Mobile Application Programming: Android

OpenGL Operation
Activities

- Apps are composed of activities
- Activities are self-contained tasks made up of one screen-full of information
- Activities start one another and are destroyed commonly
- Apps can use activities belonging to another app
Fragments

- Acts like a sub-activity
- Attached and removed from an activity using the FragmentManager
- Attachment or removal of many fragments with FragmentTransaction
- Lifecycle tied to parent activity
- Adds onAttach / onDetach and onCreateView / onDestroyView
OpenGL ES

- C-Based **Performance-Oriented** Graphics Library
- Wrapper libraries provided for Java, C#, etc.
- Produces 2D images from **2D** or **3D** geometric data
- **Mobile** version of OpenGL
  - Includes nearly all OpenGL functionality
  - Removes seldom-used or legacy features
  - Used by **non-mobile platforms** also (eg. Playstation 3)
OpenGL Environment

- android.opengl.GLSurfaceView
- GLSurfaceView.Renderer
  - GLES20 (C Library Wrapper)
    - Vertex Shader
    - Fragment Shader
    - Program
      - Uniform Variables
      - Attribute Arrays
Data read from Scene and OBJ files

OpenGL ES Primitive Processing → Vertex Shader → OpenGL ES Rasterizer

Fragments resulting from rasterization

Fragment Shader → OpenGL ES Fragment Processing → Frame Buffer
OpenGL Shading Language

- Defines a C-like language that can be compiled into GPU instructions
- Floating-point, Integer, and Boolean data types
- Vectors of these types in 2, 3, 4 sizes
- Matrices of floats in 2x2, 3x3, 4x4
- 1D-Arrays and Structures
- Special Texture types
- Operators on all these types
OpenGL Shading Language

- Variable **storage qualifiers** - uniform, attribute, varying
- Variable **precision qualifiers** - lowp, mediump, highp
- **Control Statements** - if, else
- **Loops** - for, while, do-while
- **Jumps** - break, continue, return
- **Function Definition**
- Pre-processor **Directives**
OpenGL Shading Language

- **Built-in functions** for basic mathematics, trigonometry, geometry, and texturing (e.g. exp, tan, cross, texture2D)
- **Built-in variables** to represent inputs and outputs
  - Vertex Shader
    - `gl_Position (vec4, out)`
  - Fragment Shader
    - `gl_FragCoord (vec2, in)`
    - `gl_FragColor (vec4, out)`
Host-GPU Data Transfer

- **Global values** are sent to the graphics processing hardware by changing the values of uniform variables defined in the shaders using `glGetUniformLocation` and `glUniform`* calls.

- **Geometry data** is sent to the graphics hardware by using `glBindAttribLocation`, `glEnableVertexAttribArray`, `glVertexAttribPointer`.

- **Textures** are sent to the graphics processing hardware by calling `glBindTexture` and `glTexImage2D`.

- **Drawing** is initiated by calling `glDrawArrays`.
Vertex Shader

- **Receives a vertex** from OpenGL after minimal processing
- **Modifies** incoming vertex in some way using **uniform variables** where needed
- **Outputs** the vertex
- May also output **additional data** for the **fragment shader** to use
attribute vec4 position;

void main()
{
    gl_Position = position;
}
attribute vec4 position;
uniform mat4 modelView;

void main()
{
    gl_Position = modelView * position;
}
attribute vec4 position;
uniform mat4 modelView;
uniform mat4 projection;

void main()
{
    gl_Position = projection * modelView * position;
}
attribute vec4 position;
attribute vec2 textureCoordinate;

uniform mat4 modelView;
uniform mat4 projection;

varying lowp vec2 textureCoordinateVarying;

void main()
{
    gl_Position = projection * modelView * position;
    textureCoordinateVarying = textureCoordinate;
}
attribute vec4 position;
attribute vec3 normal;
attribute vec2 textureCoordinate;

uniform mat4 modelView;
uniform mat4 projection;

varying lowp vec3 normalVarying;
varying lowp vec2 textureCoordinateVarying;

void main()
{
    gl_Position = projection * modelView * position;
    normalVarying = vec3(normalize(modelView * vec4(normal, 0.0)));
    textureCoordinateVarying = textureCoordinate;
}
attribute vec4 position;
attribute vec3 normal;
attribute vec2 textureCoordinate;

uniform mat4 modelView;
uniform mat4 projection;

uniform vec4 lightPosition;
uniform vec4 lightAmbient;
uniform vec4 lightDiffuse;

varying lowp vec3 normalVarying;
varying lowp vec2 textureCoordinateVarying;
varying lowp vec4 diffuseVarying;

void main()
{
    gl_Position = projection * modelView * position;
    normalVarying = vec3(normalize(modelView * vec4(normal, 0.0)));
    textureCoordinateVarying = textureCoordinate;

    vec4 lightVector = normalize(lightPosition - gl_Position);
    float lightIncidence = dot(lightVector, vec4(normalVarying, 1.0));
    diffuseVarying = lightDiffuse * vec4(max(lightIncidence, 0.0));
}
Fragment Shader

- Receives a fragment from OpenGL resulting from rasterizing a primitive
- Chooses a color for the fragment based on data given by vertex shader and uniform variables
- Outputs the fragment color
void main()
{
    gl_FragColor = vec4(1.0, 1.0, 0.0, 1.0);
}
uniform vec4 color;

void main()
{
    gl_FragColor = color;
}

Fragment Shader
Fragment Shader

```glsl
varying vec4 color;

void main()
{
    gl_FragColor = color;
}
```
uniform sampler2D textureUnit;

varying vec2 textureCoordinate;

void main()
{
    gl_FragColor = texture2D(textureUnit, textureCoordinate);
}
uniform vec4 color;
uniform vec4 lightPositionEyeCoords;

varying vec4 positionEyeCoords;
varying vec3 normalEyeCoords;

void main()
{
  vec3 incident = vec3(normalize(lightPositionEyeCoords - positionEyeCoords));
  vec3 normal = normalEyeCoords;
  float incidence = max(0.0, dot(incident, normal));
  gl_FragColor = color * incidence;
}
Shaders

attribute vec4 position;
attribute vec3 normal;
attribute vec2 textureCoordinate;

uniform mat4 modelView;
uniform mat4 projection;
uniform vec4 lightPosition;
uniform vec4 lightAmbient;
uniform vec4 lightDiffuse;

varying lowp vec3 normalVarying;
varying lowp vec2 textureCoordinateVarying;
varying lowp vec4 diffuseVarying;

void main()
{
  gl_Position = projection * modelView * position;
  normalVarying = vec3(normalize(modelView * vec4(normal, 0.0)));
  textureCoordinateVarying = textureCoordinate;

  vec4 lightVector = normalize(lightPosition - gl_Position);
  float lightIncidence = dot(lightVector, vec4(normalVarying, 1.0));
  diffuseVarying = lightDiffuse * vec4(max(lightIncidence, 0.0));
}

uniform vec4 color;
uniform vec4 lightPositionEyeCoords;

varying vec4 positionEyeCoords;
varying vec3 normalEyeCoords;

void main()
{
  vec3 incident = vec3(normalize(lightPositionEyeCoords - positionEyeCoords));
  vec3 normal = normalEyeCoords;
  float incidence = max(0.0, dot(incident, normal));
  gl_FragColor = color * incidence;
}