Transformations Part 3

CS GY-6533 / UY-4533

“Look At” Matrix

$$\mathbf{z} = \texttt{normalize}(\mathbf{q} - \mathbf{p})$$

$$\mathbf{y} = \text{normalize}(\mathbf{u})$$

$$\mathbf{x} = \mathbf{y} \times \mathbf{z},$$

"Look At" Matrix for eye coordinate **p** and target point **q** and up vector **u**.

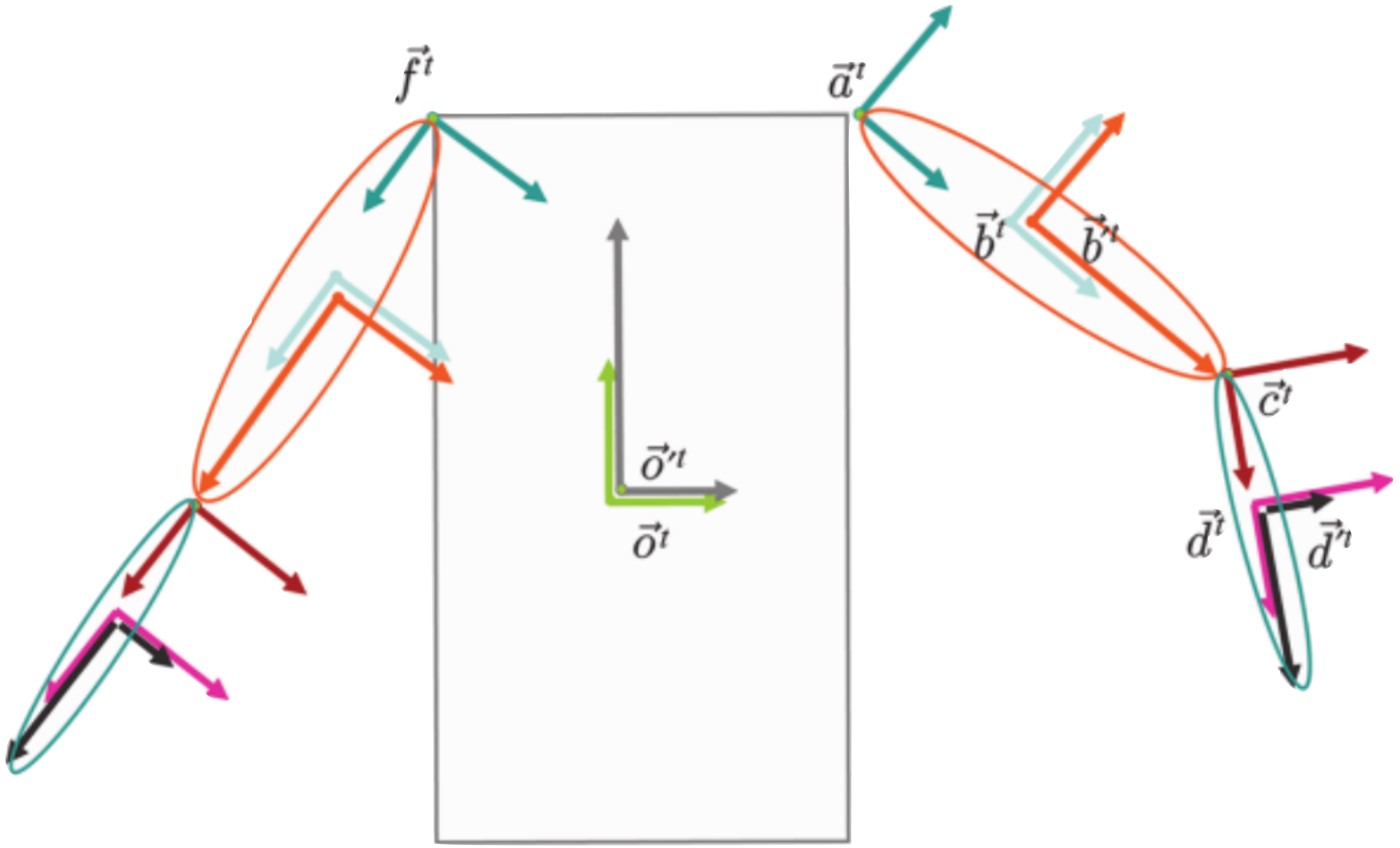
$$\begin{bmatrix} x_1 & y_1 & z_1 & p_1 \\ x_2 & y_2 & z_2 & p_2 \\ x_3 & y_3 & z_3 & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Keep in mind, the up vector here is in world space, to get a relative up vector use

