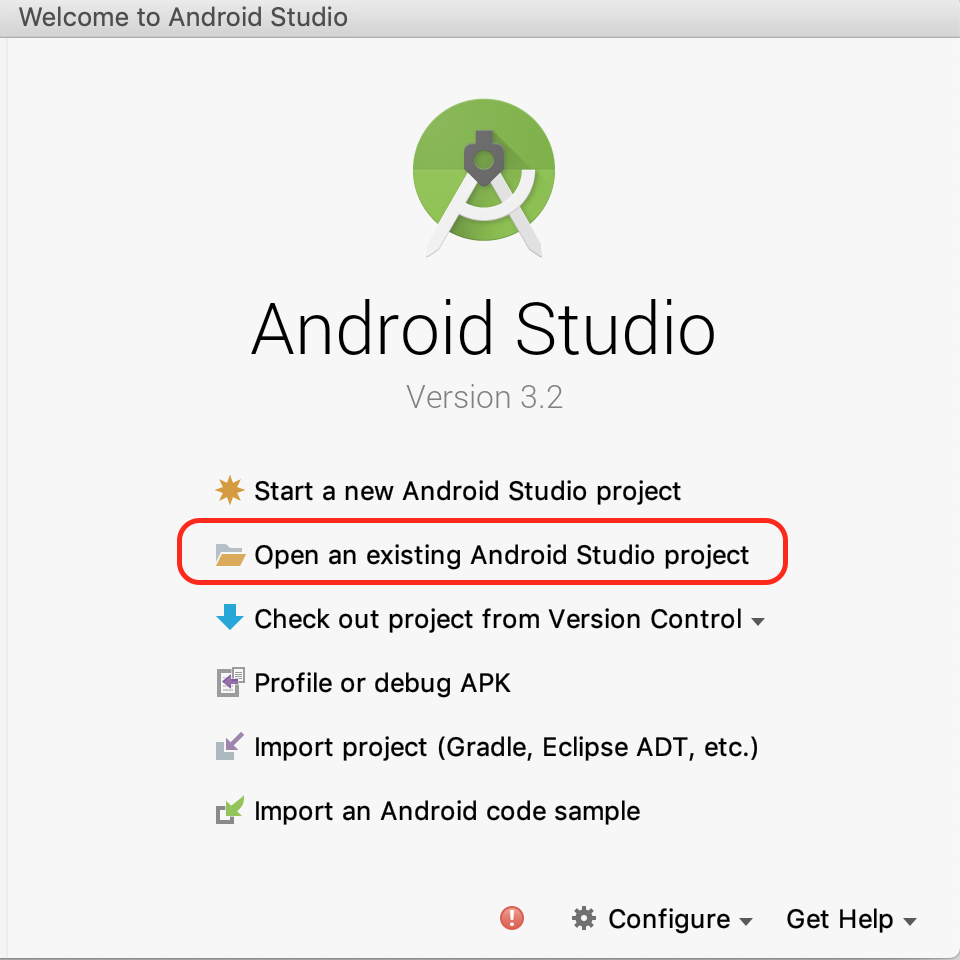


**Profesor**: Luis Felipe Botero Castaño



**INICIO DEL PROYECTO**

1. Abrir Android studio
2. Abrir el proyecto



1. Conectar el teléfono y correr la aplicación

**PRIMERA ACTIVIDAD**

1. En la clase **DepthCodelabActivity** agregar la siguiente variable:
2. private boolean isDepthSupported;
3. En el metodo **onResume()**, agregar el siguiente código despues de session = new Session(*/\* context= \*/* this);

Config config = session.getConfig();

isDepthSupported = session.isDepthModeSupported(Config.DepthMode.AUTOMATIC);

if (isDepthSupported) {

config.setDepthMode(Config.DepthMode.AUTOMATIC);

} else {

config.setDepthMode(Config.DepthMode.DISABLED);

}

session.configure(config);

1. Agregamos el mensaje que nos indica si la profundidad está soportada por el celular al inicio de la clase

private static final String DEPTH\_NOT\_AVAILABLE\_MESSAGE = "[Depth not supported on this device]";

1. Por ultimo, en el metodo onDrawFrame() dibujamos el mensaje en caso que la opción no sea soportada

if (!isDepthSupported) {

messageToShow += "\n" + DEPTH\_NOT\_AVAILABLE\_MESSAGE;

