Gráficos y Multimedia

Sesión 1: Formato 3DS.

1. Leyendo Recursos en Android

UBICACIÓN RECURSOS (raw)

- Podemos ubicar los recursos en Android en dos carpetas dentro de nuestro proyecto. Son datos privados de nuestra app y sólo se pueden leer:
  - Carpeta ***`proyecto/src/raw`*****:** en esta carpeta podemos colocar archivos en el formato que queramos (datos en bruto), desde texto hasta archivos binarios.
  - Sólo admite nombres en minúsculas.
  - Al estar ubicada en src (recursos) Android genera un ID automáticamente y es accesible desde ***`R.raw.nombre_archivo`***, siendo ésta la forma más rápida de acceder al archivo.
  - No admite una organización jerárquica, no se pueden crear carpetas en `raw`.

1. Leyendo Recursos en Android

UBICACIÓN RECURSOS (assets)

- Podemos ubicar los recursos en Android en dos carpetas dentro de nuestro proyecto. Son datos privados de nuestra app y sólo se pueden leer :
  - Carpeta ***`proyecto/assets:`*** en esta carpeta podemos colocar archivos en el formato que queramos (datos en bruto), desde texto hasta archivos binarios.
  - El nombre puede ser cualquiera (admite mayúsculas).
  - Al ser una carpeta "***apéndice***" no genera ID's para los archivos en ***R***.
  - Admite una organización jerárquica, se pueden crear carpetas en `assets`.
  - Es más lento accediendo a los datos del archivo.
  - Debemos emplear la clase `AssetManager`:

```
AssetManager assetMan = getAssets();
InputStream is = assetMan.open("prueba_asset.xml");
```

1. Leyendo Recursos en Android

UBICACIÓN RECURSOS (memoria)

- Podemos ubicar los recursos en Android externamente a nuestro proyecto, estos serían archivos de escritura y lectura. El usuario tendría acceso a los mimos. Un buen ejemplo sería un reproductor de MP3, el usuario coloca las canciones en formato MP3 en una carpeta y dice su localización a nuestra app:
  - La memoria puede ser interna o externa (desde 4.4 *KitKat* colocar apps en memoria externa es *"complicado"*, pero archivos de datos si se puede).
    - Emplearíamos las clases de lectura de Java pero abriendo el archivo con *`OpenFileInput(`***`"nombre_archivo.ext"`***`).`*
    - Si es memoria externa debemos averiguar la ruta donde está montada la misma.

1. Leyendo Recursos en Android

UBICACIÓN RECURSOS (memoria)

- Ejemplo de lectura de memoria externa:

```
try
{   File ruta_sd = Environment.getExternalStorageDirectory();
    File f = new File(ruta_sd.getAbsolutePath(), "prueba_sd.txt");

    BufferedReader fin =
        new BufferedReader(
            new InputStreamReader(
                new FileInputStream(f)));
     String texto = fin.readLine();
    fin.close();
}
catch (Exception ex)
{
    Log.e("Ficheros", "Error al leer fichero desde tarjeta SD");
}
```

2. Formato 3DS

DESCRIPCIÓN

- Formato de archivos 3D de **Autodesk**.
- El formato 3DS esta formado por pequeños trozos (*chunks*) de datos, sigue una organización jerárquica.
- Es un formato binario, por tanto es rápido de leer y ocupa poco frente a otros formatos.
- Los trozos (*chunks*) que vamos nosotros a leer son:

```
// Identificadores de los trozos (chunks) del archivo que vamos a leer
private static final int CHUNK_MAIN     = 0x4d4d;
private static final int CHUNK_OBJMESH  = 0x3d3d;
private static final int CHUNK_OBJBLOCK = 0x4000;
private static final int CHUNK_TRIMESH  = 0x4100;
private static final int CHUNK_VERTLIST = 0x4110;
private static final int CHUNK_FACELIST = 0x4120;
private static final int CHUNK_MAPLIST  = 0x4140;
private static final int CHUNK_SMOOLIST = 0x4150;
```

Formato 3DS

DESCRIPCIÓN

- Podemos tener varias mallas en el archivo. Habrá que unirlas.
- El número máximo de vértices y polígonos por malla es de 65535 (2^{16}-1).
- No tenemos normales calculadas con precisión, el código que usemos deberá recrearlas y usar en todo caso las listas de suavizado,
  - `0x4150 // Smoothing Group List`
- Emplea tipos sin signo (`unsigned`), que en Java no existen lo cual complica algo su lectura.
- Más información en: http://es.wikipedia.org/wiki/.3ds

