Attachment

Spike

Sceneform vs OpenGL ES as rendering methods on Android platform

The research conducted in this paper aims to conclude whether or not the Sceneform library is more suitable than OpenGL ES as a rendering engine for the Android platform.

Attachment

Sceneform

Sceneform is a high level scene-graph based graphics API used to accomplish rendering tasks without extended knowledge about OpenGL, which is generally a more low level API. It can be used to render 3D graphics in both AR and non-AR mobile applications. Sceneform also includes a realistic physically based renderer provided by Google's Filament graphics API.

OpenGL ES

OpenGL ES is graphics API to achieve hardware accelerated rendering on embedded systems, such as smartphones and consoles. OpenGL ES is considered a high-level API and is supported on a variety of devices. The API is supported by Android by default; making it a popular tool for Android developers.

Difference between the two

Sceneform can be considered a layer on top of OpenGL ES in some ways. It is based on a the Filament rendering engine which is again based on top of various graphics APIs including OpenGL. The main difference between the two would be that Sceneform is a high level scene graph API while OpenGL ES is a low level graphics API using hardware accelerated rendering to achieve high performance.

Sceneform and performance

Scenefor provides good performance because of the Filament rendering engine it is built upon. The Filament rendering engine's primary target is OpenGL ES 3, therefor Sceneform's performance is similar to OpenGL ES.

Development with Sceneform

The development process with Sceneform is more simple than using lower level APIs such as OpenGL. Sceneform has several inbuilt methods and tools which makes implementations and

optimization of the rendering process easier. The downside of a high level API is that there is less control over the rendering process and there are fewer features available for specific needs.

Sceneform for StreamBIM AR

Sceneform can be easily integrated with ARCore; making in an easy way to render objects in AR without having to use an extensive amount of time bridging a custom rendering engine written in OpenGL ES and ARCore. Most of the current online examples and tutorials focuses around pre-made 3D objects, luckily, Sceneform provides the ability to add custom shapes and objects dynamically through the same data that StreamBIM uses, based on vertex-, index-, normal- and material data.

OpenGL ES and performance

OpenGL ES is a higher level graphics API compared with other APIs available such as Metal and Vulkan. This means that is less performant than the other lower level APIs, however OpenGL ES is faster than using Sceneform if the user is sufficient using OpenGL ES. If the user is not sufficient using OpenGL, they might lose performance which they could have achieves using a higher level API such as Sceneform.

Development with OpenGL ES

Although OpenGL ES is more performant than Sceneform it has a considerable more complex development cycle. OpenGL ES provides more control over the rendering process, but this entails it taking more time and experience to achieve good performance in larger applications.

Conclusion

OpenGL ES provides greater performance than Sceneform but is considerable more complex as a development tool. Sceneform has a smooth transition between 3D rendering while using OpenGL ES would demand more research and a lot of time implementing AR. For the team’s needs and deadline Sceneform proves the best alternative for implementing 3D rendering in AR because of its ease of use and small performance loss over using OpenGL ES.

Attachment

Sceneform Setup and Use

Setup

Project Gradle File

Make sure that the google repository is included in the project gradle file

```
buildscript {
    ...
    repositories {
        google() <--
        ...
    }
    ...
}

allprojects {
    repositories {
       google() <--
       ...
    }
}
```

App Gradle File

Add the following blue lines to the app´s build.gradle file

```
...
android {
    ...
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
dependencies {
    ...
    implementation "com.google.ar.sceneform.ux:sceneform-ux:1.6.0"
}
```

Attachment

Add a Sceneview in an activity

```
<?xml version="1.0" encoding="utf-8"?>
...
<com.google.ar.sceneform.SceneView
       android:id="@+id/sceneView"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:background="@color/colorPrimaryDark"
       app:layout_constraintBottom_toBottomOf="parent"
       app:layout_constraintEnd_toEndOf="parent"
       app:layout_constraintStart_toStartOf="parent"
       app:layout_constraintTop_toTopOf="parent" />

...
```

Creating the scene

```
class MainActivity : AppCompatActivity() {
   private lateinit var scene: Scene  <--
   ...
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)

      scene = sceneView.scene  <-- // sceneView from layout

      ...
   }
   ...
}

override fun onPause() {  <--
   super.onPause()
   sceneView.pause()
}
```

Attachment

```
override fun onResume() {  <--
    super.onResume()
    sceneView.resume()
}
```

Creating Objects Based on Vertices and Indices

```
// Defining all the vertex data
val vertices: ArrayList<Vertex> = arrayListOf(
    Vertex.builder().setPosition(Vector3(0f, 5f, -10f)).build(),
    Vertex.builder().setPosition(Vector3(5f, -5f, -10f)).build(),
    Vertex.builder().setPosition(Vector3(-5f, -5f, -10f)).build()
)
// Defining all the index data
val indices: ArrayList<Int> = arrayListOf(2,1,0)

// Bulding a mesh based on the above data
val submesh = RenderableDefinition.Submesh.builder()
    .setTriangleIndices(indices)
    .setMaterial(material)
    .build()

// Load vertices with submesh and add nodes to scene
ModelRenderable.builder()
    .setSource(RenderableDefinition.builder()
        .setVertices(vertices)
        .setSubmeshes(arrayListOf(submesh))
        .build()
    ).build()
    .thenAccept { model ->
        // use model here
        addNodeToScene(model)
    }
```

Attachment

Add vertices/nodes to scene

```
// Add model renderable to scene graph
private fun addNodeToScene(model: ModelRenderable?) {
    model?.let {
        modelNode = Node().apply {
            setParent(scene) // or a node parent
            localPosition = Vector3(0f, 0f, 0f)
            localScale = Vector3(1f, 1f, 1f)
            name = "Custom Name"
            renderable = it
        }

        scene.addChild(modelNode)
    }
}
```

Using materials

```
MaterialFactory
    .makeOpaqueWithColor(this, BLUE) // context, color
    .thenAccept { material ->

        // set material properties with setFloat(String, value)
        material.setFloat(MaterialFactory.MATERIAL_METALLIC, 0.7f)
        material.setFloat(MaterialFactory.MATERIAL_ROUGHNESS, 0.3f)
        material.setFloat(MaterialFactory.MATERIAL_REFLECTANCE, 0.6f)

        // use material here
    }
```

Attachment

Basic Camera Movement Based on Users Input

```
private val handleTouch = Scene.OnTouchListener { hitPoint, event ->
 when(event.action) {
   MotionEvent.ACTION_UP -> { prevX = null; prevY = null }
   MotionEvent.ACTION_MOVE -> {
     if (prevX != null && prevY != null) {
       val dx = event.x - prevX!!
       val dy = event.y - prevY!!
       // pointerCount = amount of fingers touches the screen
       when (event.pointerCount) {
         1 -> {
           // screen is rotated 90deg => x = y | y = x
           rotationX += dy/10
           rotationY += dx/10

           val rotX =
                     Quaternion.axisAngle(Vector3.right(), rotationX)

           val rotY =
                     Quaternion.axisAngle(Vector3.up(), rotationY)

           scene.camera.localRotation =
                                    Quaternion.multiply(rotX, rotY)
         }
         2 -> {
           val forward = scene.camera.forward
           val scaledForward =
                               Vector3(forward.x * -dy/10,
                                       forward.y * -dy/10,
                                       forward.z * -dy/10)

           val newPosition =
           Vector3.add(scene.camera.localPosition, scaledForward)
```

Attachment

```
              scene.camera.localPosition = newPosition
            }
          }
        }

      prevX = event.x
      prevY = event.y
    }
  }

  return true
}
```