

PRÁCTICA 3: Modelado visual

Introducción

Cámara ortográfica y cámara perspectiva

Sistema multicámara

Valoración de la práctica

Objetivos

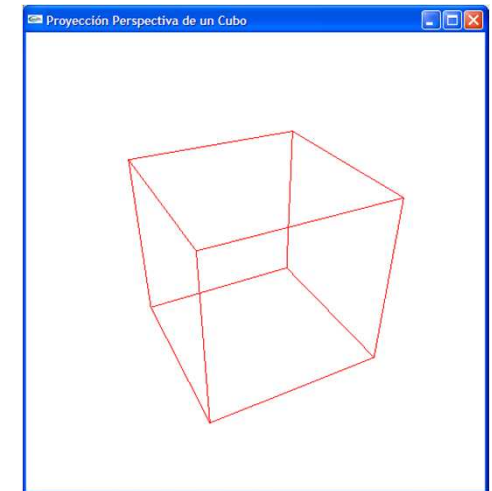
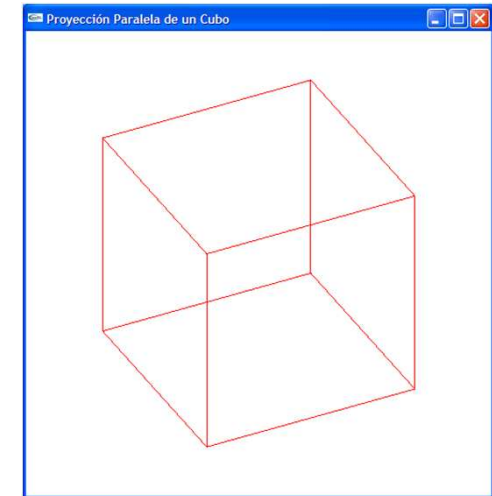
- Desarrollar clases para manejar cámaras
 - Ortográfica
 - Perspectiva
- Construir un visualizador multicámara (4 vistas)
 - Planta, Alzado y Perfil
 - Perspectiva
- Ser capaz de construir un editor/inspector de formas geométricas sencillo usando un sistema multicámara

Planificación

- 3 sesiones
- Práctica individual
- Puntuación:
 - Parte mínima (0,75 puntos) + Parte adicional (0,75)
- No se deben usar cámaras o transformaciones de OpenGL

Ficheros

- Se usará:
 - *Algebra (.h .cpp)*
 - *Primitiva (.h .cpp)*
 - *SuperficieBezier (.h .cpp)*: Implementada en la práctica 2
 - *Camara.h* : Definición de la clase y derivadas
 - *CuboPar.cpp* y *CuboPer.cpp*: Código de validación de *Camara*
- Se debe implementar:
 - *Camara.cpp*: Fichero de código fuente a completar
 - *Multivista.cpp*: Fichero de visualización multivista (se proporciona un esqueleto)
- Se debe entregar:
 - Todos los (.h .cpp) necesarios para generar los ejecutables
 - *CuboPar.exe*, *CuboPer.exe* y *Multivista.exe* (generados en modo “release” para Windows-XP)
 - Fichero *instrucciones.txt* en caso de ser necesario



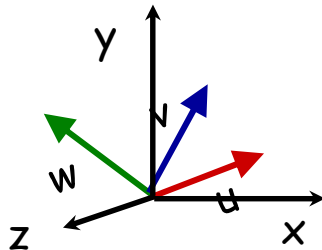
Cámara ortográfica y cámara perspectiva

- Atención especial a las operaciones de la clase *Punto*:

- `Punto homogen()` //Devuelve el punto con w=1

- Atención especial en la clase *Transformacion*:

- `Transformacion rotation(Vector u, Vector v, Vector w)`
//Acumula un giro dado por los ejes u,v,w



$$R \cdot \vec{u} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$R \cdot \vec{v} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$R \cdot \vec{w} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Cámara ortográfica y cámara perspectiva

class Camara

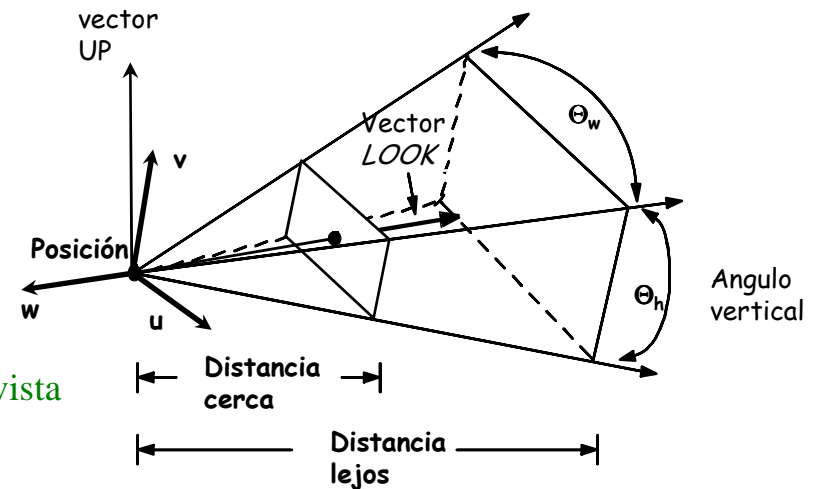
{

protected:

Punto pov; // Posición de la cámara (def: 0,0,0)
 Vector look; // Orientación de la cámara (def: 0,0,-1)
 Vector up; // Arriba en la cámara (def: 0,1,0)
 float aspectRatio; // Razón de aspecto (def: 4/3)
 float near; // Distancia al plano frontal (def: 1)
 float far; // Distancia al plano trasero (def: 10)
 Transformacion view; // Matriz de transformación al sistema de la vista
 int ready; // Indica si la cámara esta lista para disparar
 virtual void setView(); // Actualiza la vista

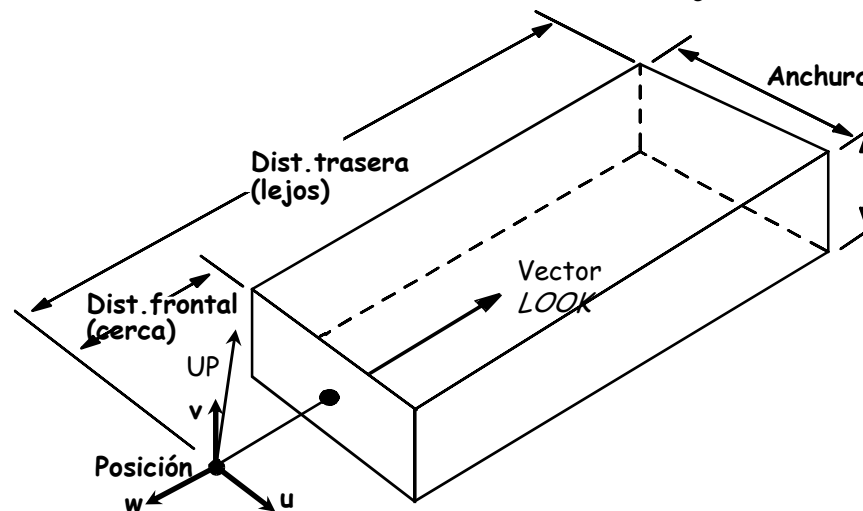
public:

Camara(); // Constructor por defecto
 void at(Punto pos); // Posiciona la cámara
 void lookAt(Punto poi); // Orienta la cámara mirando hacia el punto de interés poi
 void lookTo(Vector to); // Orienta la cámara mirando en esa dirección
 void setVertical(Vector v); // Indica el vector up
 void setAspectRatio(float ratio); // Cambia las proporciones de la foto
 void setFOV(float neardistance, float fardistance); // Indica los límites del campo visual
 Punto shot(Punto p); // Transforma un punto al sistema de referencia propio
 Transformacion getview(); // Devuelve la matriz de la vista



