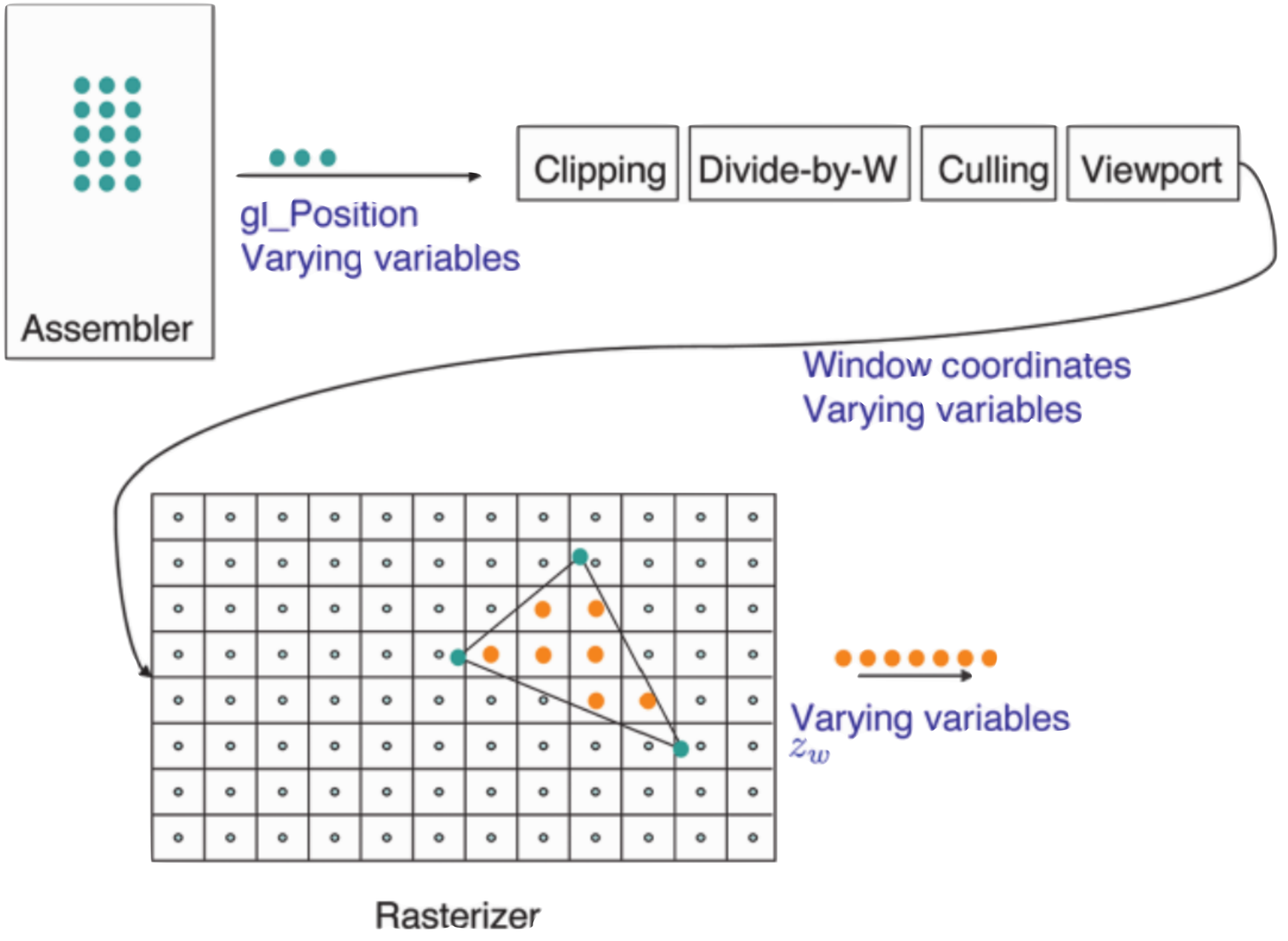
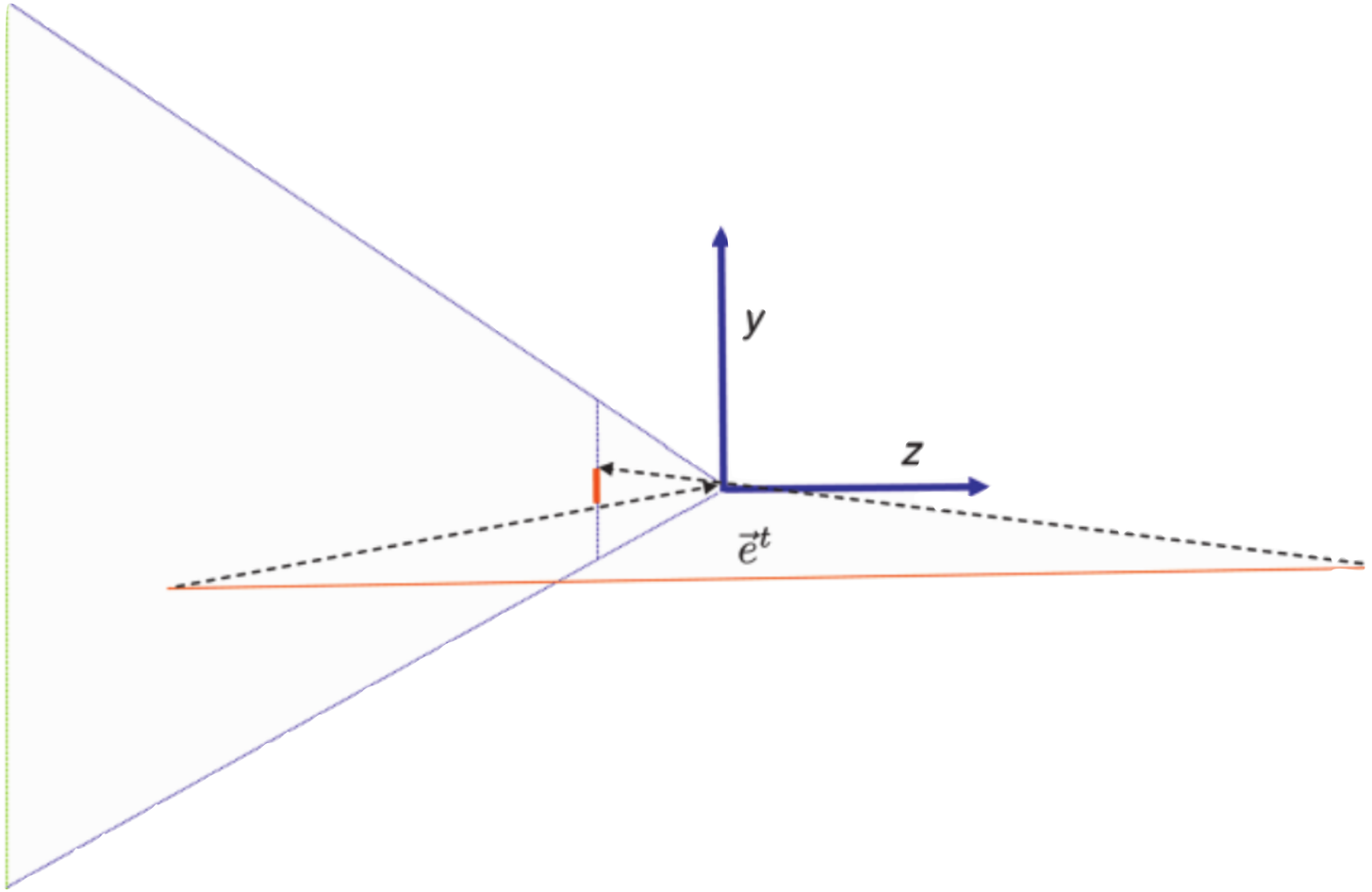
Fragment shaders and basic lighting

