Auxiliar 4 – Modelado

Modelos y Grafos

Pablo Pizarro R.

Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

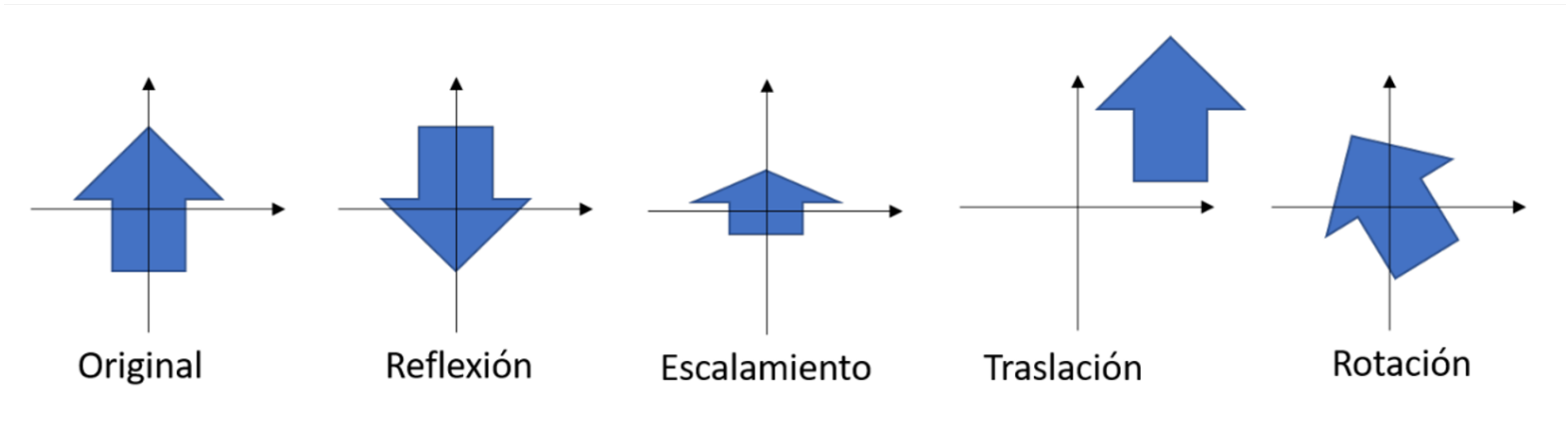
3 de abril de 2022

¿Qué veremos hoy?

1. Repaso de transformaciones
2. Implementación
3. Modelación de objetos
4. Grafo de escena
5. Ejercicio

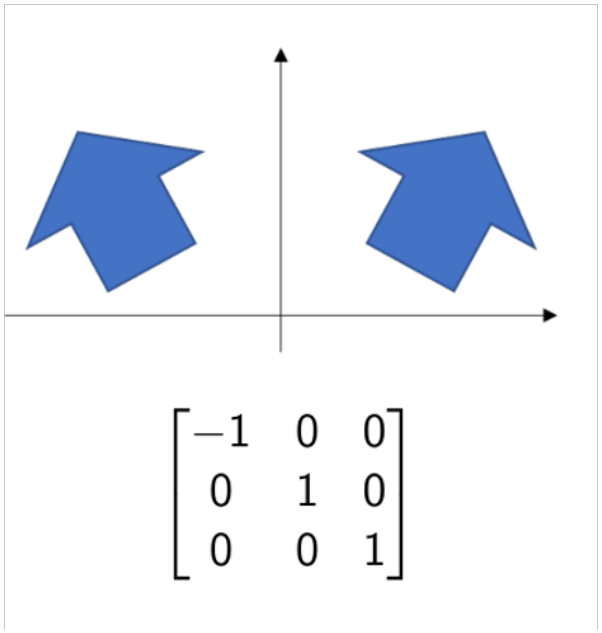
Las transformaciones

- Para operar sobre modelos en OpenGL tenemos múltiples formas de transformarlos geométricamente:

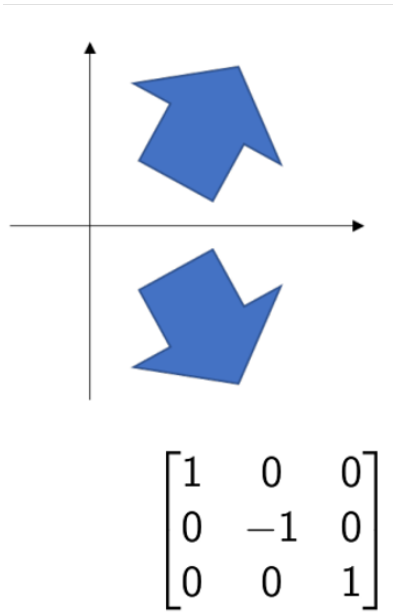


Las transformaciones – Reflexión

- Todas las transformaciones geométricas se pueden representar a través de una matriz de transformación, que modifica el estado de cada vértice del modelo.
- **Reflexión**:



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



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

Las transformaciones – Escalamiento

- **Escalamiento**: α modifica el eje x, β el y, γ el z.

$$P' = S(\alpha, \beta, \gamma)P$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Las transformaciones – Rotación

- **Rotación**: Podemos afectar cualquier dimensión de nuestro objeto.

Rotación en torno a eje Z

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotación en torno a eje Y

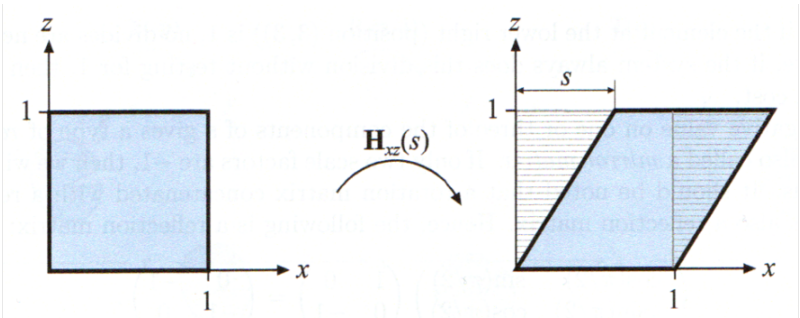
$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Rotación en torno a eje Y

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

Las transformaciones – Distorsión

- **Shearing**: Distorsión.



$$H_{xz}(s) = \begin{bmatrix} 1 & 0 & s \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Las transformaciones – Traslación

- **Traslación**: También se puede representar en forma matricial usando coordenadas homogéneas.

$$M_T = \left[\begin{array}{c|c} I & T \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Implementar las transformaciones

- Ahora que sabemos cómo se define una transformación (Matriz que opera sobre los vértices). ¿Cómo se usa en OpenGL?

Implementar las transformaciones

- Ahora que sabemos cómo se define una transformación (Matriz que opera sobre los vértices). ¿Cómo se usa en OpenGL?

  1. Definir la matriz usando numpy

Implementar las transformaciones

- Ahora que sabemos cómo se define una transformación (Matriz que opera sobre los vértices). ¿Cómo se usa en OpenGL?

  1. Definir la matriz usando numpy
  2. Con OpenGL:
     a) Se carga el modelo con OpenGL, usando el VAO
     b) Se cargan las transformaciones geométricas (matriz)

Implementar las transformaciones

- Al definir el objeto:

```
1 import grafica.basic_shapes as bs
2
3 shapeCube = bs.createRainbowCube()
4 gpuCube = es.GPUShape().initBuffers()
5 pipeline.setupVAO(gpuCube)
6 gpuCube.fillBuffers(shapeCube.vertices, shapeCube.indices,
      ↪ GL_STATIC_DRAW)
```

Implementar las transformaciones

- Al definir el objeto:

```
import grafica.basic_shapes as bs

shapeCube = bs.createRainbowCube()
gpuCube = es.GPUShape().initBuffers()
pipeline.setupVAO(gpuCube)
gpuCube.fillBuffers(shapeCube.vertices, shapeCube.indices,
    ↪ GL_STATIC_DRAW)
```

- Al dibujar el objeto (dentro del main):

```
while not glfw.window_should_close(window):
  ...
  transform = np.matmul(matriz1, matriz2)

  # Dibujamos el cubo
  glUniformMatrix4fv(glGetUniformLocation(pipeline.shaderProgram, "
    ↪ transform"), 1, GL_TRUE, transform)
  pipeline.drawCall(gpuCube)
```

Implementar las transformaciones

- ¿Qué hace `glGetUniformLocation`?
  Le dice al **shader** dónde tiene que acceder para leer la variable **transform**.

