Fundamentos de VR-AR mediante aplicaciones Gráficas 3D en Android utilizando OpenGL- Parte II

3er Foro Nacional de Tecnologías de la Información y Sistemas Computacionales

Dr. Marco Aurelio Nuño Maganda

Universidad Politécnica de Victoria
Laboratorio de Sistemas Inteligentes
mnunom@upv.edu.mx

28 y 29 de Septiembre de 2023

Outline

OpenGL

- OpenGL significa Open Graphics Library, un estándar para la progamación gráfica.
- Los comandos de gráficos son implementados por el controlador de la tarjeta gráfica y, por lo tanto, son independientes del hardware de la tarjeta gráfica, del sistema operativo y del administrador de ventanas empleado.
- Los comandos de gráficos están cercanos hardware y son suficientes para lograr la funcionalidad principal
- Varias bibliotecas y frameworks se basan en OpenGL y permiten la programación a un mayor nivel de abstracción.

Versiones OpenGL para PC

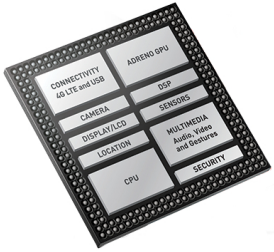
- Desde su introducción (1992) OpenGL se ha ampliado continuamente para admitir nuevas funciones de tarjetas gráficas.
- OpenGL 1.0 (1992), OpenGL 1.1 (1997), OpenGL 1.2 (1998), OpenGL 1.3 (2001), OpenGL 1.4 (2002), OpenGL 1.5 (2003)
- OpenGL 2.0 (2004), OpenGL 2.1 (2006)
- OpenGL 3.0 (2008), OpenGL 3.1 (2009), OpenGL 3.2 (2009), OpenGL 3.3 (2010)
- OpenGL 4.0 (2010), OpenGL 4.1 (2010), OpenGL 4.2 (2011), OpenGL 4.3 (2012), OpenGL 4.4 (2013), OpenGL 4.5 (2014), OpenGL 4.6 (2017)
- A partir de la versión 3.1, el Pipeline de funciones fijo ya no esta soportado, por l ocual es necesario implementar shaders, lo cual dificulta el aprendizaje.

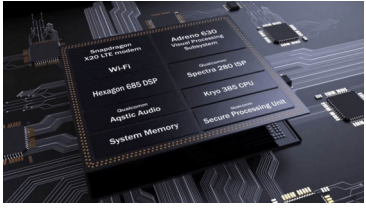
OpenGL ES y WebGL

- OpenGL ES (Embedded System) es una versión de OpenGL con funcionalidad reducida para teléfonos móviles, televisores, tabletas, etc.
- OpenGL ES 1.0 (2003): similar a OpenGL 1.3 (Pipeline Fijo)
- OpenGL ES 1.1 (2004): similar a OpenGL 1.5 (compatible con versiones anteriores)
- OpenGL ES 2.0 (2007): similar a OpenGL 2.0 (no compatible con versiones anteriores)
- OpenGL ES 3.0 (2012): similar a OpenGL 3.3
- OpenGL ES 3.1 (2014): similar a OpenGL 4.3
- OpenGL ES 3.2 (2015): similar a OpenGL 4.3
- OpenGL ES 3.3 (2017): similar a OpenGL 4.6
- WebGL esta basado en OpenGL ES 2.0 (y WebGL 2.0 en OpenGL ES 3.0) y permite gráficos 3D en páginas web (compatible con la mayoría de navegadores)

SoCs para Teléfonos Inteligentes

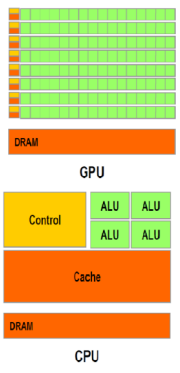
- El término SoC significa system-on-a-chip.
- Un SoC es un sistema completo contenido en un solo circuito integrado.
- La combinación de todos sus componentes en una sola unidad de procesamiento permite un ahorro de energía significativo.
- Bloques comunes en un SoC:
  - Central Processing Unit (CPU).
  - Graphics Processing Unit (GPU).
  - Image Processing Unit (ISP).
  - Digital Signal Processor (DSP).
  - Neural Processing Unit (NPU).
  - Video encoder/decoder.
  - Modems.

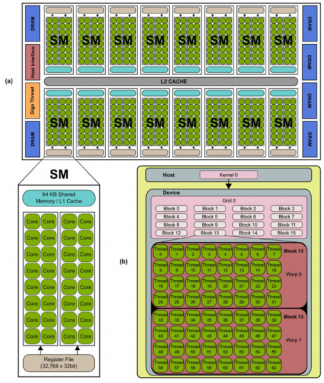




¿Qué es un GPU (Graphics Processing Unit)?

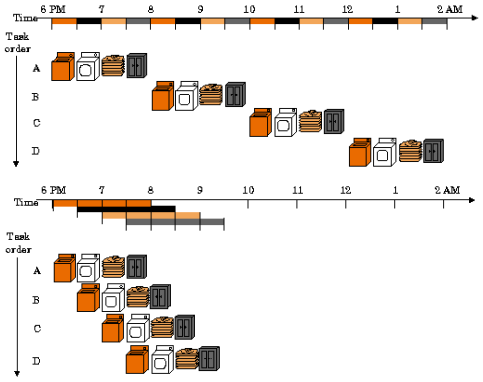
- Un GPU es un procesador formado por muchos núcleos más pequeños y especializados, especializada en operaciones de punto flotante en paralelo.
- Al trabajar conjuntamente, los núcleos ofrecen un desempeño masivo cuando se puede dividir una tarea de procesamiento y es procesada por muchos núcleos.
- Por ejemplo, el Samsung S10 utiliza un Qualcomm Snapdragon 855 SoC (tiene un CPU de 8 núcleos y un GPU dedidado para gráficos (Adreno 640).