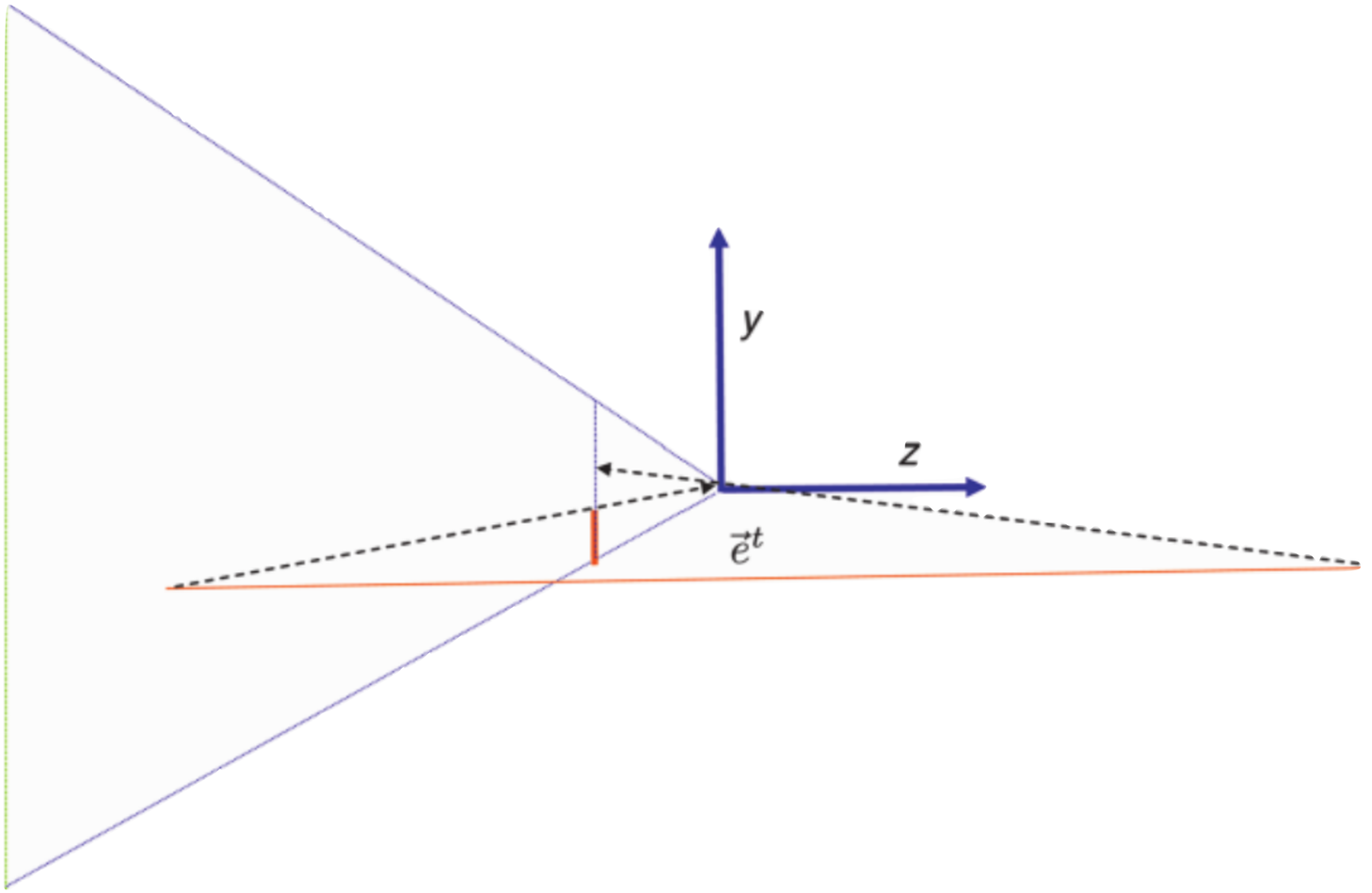
CS GY-6533 / UY-4533

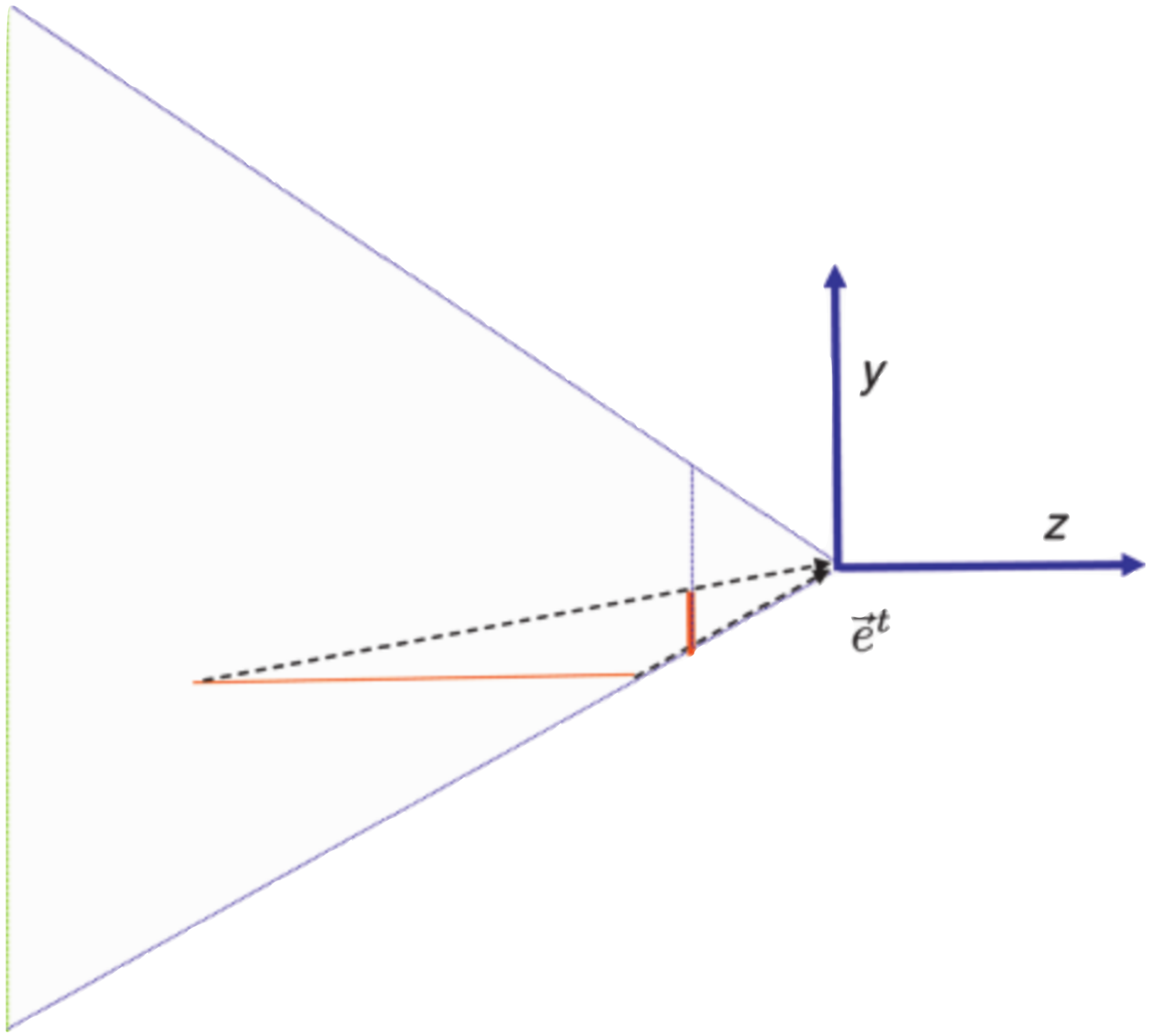
Revisiting the pipeline


Assembler
gl_Position
Varying variables
Clipping
Divide-by-W
Culling
Viewport
Window coordinates
Varying variables
Varying variables
