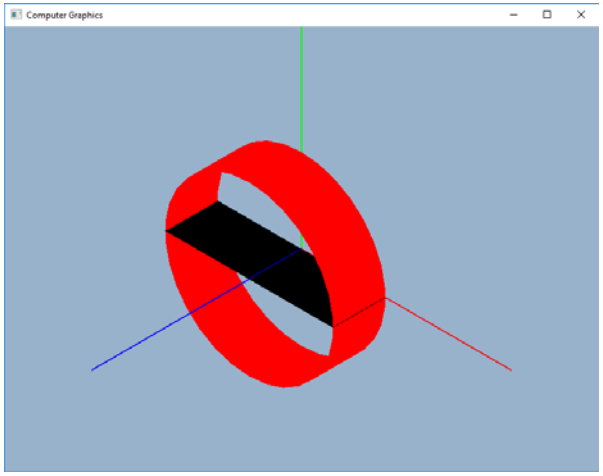
El modelo jerárquico

A. Gavilanes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Escenas por posicionamiento relativo

- Creación del rotor

```
Rotor(GLdouble r, GLdouble h) {
    ...
    cc = new Cylinder(r, r, h);
    rr = new RectangleRGB(2*r, h);
    dmat4 m = rr->getModelMat();
    m = translate(m, dvec3(0, 0, h / 2));
    m = rotate(m, radians(90.0), dvec3(1,0,0));
    rr->setModelMat(m);
}
```



- Renderizado del rotor

```
void render(dmat4 const& modelViewMat) {
    glMatrixMode(GL_MODELVIEW);
    dmat4 aMat = modelViewMat*getModelMat();
    glLoadMatrixd(value_ptr(aMat));

    cc->render(aMat);
    glColor3f(0.0, 0.0, 0.0);
    rr->render(aMat);
}
```

La clase Entity

- Cada entidad de la escena tiene un atributo **`glm::dmat4 modelMat`** que permite dibujar la entidad dentro de la escena de la que forma parte
- Cuando la entidad se renderiza, primero se sitúa dentro de la escena usando su **`modelMat`**, y luego se dibuja (lo que la forma, si tiene constituyentes):

```
void Rotor::render(dmat4 const& modelViewMat) {
    glMatrixMode(GL_MODELVIEW);
    dmat4 aMat = modelViewMat*getModelMat();
    glLoadMatrixd(value_ptr(aMat));

    ...
    // Renderizado de los elementos del rotor
    // con respecto a la matriz aMat
}
```

La clase Entity

- La matriz **`modelMat`** de cada entidad se inicializa a **`dmat4(1.0)`** y se modifica según lo requiera la colocación de la entidad dentro del objeto del que forma parte
- Ejemplos:
  - **`modelMat = dmat4(1.0)`**, para el rotor (en una escena con solo un rotor)
  - **`modelMat = dmat4(1.0)*T*R`**, para la paleta giratoria del rotor (en una escena con solo un rotor)
- Obsérvese que cada entidad de la escena manda renderizar sus elementos constituyentes llamando al método **`render(modelViewMat*modelMat)`** lo que supondrá post-multiplicar el parámetro **`modelViewMat*modelMat`** por la respectiva **`modelMat`** de la entidad que lo llama

El renderizado de Dron

```
void Dron::render(dmat4 const& modelViewMat) {
    glMatrixMode(GL_MODELVIEW);
    dmat4 aMat = modelViewMat * getModelMat();
    glLoadMatrixd(value_ptr(aMat));

    glColor3f(0.0, 0.0, 1.0);
    c->render(aMat);
    glColor3f(1.0, 0.0, 0.0);
    r1->render(aMat);
    glColor3f(1.0, 0.0, 0.0);
    r2->render(aMat);
    glColor3f(0.0, 1.0, 0.0);
    r3->render(aMat);
    glColor3f(0.0, 1.0, 0.0);
    r4->render(aMat);
}
```

