

③ Sistema de color que se usa en OpenGL, indicando qué hace cada parte, cada variable y cómo se operan.

El sistema de color empleado en OpenGL es el sistema RGB. RGB significa Red, Green, Blue y son los colores primarios aditivos. A cada uno de estos valores se le asigna un valor, en OpenGL generalmente un valor entre 0 y 1. El valor 1 significa la mayor cantidad posible de ese color y 0 ninguna cantidad. Podemos mezclar estos 3 colores para obtener una gama completa de colores. Por ejemplo, rojo puro se representa como  $(1,0,0)$ , azul  $(0,0,1)$  y verde  $(0,1,0)$ . Blanco es la combinación de los tres  $(1,1,1)$  y negro (la ausencia  $(0,0,0)$ )

④ Parámetros para definir la cámara y cómo se usan para obtener la transformación de vista.

VRP (View Reference point) : punto donde está el ojo (la cámara) Observador fijo de la escena

VUP (View up) : indica la dirección "hacia arriba". Vector

VPN (View plane normal). vector libre perpendicular al plano de proyección de la imagen, indica la dirección en la que se está mirando. Apunta hacia la cámara.

Para obtener la transformación de vista realizaremos un cambio del sistema de referencia. Trasladar y rotar la cámara para que su sistema de coordenadas coincida con el del mundo. Para ello creamos la matriz V que consigue alinear 2 sistemas de coordenadas

$$V = R[\gamma, z] \cdot R[\beta, Y] \cdot R[\alpha, X] \cdot T[-VRP_x, -VRP_y, -VRP_z]$$

donde  $\alpha, \beta, \gamma$  y  $\gamma$  son los ángulos de Euler

rotación en  $x$  a grados de

$$\alpha = \text{VPN}$$

$$\mu = \text{VUP} \times \text{VPN}^{\wedge}$$

$$v = \lambda \times u$$

Hay que rotar  $\alpha, \mu, v$  en  $x, y, z$

primero  $x$  y lo que nos da la rotación en  $Y$  y lo  $\alpha + \beta + \gamma$

## ⑤ Pasos para utilizar una imagen como textura.

Para poder aplicar una textura a la superficie de un objeto, es necesario hacer corresponder cada punto  $p = (x, y, z)$  de su superficie con un punto  $s_p = (u, v)$  del dominio de la textura. Podemos ver la textura como un envoltorio. Para aplicar la textura debemos:

- Pasar la imagen que está en forma matricial en un sistema de coordenadas normalizado, con las coordenadas  $u$  y  $v$ , siendo sus rangos entre 0 y 1. Esto permite independizar el tamaño real de la imagen de su aplicación al modelo.
- Despues asignamos a cada punto del modelo las coordenadas de textura correspondientes.

## ⑥ Pasos en OpenGL para que una escena se vea iluminada.

- Especificar las normales de los vértices.
- Dar paso a la corriente eléctrica (dando al interruptor general de la iluminación) glEnable(GL\_LIGHTING) → se deja de utilizar glColorPointer y ahora hay que utilizar materiales. Prepara a OpenGL para que utilice corriente eléctrica.
- glEnable(GL\_LIGHTi) encendemos las luces que queramos
  - definir los componentes ambiental, difusa y especular de la luz que hemos activado. glLightfv(GL\_LIGHTi, GL\_AMBIENT, vec3)  
glLightfv(GL\_LIGHTi, GL\_DIFFUSE, vec3)  
glLightfv(GL\_LIGHTi, GL\_SPECULAR, vec3)
  - definir la posición o dirección de la i-ésima fuente de luz glLightfv(GL\_LIGHTi, GL\_POSITION, pos3)
  - Hay que definir los materiales. Si no sirve de nada aplicarlos a los

## ② Normales: vectores de longitud unidad

- Normales de caras: vector unitario perpendicular a cada cara, de longitud unidad, apuntando al exterior de la malla. Calculamos la tabla de normales de las caras. Para ello hay que recorrer la tabla de caras de la malla y considerar las posiciones de sus vértices. Se calculan los vectores  $\vec{a}^j = q - p$  y  $\vec{b}^j = r - p$  <sup>vector de los siguientes</sup>, y el producto vectorial de ambos nos da  $\vec{m}^j$  el cual es ~~normal~~ perpendicular a la cara. Bastaría con normalizar  $\vec{m}^j$  para obtener el vector normal  $\vec{n}_c = \frac{\vec{m}^j}{\|\vec{m}^j\|}$

