Declaring OpenGL requirements

If your application uses OpenGL features that are not available on all devices, you must include these requirements in your [AndroidManifest.xml](https://developer.android.com/guide/topics/manifest/manifest-intro) file. Here are the most common OpenGL manifest declarations:

* **OpenGL ES version requirements** - If your application requires a specific version of OpenGL ES, you must declare that requirement by adding the following settings to your manifest as shown below.

Mapping coordinates for drawn objects

One of the basic problems in displaying graphics on Android devices is that their screens can vary in size and shape. OpenGL assumes a square, uniform coordinate system and, by default, happily draws those coordinates onto your typically non-square screen as if it is perfectly square.

To solve this problem, you can apply OpenGL projection modes and camera views to transform coordinates so your graphic objects have the correct proportions on any display.

In order to apply projection and camera views, you create a projection matrix and a camera view matrix and apply them to the OpenGL rendering pipeline. The projection matrix recalculates the coordinates of your graphics so that they map correctly to Android device screens. The camera view matrix creates a transformation that renders objects from a specific eye position.

Shape faces and winding

In OpenGL, the face of a shape is a surface defined by three or more points in three-dimensional space. A set of three or more three-dimensional points (called vertices in OpenGL) have a front face and a back face. How do you know which face is front and which is the back? Good question. The answer has to do with winding, or, the direction in which you define the points of a shape.

In this example, the points of the triangle are defined in an order such that they are drawn in a counterclockwise direction. The order in which these coordinates are drawn defines the winding direction for the shape. By default, in OpenGL, the face which is drawn counterclockwise is the front face. The triangle shown in Figure 1 is defined so that you are looking at the front face of the shape (as interpreted by OpenGL) and the other side is the back face.

Why is it important to know which face of a shape is the front face? The answer has to do with a commonly used feature of OpenGL, called face culling. Face culling is an option for the OpenGL environment which allows the rendering pipeline to ignore (not calculate or draw) the back face of a shape, saving time, memory and processing cycles.

OpenGL versions and device compatibility

Graphics programming with OpenGL ES 1.0/1.1 API is significantly different than using the 2.0 and higher versions. The 1.x version of the API has more convenience methods and a fixed graphics pipeline, while the OpenGL ES 2.0 and 3.0 APIs provide more direct control of the pipeline through use of OpenGL shaders

The OpenGL ES 3.0 API provides additional features and better performance than the 2.0 API and is also backward compatible. This means that you can potentially write your application targeting OpenGL ES 2.0 and conditionally include OpenGL ES 3.0 graphics features if they are available.

Texture compression support

Texture compression can significantly increase the performance of your OpenGL application by reducing memory requirements and making more efficient use of memory bandwidth.

Choosing an OpenGL API version

OpenGL ES 1.0 API version (and the 1.1 extensions), version 2.0, and version 3.0 all provide high performance graphics interfaces for creating 3D games, visualizations and user interfaces. Graphics progamming for OpenGL ES 2.0 and 3.0 is largely similar, with version 3.0 representing a superset of the 2.0 API with additional features. Programming for the OpenGL ES 1.0/1.1 API versus OpenGL ES 2.0 and 3.0 differs significantly, and so developers should carefully consider the following factors before starting development with these APIs:

* **Performance** - In general, OpenGL ES 2.0 and 3.0 provide faster graphics performance than the ES 1.0/1.1 APIs. However, the performance difference can vary depending on the Android device your OpenGL application is running on, due to differences in hardware manufacturer's implementation of the OpenGL ES graphics pipeline.
* **Device Compatibility** - Developers should consider the types of devices, Android versions and the OpenGL ES versions available to their customers.
* **Coding Convenience** - The OpenGL ES 1.0/1.1 API provides a fixed function pipeline and convenience functions which are not available in the OpenGL ES 2.0 or 3.0 APIs. Developers who are new to OpenGL ES may find coding for version 1.0/1.1 faster and more convenient.
* **Graphics Control** - The OpenGL ES 2.0 and 3.0 APIs provide a higher degree of control by providing a fully programmable pipeline through the use of shaders. With more direct control of the graphics processing pipeline, developers can create effects that would be very difficult to generate using the 1.0/1.1 API.
* **Texture Support** - The OpenGL ES 3.0 API has the best support for texture compression because it guarantees availability of the ETC2 compression format, which supports transparency. The 1.x and 2.0 API implementations usually include support for ETC1, however this texture format does not support transparency and so you must typically provide resources in other compression formats supported by the devices you are targeting.