close to mirror direction ↔ half vector near normal





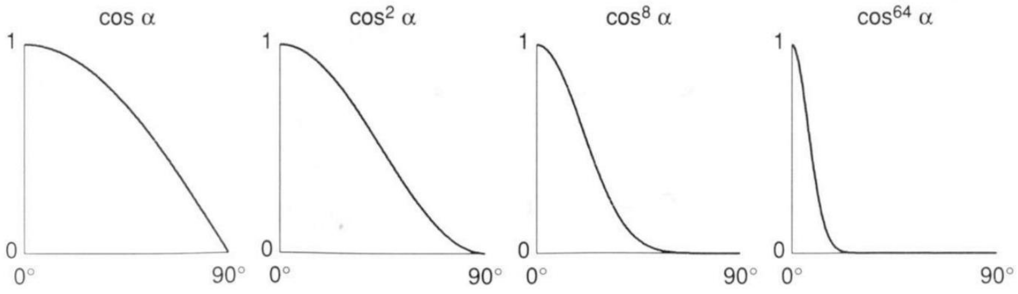
$$\begin{aligned} L_s &= k_s\,(I/r^2)\max(0,\cos\alpha)^p \\ &= k_s\,(I/r^2)\max(0,\mathbf{n}\cdot\mathbf{h})^p \end{aligned}$$

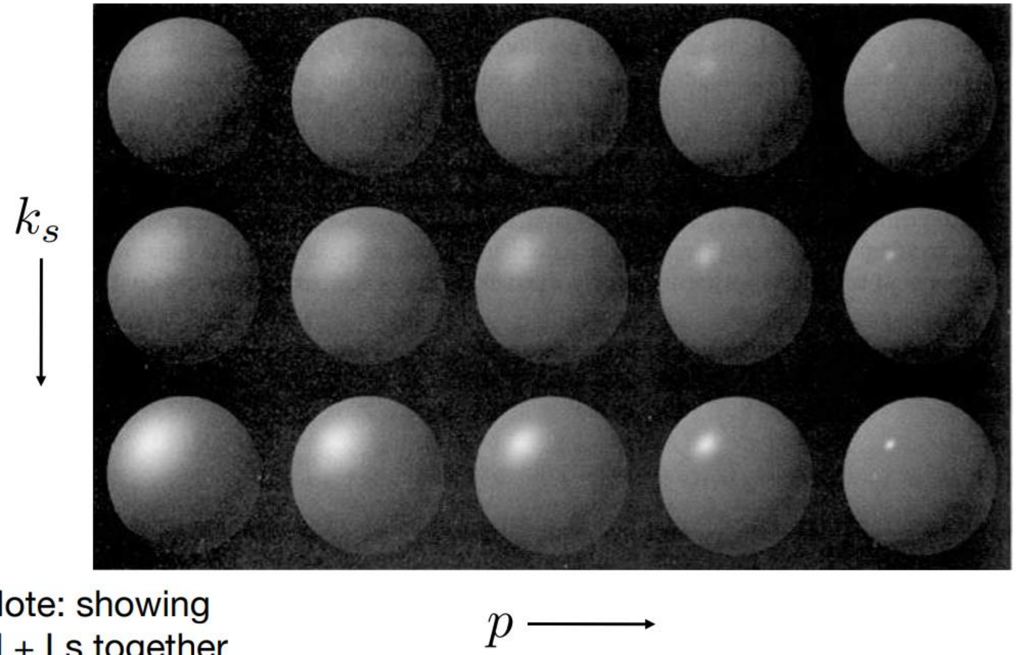
specularly reflected light

specular coefficient

Fragment Shader – Blinn-Phong Model

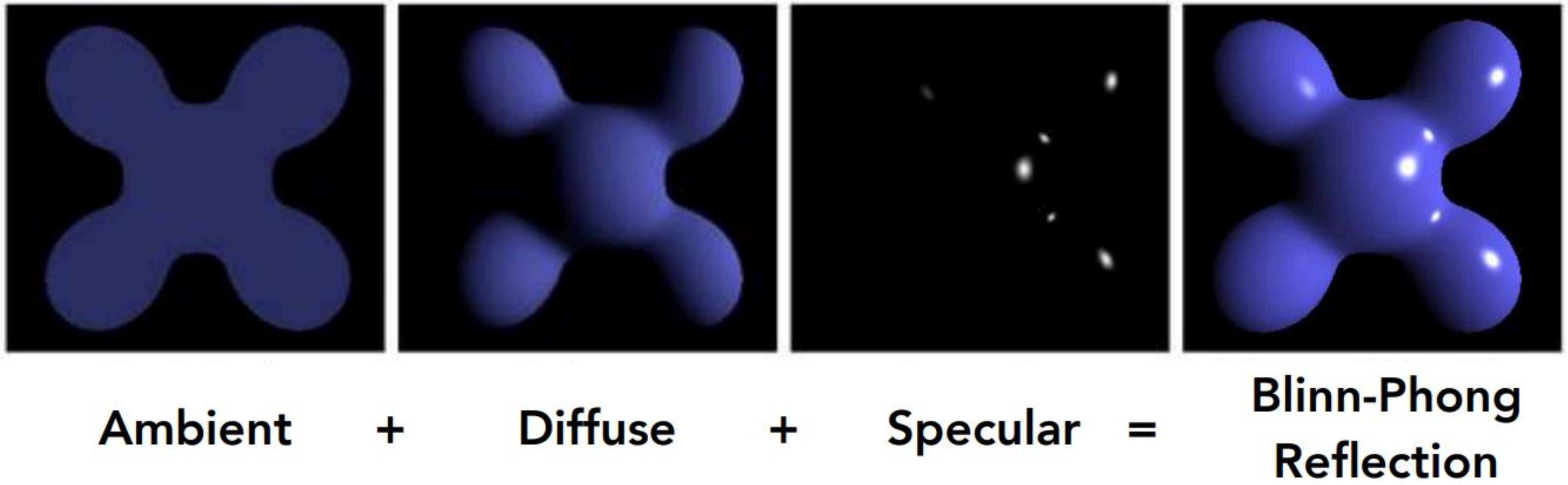
- Specular
  - brighter near mirror reflection direction
  - V close to mirror direction ↔ half vector near normal