La clase CompoundEntity

- Se puede definir una entidad compuesta como un tipo especial de entidad que está compuesta por otras entidades:

```
class CompoundEntity : public Entity
```

- Los objetos de esta clase disponen de un nuevo atributo

```
std::vector<Entity*> grObjects;
```

para guardar las entidades que los forman, junto con su accesora **`Entity* getEntity(int i)`**

- Una entidad compuesta se renderiza como una simple: se sitúa en la escena mediante su matriz **`modelMat`** y se pasa a renderizar las entidades que la forman:

```
… // Se sitúa en la escena (post-)multiplicando
  // por su matriz modelMat y se guarda el resultado en aMat
  // Se renderizan sus componentes con respecto a aMat
  for each (Entity* it in grObjects) {
     it->render(aMat);
  }
```

- En estas condiciones, la clase **`Scene`** se puede considerar que es una entidad compuesta **`class Scene : public CompoundEntity`** y su destructora, así como los métodos **`render()`** y **`update()`**, pasa a heredarlos de aquella.

La clase CompoundEntity

- La funcionalidad de esta clase se define entonces de la forma obvia:

```
CompoundEntity::~CompoundEntity() {
    for (Entity* it: grObjects) delete it;
}

void CompoundEntity::render(dmat4 const& modelViewMat) {
    glMatrixMode(GL_MODELVIEW);
    dmat4 aMat = modelViewMat * modelMat;
    glLoadMatrixd(value_ptr(aMat));
    for (Entity* it: grObjects) {
            it->render(aMat);
    }
}

void CompoundEntity::update() {
    for (Entity* it : grObjects) {
            it->update();
    }
}
```

El modelo jerárquico

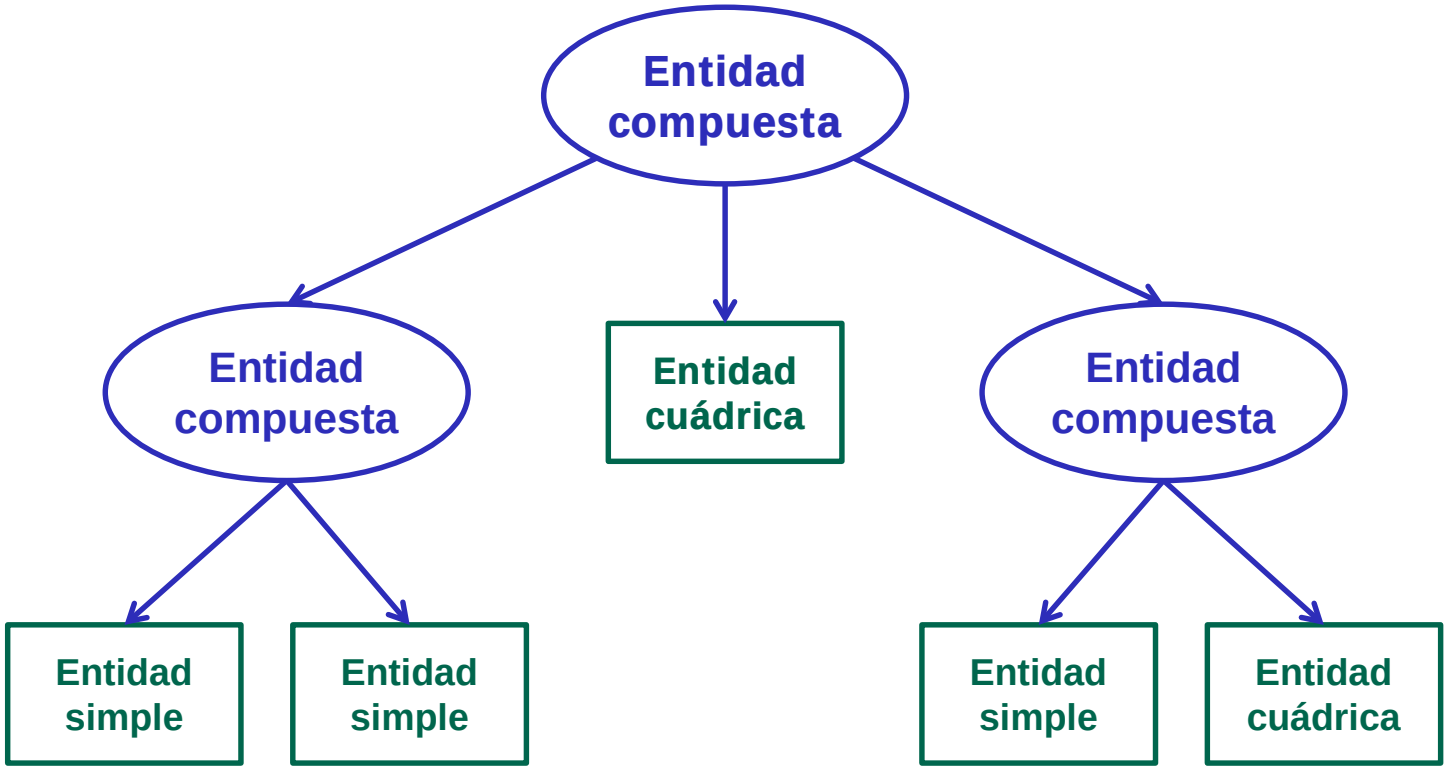
- El modelo jerárquico consiste en definir la escena como un grafo (para nosotros, un árbol) donde cada nodo es una entidad que puede ser de tres tipos:
  - Entidad simple: las formadas por una sola entidad. Son ejemplos de entidades simples las clases **`EjesRGB`** o **`RectanguloRGB`** entre otras. Sus clases heredan directamente de **`Entity`**
  - Entidad cuádrica: las que permiten dibujar objetos cuádricos usando la librería **`GLU`**. Es una entidad cuádrica la clase **`Sphere`** (y las definidas para las demás cuádricas) que hereda directamente de **`QuadricEntity`**:

```
class QuadricEntity : public Entity ...

class Sphere : public QuadricEntity ...
```

