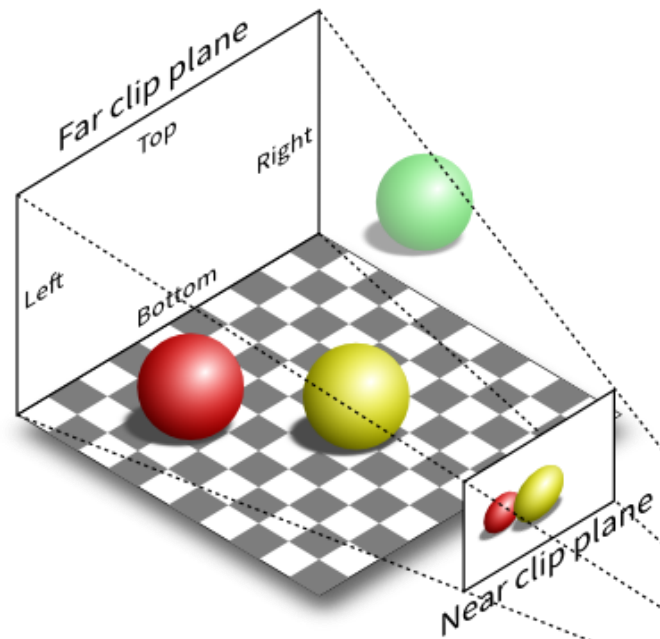
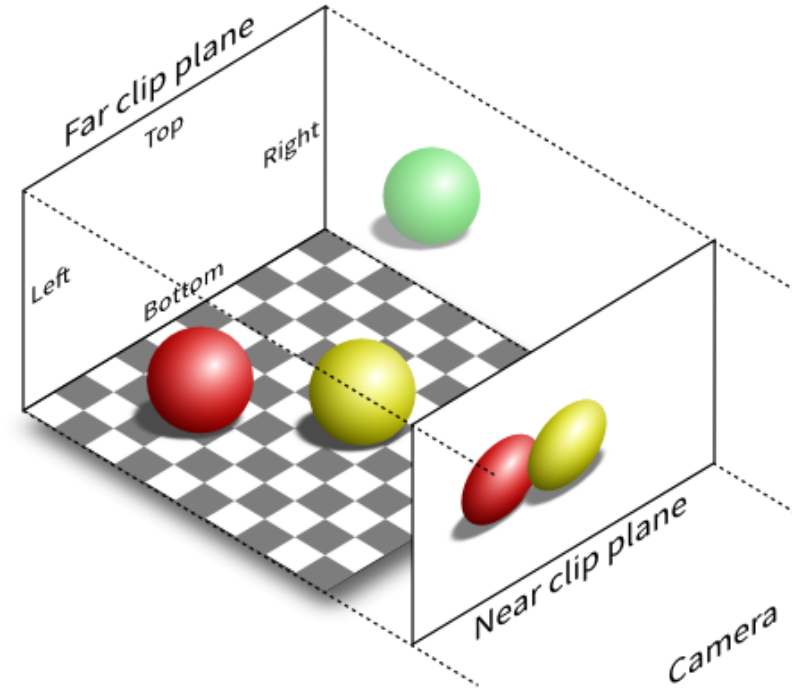