- Normales de vértices. vector unitario perpendicular al plano tangente a la superficie en la posición del vértice. Para cada vértice, el vector  $\vec{m}_v$  es un vector aproximadamente perpendicular a la superficie de la malla en la posición del vértice. La suma de los vectores normales de todas las caras adyacentes a dicho vértice.

vértice:  $\vec{m}_v = \sum_{i=0}^{k-1} \vec{m}_i$  donde  $\vec{m}_i$  es el vector perpendicular a la  $i$ -ésima cara adyacente al vértice. Hay que normalizar dicho vector.  $\vec{n}_v = \vec{m}_v / \|\vec{m}_v\|$

Para asociar colores a las normales:

```
void glNormalPointer (type, (size) stride, pointer);
```

```
void glColorPointer (size, type, stride, pointer);
```

## Ejercicios exámenes

① ¿Qué es la transformación de vista?

La transformación de vista es una transformación que permite cambiar el sistema de coordenadas. Esta transformación permite simular el posicionamiento de la cámara en cualquier posición y orientación. A partir de esos parámetros se define el nuevo sistema de coordenadas. El nuevo S.C se alinea con el S.C.M (sistema de coordenadas del mundo) aplicando transformaciones geométricas: 1 traslación y 3 rotaciones.

¿Qué parámetros de la cámara están implicados?

- VRP: Posición donde está la cámara. Origen del S.C.V. Es un punto dado en el S.C.M
- VUP: Indica la orientación hacia arriba. Es un vector dado en el S.C.M
- VPN: Hacia donde mira la cámara. Vector en S.C.M

¿Qué matriz de OpenGL almacena dicha transformación? Crear un ejemplo

```
glMatrixMode(GL_MODELVIEW);
```

```
glLoadIdentity();
```

```
glTranslatef(0,0,-10);
```

```
glRotatef(37,1,0,0);
```

```
glRotatef(45,0,1,0);
```

## Técnicas de animación

### • Animación usando fotogramas clave (keyframe) : interpolación

El animador define los parámetros de la animación en ciertos instantes concretos y el ordenador calcula los fotogramas intermedios mediante interpolación.

### • Animación por esqueletos

Un esqueleto es un modelo simplificado del personaje formado por segmentos rígidos unidos por articulaciones. Vinculados a estos segmentos se sitúan las mallas poligonales que representan la superficie del objeto animado, en lo que se denomina piel. Es un modelo jerárquico.

### • Animación procedural

Comportamiento del objeto se describe mediante un procedimiento o función. Comportamiento fáciles de generar pero difíciles de simular físicamente (rotura de un vidrio)

### • Animación por simulación física

animaciones que se pueden realizar usando las leyes de la mecánica clásica. Existen librerías que se encargan de simular detección de colisiones y dinámica de sólidos rígidos o deformables.

## ILUMINACIÓN

En OpenGL se suele usar el modelo de partículas. En este modelo, la radiación se puede describir de forma idealizada como un flujo en el espacio de partículas puntuales llamadas fotones, con trayectorias rectilíneas. En un punto P de la superficie de un objeto podemos medir la densidad de energía radiante en un instante de tiempo de los fotones de una longitud de onda concreta ( $\lambda$ ) que ponen en una determinada dirección (v).  $L(\lambda, p, v)$  se denomina radiancia y determina el tono de color y el brillo con el que observaremos el punto p cuando lo vemos desde la dirección v.

- Brillo dependrá de la cantidad de fotones
- Color " de la distribución de las longitudes de onda de dichos fotones.