Implementar las transformaciones

- ¿Qué hace glGetUniformLocation?
  Le dice al **shader** dónde tiene que acceder para leer la variable **transform**.
- Lo que hace internamente `import grafica.easy_shaders as es` :

```
#version 330

uniform mat4 transform; // Esto es lo que asocia "transform" con nuestra matriz numpy

in vec3 position;
in vec2 texCoords;

out vec2 outTexCoords;

void main()
{
  gl_Position = transform * vec4(position, 1.0f);
  outTexCoords = texCoords;
}
```

Ahora, tengo que implementar las matrices de transformación a mano?

- No. Puedes hacerla en numpy, pero también puedes usar las librerías.
- Por ejemplo, si usan `import grafica.transformations as tr` :

```
import grafica.transformations as tr

Rx = tr.rotationX(np.pi/3)
Ry = tr.rotationY(theta)
transform = tr.matmul([Rx, Ry])
...
```

Modelado de objetos

- Supongamos que queremos modelar el siguiente personaje:

Modelado de objetos

- Supongamos que queremos modelar el siguiente personaje:

- ¿Qué herramientas tenemos?

Modelado de objetos

- Supongamos que queremos modelar el siguiente personaje:

- ¿Qué herramientas tenemos?

  1. Cargar un archivo .off con la definición de vértices

Modelado de objetos

- Supongamos que queremos modelar el siguiente personaje:

- ¿Qué herramientas tenemos?

  1. Cargar un archivo .off con la definición de vértices
  2. Parametrizar cada una de las curvas que definen la superficie

Modelado de objetos

- Supongamos que queremos modelar el siguiente personaje:

- ¿Qué herramientas tenemos?

  1. Cargar un archivo .off con la definición de vértices
  2. Parametrizar cada una de las curvas que definen la superficie
  3. Modelar mediante el uso de primitivas con transformaciones o grafos de escena

Modelado de objetos

- Supongamos que queremos modelar el siguiente personaje:

- ¿Qué herramientas tenemos?

1. Cargar un archivo .off con la definición de vértices
2. Parametrizar cada una de las curvas que definen la superficie
3. Modelar mediante el uso de primitivas con transformaciones o grafos de escena
4. Dibujar cada uno de los cientos de vértices a mano

Modelado de objetos

- Supongamos que queremos modelar el siguiente personaje:

- ¿Qué herramientas tenemos?

1. Cargar un archivo .off con la definición de vértices
2. Parametrizar cada una de las curvas que definen la superficie
3. **Modelar mediante el uso de primitivas con transformaciones o grafos**
4. Dibujar cada uno de los cientos de vértices a mano

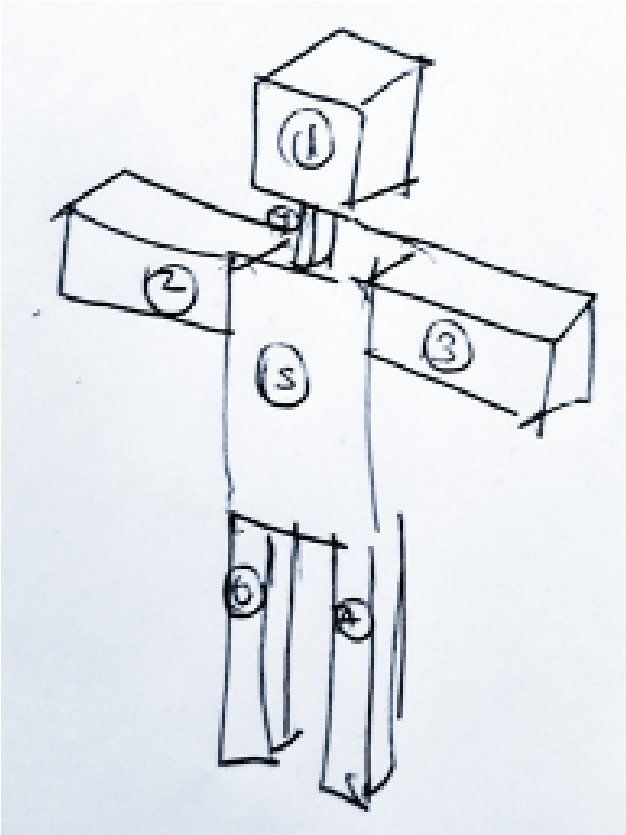
Modelado con primitivas o grafos de escena

- Básicamente tenemos dos alternativas:

  1. Ubicamos cada primitiva (cubo, esferas, etc) y las escalamos/rotamos/trasladamos para ir ensamblando la figura o modelo.
  2. Usamos grafos de escena. Estos indican el orden jerárquico del modelo, y además permiten construir escenas más complejas.