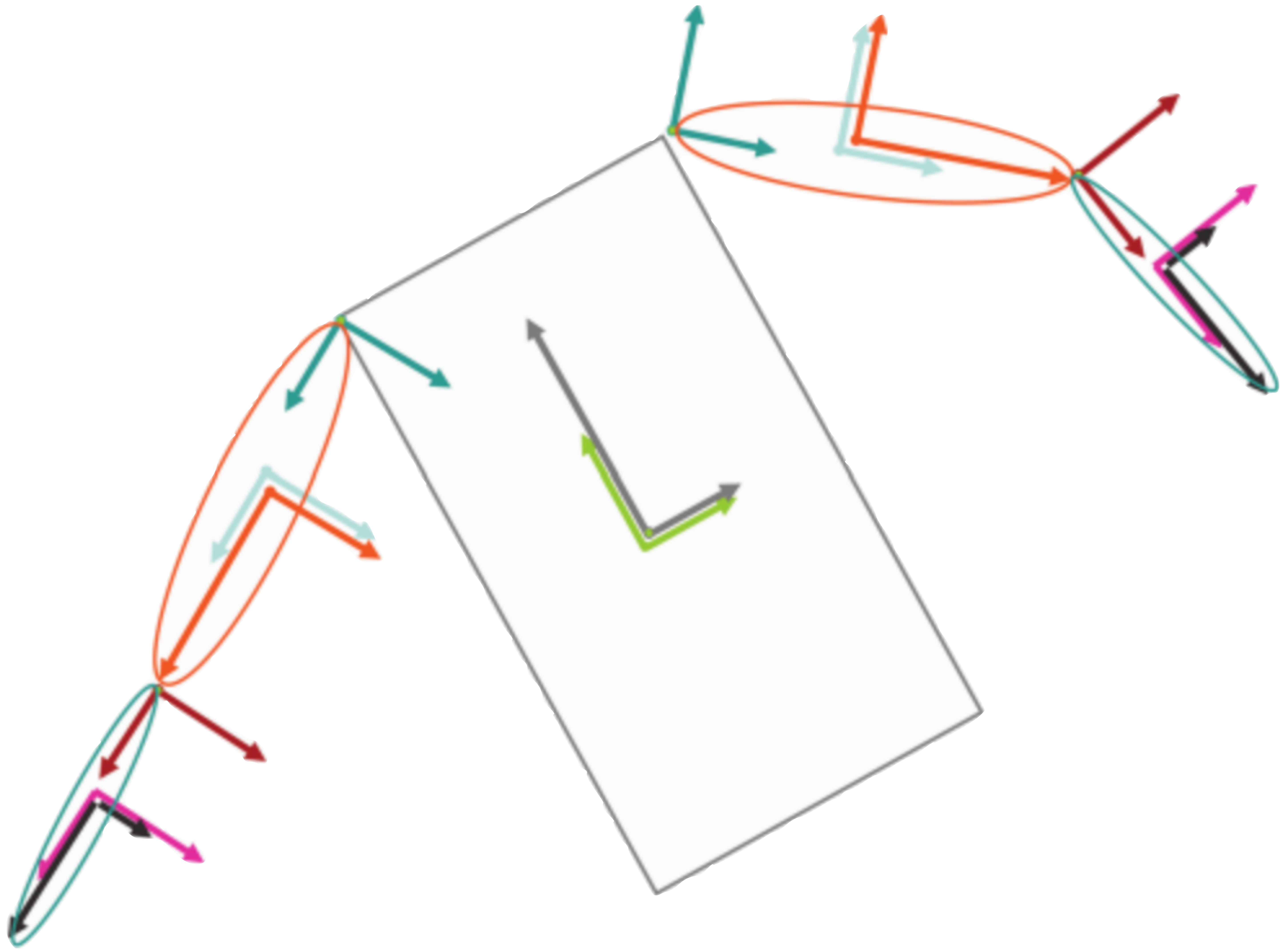
s x **z**

where **s** is **z** x **y**

Hierarchy



$\vec{f}^t$
$\vec{a}^t$
$\vec{b}^t$
$\vec{b}'^t$
$\vec{c}^t$
$\vec{d}^t$
$\vec{d}'^t$
$\vec{o}'^t$
$\vec{o}^t$



A simple scene graph.

```
Entity {
  Cvec3 t; // translation
  Cvec3 r; // rotation
  Cvec3 s; // scale

  Matrix4 modelMatrix;
  Entity *parent;
}
```

Building entity’s model matrix.

T - translation

R - $R_x R_y R_z$

S - scale

P - parent’s model matrix or identity if no parent

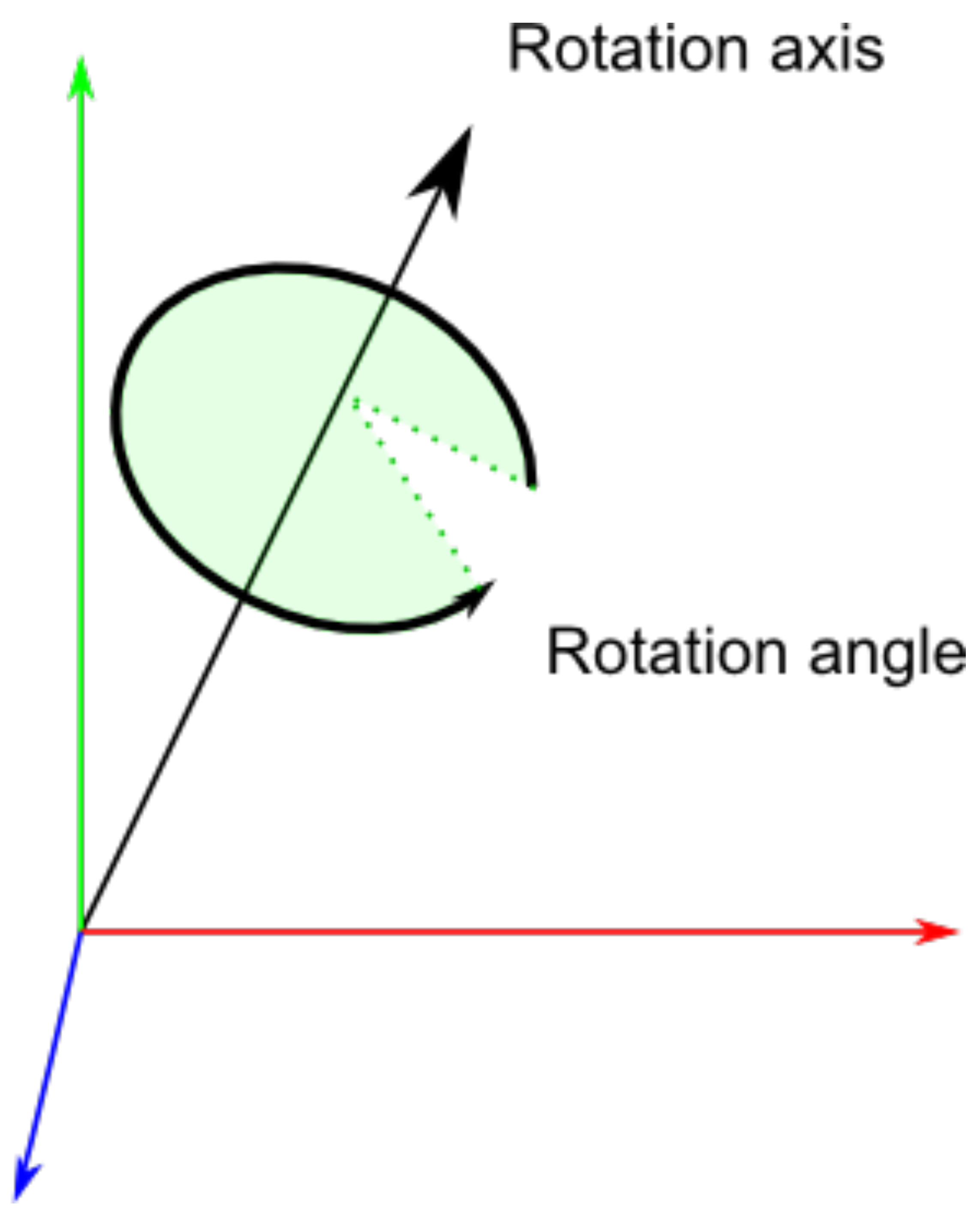
M = PTRS

Axis-angle rotation

Rotation around an axis **(z)**.