Auxiliar Proyecciones

Matriz de Proyección

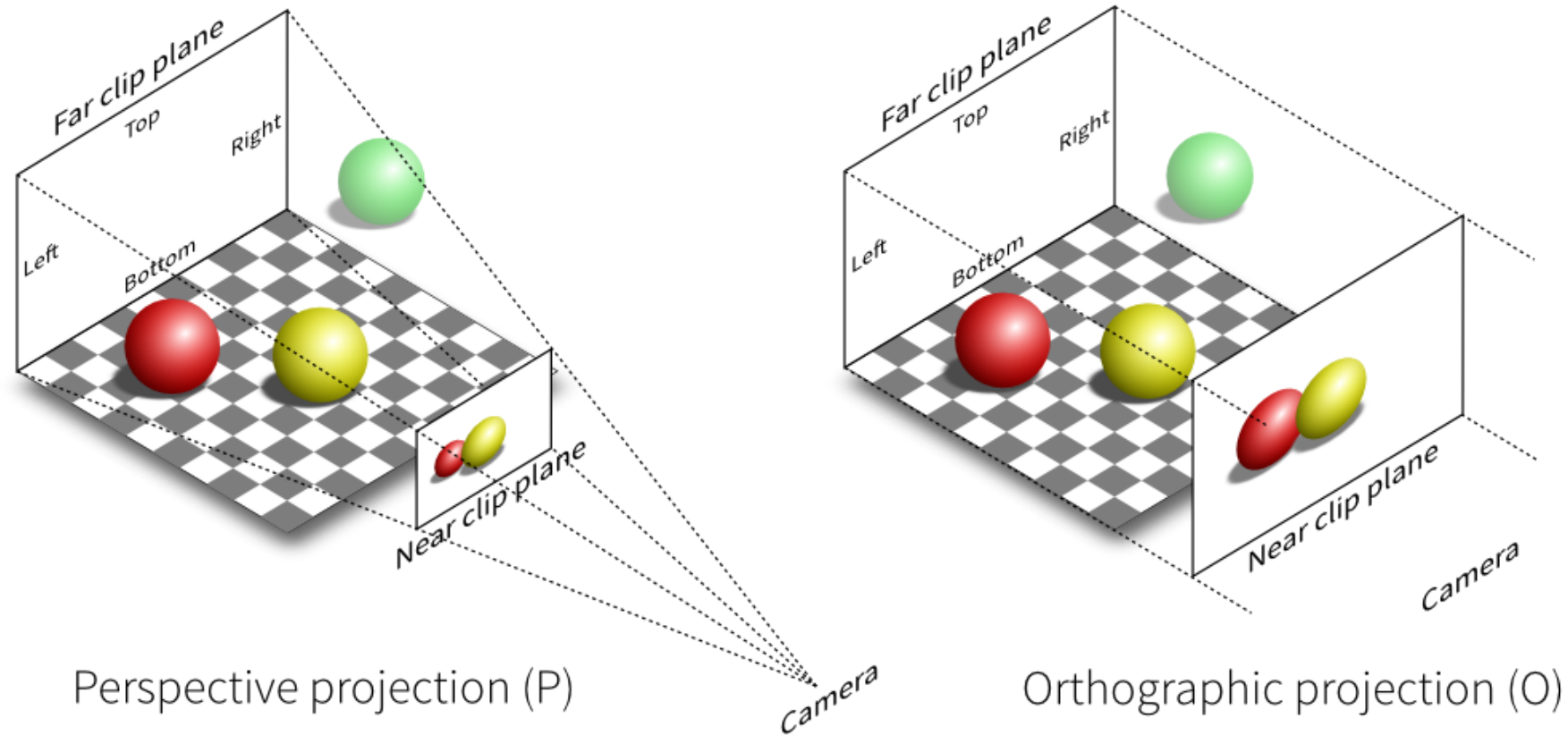


Perspective projection (P)



Orthographic projection (O)

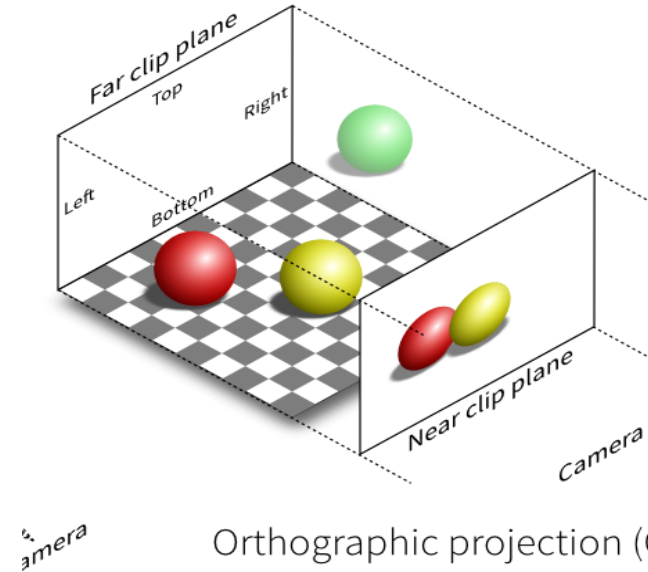
Matriz de Proyección



Matriz de Proyección Ortográfica

La diferencia se establece cambiando la matriz de proyección.

Revisar **tr.ortho(args)**



```
def ortho(left, right, bottom, top, near, far):  
    r_l = right - left  
    t_b = top - bottom  
    f_n = far - near
```

Matriz de Proyección Ortográfica

Hablemos de la matemática: requiere definir un left, right, bottom, top, near y far.

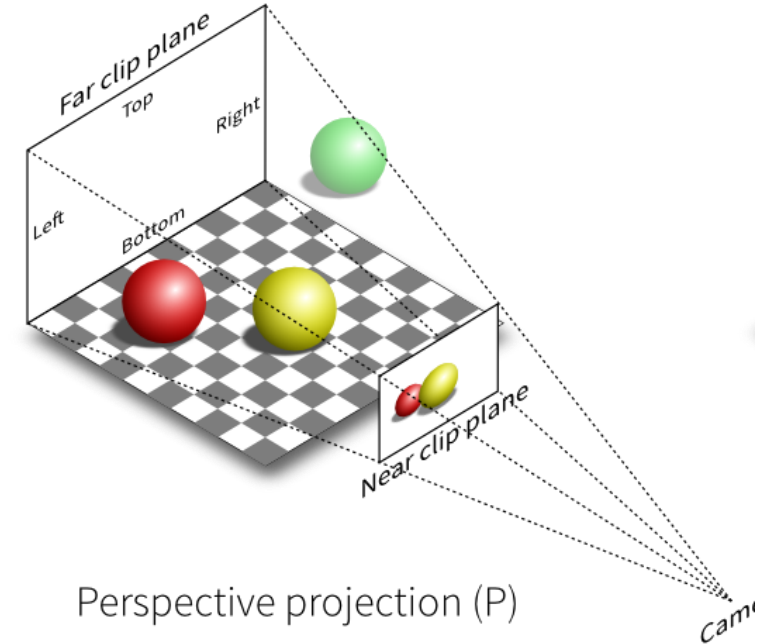
$$M_{Ortográfica} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
def ortho(left, right, bottom, top, near, far):  
    r_l = right - left  
    t_b = top - bottom  
    f_n = far - near
```

Matriz de Proyección en perspectiva

La diferencia se establece cambiando la matriz de proyección.

Revisar `tr.perspective(fovy, aspecto, near, far)`



Perspective projection (P)

```
def perspective(fovy, aspect, near, far):  
    halfHeight = np.tan(np.pi * fovy / 360) * near  
    halfWidth = halfHeight * aspect  
    return frustum(-halfWidth, halfWidth, -halfHeight, halfHeight, near, far)
```

Matriz de Proyección en perspectiva

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