Espacio RGB  $\rightarrow (r, g, b)$  donde  $0 \leq r, g, b \leq 1 \leftarrow$  máxima potencia  
 (conjunto de todas las ternas RGB)  $\uparrow$   
 color no aparece

Simplificación de radiación reflejada en OPENGL

luz directa      luz indirecta  
 1 punto      varios puntos  
2 reflexiones (especular y difusa)

Fuentes de luz  $\leftarrow$  posicionales  
 direccionales

Materiales : modelo de reflectancia de Phong

IMP

color del pixel depende  
 también de los valores de  
 color de las fuentes de luz que intervienen

Definición de Fuentes de luz : tipos y atributos

Activación / desactivación  $\rightarrow \text{glEnable} / \text{glDisable} (\text{GL\_LIGHT } i)$

Configuración de colores de una fuente  $\rightarrow \text{glLightfv} (\text{GL\_LIGHT } i, \text{GL\_POSITION}, \text{pos});$  ;  $\text{GL\_AMBIENT}$ , terna RGB  
 $\text{GL\_DIFFUSE}$  cuatro componentes  
 $\text{GL\_SPECULAR}$  con respectivos colores );

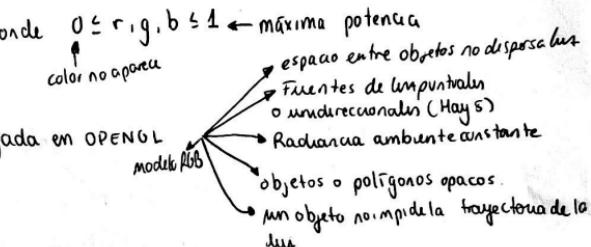
Posición / dirección  $\text{glLightfv} (\text{GL\_LIGHT } i, \text{GL\_POSITION}, \text{pos});$  ;  $\text{dir} \rightarrow (x, y, z, w)$  ;  $0 \rightarrow$  luz direccional

Observador local o infinito  $\text{glLightModel} (\text{GL\_LIGHTMODEL\_LOCAL\_VIEWER}, \text{0}) \rightarrow$  infinito (proyectivo)

$\text{GL\_FALSE} \rightarrow$  infinito (proyectivo)  
 ó  
 $\text{GL\_TRUE} \rightarrow$  perspectiva.

Especificación normales.

Definir atributos de materiales



Simplificaciones OpenGL sobre el modelo de iluminación para conseguir rasterizar de forma eficiente las escenas.

1. Las fuentes de luz son puntuales o unidireccionales, hay un nº finito de ellas (8)
2. No se considera la luz incidente en una superficie que no provenga directamente de las fuentes de luz. Para suplir esos fotones que andan "reboteando" por la escena, se crea una radiancia ambiente constante.
3. Los objetos o polígonos son totalmente opacos (sin transparencias)
4. Un objeto no impide la trayectoria de la luz (no se consideran sombras arrojadas)
5. El espacio entre los objetos no dispersa la luz (misma densidad independientemente de la distancia a la fuente).
6. En lugar de considerar todas las ondas posibles, se usa el modelo RGB.

Rotaciones:

$R_x[\alpha]$

$$f_y(p) = \cos(\alpha)p_x - \sin(\alpha)p_z$$

$$f_z(p) = \sin(\alpha)p_y + \cos(\alpha)p_z$$

$R_y[\alpha]$

$$f_x(p) = \cos(\alpha)p_x + \sin(\alpha)p_z$$

$$f_z(p) = -\sin(\alpha)p_x + \cos(\alpha)p_z$$

$R_z[\alpha]$

$$f_x(p) = \cos(\alpha)p_x + \sin(\alpha)p_y$$

$$f_y(p) = \sin(\alpha)p_x + \cos(\alpha)p_y$$

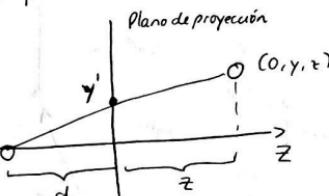
Rotación de  $X(p, i)$  de un punto  $p$ :

$$\begin{bmatrix} p_x' & p_y' & p_z' \end{bmatrix} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

## Proyección perspectiva

Calculo valor coordenadas 2D de un punto 3D

$$\frac{y}{z+d} = \frac{y'}{d}$$



$z$  es la distancia desde el punto al plano de proyección y  $d$  la distancia desde el punto de proyección al plano

near  $y' = \frac{d \cdot y}{z+d}$  y esto hace ver por qué los objetos más alejados

se ven más pequeños;

$$x' = \frac{d \cdot x}{z+d}$$

## Proyección ortogonal

Son un caso especial de proyecciones paralelas, en las cuales los proyectores son perpendiculares al plano de proyección, el centro de proyección está a una distancia infinita del plano.  $d=\infty$ .