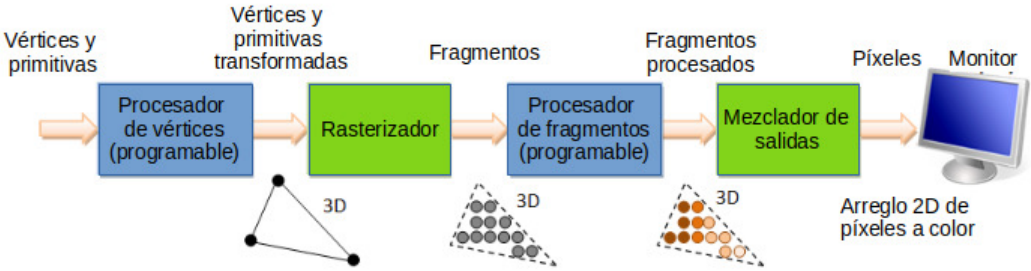


OpenGL Pipeline

- Un Pipeline es una estructura de procesamiento paralelo
- Un GPU tiene dos tipos de procesadores, incorporados en un Pipeline:
  - El vertex processor
  - El fragment processor
- Los programas son cargados en cada procesador.

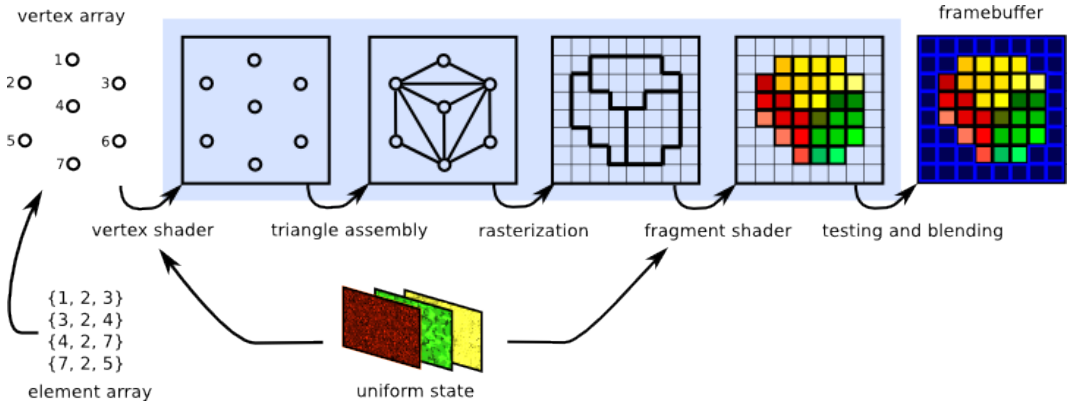


OpenGL Pipeline (2)



Pipeline de rederizado de gráficos 3D: La salida de una etapa alimenta a la entrada de la siguiente etapa. Un vértice, tiene atributos como su posición (x, y, z), color (RGB o RGBA), normales de los vértices (n_x, n_y, n_z), y textura. Una primitiva se compone de uno o más vértices. El rasterizador hace un escaneo de cada primitiva para producir un conjunto de fragmentos alineados a una cuadrícula mediante la interpolación de los vértices

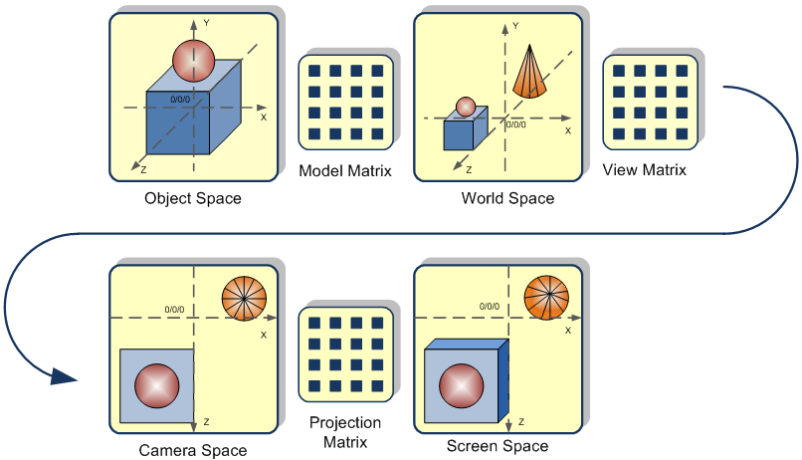
OpenGL Pipeline (3)



OpenGL Shading Language

- Las aplicaciones móviles son ejecutadas principalmente en el CPU y la memoria principal
- Para procesamiento de gráficos, los programas son ejecutados en el GPU el cual tiene su propia memoria local (memoria gráfica).
- Los programas del GPU son escritos en un lenguaje llamado Shading Language (Lenguaje de sombreado).
- La mayoría de los GPUs adoptaron el lenguaje de OpenGL shading Language (OGSL)

Transformaciones



Instrucciones para replicar los proyectos de este taller

- Siempre crear un proyecto Android Studio nuevo, seleccionando JAVA (salvo que se indique lo contrario)
- Poner un nombre significativo al proyecto (Puede empezar por el numero de demo, pero tambien incluir información de su funcionalidad)
- Se provee de una carpeta en el siguiente repositorio, la cual deben descargar. Los proyectos estan ordenados de manera secuencial y todo lo necesario para generar el respectivo proyecto esta dentro de dicha carpeta.