```
def perspective(fovy, aspect, near, far):  
    halfHeight = np.tan(np.pi * fovy / 360) * near  
    halfWidth = halfHeight * aspect  
    return frustum(-halfWidth, halfWidth, -halfHeight, halfHeight, near, far)
```

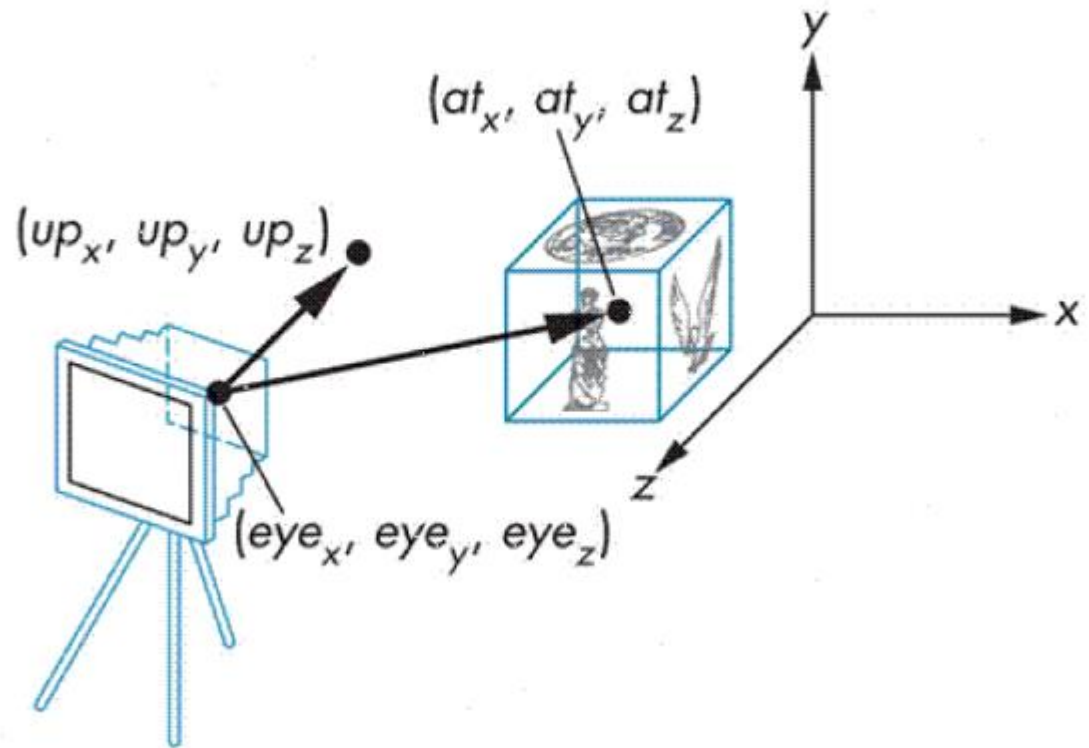
Matriz de Vista

- Importante, conocer:

\vec{up} = Vector que ve arriba

\vec{at} = Hacia donde apunta la cámara

\vec{eye} = Posición cámara



La matriz de vista es la siguiente:

$$M_{view} = \begin{bmatrix} s_x & s_y & s_z & -e \cdot s \\ u'_x & u'_y & u'_z & -e \cdot u' \\ -f_x & -f_y & -f_z & e \cdot f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde:

f : forward vector = $\frac{a-e}{||a-e||}$

s : sideways vector = $f \times u$

