Tema-1-Resumen.pdf

LosCocos

Informática Gráfica

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

WUOLAH

El más PRO del lugar puedes ser Tú.

¿Quieres eliminar toda la publi de tus apuntes?

¡Fuera Publi!
Concéntrate al máximo

Apuntes a full.
Sin publi y sin gastar coins

Para los amantes de la inmediatez, para los que no desperdician ni un solo segundo de su tiempo o para los que dejan todo para el último día.

Quiero ser PRO

4,95 / mes

TEMA 1: INTRODUCCIÓN A LA INFORMÁTICA GRÁFICA

EL PROCESO DE VISUALIZACIÓN

En aplicaciones interactivas 3D, hay tres elementos básicos: el usuario, la escena digital 3D y la imagen sintética.

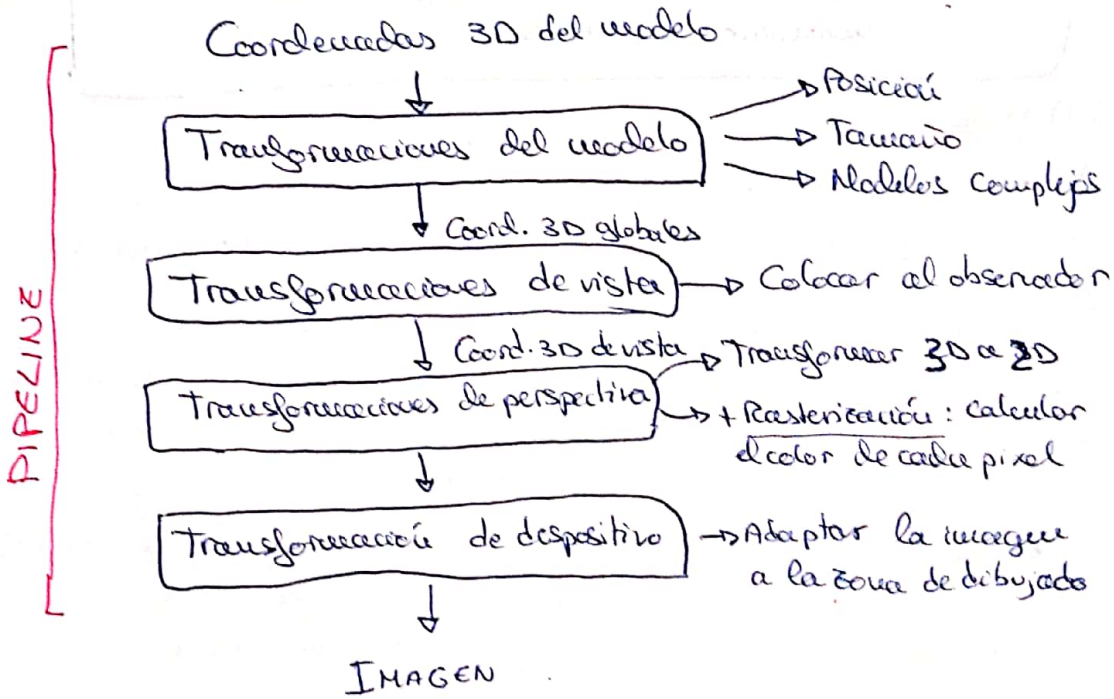
Para poder formar una imagen sintética necesitamos:

- Una persona o un objeto: con color y forma.
- Una cámara
- Luz (pues sin luz todo se vería negro).

En Ingeniería informática, necesitamos:

- Modelos digitales 3D de lo que queremos representar
  - Geometría
  - Propiedades de apariencia (textura, color...)
- Iluminación
- Una o varias cámaras, indicando para cada una
  - Lente (ángulo de visión)
  - Posición y orientación

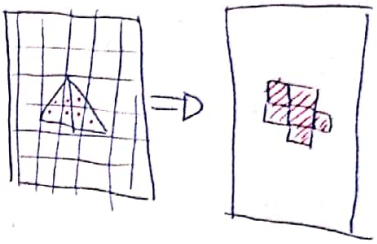
PROCESO PARA GENERAR UNA IMAGEN 2D A PARTIR DE UN MODELO 3D



RASTERIZACIÓN

Malla con colores, cuadrados y cantidades de luz. No refleja la luz ni tiene sombras.

```
Inicializar el color de todos los píxeles
Para cada primitiva P de la escena
    Encontrar conjunto S de píxeles cubiertos por P
    Para cada pixel s en S:
        calcular color de P en S
        actualizar color en s
```