  - Entidad compuesta: las formadas por varias entidades. Por ejemplo, la clase **`Dron`** es una entidad compuesta. Sus clases heredan directamente de **`CompoundEntity`**

El modelo jerárquico

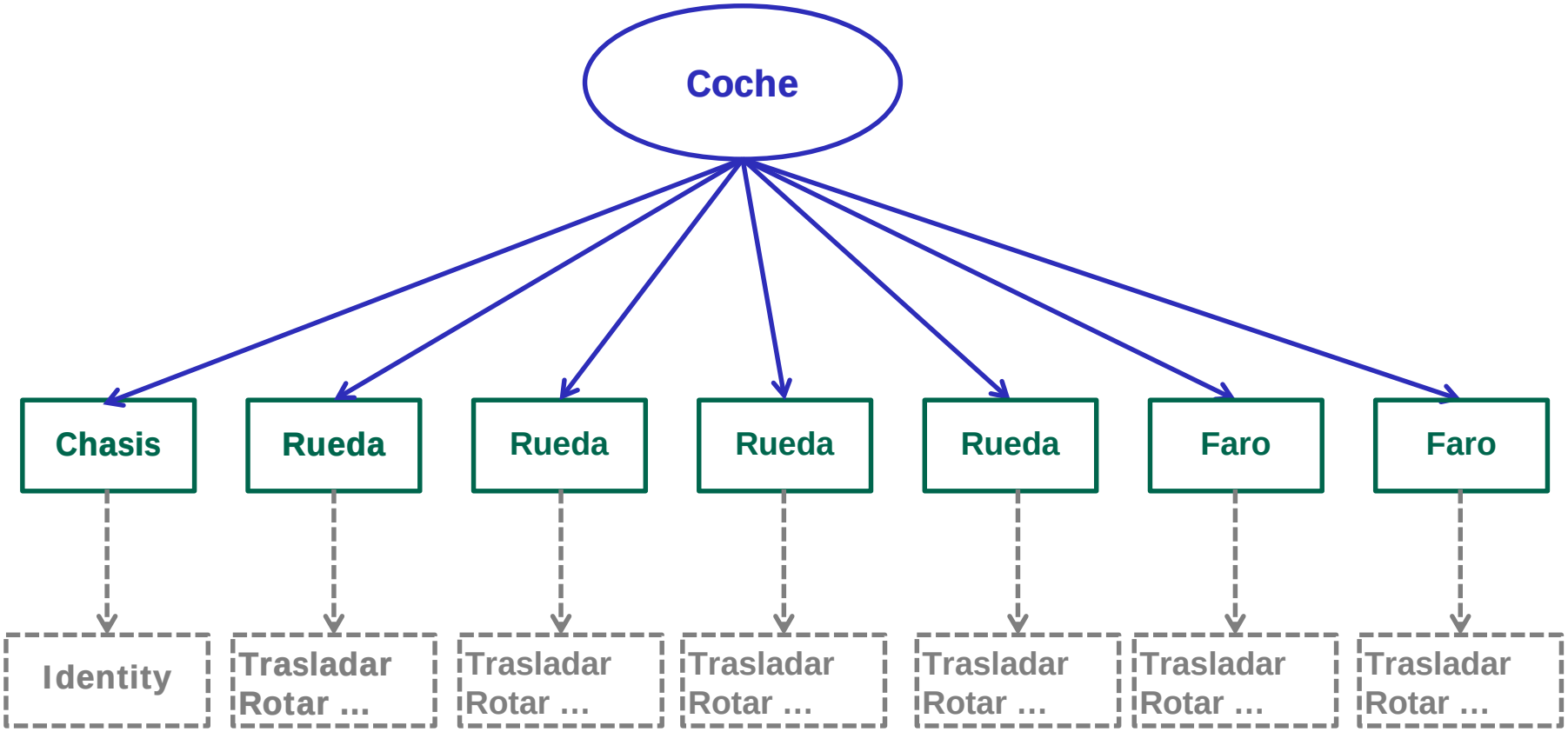


Un coche como entidad compuesta

```
// La clase Coche hereda de CompoundEntity
Coche :: Coche() {
    // Se añaden 7 elementos a la entidad compuesta Coche:
    // 1 chasis (de tipo Cubo), 4 ruedas (de tipo Rueda,
    // que es una CompoundEntity formada por un cilindro y
    // un disco) y 2 faros (de tipo Cilindro)
    for (int i=0; i<4; i++) {
        Rueda* rueda = new Rueda();
        this->entities.push_back(rueda);
        ...
    }
    // Se sitúan los 7 elementos dentro del coche
    // mediante translate, rotate, scale

    ...
}
```

Un coche en el modelo jerárquico



El modelo jerárquico

- Ventajas del modelo jerárquico:
  - Cada constituyente de una entidad compuesta debe aplicar la secuencia de transformaciones que lo colocan en la escena (global) antes de dibujarse

  En el modelo jerárquico, cada entidad guarda una única transformación que la sitúa con respecto a la entidad de la que forma parte, por tanto se dibuja más rápidamente

  - Durante el proceso de *culling*: si una entidad compuesta no es visible, no es necesario dibujarla y, por tanto, tampoco es necesario dibujar las entidades que la componen

  En el modelo jerárquico no es necesario codificar nada. Si la entidad compuesta no invoca el método **`render()`**, sus constituyentes tampoco lo harán.

  - Si una entidad compuesta se mueve, es necesario mover todas las entidades que la componen

  En el modelo jerárquico no es necesario codificar nada. Cuando la entidad compuesta se mueve, la matriz de modelado-vista se post-multiplica por la matriz **`modelMat`** de la entidad y el resultado sirve de base para las post-multiplicaciones que aplican sus constituyentes al dibujarse.

BB8 como entidad compuesta