u' : vector up redefinido a partir de $s \times f$. Esto se realiza para asegurar ortonormalidad.

Matriz de Vista: Indique vectores de view



Matriz de Vista: Indique vectores de view



Indique los vectores Up, eye y at.



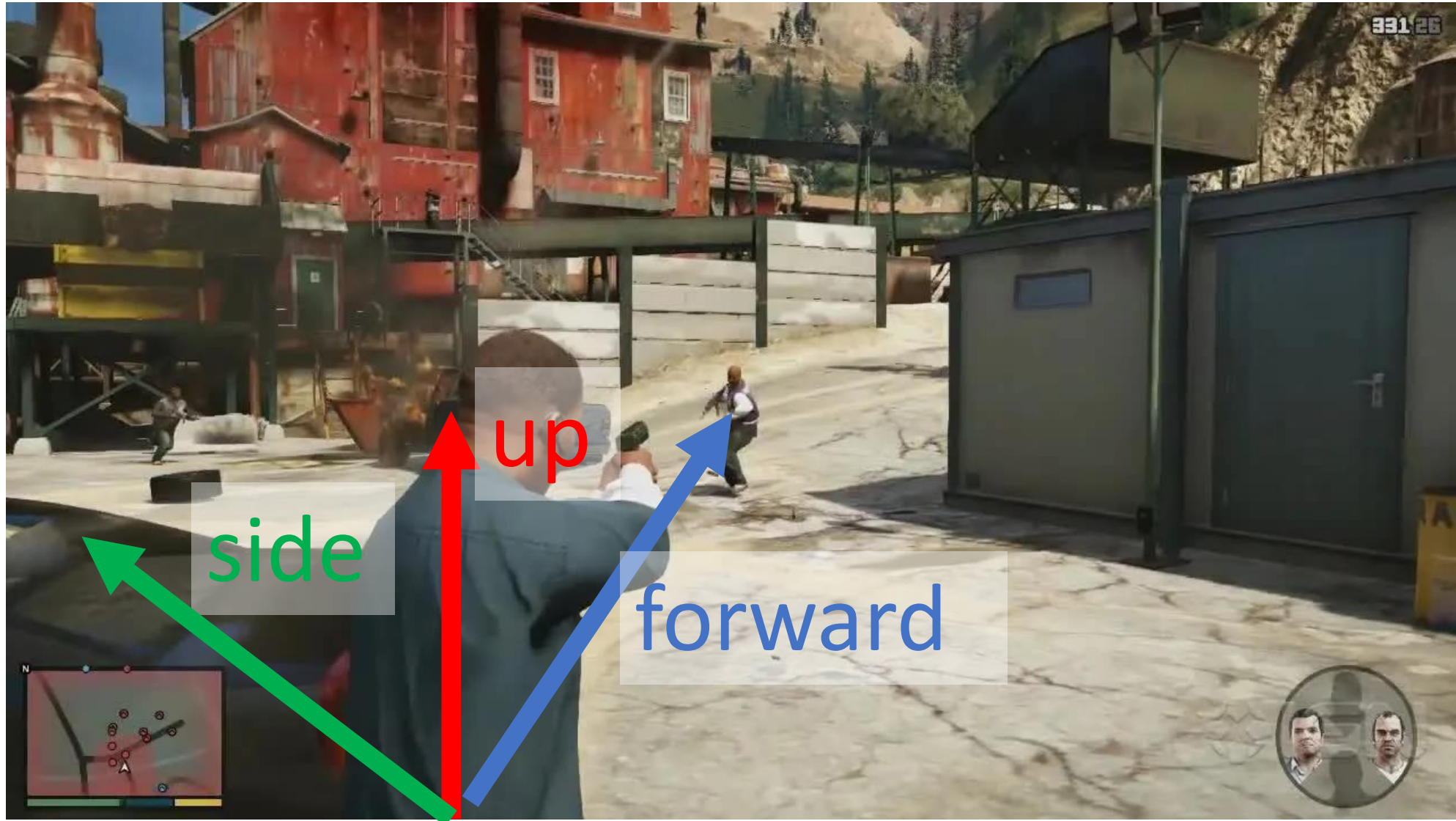
Indique los vectores Up, eye y at.



Indique los forward, side y up'.



Indique los forward, side y up'.



Indique qué shaders se usa para los elementos



Indique qué shaders se usa para los elementos



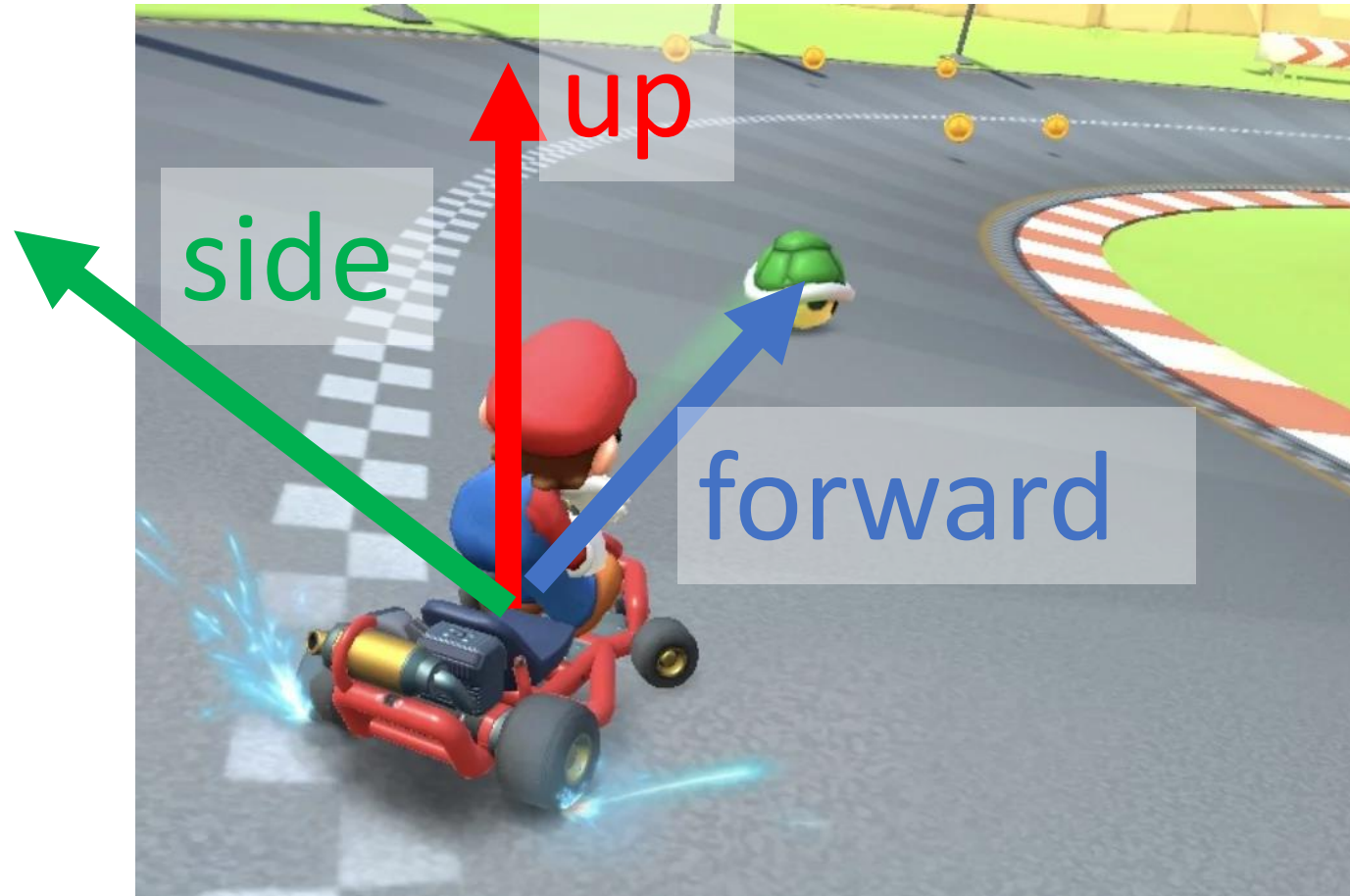
Casi todo:
shader 3D con
iluminación

Shader
2D

Indique el vector up, forward y side de Mario



Indique el vector up, forward y side de Mario



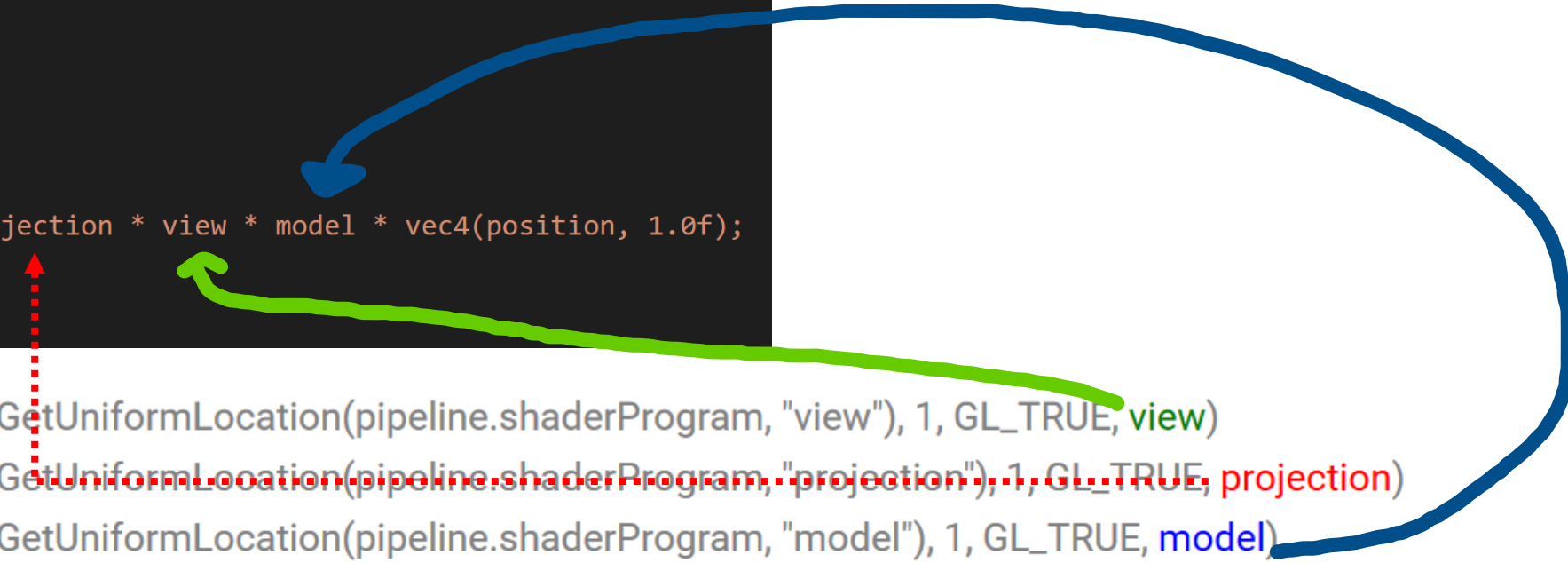
Seteando matrices en el shader