Cámara ortográfica y cámara perspectiva

```
class CamaraOrtografica: public Camara
{
protected:
    float height; // Altura de la foto (def: 2)
    void setView(); // Cálculo de la transformación de la vista
public:
    CamaraOrtografica(); // Constructor por defecto
    void setHeight(float h); // Cambia la altura del cuadro que saldrá en la foto (zoom)
    void getParam(Punto &posicion, Vector &hacia, Vector &vertical,
        float &alt, float &anch, float &cca, float &ljs)const; //Devuelve configuración
};
```



Cámara ortográfica y cámara perspectiva

```
class CamaraPerspectiva: public Camara
{
protected:
    float verticalAngle;           // Apertura vertical del objetivo
    void setView();               // Calcula de la transformación de la vista
                                // (incluida transformación perspectiva)

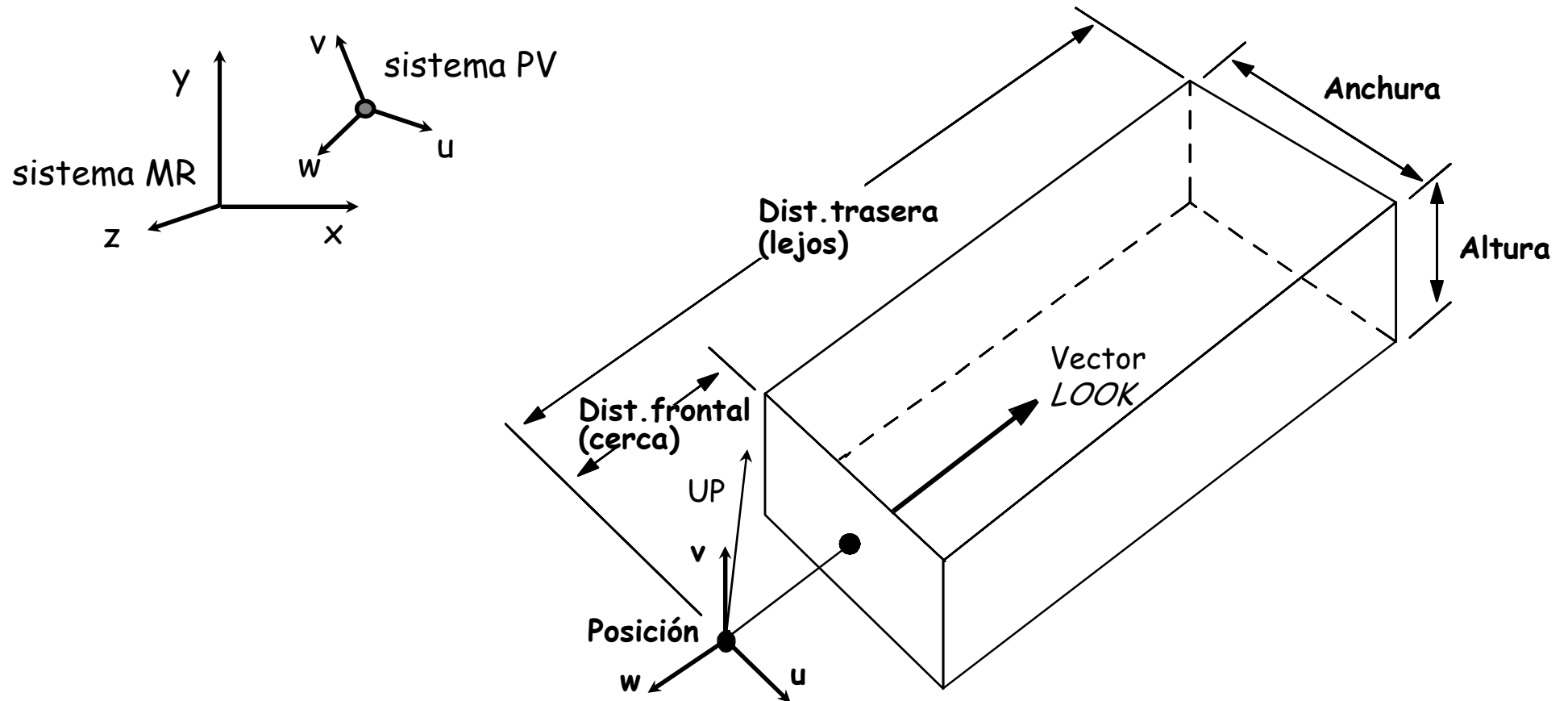
public:

    CamaraPerspectiva();          // Constructor por defecto
    void setVerticalAperture(float av); // Cambia la apertura vertical del objetivo
    void getParam(Punto &posicion, Vector &hacia, Vector &vertical,
                  float &angV, float &angH, float &cerca, float &lejos)const;

};
```

