técnicas avanzadas de gráficos

ingeniería multimedia

Seminario 4

Fachada

Departamento de Ciencia de la Computación e Inteligencia Artificial
Universidad de Alicante

Interfaces y fachadas

¿Qué es una interfaz?

¿Qué es una fachada?

Interfaz

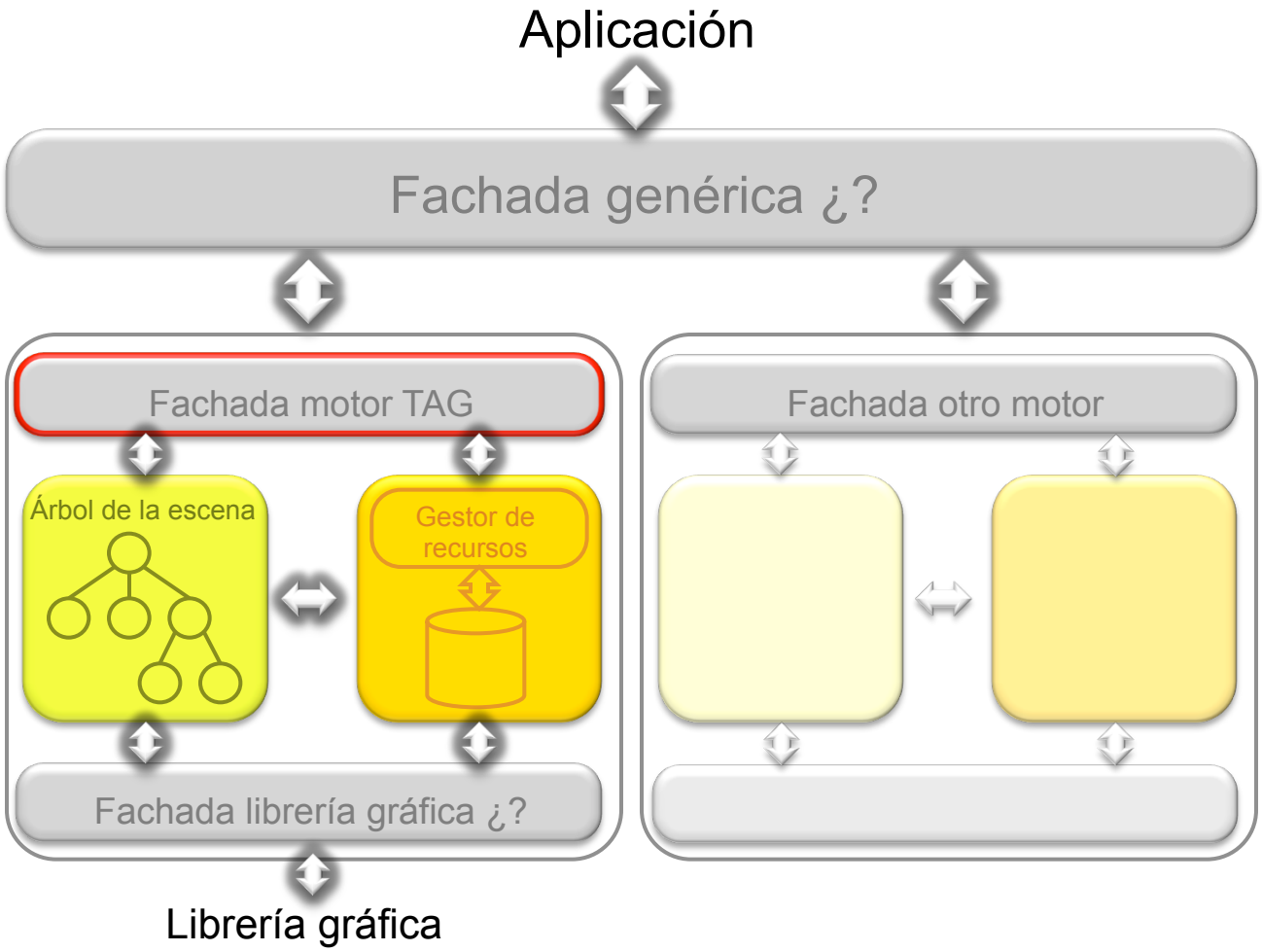
- Conexión funcional entre dos sistemas de cualquier tipo dando una comunicación entre distintos niveles.
- En un motor gráfico será su API (Application Programming Interface), es decir, el conjunto de clases y métodos que ofrece el motor gráfico para ser utilizado por otro software como una capa de abstracción.

Fachada

- Proponemos construir la interfaz del motor según un patrón Fachada (Façade)
- Patrón adecuado cuando queramos:
  - Minimizar la comunicación y las dependencias entre los subsistemas.
  - Proporcionar una interfaz simple para un subsistema complejo.
  - Preservar determinada funcionalidad para evitar comportamiento inadecuado.
  - Estructurar varios subsistemas en capas.
  - Desacoplar un sistema de sus subsistemas, haciéndolo más independiente, portable y reutilizable.