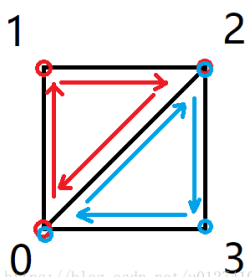
Github

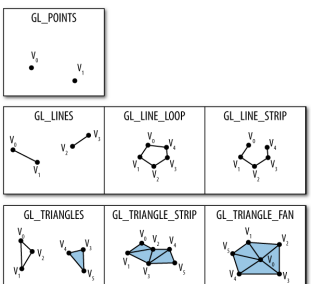
`https://github.com/mnunom-upv/Fundamentos-de-VR-AR-mediante-aplicaciones-Gr-ficas-3D-en-Android-utilizando-OpenGL`

Outline

Dibujo de Primitivas

- Para dibujar cualquier superficie en OpenGL, se debe triangular los vértices adyacentes para formar polígonos (generalmente triangulos).
- Se requieren de dos triangulos para dibujar un cuadrado.

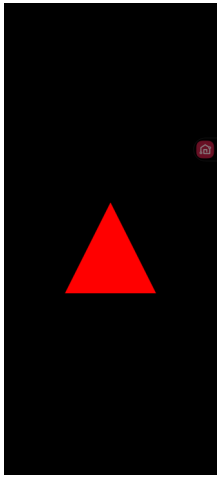




Desplegando gráficos en 2D

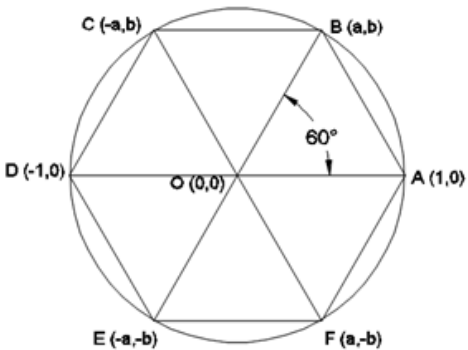
Demo #1: Triangulo 2D en OpenGL ES

- Sustituir el MainActivity.java del proyecto con el MainActivity de la carpeta del demo
- Copiar los archivos Triangle.java, MyRenderer.java y MyView.java en la carpeta del package



Experimentado con Primitivas

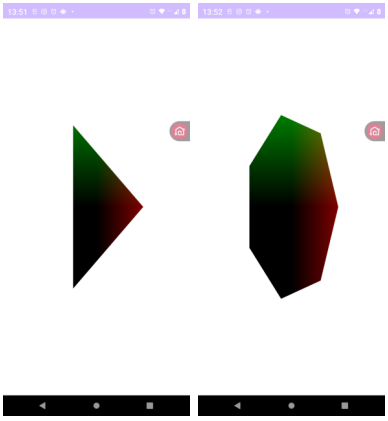
- Es posible dibujar objetos complejos si se tiene un entendimiento correcto de las primitivas
- Por ejemplo: Dibujar un Hexagono Relleno/Alambrico.



Desplegando gráficos en 2D

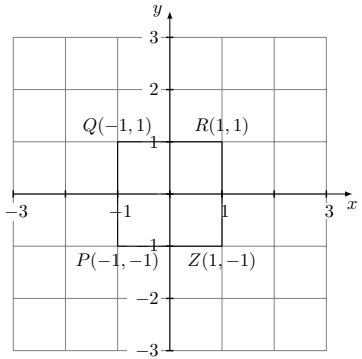
Demo #2: Dibujar un poligono regular de N lados empleando triángulos

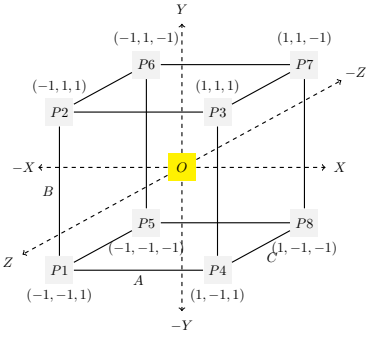
- Sustituir el MainActivity.java del proyecto con el MainActivity de la carpeta del demo
- Copiar los archivos CircleFan.java, MyGLSurfaceView.java y MyGLRenderer.java en la carpeta del package

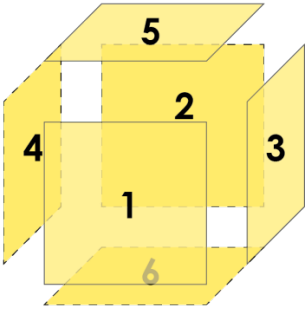


Outline

Cubo 3D


y
x
3
2
1
−1
−2
−3
−3
−1
1
3
Q(−1, 1)
R(1, 1)
P(−1, −1)
Z(1, −1)