RAY-TRACING

Generación mediante rayos de luz. Refleja la luz y posee sombras.
Es más lento pero consigue unos resultados más realistas.
El trazado de rayos consiste en seguir el camino que hace un rayo de luz desde la imagen hasta la fuente de luz, pasando por la cámara y rebotando en todos los objetos

```
Inicializar el color de todos los píxeles
Para cada pixel S de I:
    Encontrar T conjunto de primitivas que cubren a S.
    Para cada primitiva P en T
        calcular color de P en S
        actualizar color en S
```

OPEN GL

OpenGL es una API independiente de la plataforma cuyo objetivo es la creación de imágenes por rasterización independiente del hardware, lenguaje de prog., SO...
Se centra en procesar una serie de datos relativos a la imagen, crear un buffer de memoria, el framebuffer, y almacenar la imagen 2D rasterizada. No se preocupa de dónde va la imagen, por lo que tendremos que usar otras API's de interfaz de usuario para gestionar las ventanas y eventos: GLUT, java...

- ARQUITECTURA CLIENTE/SERVIDOR: CPU cliente y GPU servidor. OpenGL actúa como una máquina de estados que envía peticiones a la GPU y cambios de estado que son aplicados a no ser que se diga lo contrario.
- PIPELINE: cuando damos a dibujar, el programa hace procesamiento de primitivas, colores, texturas, rasterización, ... (>2.0)

INICIO APLICACIÓN

- glutInit(); glutInitDisplayMode(...); //Inicializar OpenGL
- glutInitWindowPosition(...); glutInitWindowSize(...);
  glutInitCreateWindow(); //Inicializaciones ventana.
- glutInitDisplayFunc(); glutInitReshapeFunc(); // Función de dibujado y redimensionado
- glutMainLoop(); //Llega a cuadar la función de dibujado en bucle infinito

PROCESO

glutMainloop mantiene una cola de eventos por cada evento ocurrido pero que no ha sido gestionado por la aplicación.

- Si la cola está vacía → inserta evento idle (desocupado)
- Si no → Se extrae primer evento y es gestionado por la aplicación

- El bucle se para si alguna función gestora cierra la aplicación.

PRIMITIVAS

La primitiva es la unidad mínima de info que puede pintar. Puntos, líneas, triángulos... Toda primitiva es un conjunto de vértices con una determinada organización.

- Un punto es una primitiva con un sólo vértice.

→ Las aplicaciones descomponen las formas complejas en triángulos y estos son enviados a OPENGL a rasterizar (son enviados al cauce gráfico)

MODO → inmediato
→ Posterior a OpenGL 2.0

MODO INMEDIATO

Las primitivas se describen como una secuencia de vértices con glVertex() < 2.0 ó arrays o buffers > 2.0, en sentido antihorario.

```
glBegin(GL_PRIMITIVA);
    glVertex(x,y,z);
    ⋮
glEnd();
```

→ GL_TRIANGLES, QUADS, LINES

PRIMITIVAS OPENGL >2.0

A partir de 2.0 se cambia el estándar al buffer, para no tener que hacer tantas llamadas a la función glVertex, utilizaremos una llamada a la función glDrawArrays() o glDrawElements().

- glDrawArrays(): la idea es tener un array con las coordenadas de los vértices y, en el momento de dibujar, pasarlos todos en bloque a la tarjeta gráfica. Otras propiedades se almacenarán en otros buffers, que se activan para ser tenidos en cuenta.

Cuando usamos glDrawArrays, usamos modo modo de interpretar la secuencia de datos, count elementos empezando por first.

El array donde están los vértices es glVertexPointer

```
float vertices[] = {...};
glEnableClientState(GL_VERTEX_POINTER);
glVertexPointer(3, GL_FLOAT, 0, vertices);
glDrawArrays(GL_TRIANGLES, 0, 3);
```

Sin embargo, OpenGL nos permite usar otra función glDrawElements, para no tener que repetir coordenadas de objetos con múltiples triángulos, a través de un buffer para los vértices y otro para las caras.

INTERCAMBIO DE BUFFERS

Se utiliza al final de la función de redibujado. OpenGL emplea un sistema de doble buffer para el dibujado. De esta forma, la imagen renderizada se realiza sobre una imagen oculta hasta que no se ejecuta el intercambio de buffers. Así evitamos ver como avanza el algoritmo de raster (↗). Esto en animación es muy importante.