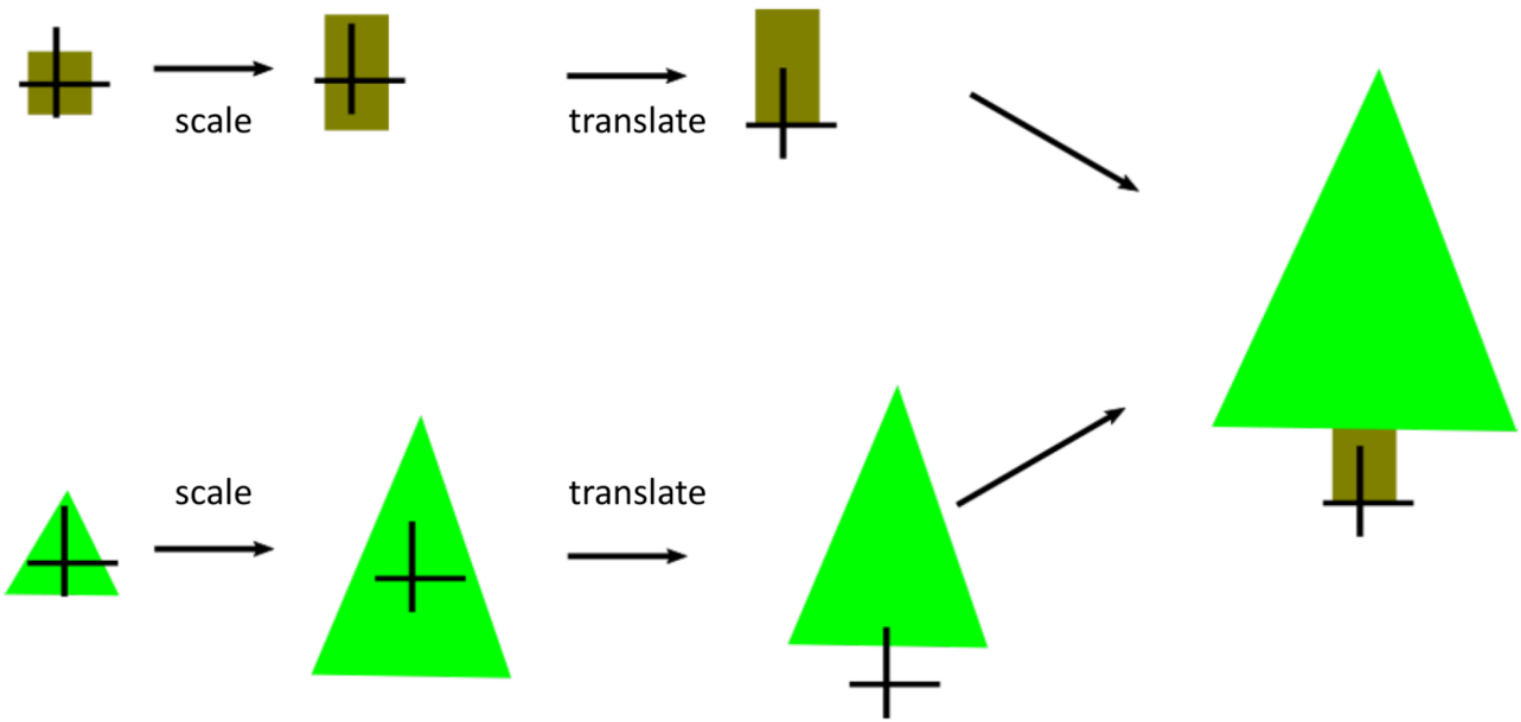
Si usamos primitivas más transformaciones

- ¿Cómo lo modelamos? Una combinación de primitivas, cubos en 3d. El objetivo es trasladarlos y escalarlos con tal de generar el modelo completo.

