

Entidades compuestas

Informática Gráfica I

Material de: **Antonio Gavilanes**
Adaptado por: **Elena Gómez y Rubén Rubio**
{mariaelena.gomez,rubenrub}@ucm.es



Contenido

1 Definición

2 Implementación

- Renderización
- Posicionamiento relativo
- Ventajas

3 Nodo ficticio

Entidades compuestas

- La clase de las entidades compuestas es una clase llamada `CompoundEntity` que hereda de `Abs_Entity`.
- Dispone de un vector donde se coleccionan las entidades que forman la entidad compuesta:

```
std::vector<Abs_Entity*> gObjects;
```

- Dispone de un método para añadir a la entidad compuesta las entidades que la forman:

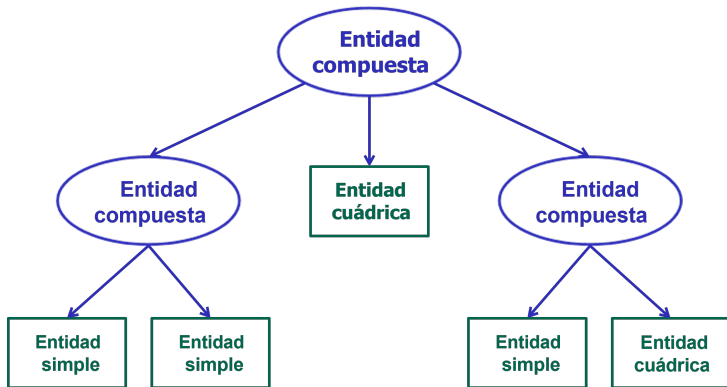
```
void addEntity(Abs_Entity* ae);
```

- La destructora vacía `gObjects` tal como lo hace `free()` de la clase `Scene`.

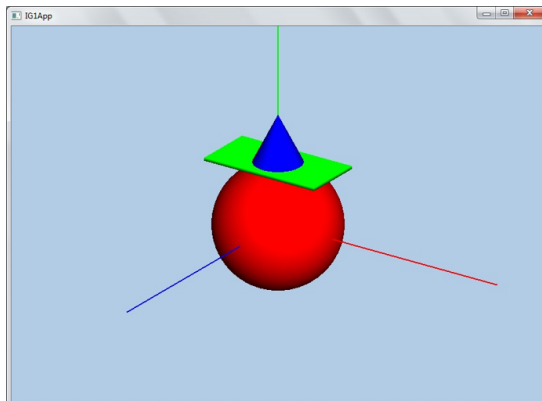
El modelo jerárquico

- 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 `Star3D`, entre otras.
Sus clases heredan directamente de `Abs_Entity`.
 - **Entidad compuesta:** las formadas por varias entidades. Por ejemplo, el objeto `capAndHat` que veremos a continuación es una entidad compuesta.
Estas clases las que heredan directamente de `CompoundEntity`.

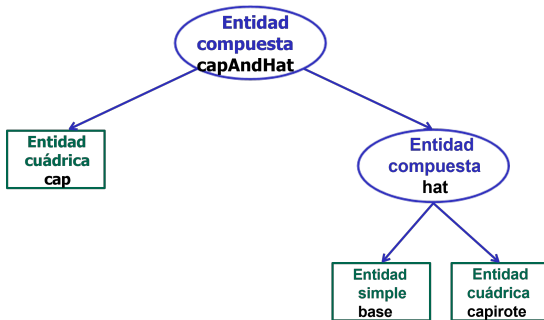
El modelo jerárquico



Ejemplo: Cabeza con sombrero



Ejemplo: Cabeza con sombrero



Ejemplo: Cabeza con sombrero

- La cabeza con sombrero es una entidad compuesta que se construye del siguiente modo:
 - Se construye la entidad compuesta `capAndHat` y se añade a la escena:

```
CompoundEntity* capAndHat = new CompoundEntity();  
gObjects.push_back(capAndHat);
```

- Se construye la cabeza `cap` como una esfera roja y se añade a la entidad compuesta `capAndHat`:

```
Sphere* cap = new Sphere(100.0);  
capAndHat->addEntity(cap);
```

- Se construye el sombrero `hat` y se añade a la entidad compuesta `capAndHat`:

```
CompoundEntity* hat = new CompoundEntity();  
capAndHat->addEntity(hat);
```


Ejemplo: Cabeza con sombrero

- La cabeza con sombrero es una entidad compuesta que se construye del siguiente modo (continuación):

