GLSL Programming/Per-Fragment Operations

The per-fragment operations are part of the OpenGL (ES) 2.0 pipeline and consist of a series of tests and operations that can modify the fragment color generated by the fragment shader before it is written to the corresponding pixel in the framebuffer. If any of the tests fails, the fragment will be discarded. (An exception to this rule is the stencil test, which can be configured to change the stencil value of the pixel even if the fragment failed the stencil and/or depth test.)

Overview

In the overview of the per-fragment operations, blue boxes represent data and gray boxes represent configurable per-fragment operations.

Fragment Data (Color and Depth)	output of the fragment shader
↓	
Pixel Ownership Test	not customizable by applications
↓	
Scissor Test	fails if fragment is outside of a specified (scissor) rectangle
↓	
Alpha Test	only in OpenGL, not in OpenGL ES 2.x; outcome depends on alpha value (opacity) of the fragment, a reference alpha value and the configured alpha test
↓	
Stencil Test	outcome depends on value in stencil buffer at corresponding pixel and the configured stencil test
↓	
Depth Test	outcome depends on depth of fragment, depth in depth buffer at the corresponding pixel and the configured depth test
↓	
Blending	outcome depends on fragment color (including its opacity), color in framebuffer at corresponding pixel and configured blend equation
↓	

Dithering	usually not customizable by applications
↓	
Logical Operations	only in OpenGL, not in OpenGL ES 2.x; outcome depends on color of fragment, color in framebuffer at corresponding pixel and configured logical (bitwise) operation
↓	
Framebuffer (Color, Depth and Stencil Value)	writing to the framebuffer can be disabled for individual channels

Specific Per-Fragment Operations

Some of the per-fragment operations are particularly important because they have established applications:

- The **depth test** is used to render opaque primitives (e.g. triangles) with correct occlusions. This is done by comparing the depth of a fragment to the depth of the frontmost previously rendered fragment, which is stored in the depth buffer. If the fragment is farther away, then the depth test fails and the fragment is discarded. Otherwise the fragment is the new frontmost fragment and its depth is stored in the depth buffer.
- **Blending** is used to render semitransparent primitives (glass, fire, flares, etc.) by blending (i.e. compositing) the color of the fragment with the color that is already stored in the framebuffer. This is usually combined with disabling writing to the depth buffer in order to avoid that semitransparent primitives occlude other primitives.
- The **stencil test** is often used to restrict rendering to certain regions of the screen, e.g. a mirror, a window or a more general “portal” in the 3D scene. It also has more advanced uses for some algorithms, e.g. shadow volumes.

More details about specific per-fragment operations can be found in the platform-specific tutorials because it depends on the platform, how the operations are configured.

Note on the Order of Per-Fragment Operations

While the specification of OpenGL imposes a certain order of the per-fragment operations, GPUs can change the order of these operations as long as this doesn't change the result. In fact, GPUs will perform many of the tests even before the fragment shader is executed whenever this is possible. In this case the fragment shader is not executed for a fragment that has failed one of the test. (One example is the so-called “early depth test.”) This can result in considerable performance gains.

Programmability of Per-Fragment Operations

Per-fragment operations are usually configurable but not programmable, i.e. the various tests can be configured in certain ways but you cannot write a program to compute the tests. However, this is changing. For example, OpenGL ES 2.0 offers no alpha test because the same functionality can be achieved by a conditional discard operation in a fragment shader, which is effectively a programmable alpha test. Another example is the (multi-vendor) OpenGL ES extension `GL_EXT_shader_framebuffer_fetch` (http://www.khronos.org/registry/gles/extensions/EXT/EXT_shader_framebuffer_fetch.txt), which offers programmable blending. (This extension is supported on most (if not all) iOS 6 devices. In beta versions of iOS 6, the extension was available as `GL_APPLE_shader_framebuffer_fetch`. On devices with Nvidia's Tegra chip, the similar extension `GL_NV_shader_framebuffer_fetch` is often available.)

Further Reading

The per-fragment operations of OpenGL ES 2.0 are defined in full detail in Chapter 4 of the “OpenGL ES 2.0.x Specification” available at the “Khronos OpenGL ES API Registry” (http://www.khronos.org/registry/gles/).

A more accessible description is given in Chapter 11 of the book “OpenGL ES 2.0 Programming Guide” by Aaftab Munshi, Dan Ginsburg and Dave Shreiner published by Addison-Wesley (see its web site (http://www.opengles-book.com/)).

< GLSL Programming