z_w
Rasterizer

Clipping


y
z
$\vec{e}^t$


y
z
$\vec{e}^t$


y
z
$\vec{e}^t$

Clip coordinates

Clipping happens in homogeneous clip coordinates before they are divided by **w**.

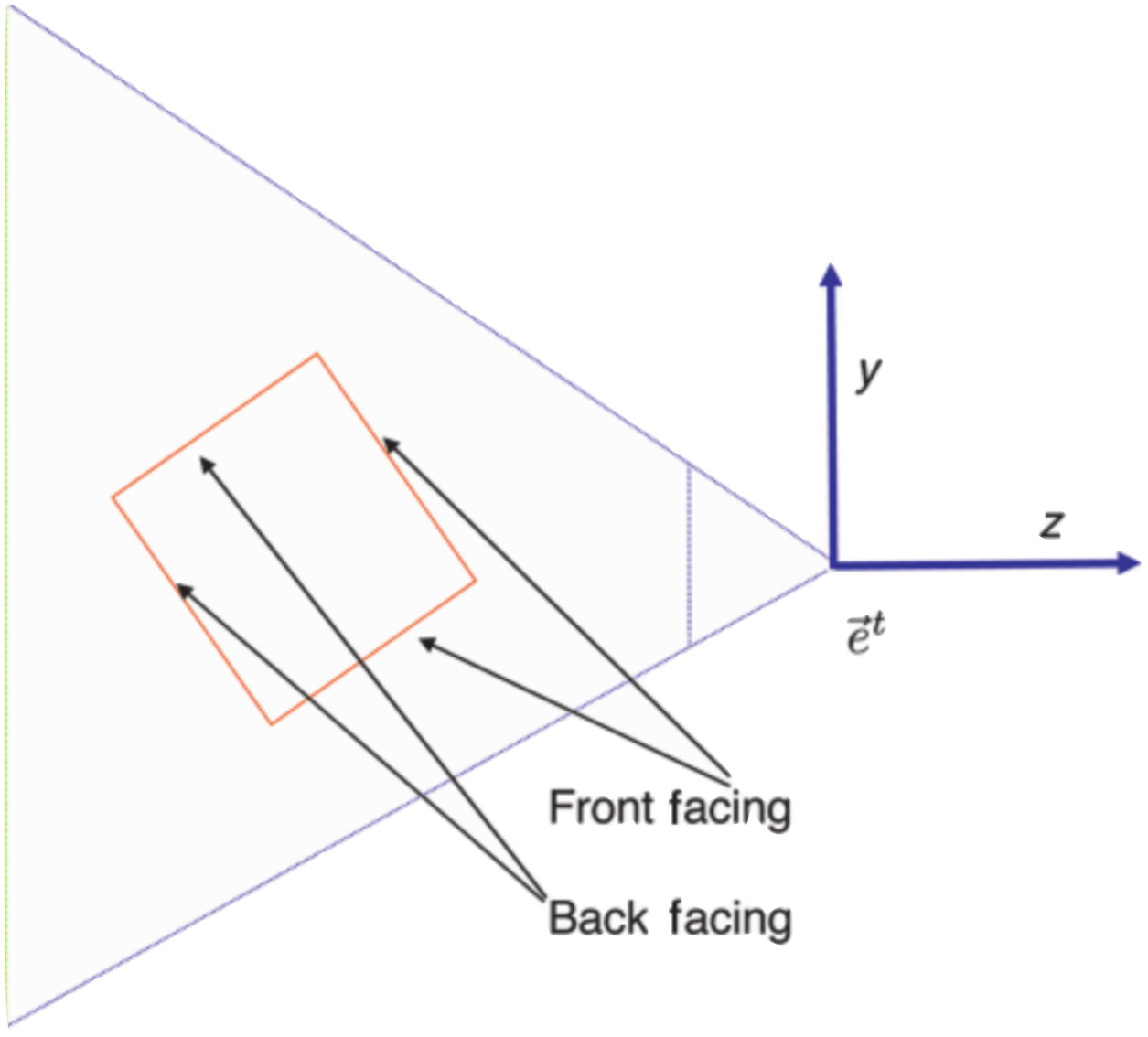
$$-1 < x_n < 1. \qquad -w_c < x_c < w_c$$