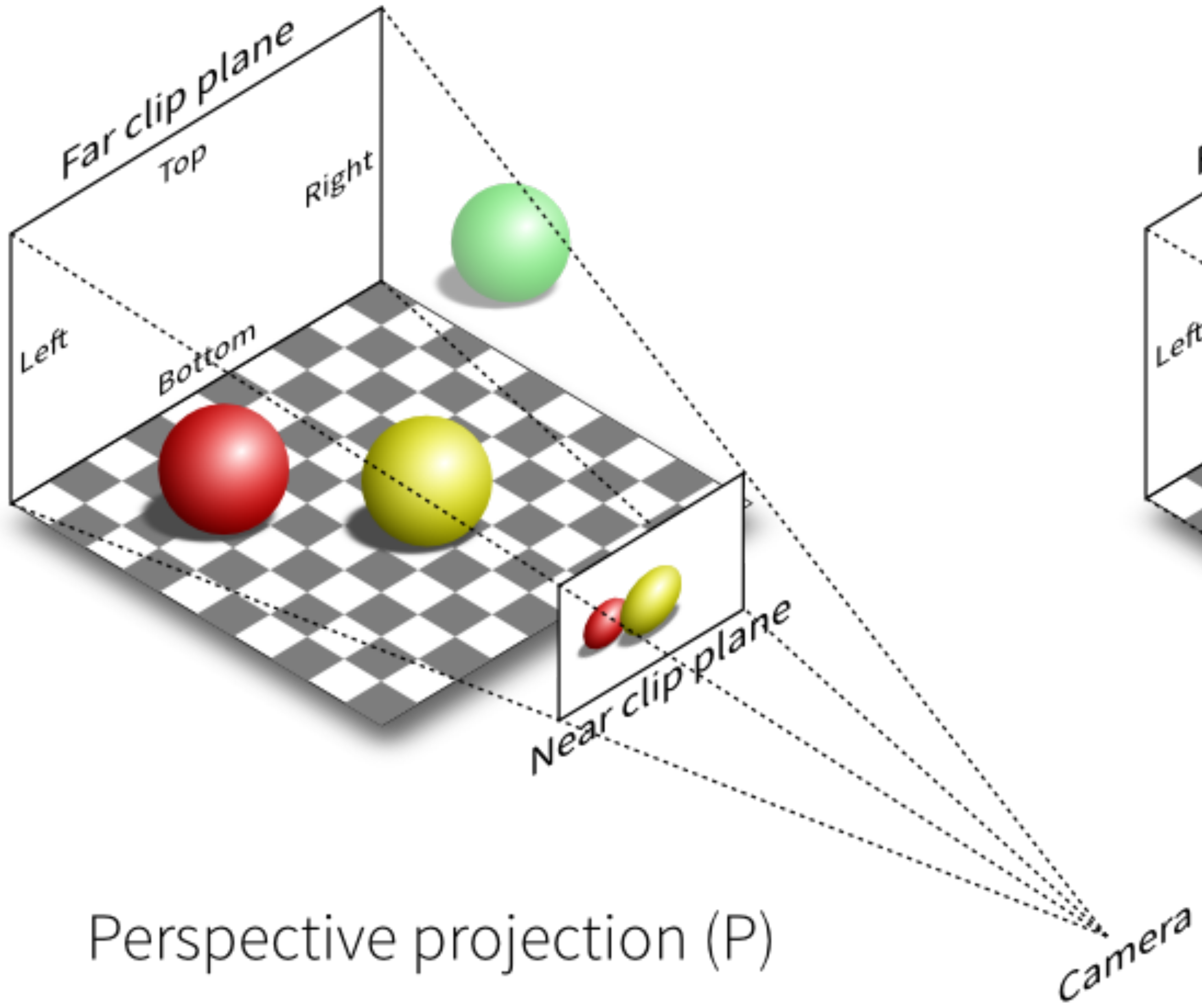
$$\begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation around an arbitrary axis **k**.

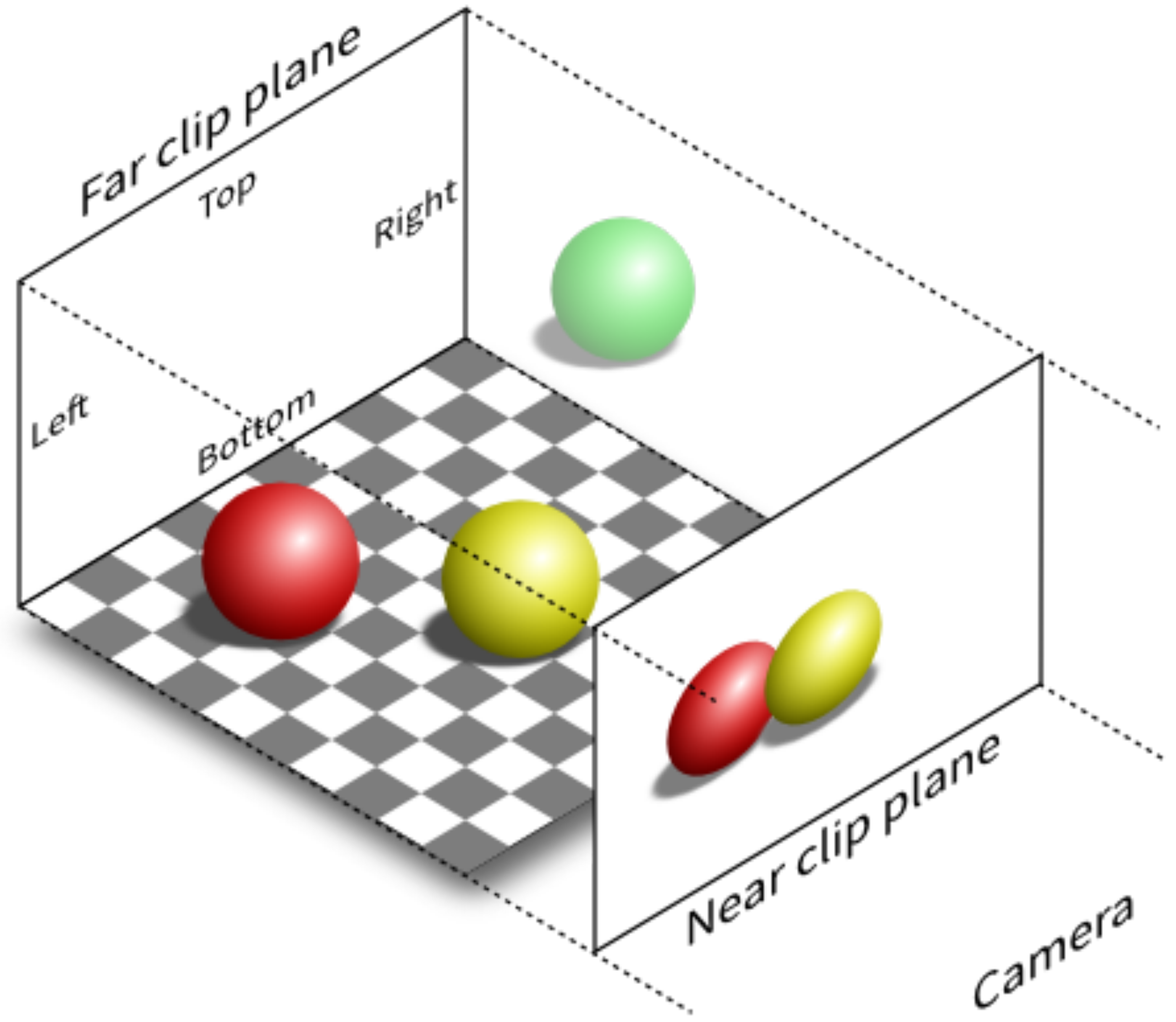


$$\begin{bmatrix} k_x^2 v + c & k_x k_y v - k_z s & k_x k_z v + k_y s \\ k_y k_x v + k_z s & k_y^2 v + c & k_y k_z v - k_x s \\ k_z k_x v - k_y s & k_z k_y v + k_x s & k_z^2 v + c \end{bmatrix},$$

Projection



Perspective projection (P)