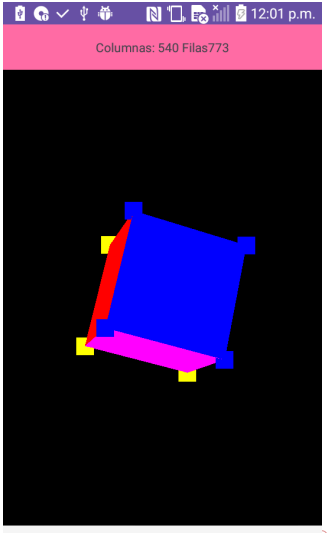

Y
−Y
X
−X
Z
−Z
O
(−1, 1, −1)
(1, 1, −1)
(−1, 1, 1)
(1, 1, 1)
P6
P7
P2
P3
P5
P8
P1
P4
B
A
C
(−1, −1, −1)
(1, −1, −1)
(−1, −1, 1)
(1, −1, 1)


5
2
4
3
1
6

Objetos 3D Complejos en OpenGL (0)

Demo #3: Cubo en OpenGL

- Sustituir el MainActivity.java del proyecto con el MainActivity de la carpeta del demo
- Copiar los archivos Cube.java y CubeRenderer.java en la carpeta del package
- Copiar el archivo *activity_main.xml* a la carpeta *res/layout*

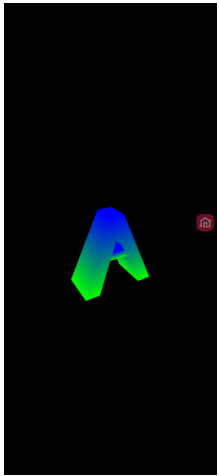


Objetos 3D Complejos en OpenGL (1)

- Una vez que se entiende como generar las coordenadas de poligonos y asignarles un color, es posible construir objetos 3D complejos.

Demo #4: Letra A en 3D

- Sustituir el MainActivity.java del proyecto con el MainActivity de la carpeta del demo
- Copiar los archivos CharacterA.java, MyRenderer.java y MyView.java en la carpeta del package

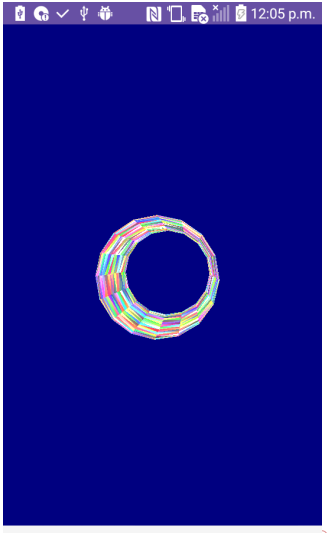


Objetos 3D Complejos en OpenGL (2)

- Es posible dibujar objetos compuestos de múltiples triángulos

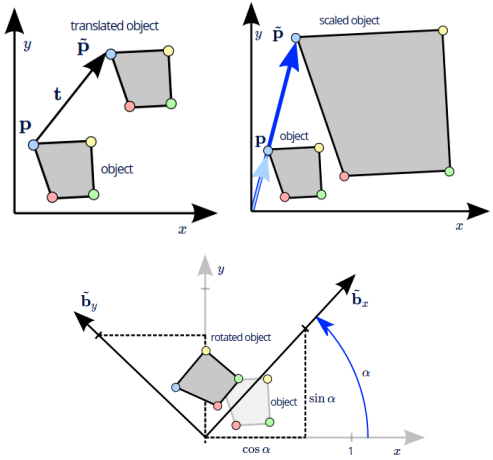
Demo #5: Cilindro Multicolores

- Sustituir el MainActivity.java del proyecto con el MainActivity de la carpeta del demo
- Copiar los archivos MyGLSurfaceRenderer.java y MyGLSurfaceView.java en la carpeta del package



Transformaciones

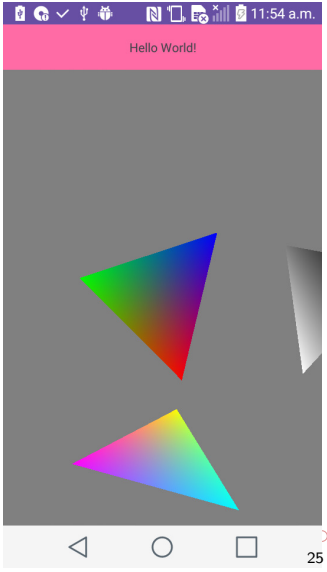
- **Translación.** Es un cambio en la posición de los objetos en cualquiera de los ejes.
- **Escalamiento.** Es un cambio en el tamaño de un objeto con respecto a sus dimensiones en cualquiera de los ejes.
- **Rotación.** La orientación de un objeto puede ser cambiada mediante un ángulo de rotación.



Aplicando Transformaciones en OpenGL

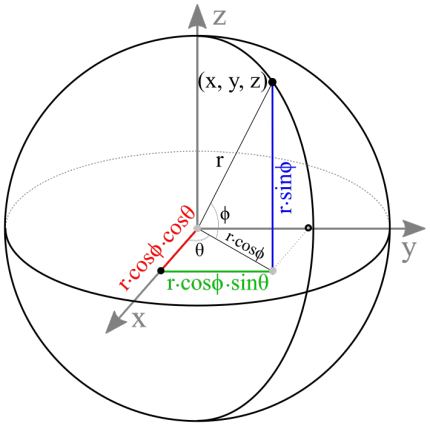
Demo #6: Transformaciones

- Sustituir el MainActivity.java del proyecto con el MainActivity de la carpeta del demo
- Copiar el archivo LessonOneRenderer.java en la carpeta del package
- Copiar el archivo *activity_main.xml* a la carpeta *res/layout*



Dibujar una esfera (1)

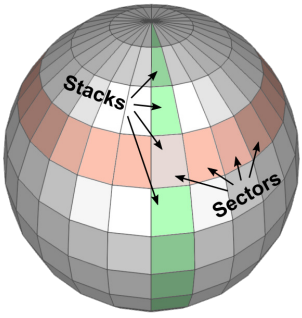
- La definición de esfera es una superficie cerrada en 3D donde cada punto de la esfera está a la misma distancia (radio) de un punto dado.
- Dado que no podemos dibujar todos los puntos en una esfera, solo tomamos muestras de una cantidad limitada de puntos dividiendo la esfera por sectores (longitud) y pilas (latitud).