$$-1 < y_n < 1 \qquad -w_c < y_c < w_c$$

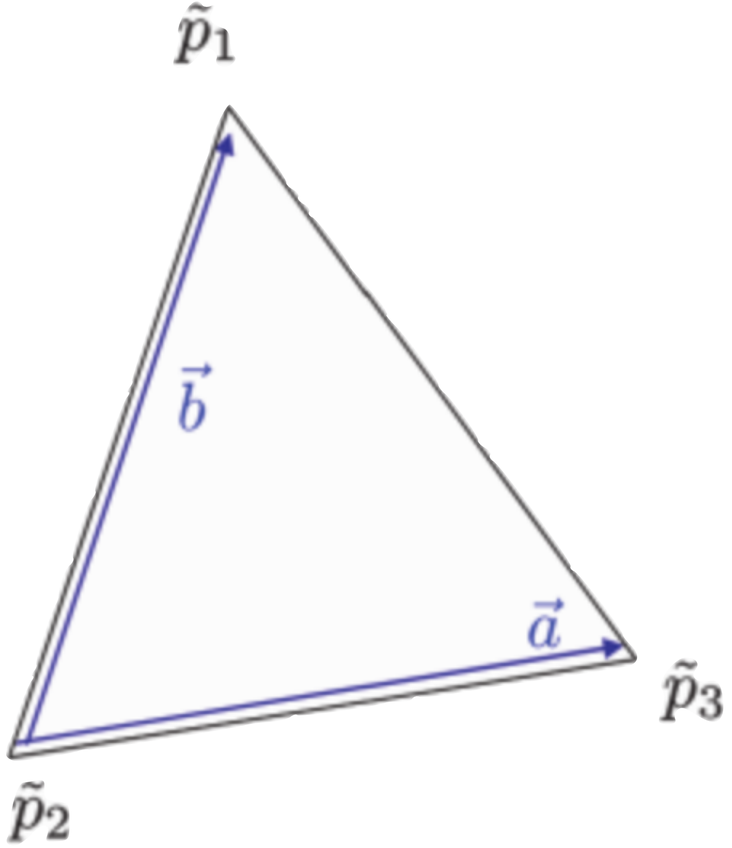
$$-1 < z_n < 1 \qquad -w_c < z_c < w_c.$$

Divide by w

Culling


y
z
$\vec{e}^t$
Front facing
Back facing

Which side is the front?



$$\vec{c} = \vec{a} \times \vec{b}.$$

$$(x_n^3 - x_n^2)(y_n^1 - y_n^2) - (y_n^3 - y_n^2)(x_n^1 - x_n^2).$$

Enabling culling in OpenGL.