3 El sombrero es una entidad compuesta por:

- Una base que es la malla de un cubo verde indexado y convenientemente escalado:

```
Cubo* base = new Cubo(100);  
// Se fija el color verde, se escala y estará colocado  
// como parte de un sombrero centrado en el origen  
hat->addEntity(base);
```

- Un capirote que es un cono:

```
Cylinder* capirote = new Cylinder(80.0, 40.0, 0);  
// Se fija el color azul, se rota y estará colocado  
// como parte de un sombrero centrado en el origen  
hat->addEntity(capirote);
```

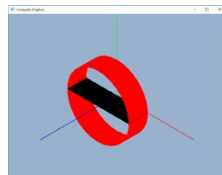
Renderización

- Las entidades compuestas heredan su método `render(modelViewMat)`, pero lo reescriben a fin de que sus entidades constituyentes se rendericen, no con respecto a la matriz de vista, sino con respecto a la matriz de modelado-vista, siendo la matriz de modelado la de la entidad compuesta,
- El proceso para renderizar una entidad compuesta es pues el siguiente:
 - Se multiplica la matriz `modelViewMat` (matriz de vista) por la matriz de modelado que la entidad compuesta tiene por ser una entidad.
 - Se carga la matriz resultante `aMat` con `upload(aMat)`.
 - Se renderizan las entidades constituyentes de `gObjects` con respecto a `aMat`.

Entidades construidas por posicionamiento relativo

- Supongamos que un rotor es una entidad con dos atributos: un tubo **tub** (cilindro rojo) y una paleta **pal** (rectángulo negro). Estas dos entidades se posicionan para renderizarse como en la figura adjunta.

```
Rotor(GLdouble r, GLdouble h) {  
    ...  
    tub = new Cylinder(r, r, h);  
    pal = new RectangleRGB(2*r, h);  
    dmat4 m = pal->getModelMat(); //m=dmatrix(1.0);  
    m = translate(m, dvec3(...));  
    m = rotate(m, radians(...), dvec3(...));  
    pal->setModelMat(m);  
}
```



Entidades construidas por posicionamiento relativo

- Renderizado de un rotor:

```
void Rotor::render(dmat4 const& modelViewMat) {  
    dmat4 aMat = modelViewMat*mModelMat;  
    upload(aMat);  
    tub->render(aMat);  
    pal->render(aMat);  
}
```

Renderizado de las entidades

- Sabemos que cada entidad de la escena tiene un atributo `glm::dmat4 mModelMat` 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 `mModelMat`, y luego se renderiza lo que la forma (si tiene constituyentes):

```
void Rotor::render(dmat4 const& modelViewMat) {  
    dmat4 aMat = modelViewMat*mModelMat;  
    upload(aMat);  
    // Renderizado de los elementos del rotor  
    // con respecto a la matriz aMat  
    ...  
}
```

Renderizado de entidades con posicionamiento relativo

- La matriz `mModelMat` de cada entidad se inicializa a `dmat4(1.0)` y se modifica según lo requiera su colocación dentro de la entidad de la que forma parte:

Ejemplo de una escena con un rotor

```
mModelMat(tub) = dmat4(1.0)); // para el tubo del rotor  
mModelMat(pal) = dmat4(1.0)*T*R} // para la paleta del rotor
```

Renderizado de entidades con posicionamiento relativo

- En general, cada entidad de la escena con matriz de modelado `mModelMat_entidad` manda renderizar cada elemento constituyente `ent` que tenga, haciendo las llamadas:

```
ent->render(modelViewMat*mModelMat_entidad)
```

lo que supondrá, a su vez, post-multiplicar el parámetro por la respectiva matriz de modelado `mModelMat_ent` de la entidad `ent`.

Renderizado de entidades con posicionamiento relativo

- Por ejemplo, en la escena del rotor, si `mModelMat_rotor` es la matriz de modelado del rotor entonces:
 - Se hará la llamada `tub->render(aMat)` para renderizar el tubo y esta llamada cargará la matriz `modelViewMat*mModelMat_rotor*dmat4(1.0)` antes de renderizar el cilindro.
 - Se hará la llamada `pal->render(aMat)` para renderizar la paleta y esta llamada cargará la matriz `modelViewMat*mModelMat_rotor*dmat4(1.0)*T*R` antes de renderizar el rectángulo.
 - La renderización de una entidad compuesta es igual, pero sus entidades constituyentes no son atributos (`tub`, `pal`) sino componentes del vector `gObjects`.

