Software Test Specification

OpenGL ES Shading Language

Filename	:	Software Test Specification.OpenGL ES Shading Language.doc
Version	:	1.0.42a External Issue
Issue Date	:	26 Feb 2010
Author	:	PCG

Contents

1. Introduction

This document specifies the tests used during SQA to validate the implementation of Open GL ES 2.0, the OpenGL ES Shading Language compilers and the correct operation of compiled shader programs on Imagination Technologies Hardware IP Cores.

1.1. Related Documentation

Title	Description
Software Test Specification.OpenGL ES 2.0.doc	The OpenGL ES 2.0 test specification.

1.2. Assumptions

This document is written with the following assumptions:

The reader is familiar with OpenGL ES in general, OpenGL ES 2.0 in particular, the OpenGL ES Shading Language, and the idea of online and offline shader compilers.

1.3. Document Scope

This specification describes the tests to be performed to verify the implementation of the OpenGL Shading Language compiler for OpenGL ES 2.0 and OpenGL ES GLSL 1.00.

2. Test Coverage

This section details the features the OpenGL Shading Language has been split into for testing and the risk associated with each feature not being tested completely.

2.1. Features for Testing

Feature	Description	Risk
Offline compiler	Functions used to compile binaries external to the OpenGL application	High
Online compiler	Functions used to compile binaries within the OpenGL application	High
Vertex Shader Special Outputs	Built-in read/write output variables a vertex shader can or must write to	High
Fragment Shader Special Inputs	Built-in read-only variables available to a fragment shader	High
Fragment Shader Special Outputs	Built-in read/write variables a fragment shader can or must write to	High
User-Defined Attribute Variables	User-defined frequently changing variables available from the OpenGL application to a vertex shader	Med
User-Defined Uniform Variables	User-defined infrequently changing variables available from the OpenGL application to either a vertex or a fragment shader	Med
User-Defined Varying Variables	User-defined variables used in a vertex shader and passed to a fragment shader	Med
Texture Maps	Functions that allow the loading of and use of textures within a shader	Med
Built-In Constants	Built-in constant values available to a shader	Low
Built-In Functions	Built-in functions for use in vertex and fragment shaders	Low
User-Defined Functions	User-defined functions that can be used in a shader program	Low
Constructors and Type Conversions	Initialisation of constructed values Explicit conversions between different data types	Med
Data Precision	The precision of data for all data types	Med
Preprocessor	The processing of source strings as part of the compilation process	High
Reserved Words	Reserved but unused keywords	Med
Arithmetic and Boolean Operations	Operations using built in operators	High
Arrays and Structures	Declaration and usage of arrays and structures	High
Variable Scoping	The scope and hiding of variables	Med

Please refer to Appendix A for full details of functions and variables being tested.

2.2. Feature Test Details

Feature	Test Details

Feature	Test Details
Offline Compiler	Syntactically correct shaders compile correctly Syntactically incorrect shaders fail to compile Generated shader binaries can be attached and used within an OpenGL application Exercise all OpenGL Shading Language features (as defined below)
Online Compiler	Syntactically correct shaders compile correctly Syntactically incorrect shaders fail to compile Generated shader binaries can be attached and used within an OpenGL application Exercise all OpenGL Shading Language features (as defined below)
Vertex Shader Special Outputs	Output variables are correctly communicated to the OpenGL application Required variables must be written to
Fragment Shader Special Inputs	All variables are correctly read from the OpenGL state Variables are read-only
Fragment Shader Special Outputs	Output variables are correctly communicated to the OpenGL application
User-Defined Attribute Variables	User defined variables are correctly read from the OpenGL application Up to Maximum number of user-defined variables are correctly read from the OpenGL application
User-Defined Uniform Variables	User defined variables are correctly read from the OpenGL application Up to Maximum number of user-defined variables are correctly read from the OpenGL application
User-Defined Varying Variables	User defined variables are correctly passed from the vertex shader to the fragment shader Up to Maximum number of user-defined variables are correctly passed from the vertex shader to the fragment shader
Texture Maps	Textures can be loaded in an OpenGL application and used within a shader Texture functions work correctly
Built-In Constants	Constant values are accessible from within a shader Constant values are correct for the OpenGL Shading Language implementation
Built-In Functions	Functions work correctly within a shader Functions accept and return correct data types
User-Defined Functions	Functions can be defined within a shader Defined function parameters work correctly Expected values are returned correctly
Constructors and Type Conversions	Data can be initialised correctly Data can be converted to different types using constructors
Data Precision	Data returned from functions, converted between types, and passed to and from shaders retains appropriate levels of precision

Feature	Test Details
Preprocessor	Preprocessor directives, operators and predefined macros are interpreted correctly
Reserved Words	Reserved keywords that are unused and should cause syntax errors
Arithmetic and Boolean Operations	Arithmetic and Boolean operators work correctly
Arrays and Structures	The semantics for creating, using and manipulating arrays and structures within a shader
Variable Scoping	Variables can be declared within a specific region of a shader program are guaranteed to be visible Variables can be redefined within a shader using a system of statically nested scopes Built in variables with global scope should be visible throughout the shader program