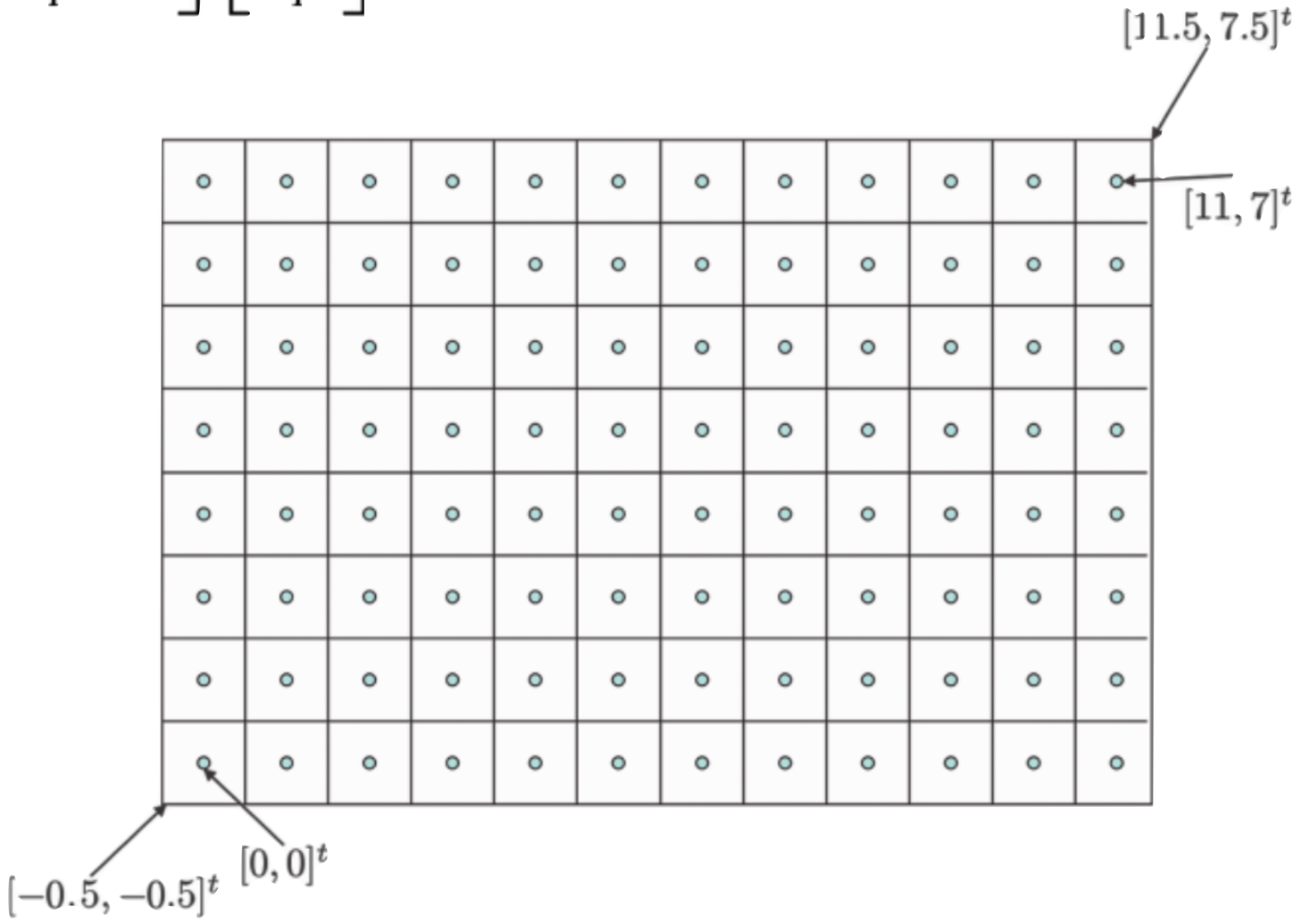
```
glEnable(GL_CULL_FACE);
```

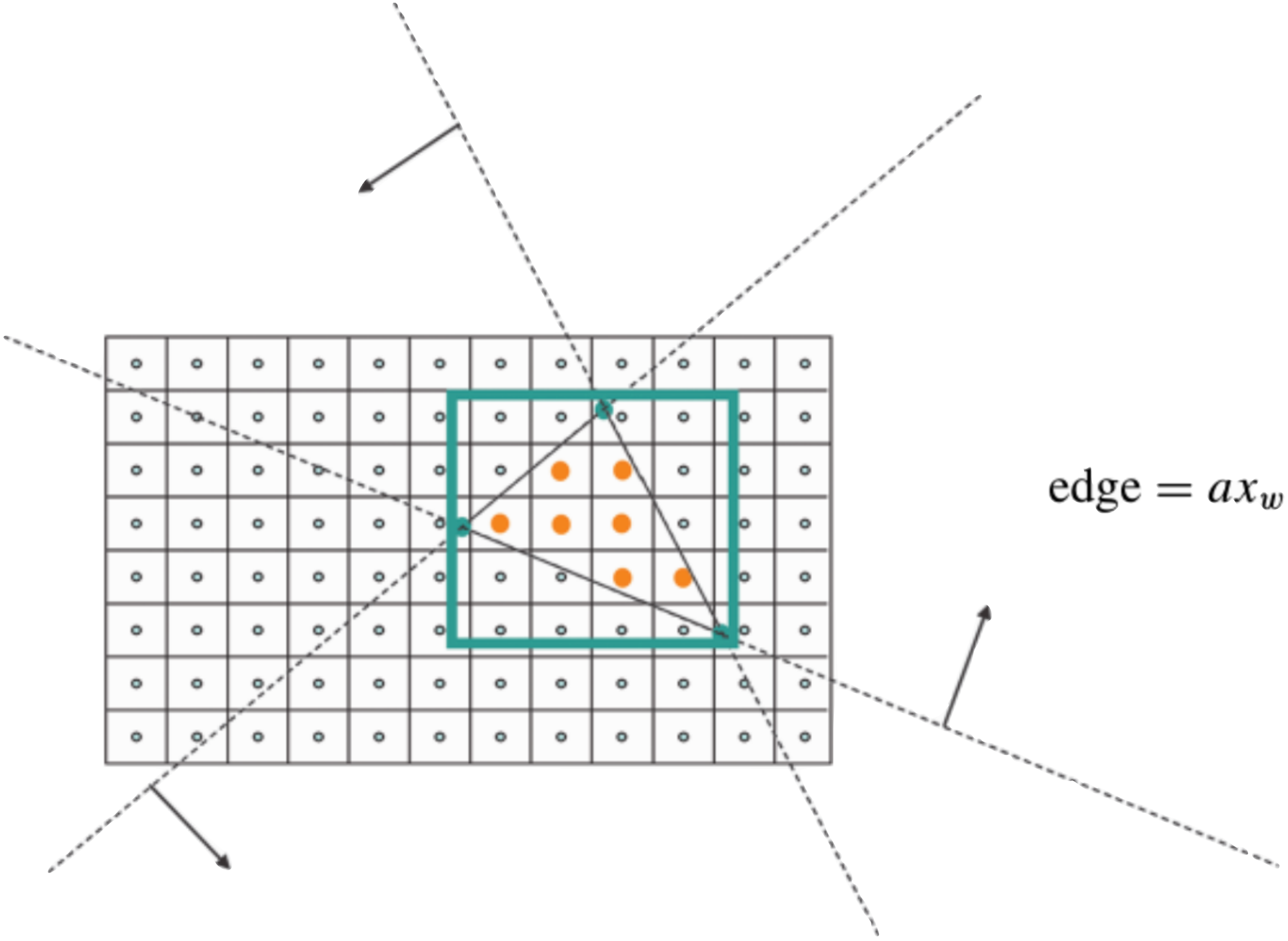
Specifying which side to cull.

```
glCullFace(GL_BACK); // GL_FRONT if we want to cull front facing
```

Viewport

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} W/2 & 0 & 0 & (W-1)/2 \\ 0 & H/2 & 0 & (H-1)/2 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \\ 1 \end{bmatrix}.$$