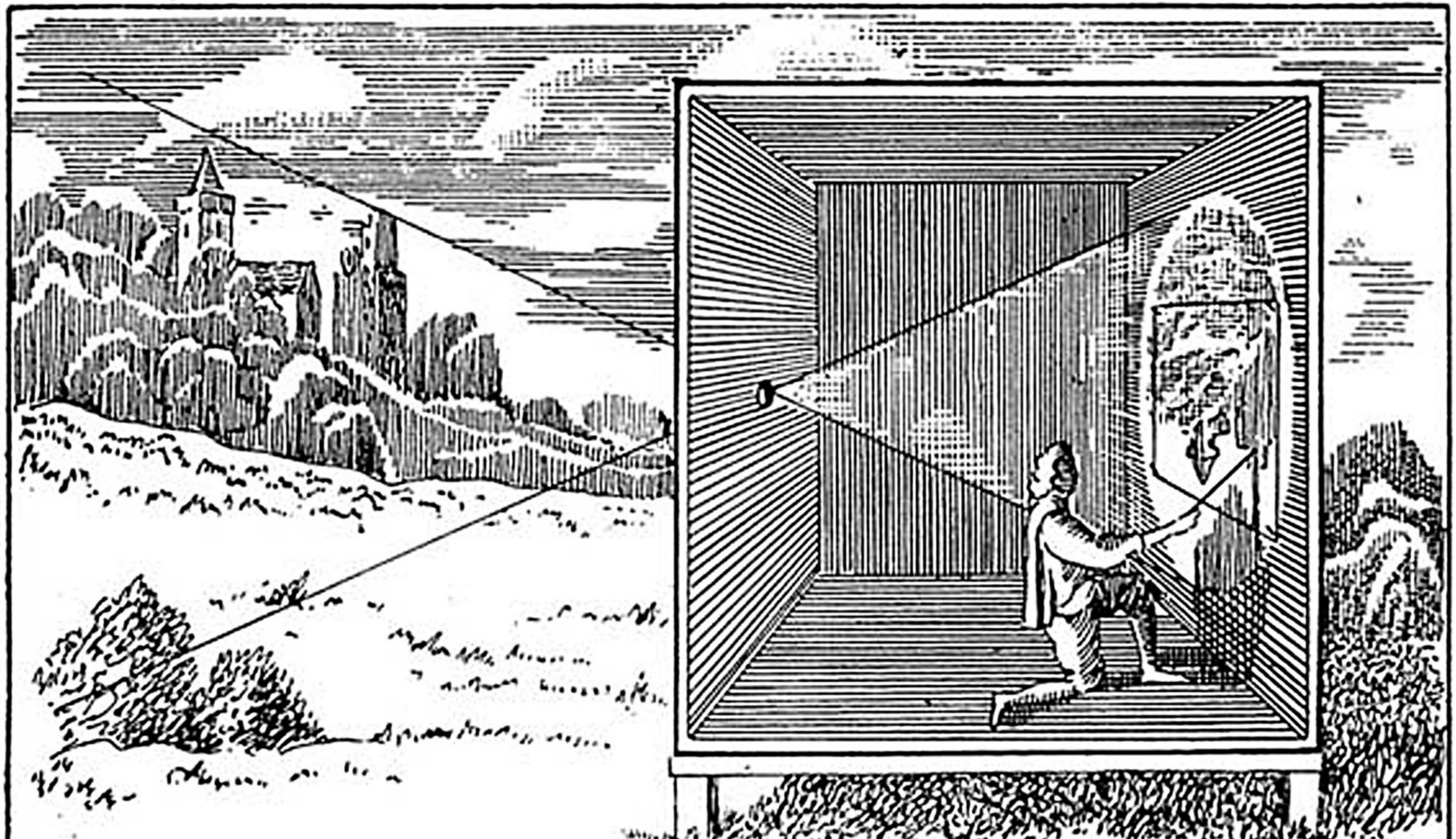
Orthographic projection (O)

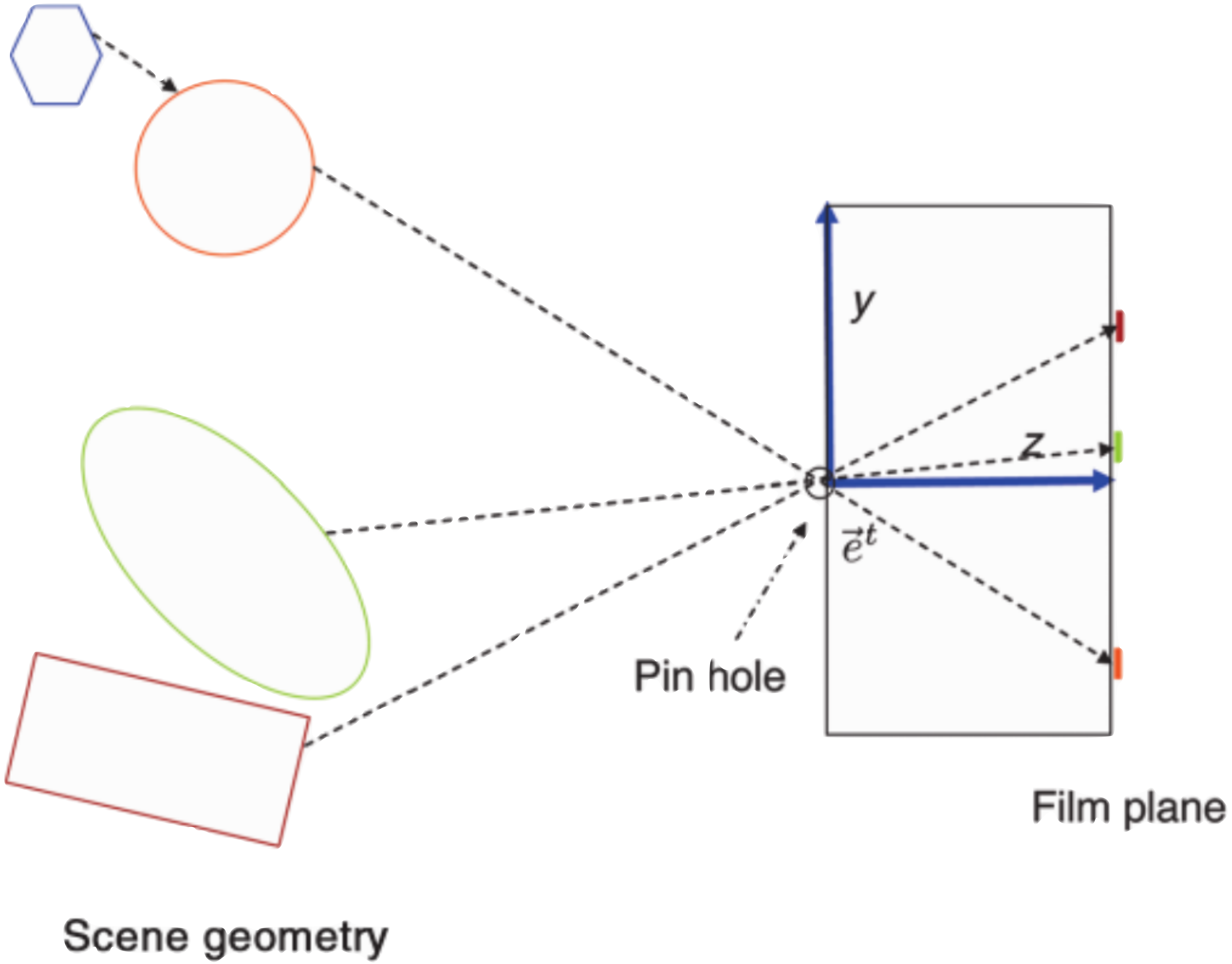
vertex.glsl

```glsl
attribute vec4 position;
attribute vec4 color;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

varying vec4 varyingColor;

void main() {
 varyingColor = color;
 gl_Position = projectionMatrix * modelViewMatrix * position;
}
```