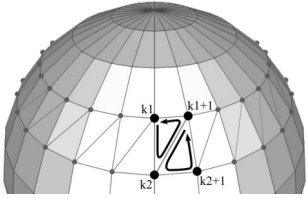


[0] http://www.songho.ca/opengl/gl_sphere.html

Dibujar una esfera (2)

- Para dibujar la superficie de una esfera en OpenGL, debe triangular los vértices adyacentes para formar polígonos. Es posible usar una sola tira triangular para renderizar toda la esfera.
- Cada sector de una pila requiere 2 triángulos.

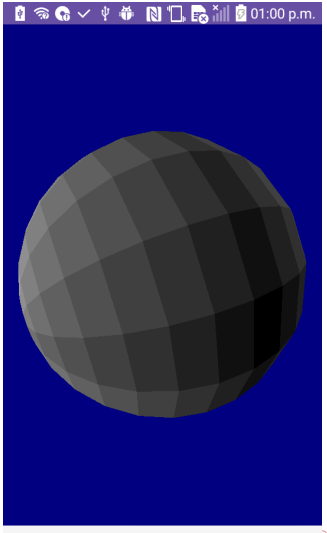




Esfera en OpenGL

Demo #7: Esfera

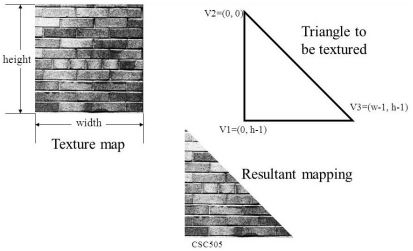
- Sustituir el MainActivity.java del proyecto con el MainActivity de la carpeta del demo
- Copiar los archivos MyGLRenderer.java y MyGLSurfaceView.java en la carpeta del packag



Outline

Mapeo de Texturas

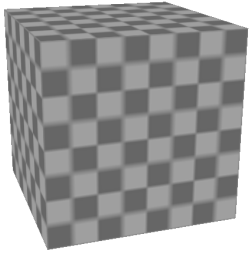
- El mapeo de texturas en gráficos por computadora se refiere a la aplicación de un tipo de superficie a una imagen 3D.
- El método común es crear una imagen de mapa de bits 2D de la textura, llamada *mapa de textura*, que luego se *envuelve* alrededor del objeto 3D.
- Para texturas mas precisas se emplean funciones matemáticas en lugar de mapas de bits.



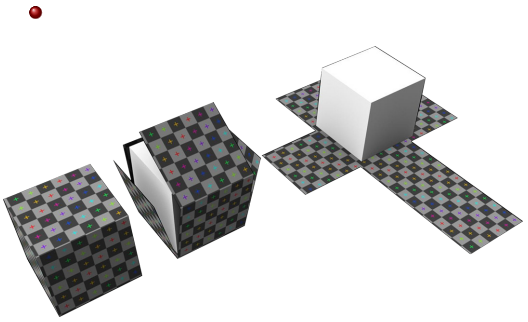
Textura en OpenGL

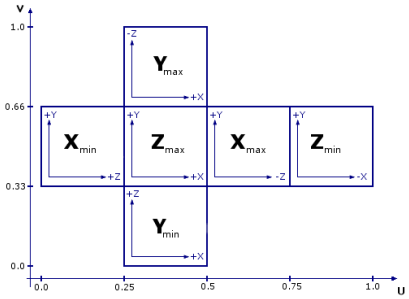
Demo #8: Textura en un Cubo

- Sustituir el MainActivity.java del proyecto con el MainActivity de la carpeta del demo
- Copiar los archivos BaseActivity.java, GLESChecker.java, ShaderHelper.java, Shape.java, ShapeRenderer.java. TextResourceReader.java, TextureHelper.java y TexturedCube.java en la carpeta del package
- Crear una carpeta dentro de la carpeta res con nombre raw. Copiar dentro los archivos con extension glsl
- Copiar las imágenes .png en la carpeta drawable



Multiples Texturas en OpenGL





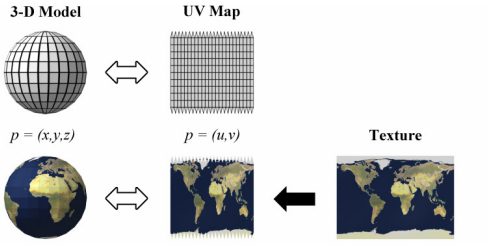
Multiples Texturas en OpenGL

Demo #9: Multiples Texturas en un Cubo

- Sustituir el *MainActivity.java* del proyecto con el *MainActivity* de la carpeta del demo
- Copiar los archivos *BaseActivity.java*, *GLESChecker.java*, *MultiTexturedCube.java*, *ShaderHelper.java*, *Shape.java*, *ShapeRenderer.java*. *TextResourceReader.java*, *TextureHelper.java* y en la carpeta del package
- Crear una carpeta dentro de la carpeta res con nombre *raw*. Copiar dentro los archivos con extension *.glsl*
- Copiar las imágenes .png en la carpeta drawable