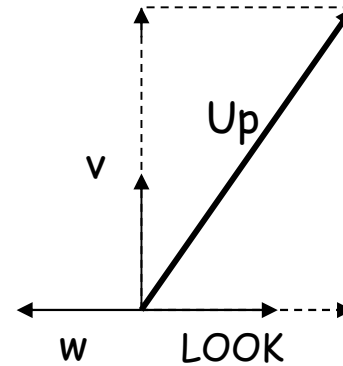
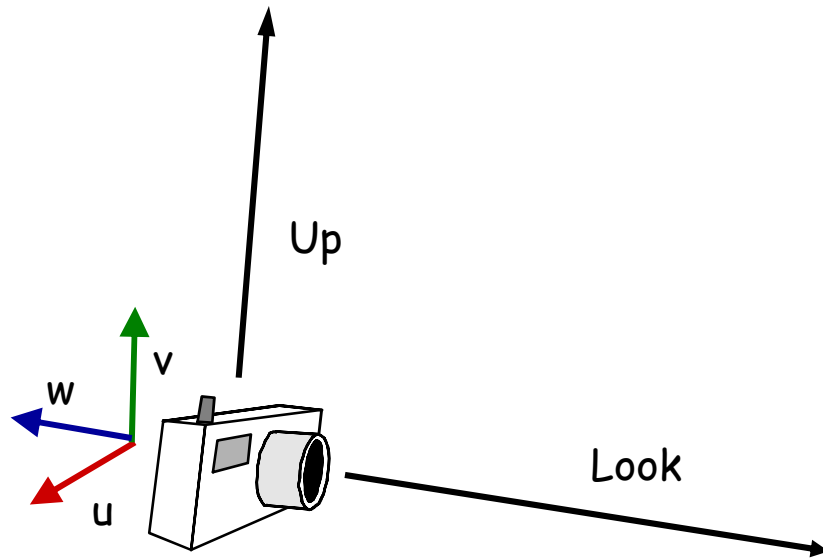