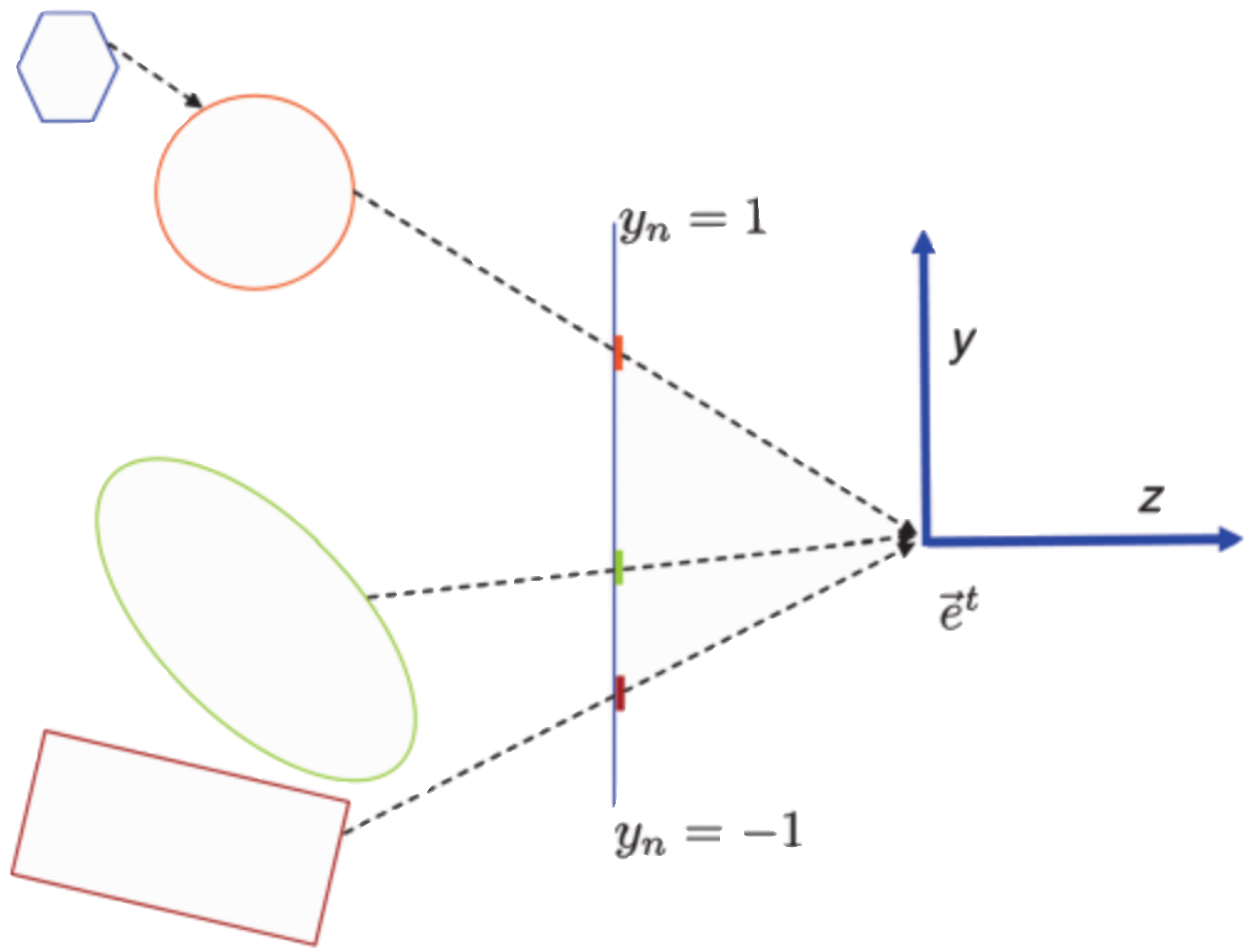
fragment.glsl

```glsl
varying vec4 varyingColor;

void main() {
    gl_FragColor = varyingColor;
}
```