}

1. Corremos la aplicación y miremos el resultado.

**SEGUNDA ACTIVIDAD**

1. En la carpeta **depht** crear la clase **DepthTextureHandler**
2. Dicha clase tendrá el siguiente contenido:

package com.google.ar.core.codelab.depth;

import static android.opengl.GLES20.GL\_CLAMP\_TO\_EDGE;

import static android.opengl.GLES20.GL\_TEXTURE\_2D;

import static android.opengl.GLES20.GL\_TEXTURE\_MAG\_FILTER;

import static android.opengl.GLES20.GL\_TEXTURE\_MIN\_FILTER;

import static android.opengl.GLES20.GL\_TEXTURE\_WRAP\_S;

import static android.opengl.GLES20.GL\_TEXTURE\_WRAP\_T;

import static android.opengl.GLES20.GL\_UNSIGNED\_BYTE;

import static android.opengl.GLES20.glBindTexture;

import static android.opengl.GLES20.glGenTextures;

import static android.opengl.GLES20.glTexImage2D;

import static android.opengl.GLES20.glTexParameteri;

import static android.opengl.GLES30.GL\_LINEAR;

import static android.opengl.GLES30.GL\_RG;

import static android.opengl.GLES30.GL\_RG8;

import android.media.Image;

import com.google.ar.core.Frame;

import com.google.ar.core.exceptions.NotYetAvailableException;

*/\*\* Handle RG8 GPU texture containing a DEPTH16 depth image. \*/*

public final class DepthTextureHandler {

private int depthTextureId = -1;

private int depthTextureWidth = -1;

private int depthTextureHeight = -1;

*/\*\**

*\* Creates and initializes the depth texture. This method needs to be called on a*

*\* thread with a EGL context attached.*

*\*/*

public void createOnGlThread() {

int[] textureId = new int[1];

glGenTextures(1, textureId, 0);

depthTextureId = textureId[0];

glBindTexture(GL\_TEXTURE\_2D, depthTextureId);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S,GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T,GL\_CLAMP\_TO\_EDGE);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

}

*/\*\**

*\* Updates the depth texture with the content from acquireDepthImage().*

*\* This method needs to be called on a thread with an EGL context attached.*

*\*/*

public void update(final Frame frame) {

try {

Image depthImage = frame.acquireDepthImage();

depthTextureWidth = depthImage.getWidth();

depthTextureHeight = depthImage.getHeight();

glBindTexture(GL\_TEXTURE\_2D, depthTextureId);

glTexImage2D(

GL\_TEXTURE\_2D,

0,

GL\_RG8,

depthTextureWidth,

depthTextureHeight,

0,

GL\_RG,

GL\_UNSIGNED\_BYTE,

depthImage.getPlanes()[0].getBuffer());

depthImage.close();

} catch (NotYetAvailableException e) {

*// This normally means that depth data is not available yet.*

}

}

public int getDepthTexture() {

return depthTextureId;

}

public int getDepthWidth() {

return depthTextureWidth;

}

public int getDepthHeight() {

return depthTextureHeight;

}

}

1. Luego en la clase **DepthCodelabActivity** vamos a hacer uso de esta nueva utilidad, para esto instanciamos nuestra nueva clase:

private final DepthTextureHandler depthTexture = new DepthTextureHandler();

1. Luego en el metodo onSurfaceCreated() agregar la textura:

depthTexture.createOnGlThread();

1. Finalmente, llenamos la textura al crear la secion

*// Add this just after "frame" is created inside onDrawFrame().*

if (isDepthSupported) {

depthTexture.update(frame);

}

**TERCERA ACTIVIDAD**

1. En la carpeta **assets/shaders** crear el archivo **background\_show\_depth\_map.vert**
2. en el archivo copiar la siguiente información:

attribute vec4 a\_Position;

attribute vec2 a\_TexCoord;

varying vec2 v\_TexCoord;

void main() {

v\_TexCoord = a\_TexCoord;

gl\_Position = a\_Position;

}

1. luego en la misma ruta crear el archivo **background\_show\_depth\_map.frag**
2. A este archivo pegarle la siguiente información:

precision mediump float;

uniform sampler2D u\_Depth;

varying vec2 v\_TexCoord;

const highp float kMaxDepth = 8000.0; *// In millimeters.*

const float kDepthOffsets = 8192.0; *// 1 << 13*

float GetDepthMillimeters(vec4 depth\_pixel\_value) {

return 255.0 \* (depth\_pixel\_value.r + depth\_pixel\_value.g \* 256.0);