Fachada


recuerda


Aplicación
Fachada genérica ¿?
Fachada motor TAG
Árbol de la escena
Gestor de recursos
Fachada librería gráfica ¿?
Fachada otro motor
Librería gráfica

Fachada del motor

- Utilidad de la fachada
  - Separa el motor de la aplicación ➔ el motor puede sustituirse fácilmente por otro y también reutilizarse
  - Proporciona una única clase como interfaz con todo el motor ➔ oculta la complejidad de uso del motor
  - Evita operaciones ilegales sobre el motor ➔ mantiene la integridad del árbol de la escena y de todos los datos
  - Proporciona funciones de utilidad ➔ facilita muchas acciones que debería hacer la aplicación en otro caso

Una propuesta de fachada

TMotorTAG
TNodo* escena TGestorRecursos *gestorRecursos *Atributos para mantenimiento de las cámaras, luces y viewports*
TNodo *crearNodo (TNodo *padre, TEndidad *ent) TTransform *crearTransform (…) TCamara *crearCamara (…) TLuz *crearLuz (…) TMalla *crearMalla (char *fichero, …) void draw (…) *Métodos para el registro y manejo de las cámaras* *Métodos para el registro y manejo de las luces* *Métodos para el registro y manejo de los viewports*

Crear nodos


```
TNodo *TMotorTAG :: crearNodo (TNodo *padre, TEndidad *ent)
{
        crear nodo y asociarle la entidad
        añadir el nuevo nodo como hijo del nodo padre
        devolver la referencia al nodo creado
}
```

Crear entidades


```
TTransform *TMotorTAG :: crearTransform (…) {
    Crear transformación y devolverla
}

TCamara *TMotorTAG :: crearCamara (…) {
   Crear cámara y devolverla
}

TLuz *TMotorTAG :: crearLuz (…) {
    Crear luz y devolverla
}

TMalla *TMotorTAG :: crearMalla (char* fichero, …) {
    Pedir el recurso malla al gestor de recursos a partir del fichero
    Crear la malla a partir del recurso malla y devolverla
}
```

Un ejemplo de uso


```
TMotorTAG *motor = new TMotorTAG (…)

// Creamos algunas entidades y las modificamos

TTransform *transf1 = motor->crearTransform (...)
TTransform *transf2 = motor->crearTransform (...)

transf1->escalar (0.5, 0.25, 0.5)
transf1->trasladar (0,0,-300)

// Creamos algunos nodos y les asociamos las entidades anteriores

TNodo *nodoTransf1 = motor->crearNodo (motor->nodoRaiz (), transf1)
TNodo *nodoTransf2 = motor->crearNodo (nodoTransf1, transf2)

//Creamos más entidades y nodos

TMalla *malla1 = motor->crearMalla ("mimalla.obj", …)
TNodo *nodoMalla = motor->crearNodo (nodoTransf2, malla1)
transf2->rotar (-PI, 1, 1, 1)
```

Luces y cámaras

- En las librerías gráficas (y particularmente en OpenGL) las luces y las cámaras se activan al principio del dibujado
- Nosotros las tenemos en el árbol, en posiciones desconocidas a priori
- Dos posibles soluciones:
  - Mediante varias pasadas recorriendo el árbol: para luces, para cámara y para dibujar.
  - Mediante el registro de las luces y cámaras

Varios recorridos del árbol

- Cambiar las funciones draw, beginDraw y endDraw de los nodos y entidades, para pasarles un parámetro indicando la pasada: luces, cámaras o dibujar.
- La función beginDraw de las entidades cambia:
  - Para las luces, sólo realiza acciones si la pasada es de luces
  - Para las cámaras, sólo realiza acciones si la pasada es de cámaras
  - Para el resto de entidades, sólo realiza acciones (las que ya vimos) si la pasada es de dibujar

Varios recorridos : luz

- Cuando se ejecuta esta función, la matriz activa de la librería gráfica (MODELVIEW) contiene la posición de esa luz
- Para activar/desactivar las luces: podemos ponerlas y quitarlas del árbol o podemos añadirles un atributo indicando si están activas

```
void *TLuz :: beginDraw (int pasada)
{
        si (pasada == LUZ && luz activa) {
                obtener la posición de la luz de la matriz activa (MODELVIEW)
                activar la luz en la librería gráfica
                colocar la luz en la posición obtenida
         }
}
```

Varios recorridos: cámara

- Cuando se ejecuta esta función, la matriz activa de la librería gráfica (MODELVIEW) contiene la posición de esa cámara
- Para activar/desactivar la cámara: podemos ponerla o quitarla del árbol o podemos añadirle un atributo indicando si está activa. En este caso, sólo tenemos una cámara activa a la vez
- En este caso, la matriz de posición de la cámara debe invertirse y guardarse para cargarse al principio de dibujado en la matriz MODELVIEW

```
void *TCamara :: beginDraw (int pasada)
{
        si (pasada == CAMARA && camara activa) {
                obtener la matriz de posición de la cámara (MODELVIEW)
                invertir esa matriz y devolverla para utilizarla en el dibujado
         }
}
```