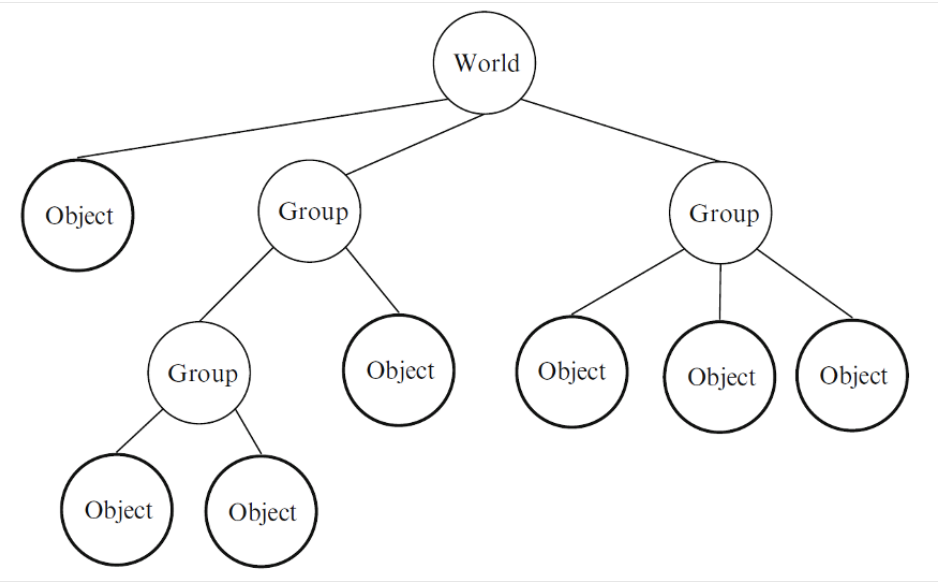


A TPOSE lo podemos representar con un total de 7 elementos, uno para la cabeza, otro para el cuello, dos para los brazos, uno para el torso y dos para las piernas

Usando grafos de escena



Grafos – Mayor nivel para la definición de modelos



- Cada hoja corresponde a un objeto básico
- Cada arco corresponde a una transformación
- Cada nodo interno corresponde a una agrupación

Grafos

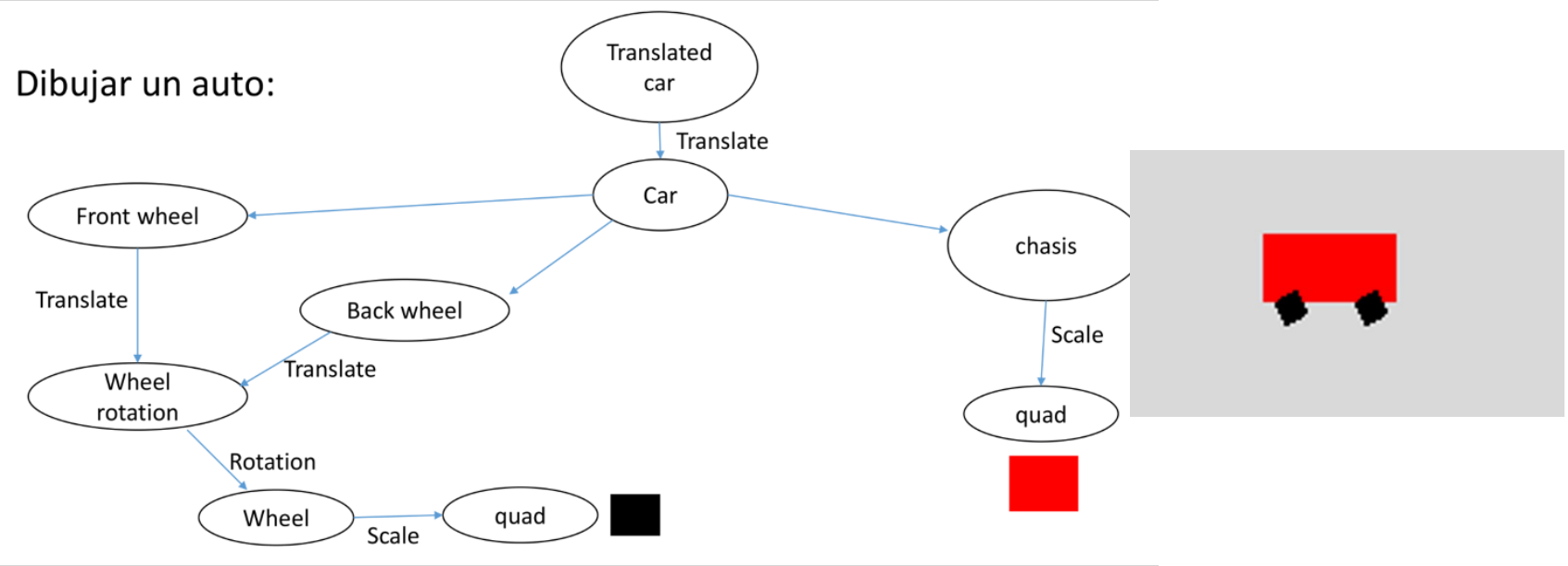
- ¿Qué tengo que usar para implementarlos?, ¿Hay que programar algún objeto?
- La librería del curso trae una definición sencilla llamada **SceneGraphNode**:

```
1  class SceneGraphNode:
2
3    def ___init___(self, name):
4      self.name = name
5      self.transform = tr.identity()
6      self.childs = []
7
8    def clear(self):
9      for child in self.childs:
10       child.clear()
```