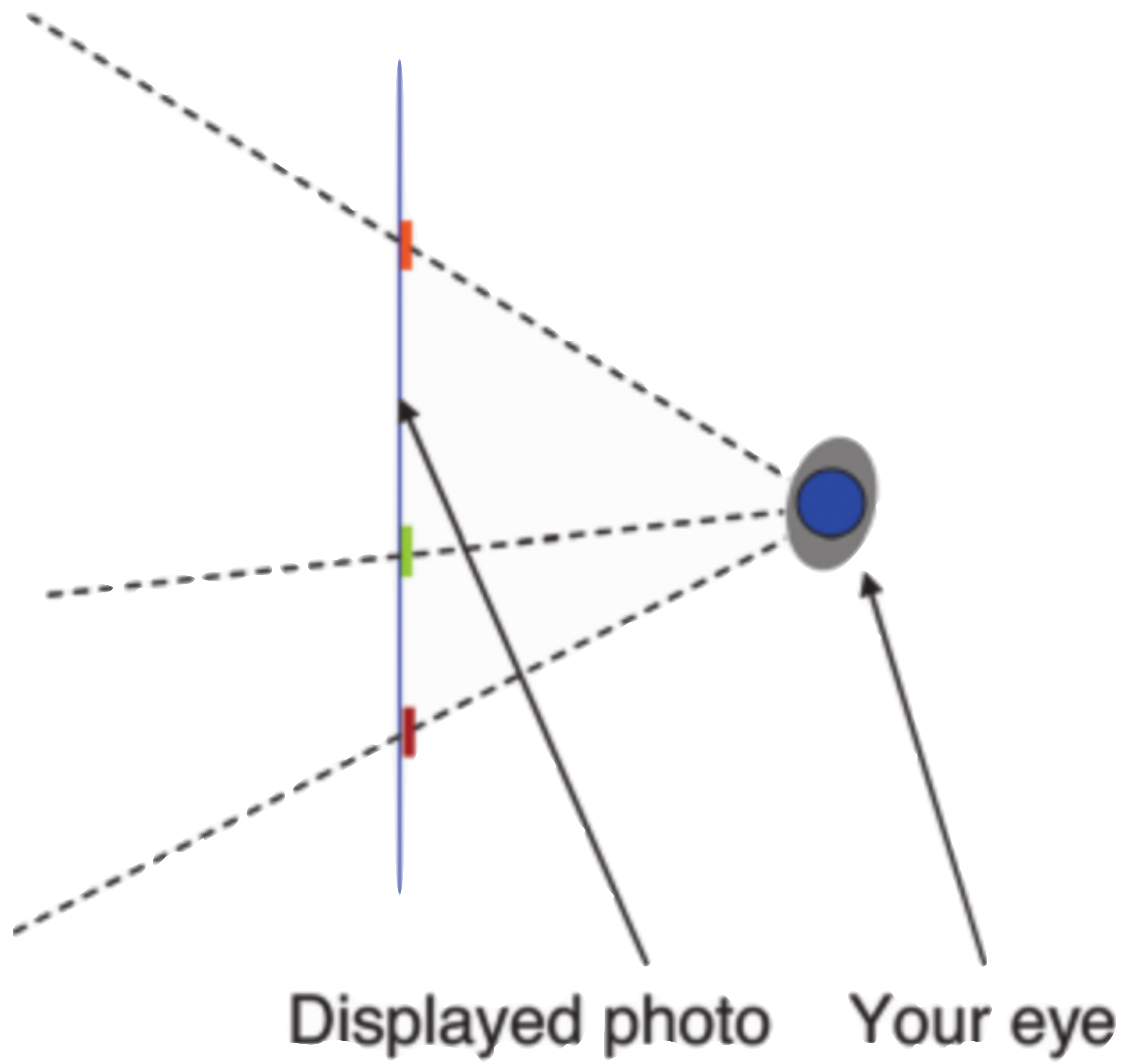




y
z
$\vec{e}^t$
Pin hole
Film plane
Scene geometry



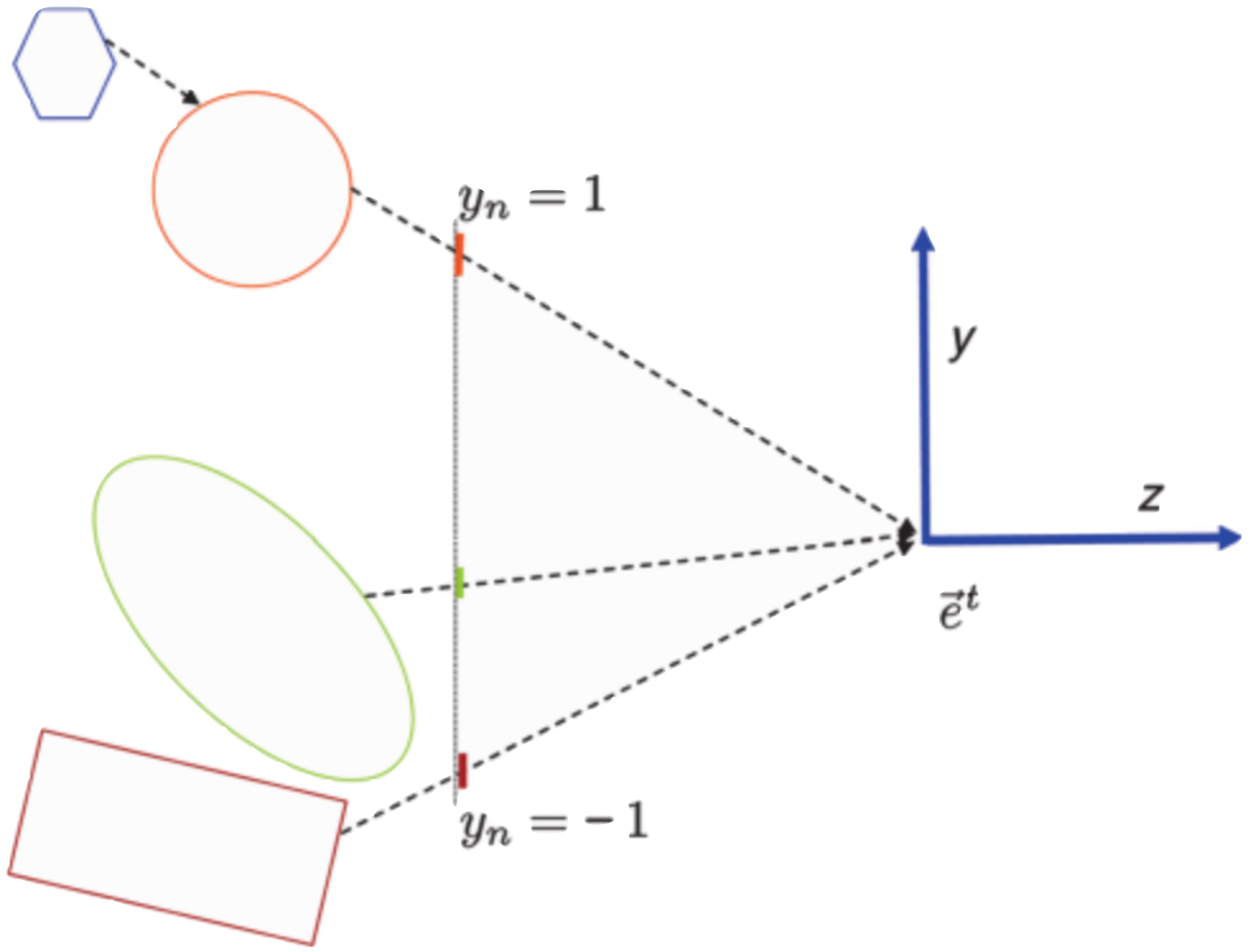
Scene geometry Film plane at z=-1


Displayed photo
Your eye

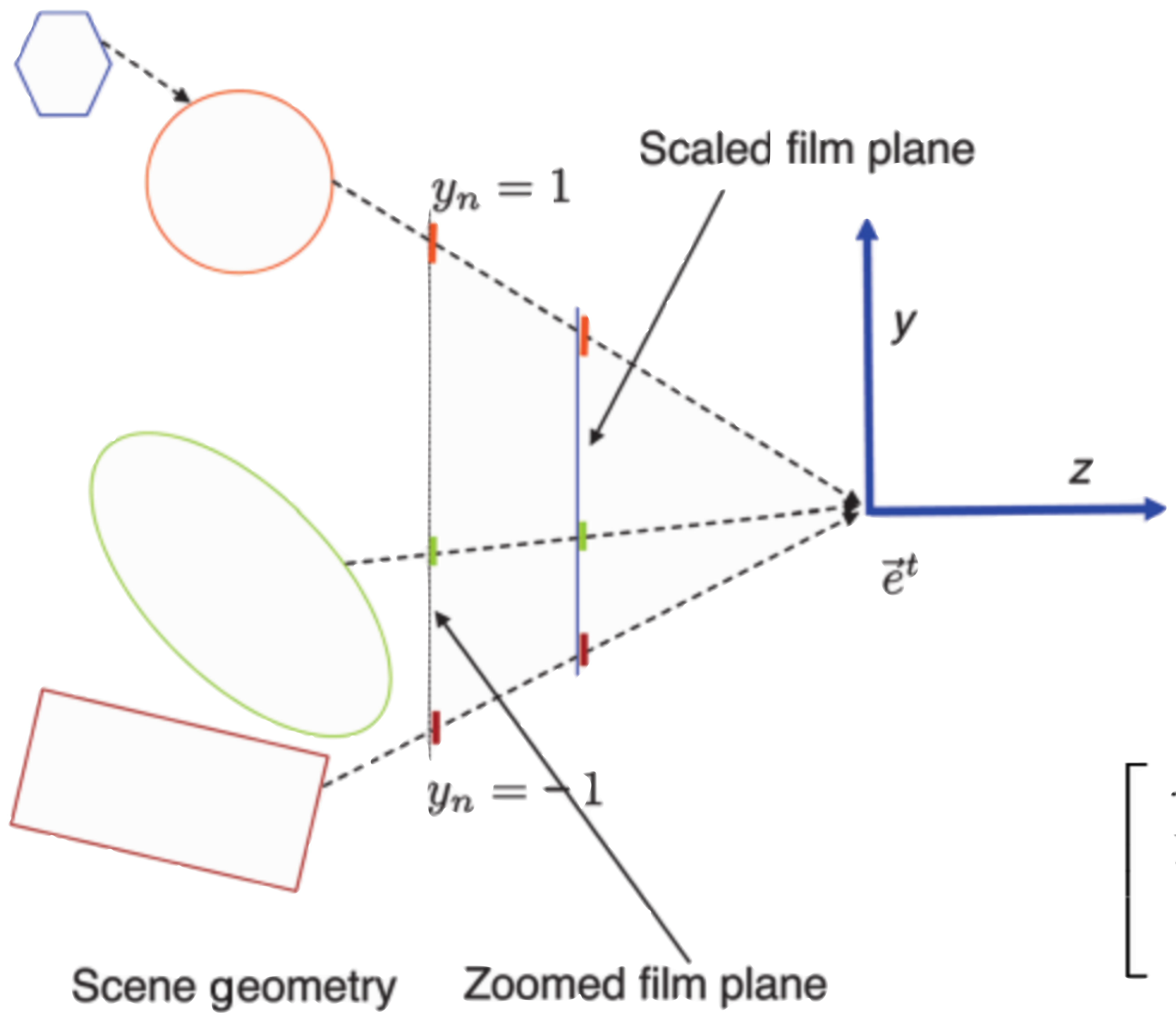
$$x_n = -\frac{x_e}{z_e}$$

$$y_n = -\frac{y_e}{z_e}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ - & - & - & - \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ - \\ w_c \end{bmatrix} = \begin{bmatrix} x_n w_n \\ y_n w_n \\ - \\ w_n \end{bmatrix},$$