Registro de luces y cámaras

- Evita varias pasadas del árbol
- Consiste en llevar un registro de las luces y cámaras para poder acceder directamente a ellas al principio del dibujado
- Mantendremos un registro (un array) de:
  - Luces: registro de nodos que contienen luces y de cuáles están activas
  - Cámaras: registro de nodos que contienen cámaras y de cuál es la cámara activa

Registro de luces


```
// Crear y registrar luz en la aplicación
TLuz *luz = motor->crearLuz (…)
TNodo *nodoLuz = motor->crearNodo (padre, luz);
int nLuz = motor->registrarLuz (nodoLuz)
motor->setLuzActiva (nLuz)
…
```

```
TMotorTAG :: draw (…) {
        …
        para cada nodoLuz activo del registro de luces {
                recorrer el árbol a la inversa desde nodoLuz hasta la raiz
                guardar el recorrido en una lista auxiliar de nodos de transformación
                invertir la lista auxiliar
                recorrer la lista auxiliar multiplicando las matrices en una matriz auxiliar
                obtener la posición de la luz a partir de la matriz auxiliar
                posicionar y activar la luz en la librería gráfica
        } …
}
```

Registro de cámaras


```
// Crear y registrar cámara en la aplicación
TCamara *camara= motor->crearCamara (…)
TNodo *nodoCamara = motor->crearCamara (padre, camara);
int nCamara = motor->registrarCamara (nodoCamara)
motor->setCamaraActiva (nCamara)
…
```

```
TMotorTAG :: draw (…) {
        …
        nodoCamara = getCamaraActiva ();
        recorrer el árbol a la inversa desde nodoCamara hasta la raiz
        guardar el recorrido en una lista auxiliar de nodos de transformación
        invertir la lista auxiliar
        recorrer la lista auxiliar multiplicando las matrices en una matriz auxiliar
        invertir la matriz auxiliar
        cargar la matriz auxiliar en la matriz MODELVIEW de la librería gráfica
        …
}
```

Viewports

- Los viewports no forman parte del árbol, porque no son datos de la escena
- Proponemos un registro de viewports como el propuesto para luces y cámaras
- Para cada registro sabremos: posición (x,y) en la ventana y tamaño (alto, ancho)
- También mantendremos qué viewport está activo (sólo uno cada vez)
- En la función TMotorTAG :: Draw fijaremos el viewport de la librería gráfica con los datos del viewport activo

Un ejemplo de uso


```
TMotorTAG *motor = new TMotorTAG (…)
TTransform *transf1 = motor->crearTransform (...)
TTransform *transf2 = motor->crearTransform (…)
TTransform *transf3 = motor->crearTransform (…)
TLuz *luz = motor->crearLuz (…)
TCamara *camara= motor->crearCamara (…)
TMalla *malla1 = motor->crearMalla ("mimalla.obj", …)
transf1->escalar (0.5, 0.25, 0.5)
transf1->trasladar (0,0,-300)
transf3->trasladar (10, 10, 0)
TNodo *nodoTransf1 = motor->crearNodo (motor->nodoRaiz (), transf1)
TNodo *nodoTransf2 = motor->crearNodo (motor->nodoRaiz (), transf2)
TNodo *nodoTransf3 = motor->crearNodo (nodoTransf2, transf3)
TNodo *nodoMalla = motor->crearNodo (nodoTransf2, malla1)
TNodo *nodoLuz = motor->crearNodo (nodoTransf1, luz);
TNodo *nodoCamara = motor->crearCamara (nodoTransf3, camara);
int nViewport = motor->registrarViewport (x, y, alto, ancho, …)
motor->setViewportActivo (nViewport)
// Sólo si se ha optado por la opción de registrar luces y cámaras
int nCamara = motor->registrarCamara (nodoCamara)
motor->setCamaraActiva (nCamara)
int nLuz = motor->registrarLuz (nodoLuz)
motor->setLuzActiva (nLuz)
```

Dibujado

// Si se ha optado por varios recorridos del árbol
TMotorTAG :: draw (…) {
    Inicializar la librería gráfica como sea necesario
    escena->draw (LUZ)
    Inicializar el viewport activo con la librería gráfica
    escena->draw (CAMARA)
    cargar en la matriz MODELVIEW la matriz de la cámara
    escena->draw (DIBUJAR)
}

// Si se ha optado por registrar las luces y las cámaras
TMotorTAG :: draw (…) {
    Inicializar la librería gráfica como sea necesario
    Inicializar las luces como se ha explicado
    Inicializar el viewport activo con la librería gráfica
    Inicializar la cámara como se ha explicado
    escena->draw (…)
}