```
vertex_shader = """
    #version 130

    uniform mat4 projection;
    uniform mat4 view;
    uniform mat4 model;

    in vec3 position;
    in vec3 color;

    out vec3 newColor;
    void main()
    {
        gl_Position = projection * view * model * vec4(position, 1.0f);
        newColor = color;
    }
    """
```



```
glUniformMatrix4fv(glGetUniformLocation(pipeline.shaderProgram, "view"), 1, GL_TRUE, view)
glUniformMatrix4fv(glGetUniformLocation(pipeline.shaderProgram, "projection"), 1, GL_TRUE, projection)
glUniformMatrix4fv(glGetUniformLocation(pipeline.shaderProgram, "model"), 1, GL_TRUE, model)
```

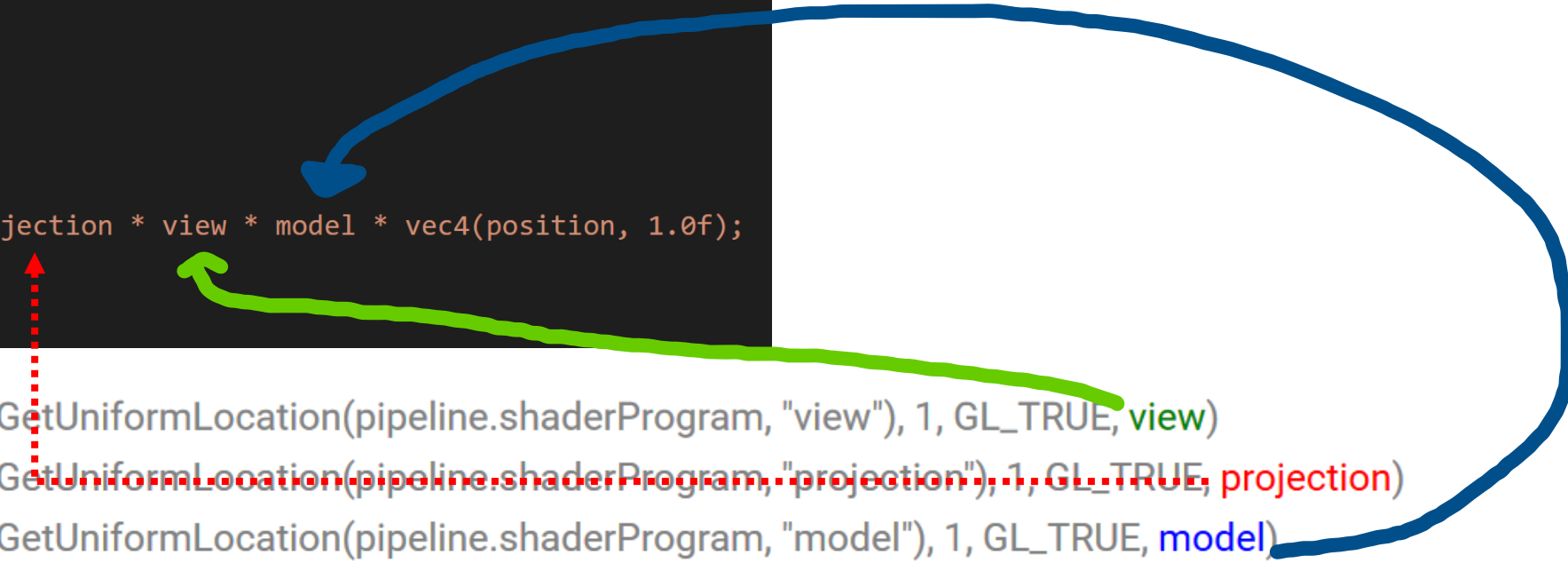
Seteando matrices en el shader

```
vertex_shader = """
    #version 130

    uniform mat4 projection;
    uniform mat4 view;
    uniform mat4 model;

    in vec3 position;
    in vec3 color;

    out vec3 newColor;
    void main()
    {
        gl_Position = projection * view * model * vec4(position, 1.0f);
        newColor = color;
    }
    """
```