UV mapping

- El mapeo UV es el proceso de modelado 3D de proyectar una imagen 2D en la superficie de un modelo 3D para el mapeo de texturas. Las letras U y V denotan los ejes de la textura 2D porque X, Y y Z ya se utilizan para denotar los ejes del objeto 3D en el espacio modelo,



Textura sobre una esfera

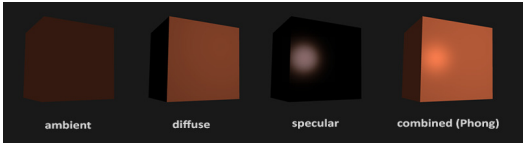
Demo #10: Textura sobre una esfera

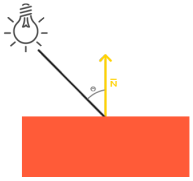
- Sustituir el *MainActivity.java* del proyecto con el *MainActivity* de la carpeta del demo
- Copiar los archivos *BaseActivity.java*, *GLESChecker.java*, *Globe.java*, *ShaderHelper.java*, *Shape.java*, *ShapeRenderer.java*. *TextResourceReader.java* y *TextureHelper.java* en la carpeta del package
- Crear una carpeta dentro de la carpeta res con nombre *raw*. Copiar dentro los archivos con extension *.glsl*
- Copiar las imágenes .png en la carpeta drawable

Outline

Iluminación Ambiente y Difusa

- Iluminación ambiental: lo que los objetos nunca están completamente oscuros.
- Iluminación difusa: simula el impacto direccional que tiene un objeto ligero sobre un objeto.
- Iluminación especular: simula el punto brillante (del color de la fuente) de una luz que aparece sobre objetos brillantes.

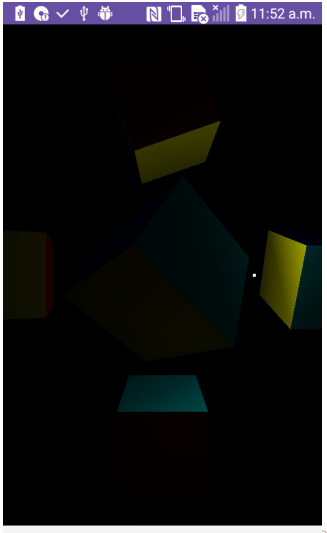




Demos de Iluminación (1)

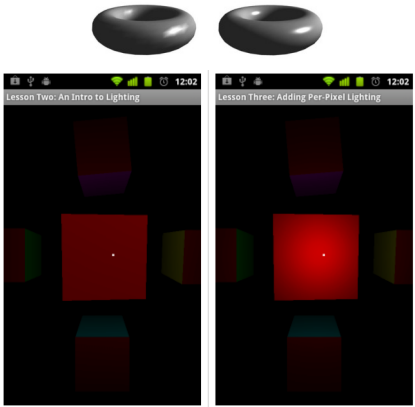
Demo #11: Iluminación por Vertices

- Sustituir el *MainActivity.java* del proyecto con el *MainActivity* de la carpeta del demo
- Copiar el archivo *LessonTwoRenderer.java* en la carpeta del package



Iluminación por Fragmentos

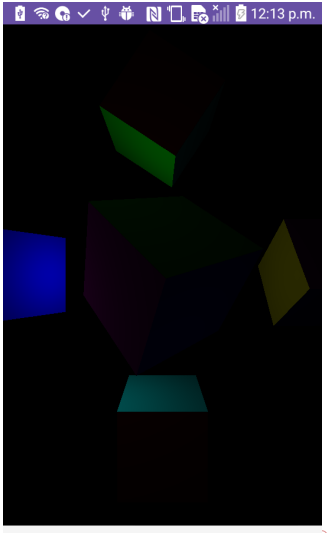
- En la iluminación por vértice, la cara frontal del cubo muestra un sombreada plana sin evidencia de indicencia de luz. Los cuatro puntos de la cara frontal son equidistantes de la luz, y intensidad en cada puntos es interpolada mediante los triángulos que forman la cara frontal.
- En la iluminación por fragmento hay un efecto mucho más notorio en la misma cara del cubo.



Demos de Iluminación (2)

Demo #12: Iluminación por Fragmentos

- Sustituir el *MainActivity.java* del proyecto con el *MainActivity* de la carpeta del demo
- Copiar los archivos *LessonTwoRenderer.java* y *LessonThreeRenderer.java* en la carpeta del package



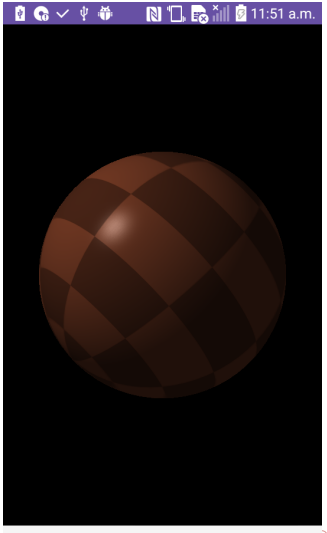
Textura e Iluminación

- Mapear texturas permite construir ambientes 3D de apariencia realista.
- Para lograr esto, es necesario enviar al *Vertex Shader* una matriz con información de coordenadas de textura.
- Para agregar realismo a las escenas, es necesario combinar texturas más iluminación, siendo requerido un código más complejo en el *Fragment Shader* para lograr dicha tarea.