## Definición de la matriz de proyección en OpenGL

glMatrixMode(GL\_PROJECTION); → Hasta que digamos lo contrario, todas las operaciones se aplicarán sobre la matriz de proyección.  
glLoadIdentity();

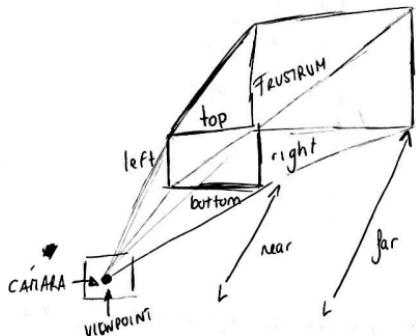
Perspectiva → glFrustum(left, right, bottom, top, near, far);

Ortográfica → glOrtho(left, right, bottom, top, near, far);

## Recortado

Se conserva cierto valor de  $z$  que permite pasar de coordenadas de vista a coordenadas de recorte. Una vez que se tienen las coordenadas de recorte de los vértices, se comprueba qué primitivas están dentro o fuera del volumen de visualización.

## Volumen de visualización



FRUSTUM = Volumen de visualización  
parte de la escena que se considera  
a la hora de generar la imagen 2D.

Todo lo que esté mas cerca o lejos  
respectivamente del plano se ignora

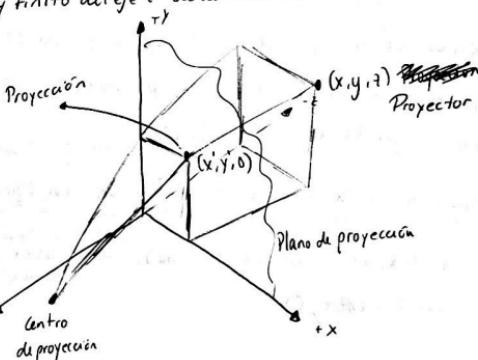
## La transformación de Proyección

### Proyecciones

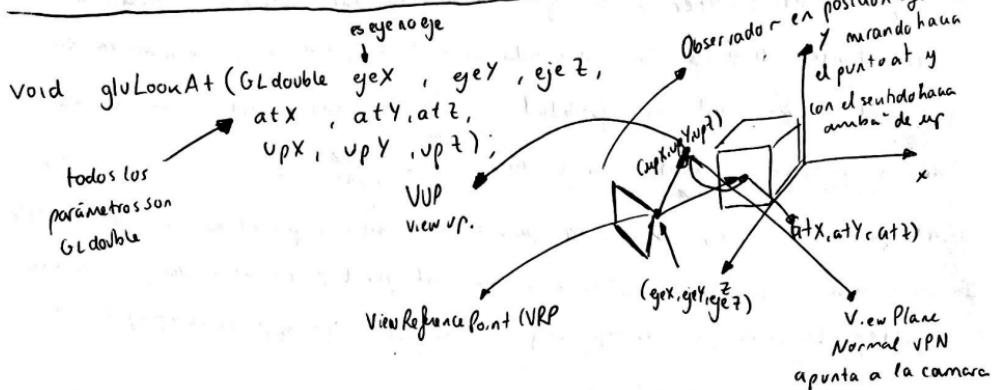
Ortográfica : se consigue ubicando el centro de proyección en el infinito del eje  $\vec{z}$  de la cámara de forma que todos los proyectores son paralelos  
Perspectiva con un punto de fuga : se genera con un único centro de proyección  
en un punto concreto y finito del eje  $\vec{z}$  de la cámara.