```
glUniformMatrix4fv(glGetUniformLocation(pipeline.shaderProgram, "view"), 1, GL_TRUE, view)
glUniformMatrix4fv(glGetUniformLocation(pipeline.shaderProgram, "projection"), 1, GL_TRUE, projection)
glUniformMatrix4fv(glGetUniformLocation(pipeline.shaderProgram, "model"), 1, GL_TRUE, model)
```

Mallas de polígonos:

- Archivos OBJ
- Archivos OFF

Mallas de polígonos:

- Archivos OBJ:

Definen vértices, texturas y normal

Se guarda el índice de cada línea

Y luego se forman caras (faces) con estos índices

Las caras indican
vértice/textura/normal

f v1/vt1/vn1 v2/vt2/vn2

```
v 0.437500 0.164062 0.765625
v -0.437500 0.164062 0.765625
vt 0.498072 0.552315
vt 0.264218 0.550140
vn 0.6650 -0.2008 0.7194
vn -0.6650 -0.2008 0.7194
vn 0.8294 -0.3036 0.4689
```

```
f 47/1/1 1/2/1 3/3/1 45/4/1
f 4/5/2 2/6/2 48/7/2 46/8/2
```

Mallas de polígonos:

Pueden usar el `obj_reader.py` para leer archivos `.OBJ` o `.OFF`

Pueden descargar modelos 3D de `.OBJ` o `.OFF` en internet. Úsenlos para que sus tareas se vean más elaboradas y más bonitas con poco esfuerzo 😊

Comencemos la clase...

Pregunta 1: Práctica de astronauta

- H. Simpson está practicando para ser astronauta. Tenemos un brazo giratorio. Describa los vectores necesarios para lograr una simulación así en OpenGL vista desde arriba.

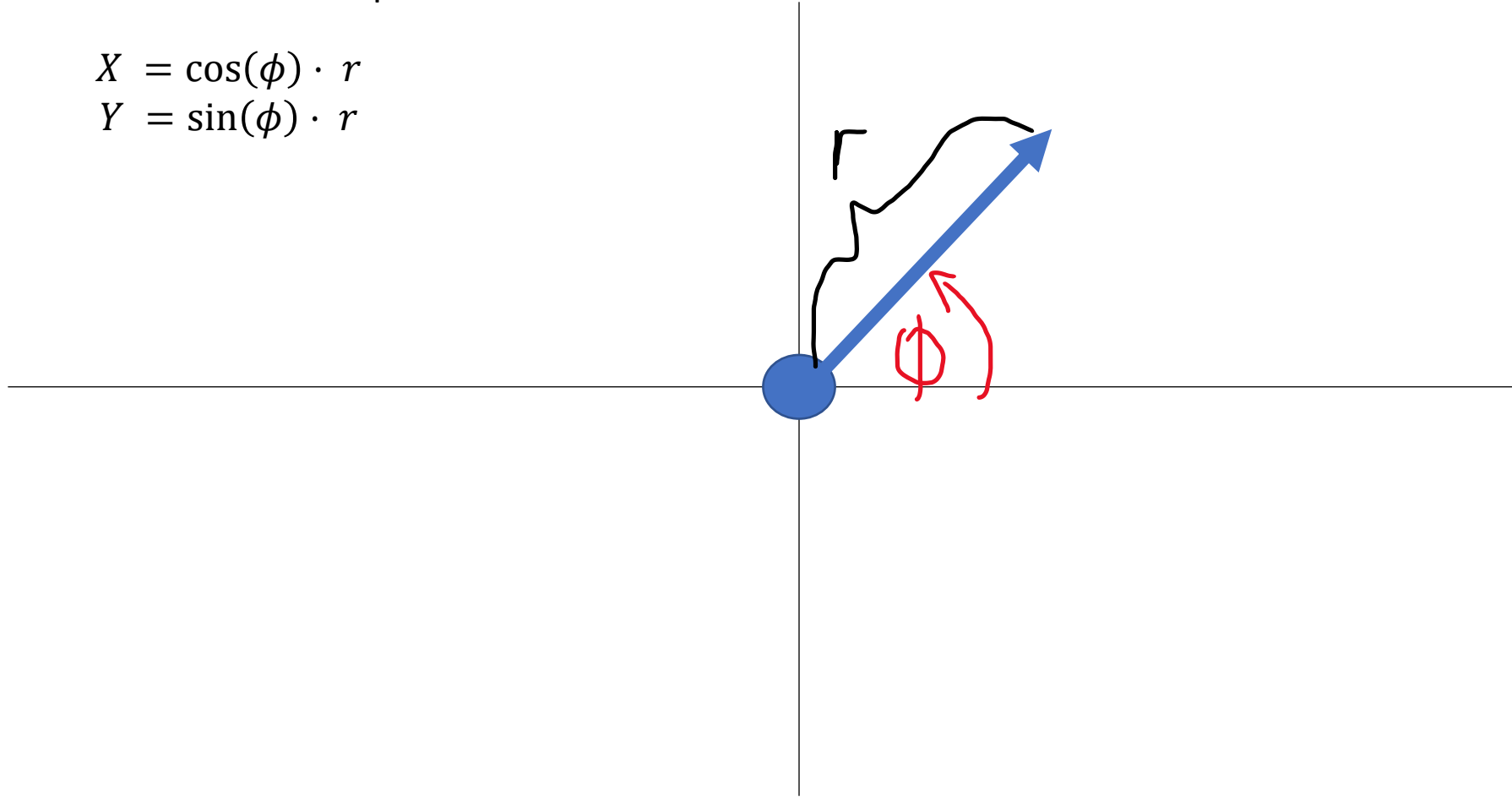


Pregunta 1: Práctica de astronauta

Usamos coordenadas polares

$$X = \cos(\phi) \cdot r$$

$$Y = \sin(\phi) \cdot r$$



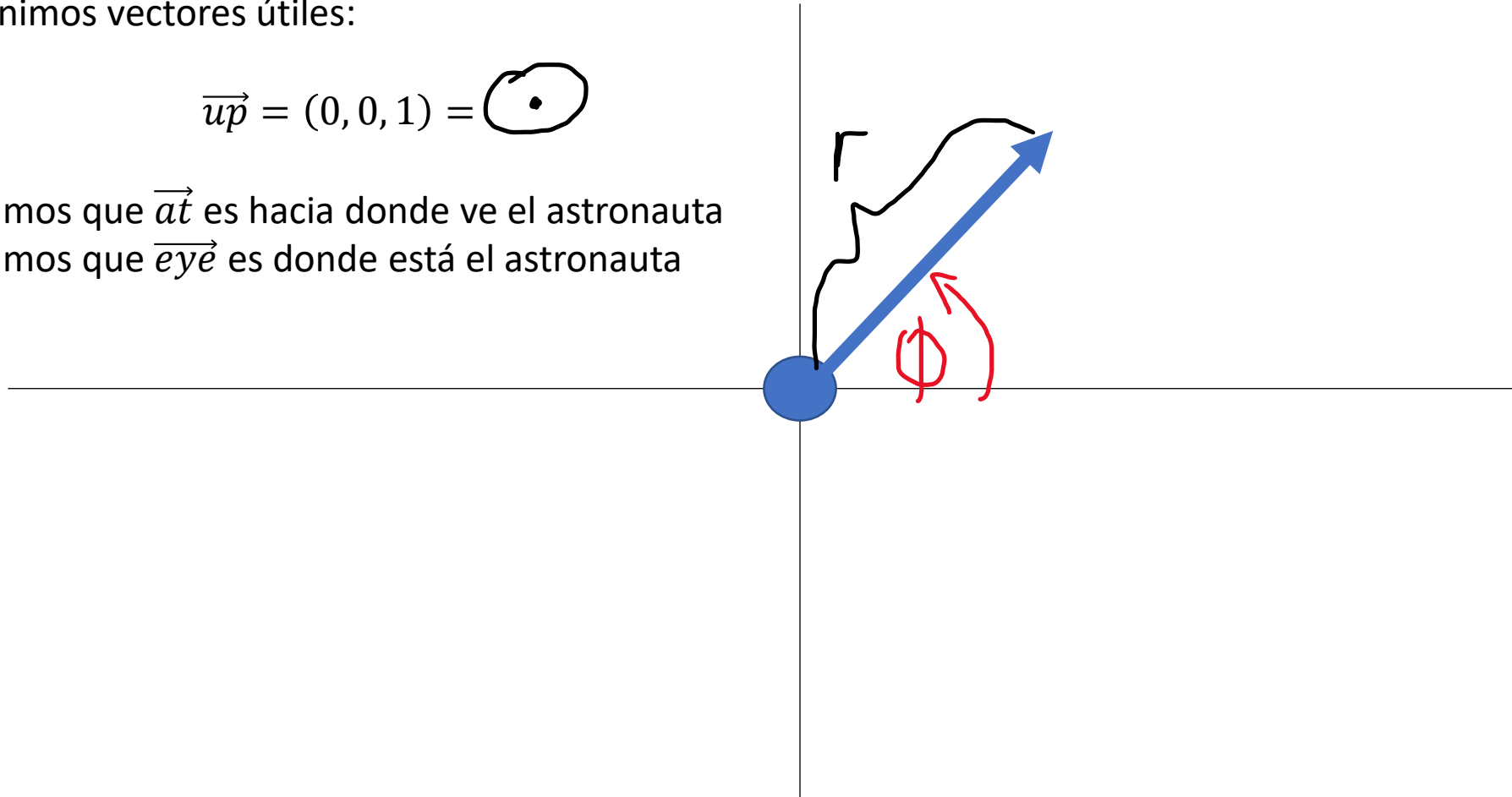
Pregunta 1: Práctica de astronauta

Definimos vectores útiles:

$$\vec{up} = (0, 0, 1) = \text{un círculo con un punto en el centro}$$

Digamos que \vec{at} es hacia donde ve el astronauta

Digamos que \vec{eye} es donde está el astronauta