}

*// Returns an interpolated color in a 6 degree polynomial interpolation.*

vec3 GetPolynomialColor(in float x,

in vec4 kRedVec4, in vec4 kGreenVec4, in vec4 kBlueVec4,

in vec2 kRedVec2, in vec2 kGreenVec2, in vec2 kBlueVec2) {

*// Moves the color space a little bit to avoid pure red.*

*// Removes this line for more contrast.*

x = clamp(x \* 0.9 + 0.03, 0.0, 1.0);

vec4 v4 = vec4(1.0, x, x \* x, x \* x \* x);

vec2 v2 = v4.zw \* v4.z;

return vec3(

dot(v4, kRedVec4) + dot(v2, kRedVec2),

dot(v4, kGreenVec4) + dot(v2, kGreenVec2),

dot(v4, kBlueVec4) + dot(v2, kBlueVec2)

);

}

*// Returns a smooth Percept colormap based upon the Turbo colormap.*

vec3 PerceptColormap(in float x) {

const vec4 kRedVec4 = vec4(0.55305649, 3.00913185, -5.46192616, -11.11819092);

const vec4 kGreenVec4 = vec4(0.16207513, 0.17712472, 15.24091500, -36.50657960);

const vec4 kBlueVec4 = vec4(-0.05195877, 5.18000081, -30.94853351, 81.96403246);

const vec2 kRedVec2 = vec2(27.81927491, -14.87899417);

const vec2 kGreenVec2 = vec2(25.95549545, -5.02738237);

const vec2 kBlueVec2 = vec2(-86.53476570, 30.23299484);

const float kInvalidDepthThreshold = 0.01;

return step(kInvalidDepthThreshold, x) \*

GetPolynomialColor(x, kRedVec4, kGreenVec4, kBlueVec4,

kRedVec2, kGreenVec2, kBlueVec2);

}

void main() {

vec4 packed\_depth = texture2D(u\_Depth, v\_TexCoord.xy);

highp float depth\_mm = GetDepthMillimeters(packed\_depth);

highp float normalized\_depth = depth\_mm / kMaxDepth;

vec4 depth\_color = vec4(PerceptColormap(normalized\_depth), 1.0);

gl\_FragColor = depth\_color;

}

1. Ahora actualizamos el archivo **BackgroundRenderer** agregando estas dos texturas

private static final String DEPTH\_VERTEX\_SHADER\_NAME = "shaders/background\_show\_depth\_map.vert";

private static final String DEPTH\_FRAGMENT\_SHADER\_NAME = "shaders/background\_show\_depth\_map.frag";

1. Modificamos la clase **BackgroundRenderer** para que esté en capadidad de mostrar ambas texturas:

private int depthProgram;

private int depthTextureParam;

private int depthTextureId = -1;

private int depthQuadPositionParam;

private int depthQuadTexCoordParam;

1. Agregamos un método para llenar dichas texturas:

public void createDepthShaders(Context context, int depthTextureId) throws IOException {

int vertexShader =

ShaderUtil.loadGLShader(

TAG, context, GLES20.GL\_VERTEX\_SHADER, DEPTH\_VERTEX\_SHADER\_NAME);

int fragmentShader =

ShaderUtil.loadGLShader(

TAG, context, GLES20.GL\_FRAGMENT\_SHADER, DEPTH\_FRAGMENT\_SHADER\_NAME);

depthProgram = GLES20.glCreateProgram();

GLES20.glAttachShader(depthProgram, vertexShader);

GLES20.glAttachShader(depthProgram, fragmentShader);

GLES20.glLinkProgram(depthProgram);

GLES20.glUseProgram(depthProgram);

ShaderUtil.checkGLError(TAG, "Program creation");

depthTextureParam = GLES20.glGetUniformLocation(depthProgram, "u\_Depth");

ShaderUtil.checkGLError(TAG, "Program parameters");

depthQuadPositionParam = GLES20.glGetAttribLocation(depthProgram, "a\_Position");

depthQuadTexCoordParam = GLES20.glGetAttribLocation(depthProgram, "a\_TexCoord");

this.depthTextureId = depthTextureId;

}

1. Agregamos un metodo para dibujar estas texturas:

*// Put this at the bottom of the file.*

public void drawDepth(@NonNull Frame frame) {

if (frame.hasDisplayGeometryChanged()) {

frame.transformCoordinates2d(

Coordinates2d.OPENGL\_NORMALIZED\_DEVICE\_COORDINATES,

quadCoords,

Coordinates2d.TEXTURE\_NORMALIZED,

quadTexCoords);

}

if (frame.getTimestamp() == 0 || depthTextureId == -1) {

return;

}

*// Ensure position is rewound before use.*

quadTexCoords.position(0);

*// No need to test or write depth, the screen quad has arbitrary depth, and is expected*

*// to be drawn first.*

GLES20.glDisable(GLES20.GL\_DEPTH\_TEST);