### Elementos :

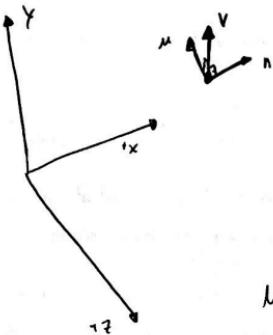
- Proyector: recta que pasa por el punto  $a$  y el centro de proyección.
- Centro de proyección: punto donde pasan y convergen todos los proyectores
- Plano de proyección: plano 2D donde se forma la imagen



# CÁMARA : TRANSFORMACIÓN DE VISTA Y PROYECCIÓN



## SISTEMA DE REFERENCIA DE LA CÁMARA



$n \rightarrow$  vector VPN

$u \rightarrow$  perpendicular al plano que forman VUP y VPN

$$u = \frac{VUP \times VPN}{\|VUP \times VPN\|}$$

$$v \rightarrow \text{ortogonal a } n \text{ y } u \quad v = \frac{n \times u}{\|n \times u\|}$$

Una transformación de vista no es más que un cambio de sistema de referencia. Trasladar y rotar la cámara para que su sistema de coordenadas coincida con el del mundo

El algoritmo de ZBuffer es un algoritmo utilizado para calcular el pixel más cercano al plano de proyección. Para cada primitiva que se dibuje, se almacene en el pixel del Zbuffer su valor de profundidad normalizado dentro del volumen de visualización, es decir 0 (<sup>negro</sup><sub>dibujar</sub>) lo más cercano, y 1 (<sup>blanco</sup><sub>lejos</sub>) lo más cercano al plano far. De esta forma si el valor de  $z$  de la primitiva es menor que el actualmente existente en la posición del buffer, se sustituye el color del pixel por ese nuevo valor y se actualiza el Zbuffer. La principal ventaja es que es el peor de los casos su complejidad es proporcional al nº de primitivas.

## Sistemas de partículas

Un sistema de partículas es un conjunto de muchas diminutas partículas que juntas representan un objeto difuso. Ej: fuego. Una partícula en singular es un elemento con una serie de propiedades comunes a todas las partículas de su clase, pero con distintos valores para cada una de las partículas.

Ciclo de vida:

- Generación: se generan aleatoriamente en el emisor, para el cual se ha definido una forma y posición determinada que puede variar. Valores iniciales pueden seguir un patrón o ser aleatorios.
- Dinámica de partículas: atributos o propiedades de las partículas cambian en cada frame. Algunos atributos pueden definirse en función de otros. Esta interacción puede ser controlada por motores físicos más o menos complejos.
- Extinción. Cada partícula tiene dos propiedades que controlan su existencia: edad y tiempo máximo de vida. Edad: nº frames lleva viva partícula. T.M.V: cuando edad = T.M.V muere. Extinción prematura:
  - Se sale del área de visión
  - Golpea el suelo una chispa de fuego
  - algún atributo alcanza un umbral

## Selección en OpenGL

### Buffer de selección

Miando el modo GL\_SELECT OpenGL devuelve el nº de objetos seleccionados como resultado de selección y la información asociada a estos en un buffer de enteros. OpenGL devuelve en el buffer de selección la secuencia de objetos pintados en el pequeño volumen de recorte generado.

### Selección ~~para~~ con codificación por colores

Usar un código de color por cada objeto seleccionable. Se trata de crear una función de dibujado distinta para cuando queremos seleccionar, y cuando el usuario hace clic, se pinta la escena "para seleccionar" en el buffer trásero y se lee el color del pixel donde el usuario ha hecho clic. Si no se hace un intercambio de buffers, el programa seguirá su proceso natural. Pasos:

- Llamar a `dibuja_seleccion()`
- Leer el pixel `(x,y)`
- Averiguar a qué objeto hemos asignado el color de dicho pixel
- No intercambiar buffers.

3 formas:

- Intersección rayo-escena : desde el pixel hasta donde está mirando el observador, lanza un rayo y ver donde colisiona.

1. Usando el volumen de visualización  
- Recortando subvolumen centrado en la posición dada. Recortar el volumen de visualización sólo al pixel donde se ha hecho click y ver qué objeto se está viendo porque es el que está más cercano.

2. Codificando el id de objeto como color y leyendo el framebuffer

3. Usando una pila de nombres

4. Asignando un nombre a cada objeto

5. Comprobando si el nombre coincide

6. Creando el nombre

7. Comprobando si el nombre coincide

8. Enlazar el programa de shaders

9. Especificar que ese programa de shaders se usará en lugar de las funciones del pipeline fijo.

## Fragment shader

Procesador de fragmentos / píxeles, operan sobre estos para determinar su color.