Textura e Iluminación

Demo #13: Textura e Iluminación sobre una Esfera

- Sustituir el *MainActivity.java* del proyecto con el *MainActivity* de la carpeta del demo
- Copiar los archivos *BaseActivity.java*, *GLESChecker.java*, *ShaderHelper.java*, *Shape.java*, *ShapeRenderer.java*, *Shere.java*, *TextResourceReader.java y TextureHelper.java* en la carpeta del package
- Crear una carpeta dentro de la carpeta *res* con nombre *raw*. Copiar dentro los archivos con extension *.glsl*
- Copiar las imágenes .png en la carpeta drawable



Outline

Realidad Virtual OpenGLES Android

- Para este demo, es necesario contar con un teléfono de gama media hacia arriba, que incluye sensores de movimiento (acelerómetros y giroscopios)
- De manera opcional, se puede utilizar el adaptador de realidad virtual para una interacción inmersiva

Realidad Virtual OpenGLES Android

Demo #14: Realidad Virtual (1)

- Sustituir el *MainActivity.java* del proyecto con el *MainActivity* de la carpeta del demo
- Copiar **todos** los archivos .java de la carpeta a **EXCEPCION** de *ExampleInstrumentedTest.java y ExampleUnitTest.java* en la carpeta del package
- Crear una carpeta dentro de la carpeta *res* con nombre *raw*. Copiar dentro los archivos con extension *.shader*
- Agregar la siguiente linea a la sección dependencias del archivo build.gradle.kts:

```
implementation ("com.google.protobuf.nano:protobuf-javanano:3.0.0-alpha-2")
```

Realidad Virtual OpenGLES Android

Demo #14: Realidad Virtual (2)

- Agregar los siguientes permisos al archivo Android Manifest

```
1       <uses-feature android:name="android.hardware.sensor.accelerometer" android:required="true"/>
2       <uses-feature android:name="android.hardware.sensor.gyroscope" android:required="true"/>
3       <uses-permission android:name="android.permission.HIGH_SAMPLING_RATE_SENSORS"/>
4       <uses-permission android:name="android.permission.NFC" />
5       <uses-permission android:name="android.permission.VIBRATE" />
6       <uses-feature android:glEsVersion="0x00020000" android:required="true" />
```

Realidad Virtual OpenGLES Android

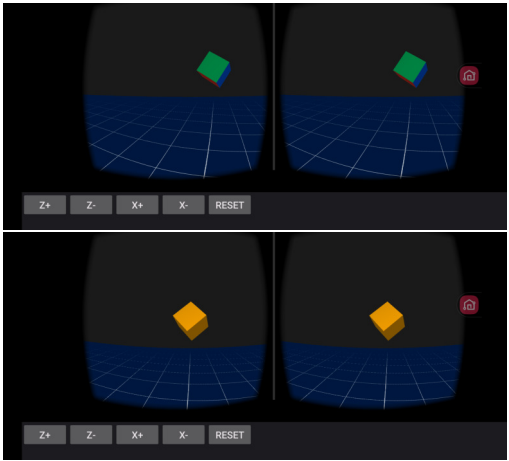
Demo #14: Realidad Virtual (3)

- Reemplazar el *activity_main.xml*, cambiando por package de su proyecto (lineas resaltadas):

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:id="@+id/ui_layout"
4      android:orientation="vertical"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent" >
7      <LinearLayout
8          android:layout_width="match_parent"
9          android:orientation="horizontal"
10         android:layout_weight="5"
11         android:layout_height="0dp">
12         <com.example.realidadvirtual_2023.CardboardView
13             android:id="@+id/cardboard_view"
14             android:layout_width="match_parent"
15             android:layout_height="match_parent"/>
16         <com.example.realidadvirtual_2023.CardboardOverlayView
17             android:id="@+id/overlay"
18             android:layout_width="match_parent"
19             android:layout_height="match_parent" />
20     </LinearLayout>
21     <!-- EStе panel es temporal, solo para depuracion -->
```

Realidad Virtual OpenGLES Android

- La vista es afectada por los sensores del telefono (acelerómetros y giroscopios)
- La posición en el plano XZ es afectada por los botones ubicados en la parte inferior



Outline

OpenCV en Android

Demo #15: OpenCV Android - Parte I (1)

- Descomprimir la carpeta opencv-4.8.0-android-sdk.zip y copiar la carpeta sdk a la carpeta raiz del proyecto Android
- Editar Manualmente el archivo build.gradle de la carpeta SDK y hacer los siguiente cambios:
  - Comentar las siguientes lienas:

    ```
    apply plugin: 'com.android.library'
    apply plugin: 'kotlin-android'
    ```

  - E inmediatamente abajo agregar la siguiente linea:

    ```
    plugins { id("com.android.library") }
    ```

  - Agregar la siguiente linea dentro de android { ... }

    ```
    namespace "org.opencv"
    ```

  - Dentro del grupo android { } - Cambiar el numero despues de compileSdkVersion y targetSdkVersion a 33

    ```
    compileSdkVersion 33
    targetSdkVersion 33
    ```

OpenCV en Android

Demo #15: OpenCV Android - Parte I (2)

- Dentro del grupo android { }, agregar lo siguiente:

```
buildFeatures {
    aidl = true
    buildConfig = true
}
```

- Agregar las sigueintes lineas al final del settings.graddle.kts (Sustituir por la ruta donde esta la carpeta SDK dentro del proyecto)

```
include(":sdk")
project(":sdk").projectDir = File("/home/marco/AndroidStudioProjects/OpenCV_JavaFin5/sdk")
```

- Agregar al archivo *gradle.properties* la siguiente linea

```
android.defaults.buildfeatures.buildconfig=true
```

- Agregar en la seccion dependencias del archivo *build.gradle.kts (Module:app)* la siguiente linea:

```
implementation(project(mapOf("path" to ":sdk")))
```

OpenCV en Android

Demo #15: OpenCV Android - Parte I (3)

- Agregar las siguientes lineas dentro del onCreate del Main Activity

```
if (OpenCVLoader.initDebug()) {
            Toast.makeText(getApplicationContext(),"OpenCV loaded"+OpenCVLoader.OPENCV_VERSION,Toast.LENGTH_SHORT).show();
        }
```

- Para corregir el error sobre la palabra *OpenCVLoader*, ubiquen el cursor el dicha palabra y presione la combinación ALT + ENTER. En la primera opción debe agregar el import a la sección de imports del proyecto.
- Repetir el paso anterior para la palabra *Toast*.
- Al ejecutar la aplicación, debe aparecer un mensaje que confirma que OpenCV fue cargado (con la versión específica descargada)

OpenCV en Android

Demo #15: OpenCV Android - Parte 2 (1)

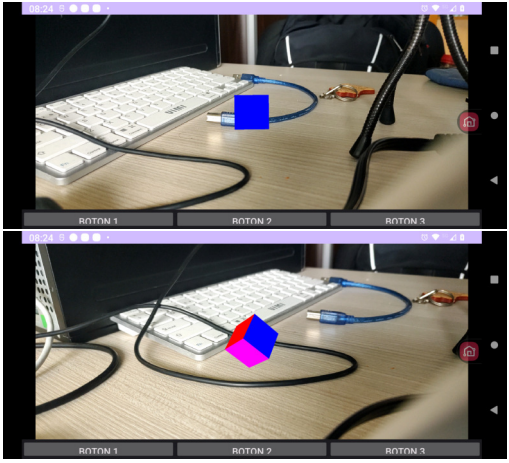
- Agregar las siguientes lineas al archivo *AndroidManifest.xml*:

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:name="android.hardware.camera" android:required="false"/>
<uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
<uses-feature android:name="android.hardware.camera.front" android:required="false"/>
<uses-feature android:name="android.hardware.camera.front.autofocus" android:required="false"/>
```

- Agregar el archivo *color_blob_detection_surface_view* al la carpeta *res/layout*
- Agregar los archivos *CubeRenderer.java*, *Cube.java*, *ColorBlobDetection.java* a la carpeta del Package
- Sustitir el archivo *MainActivity* del proyecto por el ubicado en la Carpeta

OpenCV en Android

- La vista es afectada por los sensores del telefono (acelerómetros y giroscopios)
- La posición en el plano XZ es afectada por los botones ubicados en la parte inferior



Realidad Aumentada en Android

Demo #16: Realidad Aumentada en Kotlin

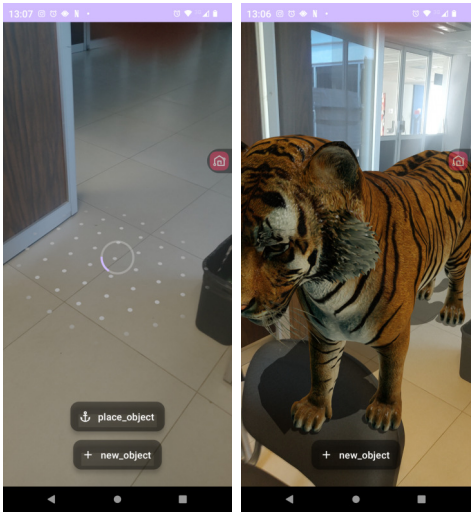
- Se debe crear un proyecto NUEVO, pero seleccionar Kotlin como lenguaje
- Agregar la siguiente linea a la sección dependencias del archivo build.gradle.kts y sincronizar el proyecto:

```
implementation ("io.github.sceneview:arsceneview:0.9.0")
```

- Sustituir el MainActivity.kt del proyecto con el MainActivity de la carpeta del demo
- Copiar el archivo *activity_main.xml* a la carpeta *res/layout*
- Copiar los archivos *ic_add.xml e ic_anchor.xml* a la carpeta *res/drawable*
- Buscar superficie con textura (aparecera una matriz de puntos) → Place Object → New Object → Place Object

Realidad Aumentada en Android

- El objeto es fijado a la superficie texturizada seleccionada por el usuario
- A partir de los sensores del teléfono, se cambia la vista manteniendo el objeto “fijo” a la superficie
- Es posible agregar otros objetos (estan en formato GLB)



Outline

Conclusiones

Recapitulando lo visto en esta presentación

- Revisamos conceptos relacionados con OpenGL, GLSL, CPUs, GPUs
- Explicamos primitivas 2D, 3D, y se desarrollaron prototipos para su implementación
- Comprendimos los diferentes modelos de iluminacion
- Integramos graficos 2D con una aplicación de Realidad Virtual simple
- Integramos graficos 2D con una aplicación de Realidad Aumentada simple