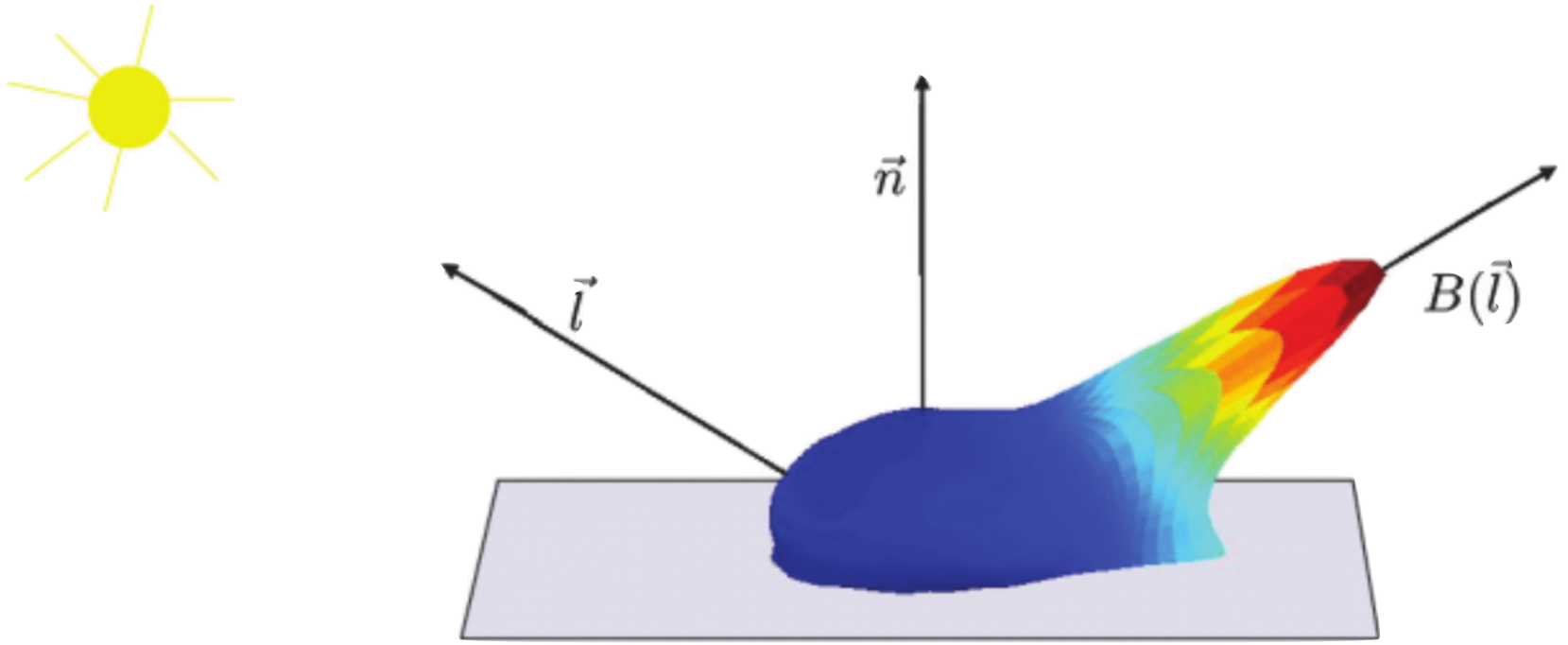


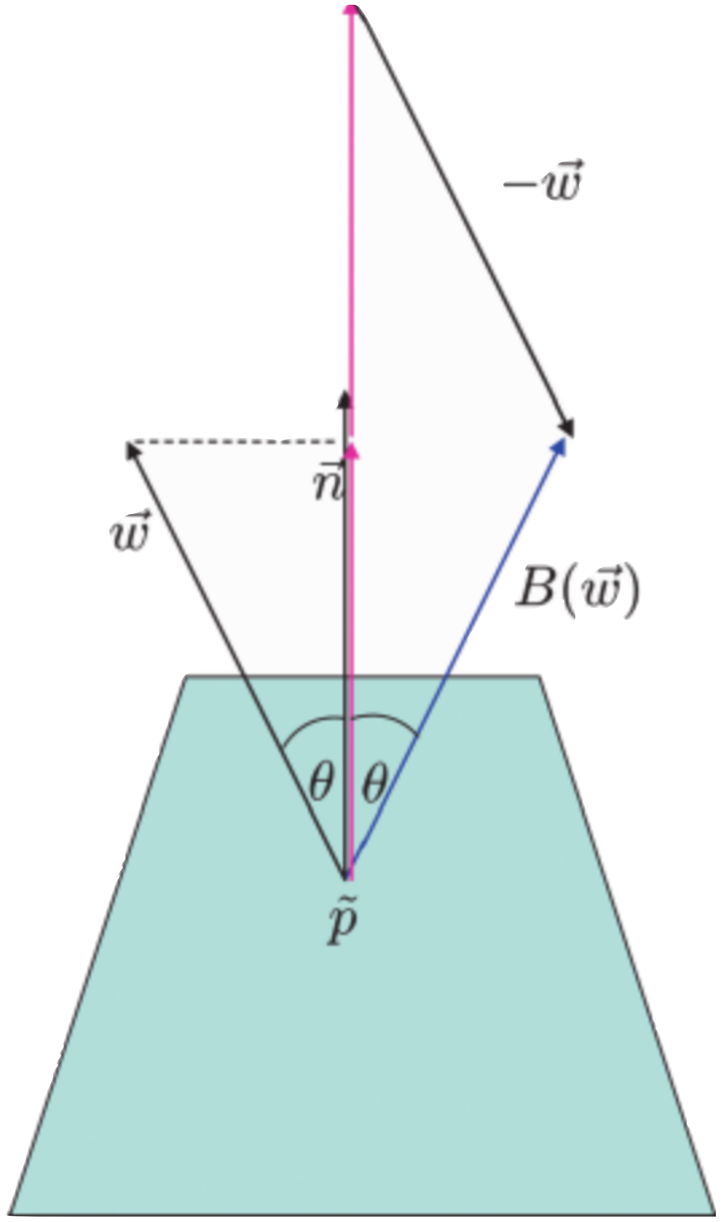


$$\text{edge} = ax_w + by_w + c,$$

Fragment shaders

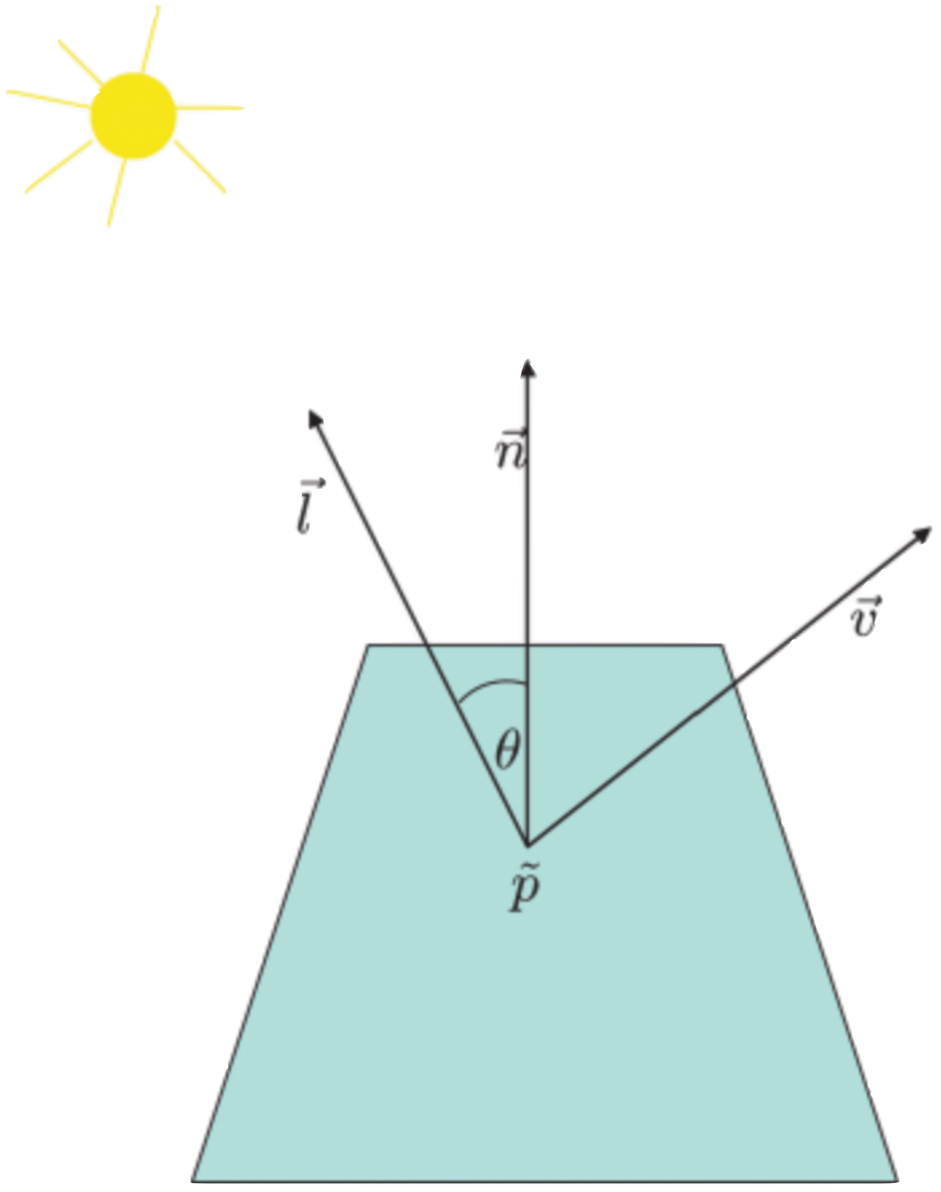
Basic lighting


$\vec{n}$
$\vec{l}$
$B(\vec{l})$



$$B(\vec{w}) = 2(\vec{w} \cdot \vec{n})\vec{n} - \vec{w}.$$

Diffuse



$\vec{n} \cdot \vec{l}$

Vertex program

```
attribute vec4 position;
attribute vec4 normal;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform mat4 normalMatrix;

varying vec3 varyingNormal;

void main() {
    varyingNormal = normalize((normalMatrix * normal).xyz);
    gl_Position = projectionMatrix * modelViewMatrix * position;
}
```

Fragment program

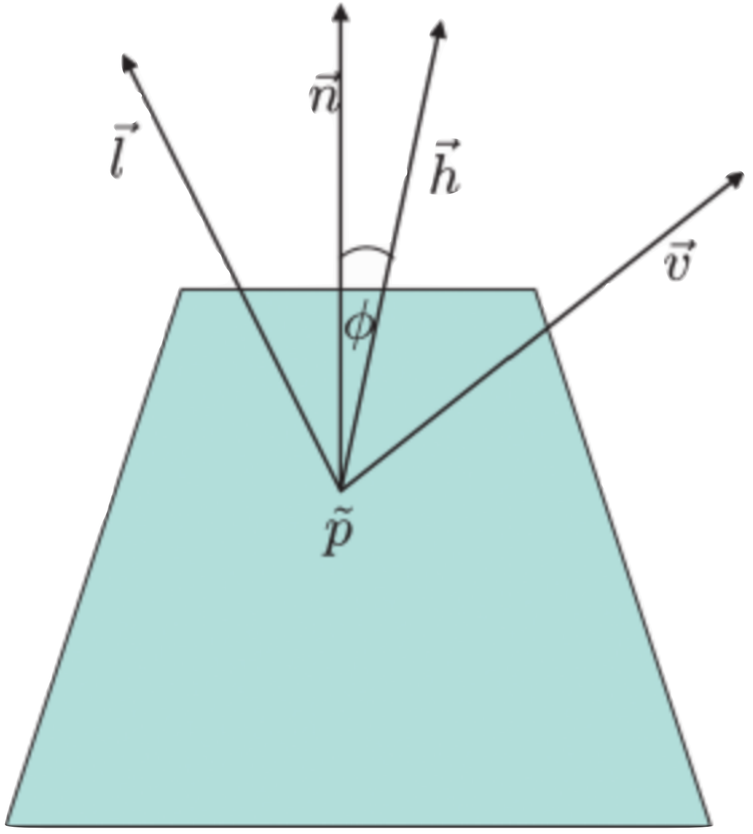
```
varying vec3 varyingNormal;
uniform vec3 uColor;
uniform vec3 lightDirection;

void main() {
  float diffuse = max(0.0, dot(varyingNormal, lightDirection));
  vec3 intensity = uColor * diffuse;
    gl_FragColor = vec4(intensity.xyz, 1.0);
}
```

Light direction must be in eye space!