Cámara ortográfica y cámara perspectiva

- Cálculo de u , v y w



Cámara ortográfica y cámara perspectiva

- Cálculo de u, v y w



$$\vec{w} = \frac{-\vec{LOOK}}{|\vec{LOOK}|}$$

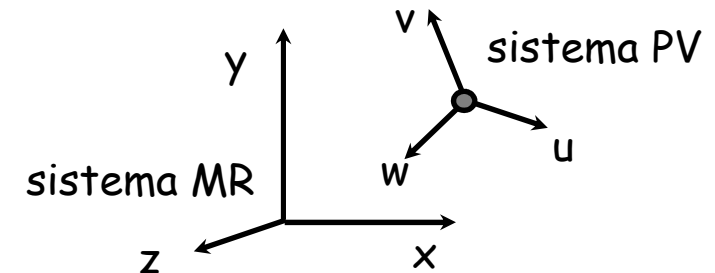
$$\vec{U} = (\vec{UP} \times \vec{w})$$

$$\vec{u} = \frac{\vec{U}}{|\vec{U}|}$$

$$\vec{v} = \vec{w} \times \vec{u}$$

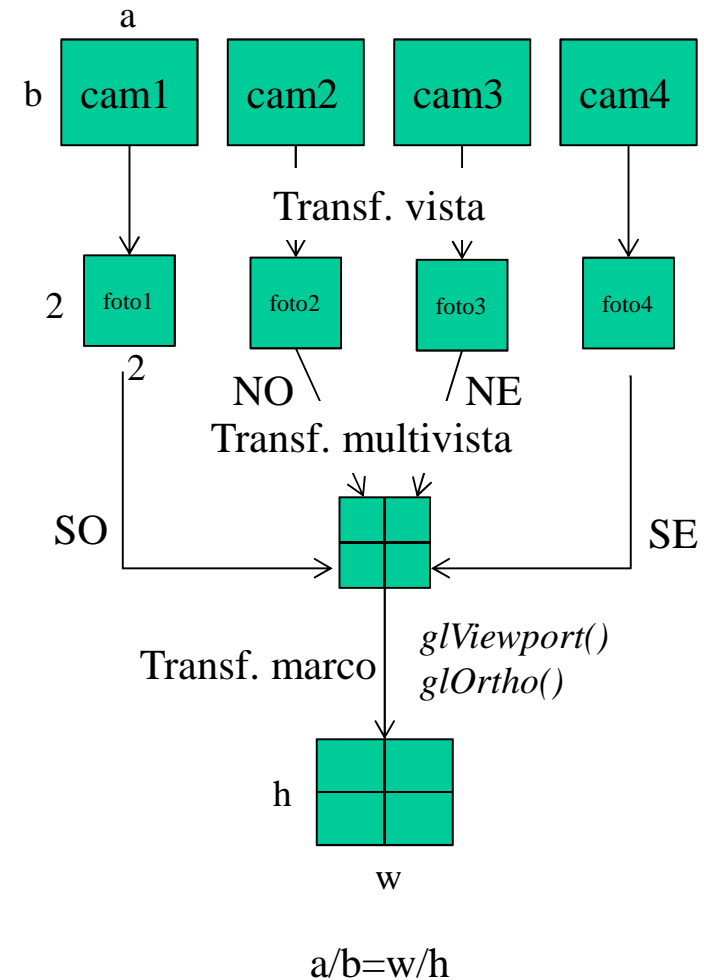
Cámara ortográfica y cámara perspectiva

- Calcular vista *Camara::setView()*
 1. Obtener u, v, w .
 2. $view = Rn(u, v, w) \cdot T(-pov)$
 3. *ready*
- *CamaraOrtografica::setView()*
 4. Calcular el escalado canónico
 5. Acumular la traslación del plano -cerca al origen
 6. $view = Sn() \cdot T(cerca) \cdot Rn(u, v, w) \cdot T(-pov)$ ($0 < cerca < 1$)
 7. Proyección ortográfica (se omite)
- *CamaraPerspectiva::setView()*
 4. Transformación perspectiva-paralela
 5. Escalado canónico
 6. $view = Mpp() \cdot Sn() \cdot Rn(u, v, w) \cdot T(-pov)$
 7. Proyección ortográfica (se omite por usos futuros)
- Fotografiar *Camara::shot()*
 - Si *not ready*, calcular vista
 - Aplicar la transformación de la vista (transformar p con $view$)
 - Calcular el **homogéneo** por si viene de perspectiva
 - Devolver el punto (la x, y es la proyección del punto)