GLES20.glDepthMask(false);

GLES20.glActiveTexture(GLES20.GL\_TEXTURE0);

GLES20.glBindTexture(GLES20.GL\_TEXTURE\_2D, depthTextureId);

GLES20.glUseProgram(depthProgram);

GLES20.glUniform1i(depthTextureParam, 0);

*// Set the vertex positions and texture coordinates.*

GLES20.glVertexAttribPointer(

depthQuadPositionParam, COORDS\_PER\_VERTEX, GLES20.GL\_FLOAT, false, 0, quadCoords);

GLES20.glVertexAttribPointer(

depthQuadTexCoordParam, TEXCOORDS\_PER\_VERTEX, GLES20.GL\_FLOAT, false, 0, quadTexCoords);

*// Draws the quad.*

GLES20.glEnableVertexAttribArray(depthQuadPositionParam);

GLES20.glEnableVertexAttribArray(depthQuadTexCoordParam);

GLES20.glDrawArrays(GLES20.GL\_TRIANGLE\_STRIP, 0, 4);

GLES20.glDisableVertexAttribArray(depthQuadPositionParam);

GLES20.glDisableVertexAttribArray(depthQuadTexCoordParam);

*// Restore the depth state for further drawing.*

GLES20.glDepthMask(true);

GLES20.glEnable(GLES20.GL\_DEPTH\_TEST);

ShaderUtil.checkGLError(TAG, "BackgroundRendererDraw");

}

1. Por último debemos agregar un boton. En la activity **DepthCodelabActivity** importamos la libreria con los botones:

import android.widget.Button;

1. agregamos el botón en off por defecto

private boolean showDepthMap = false;

1. Agregar la acción al botón

final Button toggleDepthButton = (Button) findViewById(R.id.toggle\_depth\_button);

toggleDepthButton.setOnClickListener(

view -> {

if (isDepthSupported) {

showDepthMap = !showDepthMap;

toggleDepthButton.setText(showDepthMap ? R.string.hide\_depth : R.string.show\_depth);

} else {

showDepthMap = false;

toggleDepthButton.setText(R.string.depth\_not\_available);

}

});

1. Agregamos los mensajes en el archivo **res/values/strings.xml**

<string translatable="false" name="show\_depth">Show Depth</string>

<string translatable="false" name="hide\_depth">Hide Depth</string>

<string translatable="false" name="depth\_not\_available">Depth Not Available</string>

1. Agregamos el boton en la Ui de la aplicación:

<Button

android:id="@+id/toggle\_depth\_button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_margin="20dp"

android:gravity="center"

android:text="@string/show\_depth"

android:layout\_alignParentRight="true"

android:layout\_alignParentTop="true"/>

1. Por último, agregamos en la clase **DepthCodelabActivity**

*// Add to onSurfaceCreated() after backgroundRenderer.createonGlThread(/\*context=\*/ this);*

backgroundRenderer.createDepthShaders(*/\*context=\*/* this, depthTexture.getDepthTexture());

1. Corremos la aplicación

**CUARTA ACTIVIDAD**

1. Podríamos tener una animación a la hora de ver la profundidad, para esto, primero modificamos nuestras texturas. En el archivo **background\_show\_depth\_map.frag,** agregamos al inicio

uniform float u\_DepthRangeToRenderMm;

const float kDepthWidthToRenderMm = 350.0;

1. Luego en el metodo main del shadder agregamos:

gl\_FragColor.a = clamp(1.0 - abs((depth\_mm - u\_DepthRangeToRenderMm) / kDepthWidthToRenderMm), 0.0, 1.0);

1. Luego en la clase **BackgroundRenderer** agregamos:

private static final float MAX\_DEPTH\_RANGE\_TO\_RENDER\_MM = 10000.0f;

private float depthRangeToRenderMm = 0.0f;

private int depthRangeToRenderMmParam;

1. En el método **createDepthShaders**

*// Enables alpha blending.*

GLES20.glEnable(GLES20.GL\_BLEND);

GLES20.glBlendFunc(GLES20.GL\_SRC\_ALPHA, GLES20.GL\_ONE\_MINUS\_SRC\_ALPHA);

*// Updates range each time draw() is called.*

depthRangeToRenderMm += 50.0f;

if (depthRangeToRenderMm > MAX\_DEPTH\_RANGE\_TO\_RENDER\_MM) {

depthRangeToRenderMm = 0.0f;

}

*// Passes latest value to the shader.*

GLES20.glUniform1f(depthRangeToRenderMmParam, depthRangeToRenderMm);

1. Ahora corremos nuestra aplicación y vemos una animación