If we want to define a light direction in world space, we must use the transpose of inverse of our view matrix (with translation zero) to rotate the light direction vector into eye space.

Specular



$$\vec{h} = \text{normalize}(\vec{v} + \vec{l})$$

Vertex program

```
attribute vec4 position;
attribute vec4 normal;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform mat4 normalMatrix;

varying vec3 varyingNormal;
varying vec3 varyingPosition;

void main() {
    varyingNormal = normalize((normalMatrix * normal).xyz);
    vec4 p = modelViewMatrix * position;
    varyingPosition = p.xyz;
    gl_Position = projectionMatrix * p;
}
```

Fragment program

```
varying vec3 varyingNormal;
varying vec3 varyingPosition;

uniform vec3 uColor;
uniform vec3 lightDirection;

void main() {
    float diffuse = max(0.0, dot(varyingNormal, lightDirection));
    vec3 v = normalize(-varyingPosition);
    vec3 h = normalize(v + lightDirection);

    float specular = pow(max(0.0, dot(h, varyingNormal)), 64.0);
    vec3 specularHighlight = vec3(1.0, 1.0, 1.0) * specular;

    vec3 intensity = (uColor * diffuse) + specularHighlight;
    gl_FragColor = vec4(intensity.xyz, 1.0);
}
```