- Para importarla, basta con `import grafica.easy_shaders as es`

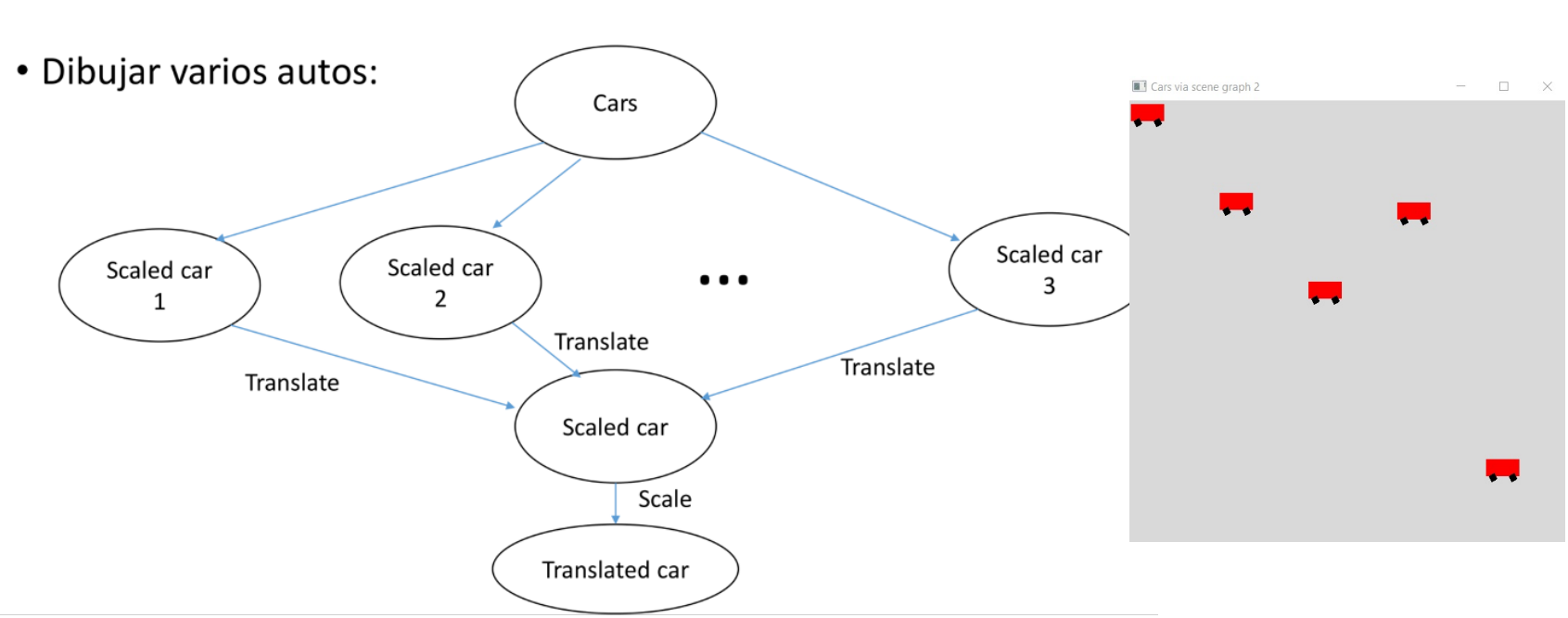
Grafos – Ejemplo auto

- Ver ejemplo ex_scene_graph_2dcars.py



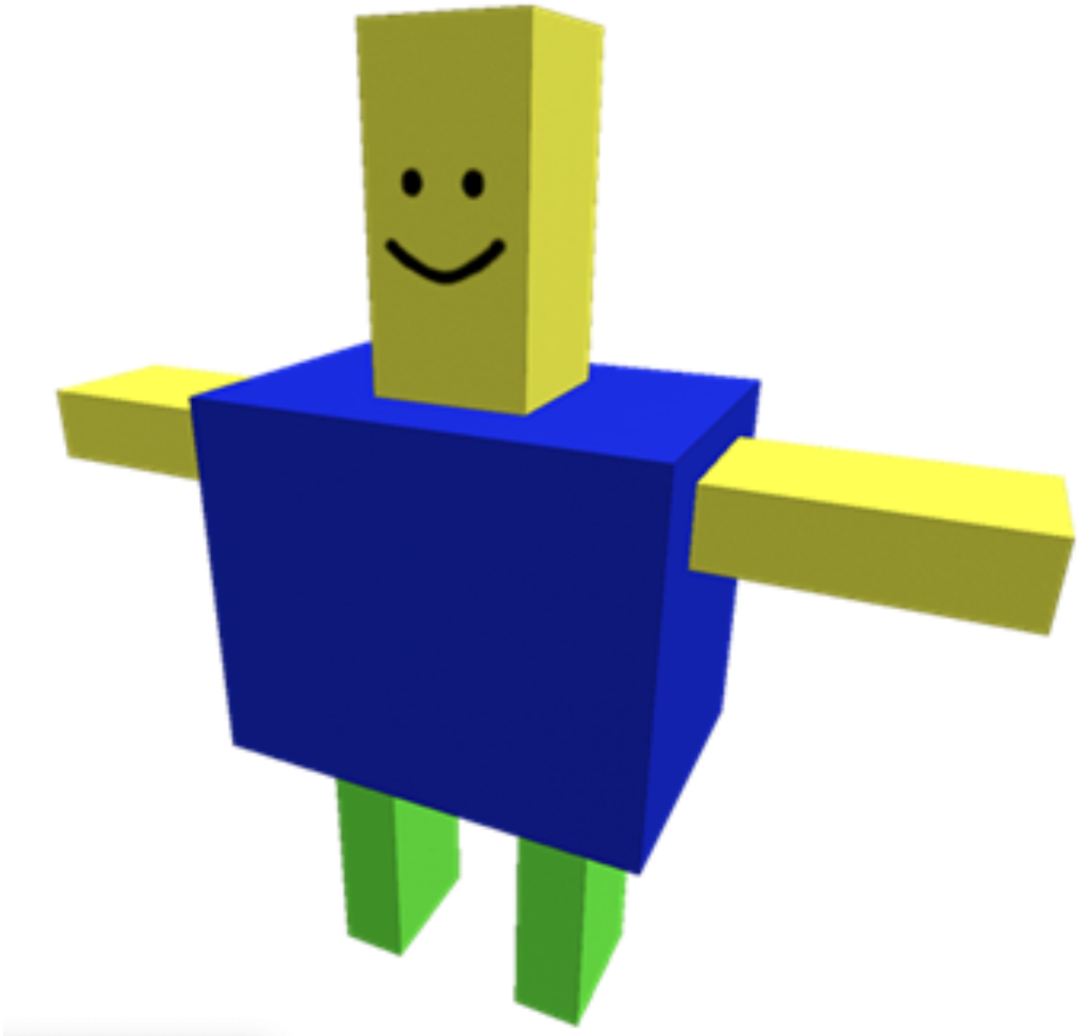
Grafos – Ejemplo auto, ¿Y si tengo muchos?

- Ver ejemplo ex_scene_graph_2dcars.py



Ejemplo de hoy

- Modelaremos nuestro EPIC-TPOSE con primitivas y grafos de escena.



Gracias por su atención