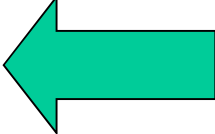
Sistema multicámara

- El objetivo de esta parte es mostrar en la misma pantalla varias vistas de un mismo objeto
- Se utilizan 4 cámaras (mirando al origen)
 - Planta: mirando desde y
 - Alzado: mirando desde z
 - Perfil: mirando desde x
 - Perspectiva: punto de vista a elegir
- Las cámaras deben tener la misma relación de aspecto que el marco
- Se toman 4 fotos, cada una con una cámara
- Se usan 4 transformaciones NO,NE,SE,SO que se aplican a cada foto para situarlas en la zona adecuada de la ventana canónica
- Se aplica la transformación del dispositivo (ventana-marco). Esto lo lleva a cabo OpenGL con las instrucciones `glViewport()` y `glOrtho()`

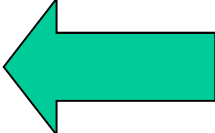


Sistema multicámara

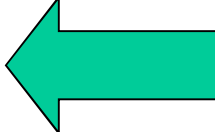
```
#define ALTO 8.0          //Medida de la ventana del mundo real
CamaraPerspectiva canon; //Cámaras
CamaraOrtografica cenital,perfil,frontal;
//Transformaciones que se aplican a las fotos para situarlas en la ventana
Transformacion NO,NE,SO,SE;
...
void myinit(void){
    ...
    //Camara perspectiva
    canon.at(Punto(2.0,0.5,3.0));
    canon.lookAt(Punto());
    canon.setVertical(Vector(0.0,1.0,0.0));
    //Camara en el techo
    ...
    //Transformaciones de las fotos
    NO.translation(Vector(-0.5,0.5,0.0));
    NO.scale(0.5,0.5);
    ...
}
...
void reshape(int w, int h){
    ...
    //Mantener la razón de aspecto
    canon.setAspectRatio(w/(float)h);
    ...
}
```



Configurar
cada cámara



Definición
multivista



Razón de
aspecto

Sistema multicámara

//Dibujo

```
void display(void)
{
```

```
    ...
```

```
    //Dibujo vista perfil
```

```
    displayAxis(perfil, SO);
```

```
    displayShape(perfil, SO);
```

```
    ...
```

```
    glFlush();
```

```
}
```

```
void displayAxis(Camara &cam, Transformacion multivista)
```

```
//Fotografía con la cámara cam y aplica multivista
```

```
{
```

```
    ...
```

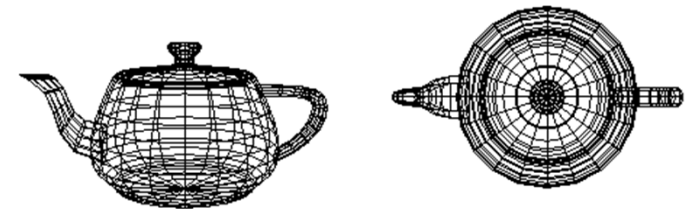
```
}
```

```
void displayShape(Camara &cam, Transformacion multivista)
```

```
{
```

```
    ...
```

```
}
```



Valoración de la práctica

- La práctica puntúa **1,5 puntos**
- **Parte mínima.** Se obtienen **0,75 puntos** si:
 - Se construye correctamente la clase *Camara* y sus derivadas según los requisitos
 - Dibujo correcto de *CuboPar* y *CuboPer*
 - Se construye un programa que dibuje un *Cubo* y unos ejes, usando cuatro cámaras (planta, alzado, perfil y perspectiva)
- **Parte adicional.** Se valorará, para los **0,75 puntos** restantes, lo siguiente:
 - La interacción mediante ratón en la vista perspectiva para variar el punto de vista (posición y zoom) manteniendo el punto de interés (origen)
 - La animación de la figura en la vista perspectiva
 - El dibujo de una tetera utilizando *Teapot.h* y *SuperficieBezier*
 - La edición interactiva de la figura geométrica por arrastre de puntos en las vistas de planta, alzado y perfil actualizándose el resto de vistas convenientemente