Light color

Fragment program

```
varying vec3 varyingNormal;
varying vec3 varyingPosition;
uniform vec3 uColor;

uniform vec3 lightDirection;
uniform vec3 lightColor;
uniform vec3 specularLightColor;

void main() {
    float diffuse = max(0.0, dot(varyingNormal, lightDirection));
    vec3 diffuseColor = lightColor * diffuse;

    vec3 v = normalize(-varyingPosition);
    vec3 h = normalize(v + lightDirection);
    float specular = pow(max(0.0, dot(h, varyingNormal)), 64.0);
    vec3 specularHighlight = specularLightColor * specular;

    vec3 intensity = (uColor * diffuseColor) + specularHighlight;
    gl_FragColor = vec4(intensity.xyz, 1.0);
}
```

Multiple lights

```
varying vec3 varyingNormal;
varying vec3 varyingPosition;
uniform vec3 uColor;

struct Light {
    vec3 lightDirection;
    vec3 lightColor;
    vec3 specularLightColor;
};

uniform Light lights[2];

void main() {
    vec3 diffuseColor = vec3(0.0, 0.0, 0.0);
    vec3 specularColor = vec3(0.0, 0.0, 0.0);

    for(int i=0; i< 2; i++) {
        float diffuse = max(0.0, dot(varyingNormal, lights[i].lightDirection));
        diffuseColor += lights[i].lightColor * diffuse;
        vec3 v = normalize(-varyingPosition);
        vec3 h = normalize(v + lights[i].lightDirection);
        float specular = pow(max(0.0, dot(h, varyingNormal)), 64.0);
        specularColor += lights[i].specularLightColor * specular;
    }

     vec3 intensity = (uColor * diffuseColor) + specularColor;
    gl_FragColor = vec4(intensity.xyz, 1.0);
}
```

Setting array and structure uniforms.

```
lightDirectionUniformLocation0 = glGetUniformLocation(program, "lights[0].lightDirection");
lightDirectionUniformLocation1 = glGetUniformLocation(program, "lights[1].lightDirection");
```

Positional lights

Light direction is direction from the fragment position to the light position.

```
varying vec3 varyingNormal;
varying vec3 varyingPosition;
uniform vec3 uColor;

struct Light {
    vec3 lightPosition;
    vec3 lightColor;
    vec3 specularLightColor;
};

uniform Light lights[2];

void main() {
    vec3 diffuseColor = vec3(0.0, 0.0, 0.0);
    vec3 specularColor = vec3(0.0, 0.0, 0.0);

    for(int i=0; i< 2; i++) {
        vec3 lightDirection = -normalize(varyingPosition-lights[i].lightPosition);
        float diffuse = max(0.0, dot(varyingNormal, lightDirection));
        diffuseColor += (lights[i].lightColor * diffuse) * attenuation;

        vec3 v = normalize(-varyingPosition);
        vec3 h = normalize(v + lightDirection);
        float specular = pow(max(0.0, dot(h, varyingNormal)), 64.0);
        specularColor += lights[i].specularLightColor * specular;
    }

  vec3 intensity = (uColor * diffuseColor) + specularColor;
    gl_FragColor = vec4(intensity.xyz, 1.0);
}
```

Light positions must be defined in eye space!

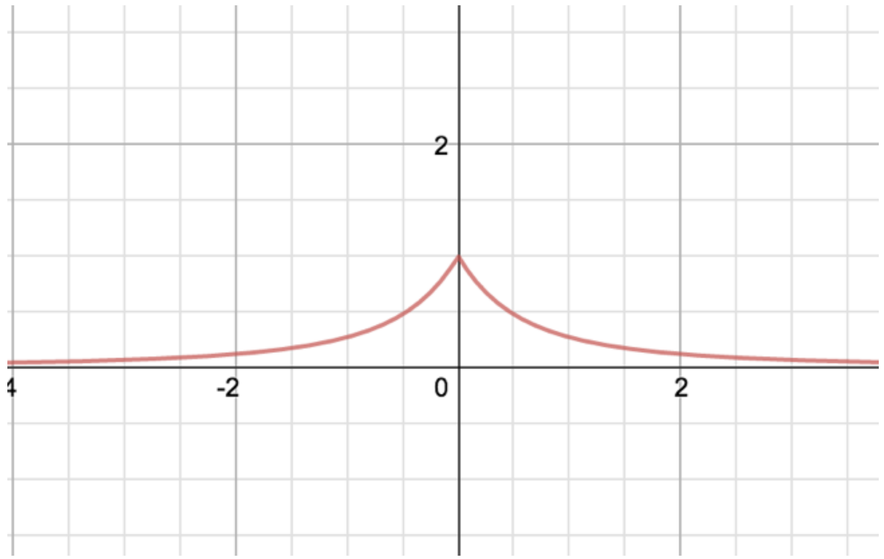
Attenuation

Light attenuation

Basic attenuation function.

$$\frac{1}{1+a|x|+b|x|^2}$$

```
1.0 / (1.0 + a*dist + b*dist*dist)
```



See how a and b values affect the attenuation graph:

https://www.desmos.com/calculator/nmnaud1hrw

```
varying vec3 varyingNormal;
varying vec3 varyingPosition;
uniform vec3 uColor;

struct Light {
    vec3 lightPosition;
    vec3 lightColor;
    vec3 specularLightColor;
};

uniform Light lights[2];

float attenuate(float dist, float a, float b) {
    return 1.0 / (1.0 + a*dist + b*dist*dist);
}

void main() {
    vec3 diffuseColor = vec3(0.0, 0.0, 0.0);
    vec3 specularColor = vec3(0.0, 0.0, 0.0);

    for(int i=0; i< 2; i++) {
        vec3 lightDirection = -normalize(varyingPosition-lights[i].lightPosition);
        float diffuse = max(0.0, dot(varyingNormal, lightDirection));
        float attenuation = attenuate(distance(varyingPosition, lights[i].lightPosition) / 5.0, 2.7, 5.0);
        diffuseColor += (lights[i].lightColor * diffuse) * attenuation;

        vec3 v = normalize(-varyingPosition);
        vec3 h = normalize(v + lightDirection);
        float specular = pow(max(0.0, dot(h, varyingNormal)), 64.0);
        specularColor += lights[i].specularLightColor * specular * attenuation;
    }

  vec3 intensity = (uColor * diffuseColor) + specularColor;
    gl_FragColor = vec4(intensity.xyz, 1.0);
}
```