Variables:

- Uniformes. proporcionadas por el sistema o por la aplicación. Variables globales solo lectura.
- Entrada. variables de entrada son las de salida de la etapa anterior, que vienen de otros procesadores. Son los datos que se interpolan dentro de cada primitiva para obtener información suficiente para el fragment shader y poder calcular el color de cada pixel.
  - Color ( $\text{fFragColor}$ )
  - Coordenadas de textura. ( $\text{vST}$ )
- De salida. debe contener al menos una variable de salida para almacenar el color del pixel  $\rightarrow \text{out vec4 fFragColor.}$

**IN DEL FS DEBEN COINCIDIR UNO A UNO CON OUT DEL VERTEX SHADER**

Posición Fragmento

Variables de Entrada

Variables Uniformes

Fragment Shader   $\text{fFragColor}$

Textures

## Esquema aplicación OpenGL 4x

1. Leer código fuente shaders y guardarlo
2. Leer archivos en una cadena de texto
3. Crear un objeto shader para cada una de las cadenas que contienen al shader
4. Asignar a cada objeto la cadena con el código fuente de su shader.
5. Compilar cada objeto
6. Crear un programa shader global
7. Conectar shaders con programa global
8. Enlazar el programa de shaders
9. Especificar que ese programa de shaders se usará en lugar de las funciones del pipeline fijo.

**NO ES LO MISMO UN SHADER  
QUE UN PROGRAMA SHADER**

## PROGRAMACIÓN EN GPU

Shaders están escritos en un lenguaje denominado GLSL (OpenGL Shading Language)

Encaya mejor en los procesadores gráficos

Vertex shader se ocupa de la geometría y fragment shader de generar la imagen final.

### Vertex shader

Toma la información de los vértices y su entorno que estén almacenados en el sistema OpenGL y los prepara en un conjunto de variables uniformes y de atributo, de forma que se puedan realizar cálculos sobre éstos.

Procesador de vértices trabaja sobre geometría, normalmente en coordenadas del modelo y produce geometría que está referenciada en el sistema de coordenadas de vista 3D.

La parte primordial del vertex shader es tomar las variables de atributo y usarlas o copiarlas en variables de salida para posteriores shaders.

Existen varios tipos de variables:

- Variables de atributo. Almacenan las propiedades del vértice que se está procesando y son variables de entrada. Sólo leídas en vertex shader.
- Variables uniformes. Constantes a lo largo del procesamiento de una primitiva, sólo de lectura.
- Variables de salida. Resultados del procesamiento, se usan como variables de sólo escritura.

Variables obligatorias → GL\_POSITION

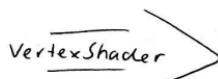
Trabajo del procesador de vértices:

Posición del vértice

Variables de atributo

Variables uniformes

Texturas



Variables de salida (y color)  
gl\_Position.

Pasos para utilizar una imagen como textura.

1. Hacer corresponder a cada punto p de su superficie con un punto  $s = (u, v)$  del dominio de la textura.
2. Asignar las coordenadas de textura en el objeto
3. Activar las texturas en OpenGL con `glEnable(GL_TEXTURE_2D)`; para usarlas sustituyendo la textura al color
4. Cargar las texturas en el sistema gráfico.  
Alojaren memoria RAM una matriz con los colores de sus texels.
  - 4.1 Crear o generar un nuevo identificador de textura único.
  - 4.2 Activar la textura con id 'idTex' `glBindTexture(GL_TEXTURE_2D, idTex);`
  - 4.3 Especificar cuál será la imagen de textura asociada al identificador `tex;`   
textura activa con `glTexImage2D();` Podemos usar con GLU `glBuild2DMaps.`
5. También a la hora de dibujar es necesario indicar antes donde está la tabla de coordenadas de textura con `glTexCoordPointer.`

Texel: se representa con 3 bytes  $l_{tex} (0 \text{ y } 255)$  que codifican la proporción rojo verde y azul  
(texel element)