Note: showing
Ld + Ls together

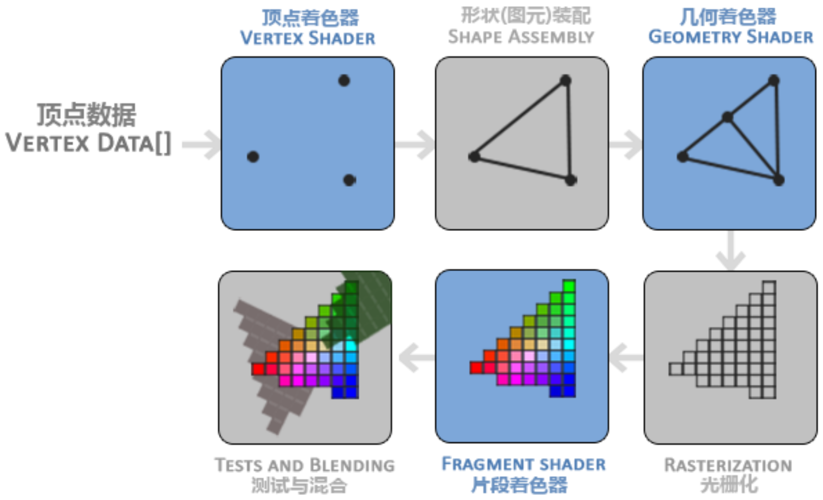
Fragment Shader – Blinn-Phong Model



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

Fragment Shader

```
void main() {
    // ambient
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;

     // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(LightPos - FragPos);
    vec3 diffuse = max(dot(norm, lightDir), 0.0) * lightColor;

    // specular
    float specularStrength = 0.5;
    vec3 viewDir = normalize(-FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
    vec3 specular = specularStrength * spec * lightColor;

    vec3 result = (ambient + diffuse + specular) * objectColor;
    FragColor = vec4(result, 1.0);
}
```

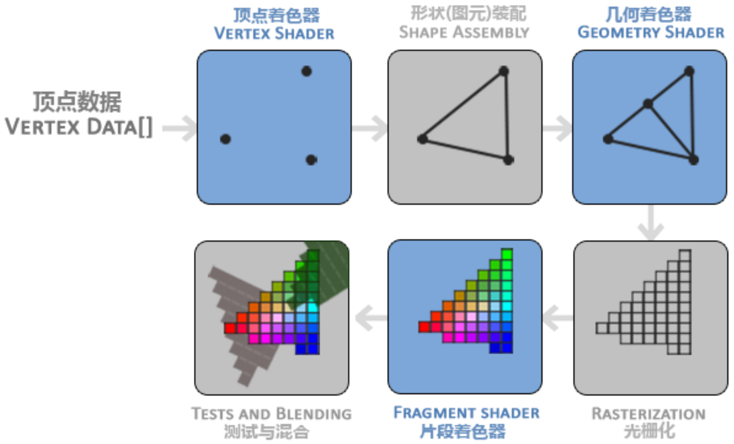



```
#version 330 core
out vec4 FragColor;

in vec3 FragPos;
in vec3 Normal;
in vec3 LightPos;

uniform vec3 lightColor;
uniform vec3 objectColor;
```

Last step – depth test and alpha blend

- Enable depth test to decide whether or not to draw current triangle.

```
glEnable(GL_DEPTH_TEST);
```



Z-Buffer Visibility Tests

Reference & Supplementary material

- LearnOpenGL CN
  - https://learnopengl-cn.github.io/
- GAMES101-现代计算机图形学入门-闫令琪 Lec.4
  - https://www.bilibili.com/video/BV1X7411F744?p=4&share_source=copy_web&vd_source=26e548adcc52816031d5eccbe815724b
- Transformation in OpenGL
  - https://blog.xehoth.cc/CG/OpenGL/Transformation-OpenGL/
- Shader Toy
  - https://www.shadertoy.com/browse

Thank You

Have fun with homework1. Have a nice holiday…