Ventajas del modelo jerárquico

- Cada constituyente de una entidad compuesta debe aplicar, antes de dibujarse, la secuencia de transformaciones que la colocan en la escena.
- 👉 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 el renderizado es más rápido.

Ventajas del modelo jerárquico

- 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.

Ventajas del modelo jerárquico

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

Nodo ficticio

- La técnica del **nodo ficticio** se utiliza para facilitar los movimientos de ciertas entidades de la escena.
- Consiste en crear un nodo ficticio **NF** por encima de (que tiene como hijo a) otro nodo **N** de forma que la animación de **N** se hace recaer sobre **NF**, pero de manera que esta afecte a **N**.
- El nodo ficticio **NF** es una entidad compuesta que, por tanto, solo tiene un constituyente **N**.

Ejemplo

El efecto de rotar **NF** será que rote **N**.

Nodo ficticio

Ejemplo 1

Hacer girar un triángulo RGB (perdiendo la horizontalidad) sobre una circunferencia de radio 200.

- 1 Se crea el nodo ficticio y el triángulo como constituyente de él:

```
CompoundEntity* inventedNode = new CompoundEntity();  
RGBTriangle* tr = new RGBTriangle(lado);  
inventedNode->addEntity(tr);
```

- 2 Se sitúa el triángulo `tr` sobre la circunferencia:

```
tr->setModelMat(translate(inventedNode->modelMat(),  
    dvec3(200, 0, 0)));
```

Nodo ficticio

Ejemplo 1

Hacer girar un triángulo RGB (perdiendo la horizontalidad) sobre una circunferencia de radio 200.

- 3 Se hace rotar el nodo ficticio:

```
inventedNode->setModelMat(rotate(inventedNode->modelMat(),  
    radians(3.0), dvec3(0, 0, 1)));
```

Nodo ficticio

Ejemplo 2

Hacer girar un triángulo RGB (perdiendo la horizontalidad) sobre una circunferencia de radio 200 y hacer que gire, en torno de él, un triángulo RGB más pequeño, en sentido horario contrario al del triángulo mayor:

- 1 Se crea un primer `inventedNode` y se le añaden, como constituyentes, el triángulo mayor `tr` y un segundo `inventedNode2`, ambos desplazados a la posición (200, 0, 0):

```
dvec3 pos = dvec3(200, 0, 0);  
CompoundEntity* inventedNode = new CompoundEntity();  
RGBTriangle* tr = new RGBTriangle(side);  
inventedNode->addEntity(tr);  
tr->setModelMat(translate(inventedNode->modelMat(), pos));  
...
```

Nodo ficticio

Ejemplo 2

Hacer girar un triángulo RGB (perdiendo la horizontalidad) sobre una circunferencia de radio 200 y hacer que gire, en torno de él, un triángulo RGB más pequeño, en sentido horario contrario al del triángulo mayor:

- 1 Se crea un primer `inventedNode` y se le añaden, como constituyentes, el triángulo mayor `tr` y un segundo `inventedNode2`, ambos desplazados a la posición (200, 0, 0):

...

```
CompoundEntity* inventedNode2 = new CompoundEntity();
inventedNode->addEntity(inventedNode2);
inventedNode2->setModelMat(
    translate(inventedNode->modelMat(), pos));
```


Nodo ficticio

Ejemplo 2

Hacer girar un triángulo RGB (perdiendo la horizontalidad) sobre una circunferencia de radio 200 y hacer que gire, en torno de él, un triángulo RGB más pequeño, en sentido horario contrario al del triángulo mayor:

- 2 Se crea el triángulo menor `trLittle` y se añade, desplazado, al segundo nodo ficticio:

```
RGBTriangle* trLittle = new RGBTriangle(side-20);  
inventedNode2->addEntity(trLittle);  
trLittle->setModelMat(translate(trLittle->modelMat(),  
    dvec3(-100, 0, 0))));
```

Nodo ficticio

Ejemplo 2

Hacer girar un triángulo RGB (perdiendo la horizontalidad) sobre una circunferencia de radio 200 y hacer que gire, en torno de él, un triángulo RGB más pequeño, en sentido horario contrario al del triángulo mayor:

- 3 Se hacen rotar los nodos ficticios, en sentidos horarios opuestos:

```
inventedNode->setModelMat(rotate(inventedNode->modelMat(),  
    radians(3.0), dvec3(0, 0, 1)));  
inventedNode2->setModelMat(rotate(inventedNode2->modelMat(),  
    radians(-10.0), dvec3(0, 0, 1)));
```