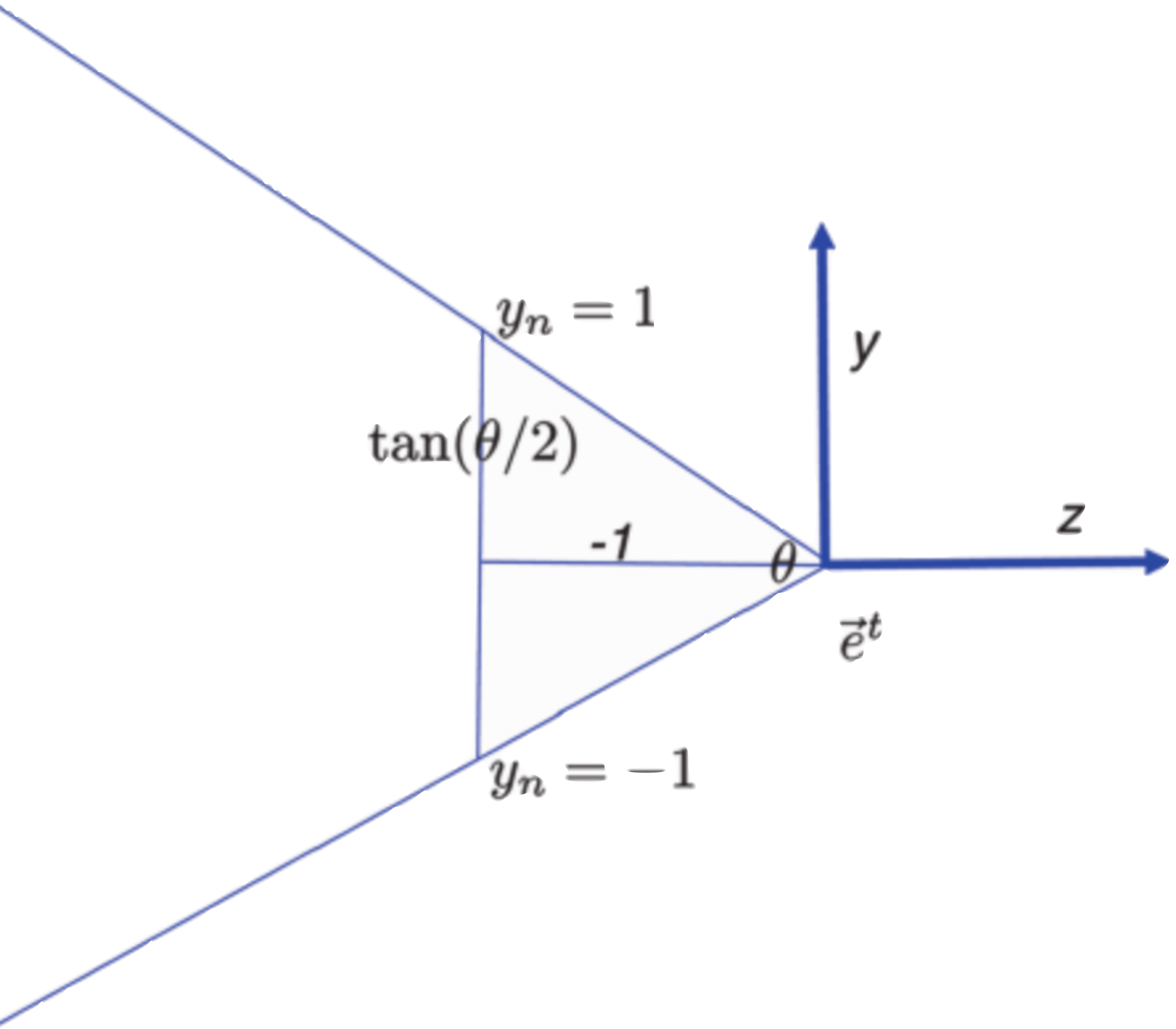
$y_n = 1$
y
z
$\vec{e}^t$
$y_n = -1$



$$x_n = \frac{x_e n}{z_e}$$

$$y_n = \frac{y_e n}{z_e}.$$

$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ - \\ w_n \end{bmatrix} = \begin{bmatrix} -n & 0 & 0 & 0 \\ 0 & -n & 0 & 0 \\ - & - & - & - \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}.$$



$$\begin{bmatrix} \frac{1}{\tan(\frac{\theta}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\theta}{2})} & 0 & 0 \\ - & - & - & - \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