Formato 3DS

IMPLEMENTACIÓN

- No podemos emplear la clase `DataInputStream` (tiene métodos como `readShort(), readFloat(),` etc.), ya que sólo permite leer datos grabados con `DataOutpuStream`.
- Emplearemos vectores de `floats` y de `integers` para almacenar los datos temporales y crear al final un `ByteBuffer` (memoria en el JNI, Java Native Interface).
- Debemos leer los datos byte a byte (unsigned) con la clase InputStream y recrear los datos nosotros, debemos poder leer:
  - **`unsigned char (1 byte)`**
  - **`unsigned byte (1 byte)`**
  - **`unsigned short (2 bytes)`**
  - **`unsigned int (4 bytes)`**
  - **`float (4 bytes)`**

Formato 3DS

IMPLEMENTACIÓN. Lectura de float

- Para leer un `float` usaremos este código:

```
public float readFloat(InputStream is) {
  int a, b, c, d;

  try {
      a = is.read();
      b = is.read();
      c = is.read();
      d = is.read();

  } catch (IOException e) {
      throw new RuntimeException("No se pudo abrir el recurso", e);
  } catch (Resources.NotFoundException nfe) {
      throw new RuntimeException("Recurso no encontrado", nfe);
  }
  return Float.intBitsToFloat(a + (b << 8) + (c << 16) + (d <<24));
}
```