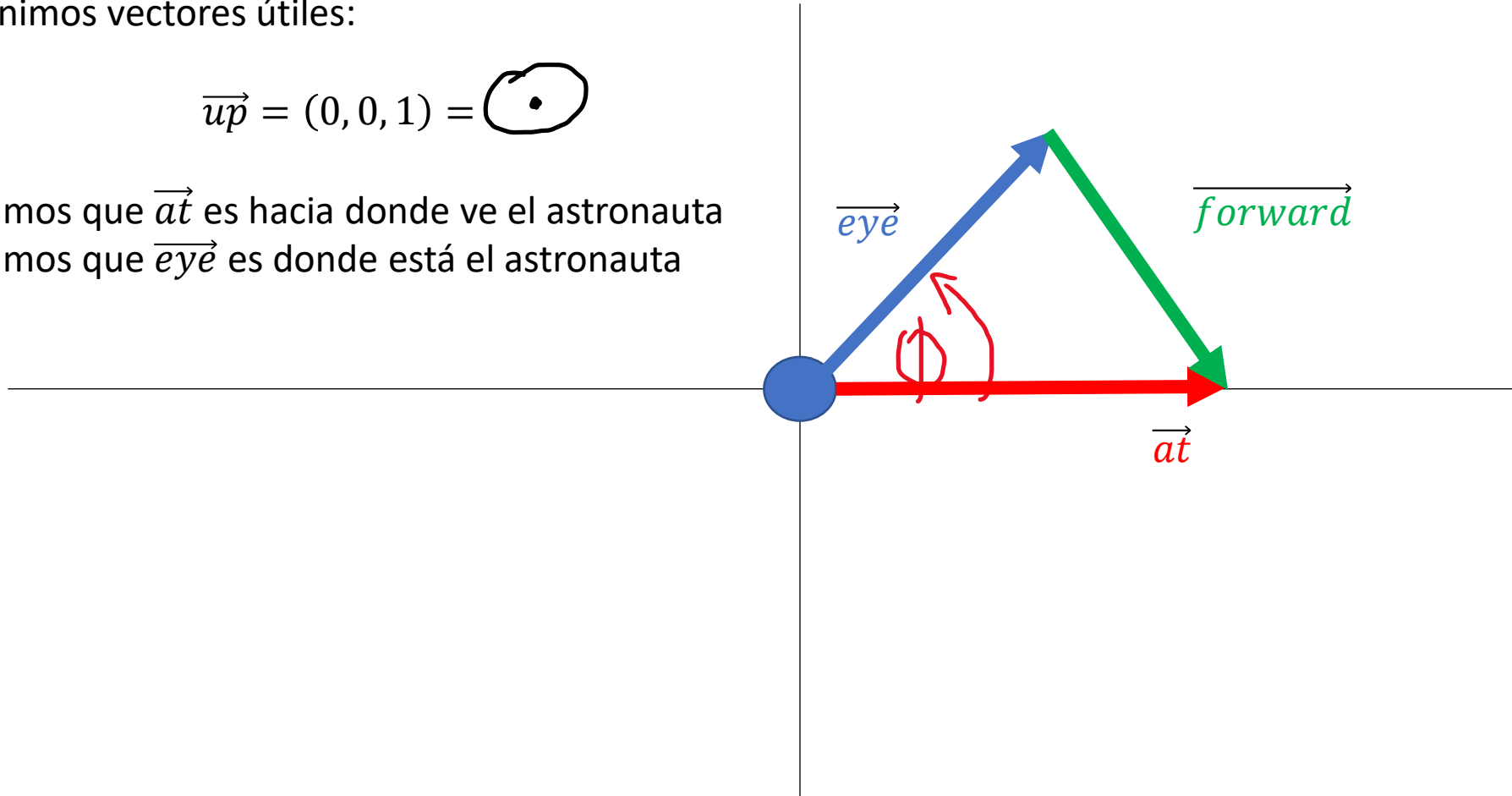
Pregunta 1: Práctica de astronauta

Definimos vectores útiles:

$$\vec{up} = (0, 0, 1) = \text{[sketch of a dot inside a circle]}$$

Digamos que \vec{at} es hacia donde ve el astronauta

Digamos que \vec{eye} es donde está el astronauta



Pregunta 2:

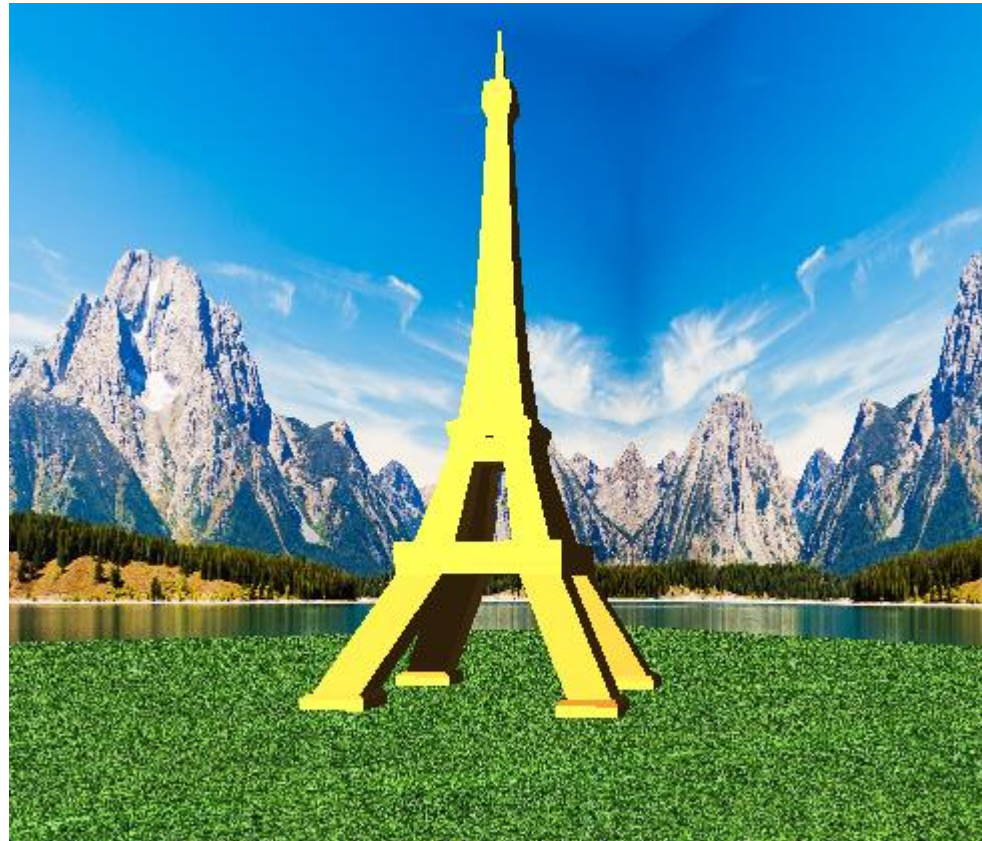
- Partiendo con el archivo Pregunta1.py, encuentre la matriz de vista en el código y modifique el vector up de distintas formas. Trate de predecir qué ocurre cuando el vector up es:
- $[0.0, 0.0, -1.0]$
- $[1.0, 1.0, 0]$

Pregunta 3:

- Modifique el archivo para que al usar las teclas A y D, gire la cabeza hacia la izquierda o derecha respectivamente.
- Si se apreta W, se mueve hacia delante.
- Si se apreta S, se retrocede.

Pregunta 4:

- Agregue el Modelo 3D de la Torre Eiffel al centro de la SkyBox.



¿Preguntas?