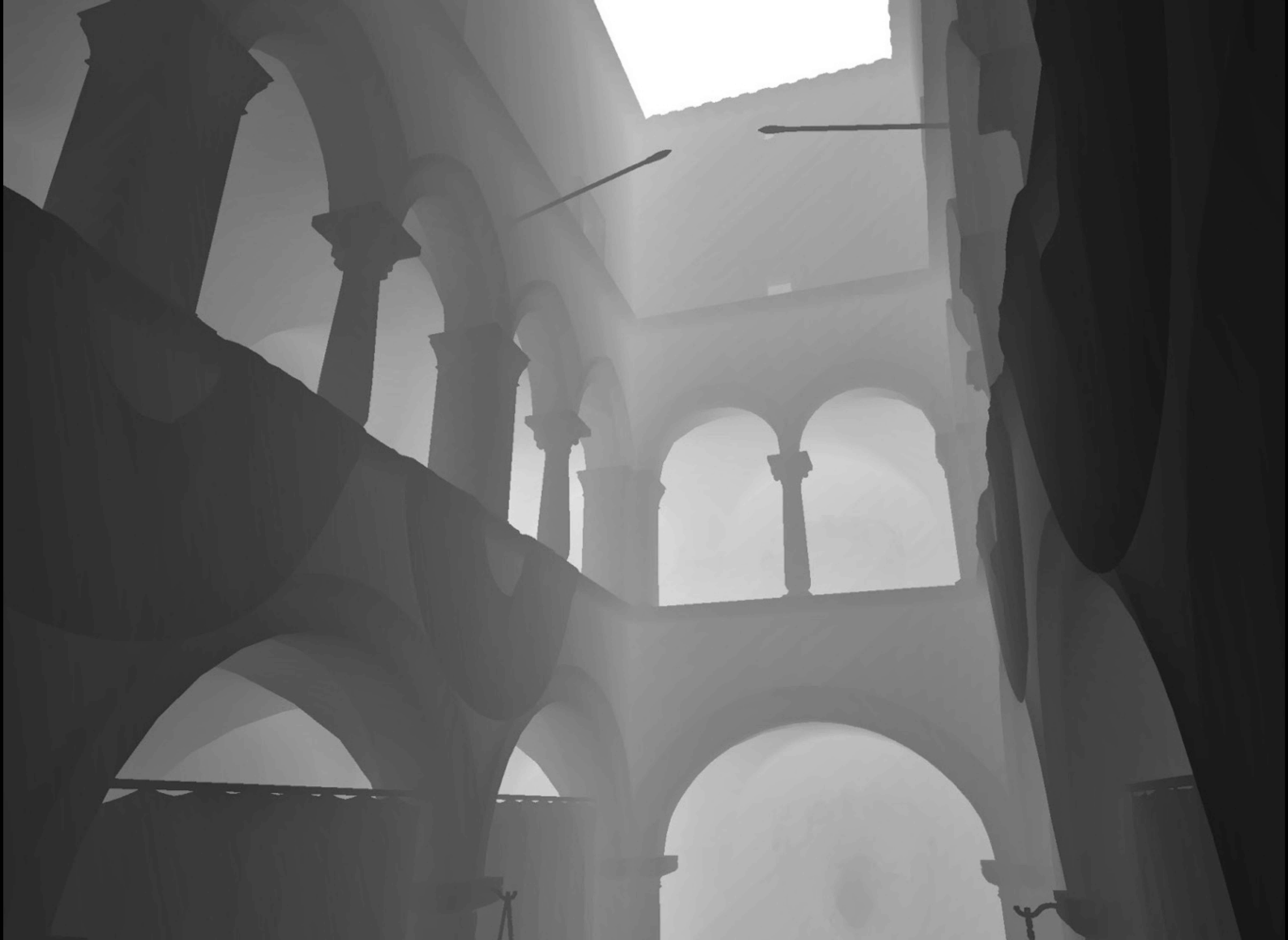
$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ - \\ w_n \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ - & - & - & - \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

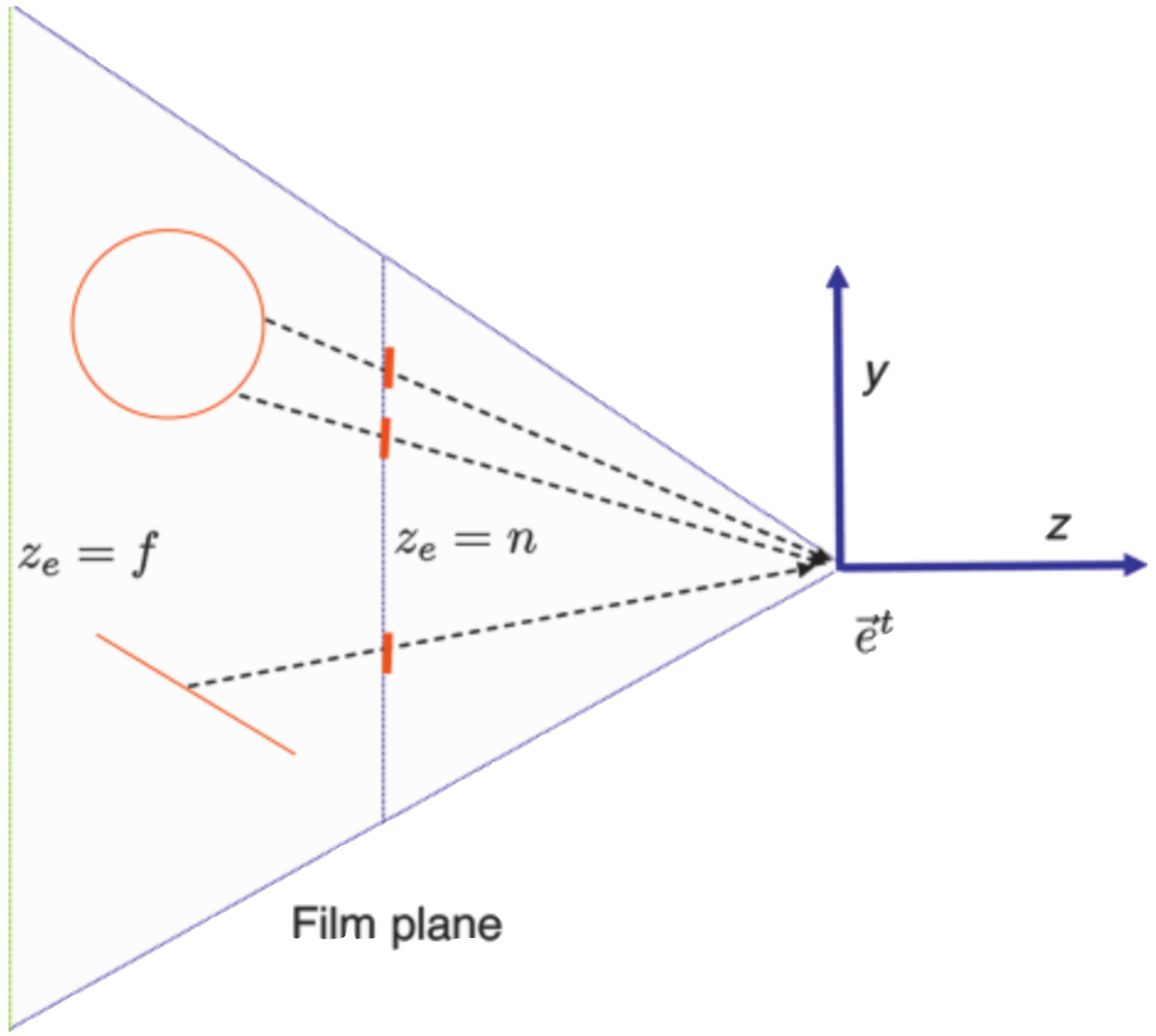
$$= \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ - & - & - & - \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}.$$

$$\begin{bmatrix} \frac{1}{a\ \tan(\frac{\theta}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\theta}{2})} & 0 & 0 \\ \text{—} & \text{—} & \text{—} & \text{—} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Depth

Depth Buffer




y
z
$z_e = f$
$z_e = n$
$\vec{e}^t$
Film plane

$$\begin{bmatrix} \frac{1}{a\,\tan(\frac{\theta}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\theta}{2})} & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Enabling depth testing in OpenGL

```
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    // ..
}

void init() {
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);
    // ..
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    // ..

}
```

Assignment 2

- Render a simple 3D scene using cubes.

- At least 3 objects must be in a hierarchy.