Formato 3DS

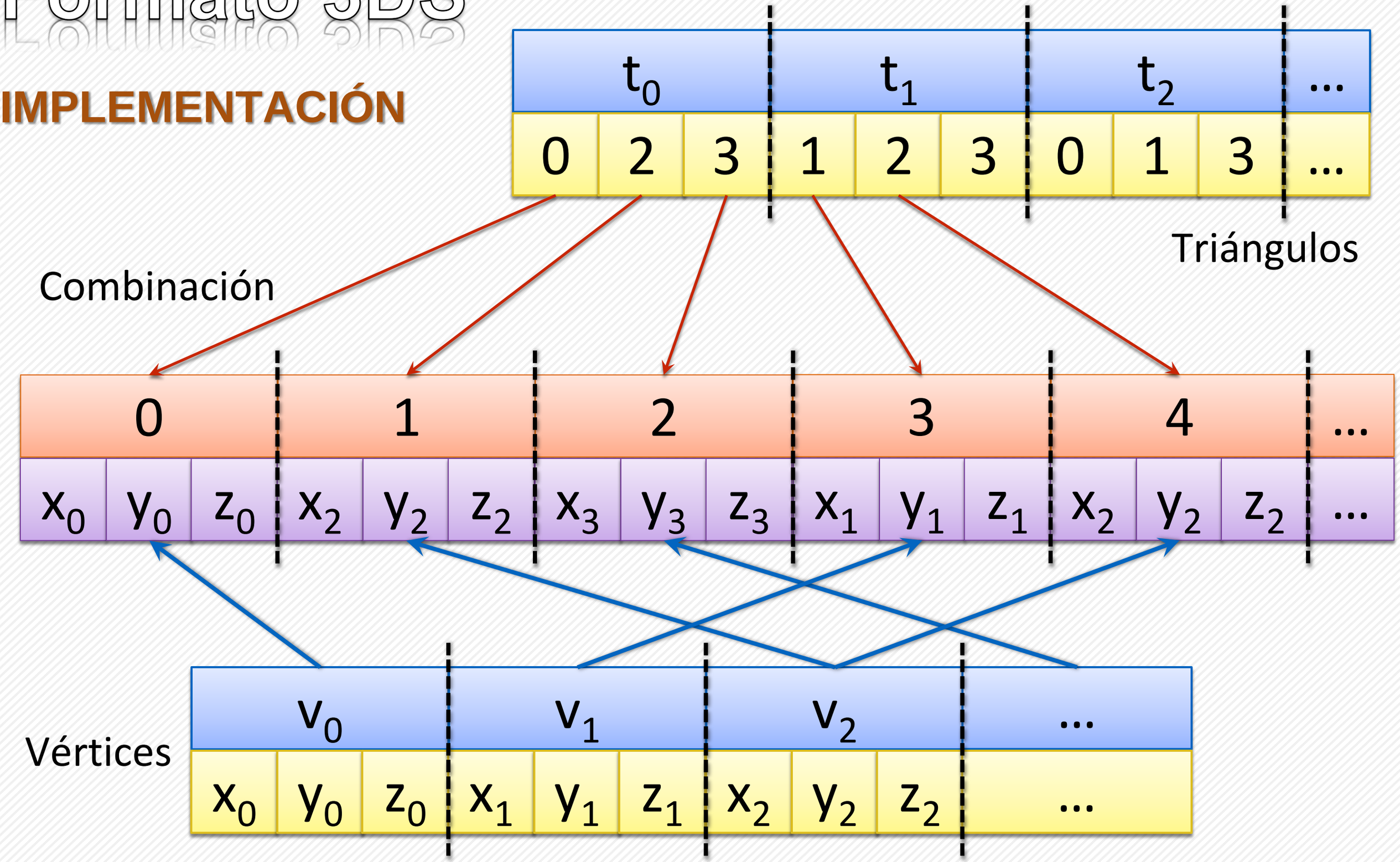
IMPLEMENTACIÓN

- Del archivo 3DS vamos a leer:
  - La lista de vértices, vector de tres `floats` (x, y, z) por vértice.
  - La lista de polígonos, vector de tres `integers` (v_0, v_1 ,v_2) por triángulo.
  - La lista de coordenadas de textura (si existe), vector de dos `floats` (u, v) por vértice. Deberá coincidir en tamaño con el número de vértices.

- Para finalizar deberemos reorganizar (*expandir*) y combinar la información de ambos vectores (vértices y polígonos) de forma que tendremos un vector de tamaño número de polígonos*9 (3 floats por vértice, es decir 3x3=9), esto se realizará de la siguiente manera:

Formato 3DS

IMPLEMENTACIÓN



Formato 3DS

IMPLEMENTACIÓN

- Veamos ahora el código con detalle, en Moodle debemos descargar el archivo:
  - Código de la clase Resource3DSReader v1.0

Formato 3DS

IMPLEMENTACIÓN

- En el constructor de `OpenGLRenderer()` añadimos el siguiente código:

```
// Lee un archivo 3DS desde un recurso
obj3DS = new Resource3DSReader();
obj3DS.read3DSFromResource(context, R.raw.mono);
```

- En la clase declaramos:

```
Resource3DSReader obj3DS;
```

- Y cambiamos la siguiente línea de 2 a 3 (ahora tenemos x, y, z):

```
~~private static final int POSITION_COMPONENT_COUNT = 2;~~
private static final int POSITION_COMPONENT_COUNT = 3;
```

Formato 3DS

IMPLEMENTACIÓN

```
@Override
public void onDrawFrame(GL10 glUnused) {
    // Clear the rendering surface.
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glLineWidth(2.0f);

    // Añadimos una rotación
    final float[] modelMatrix = new float[16];
    setIdentityM(modelMatrix, 0);
    // Rotación de 1º alrededor del eje y
    rotateM(modelMatrix, 0, -1.0f, 0f, 1f, 0f);

    final float[] temp = new float[16];
    multiplyMM(temp, 0, projectionMatrix, 0, modelMatrix, 0);
    System.arraycopy(temp, 0, projectionMatrix, 0, temp.length);
```

Formato 3DS

IMPLEMENTACIÓN

```
        // Envía la matriz de proyección al shader
        glUniformMatrix4fv(uMatrixLocation, 1, false, projectionMatrix, 0);

        // Actualizamos el color (Marrón)
        glUniform4f(uColorLocation, 0.78f, 0.49f, 0.12f, 1.0f);
```

```
        // Dibujamos el objeto
        glDrawArrays(GL_TRIANGLES, 0, obj3DS.numPol*3);

        // Actualizamos el color (Blanco)
        glUniform4f(uColorLocation, 1.0f, 1.0f, 1.0f, 1.0f);

        glDrawArrays(GL_LINES, 0, obj3DS.numPol*3);
        //for (i=0; i<obj3DS.numPol; i++)
            //glDrawArrays(GL_LINE_LOOP, i*3, 3);
    }
```