3. Test Catalogue

3.1. Functional Tests

Test ID	Application	Description	Verif.	Source
GES_FN01	GLSLEStriangle	Minimal GLSL smoke test	M	IMG
GES_FN02	GLSLESparser	Tests the correct parsing of shaders	A	IMG
GES_FN03	GLSLESofflineparser*	Tests the correct parsing of shaders using an offline compiler	A	IMG
GES_FN04	GLSLEStestkit	Tests all OpenGL Shading Language features	A	IMG
GES_FN05	GLSLESgolden*	Tests a library of “golden” compiled shader binaries for correct and compatible operation with forward driver revisions	M	IMG

** tests are not yet implemented and are subject to change*

3.2. Non-Functional Tests

3.2.1. Stress Tests

Test ID	Application	Description	Verif.	Source
GES_SR01	GLSLESvertexstress*	Stresses the vertex shader part of the driver and hardware	M	IMG
GES_SR02	GLSLESfragmentstress*	Stresses the fragment shader part of the driver and hardware	M	IMG

**tests are not yet available and are subject to change*

4. Test Cases

This section details the test cases required for the OpenGL Shading Language compiler, the test groups they belong to, and the failure criteria for the defined tests.

4.1. Test Case Definitions

4.1.1. GES_FN01 - Triangle Test

Test Cases

Test Case ID	Function/Variable Under Test	Description	Expected Result
101	ALL	Draw a triangle. Transform the vertices with a vertex shader and colour the fragments with a fragment shader	Transformed triangle is displayed and is the correct colour

4.1.2. GES_FN02 - Parser Test

Test Groups

Test Group ID	Test Group Name
100	Syntactically Correct Shaders
200	Syntactically Incorrect Shaders

Test Cases

Test Case ID	Function/Variable Under Test	Description	Expected Result
101	glCompileShader glCreateProgram glAttachShader glLinkProgram glUseProgram	Attempt to compile a large number of syntactically correct shaders, using each of the OpenGL Shading Language features	Shaders will compile, link, and attach to an OpenGL application
201	glCompilerShader	Attempt to compile a large number of syntactically incorrect shaders	Shaders will fail to compile

4.1.3. GES_FN03 - Offline Parser Test

Test Groups

Test Group ID	Test Group Name
100	Syntactically Correct Shaders
200	Syntactically Incorrect Shaders

Test Cases

Test Case ID	Function/Variable Under Test	Description	Expected Result

Test Case ID	Function/Variable Under Test	Description	Expected Result
101	glGetShaderPrecisionFormatOES glShaderBinaryOES glReleaseShaderCompilerOES glCreateProgram glAttachShader glLinkProgram glUseProgram	Attempt to compile a large number of syntactically correct shaders, using each of the OpenGL Shading Language features	Shaders will compile, link, and attach to an OpenGL application
201		Attempt to compile a large number of syntactically incorrect shaders	Shaders will fail to compile

4.1.4. GES_FN04 - Testkit

Test Groups

Test Group ID	Test Group Name
300	Vertex Shader Special Outputs
400	Fragment Shader Special Inputs
600	Fragment Shader Special Outputs
700	User-Defined Attribute Variables
800	User-Defined Uniform Variables
900	User-Defined Varying Variables
1000	Texture Maps
1100	Built-In Constants
1300	Built-In Functions – Trigonometry
1400	Built-In Functions – Exponential
1500	Built-In Functions – Common
1600	Built-In Functions – Geometric
1700	Built-In Functions – Fragment Processing
1800	Built-In Functions – Matrix
1900	Built-In Functions – Vector Relational
2000	Built-In Functions – Texture Lookup
2100	Built-In Functions – Texture Lookup with LOD
2200	Built-In Functions – Noise
2300	User-Defined Functions
2400	Constructors and Type Conversions
2500	Data Precision

Test Group ID	Test Group Name
2600	OpenGL ES Invariance
2700	Preprocessor
2800	Reserved Words
2900	Arithmetic and Boolean Operations
3000	Arrays and Structures
3100	Variable Scoping

Test Cases

Test Case ID	Function/Variable Under Test	Description	Expected Result
301	gl_Position	For a range of vertices, write the vertex position in clipping coordinates	The vertices should be positioned correctly when finally written to the frame buffer The compiler should fail if **gl_Position** is not written to
303	gl_Position	For a range of vertices, do not write to **gl_Position**	Behaviour is undefined, but a warning should be issued
304	gl_PointSize	With **GL_VERTEX_PROGRAM_POINT_SIZE** enabled, specify a range of point sizes	The diameter of each point primitive, in pixels, should be used when finally written to the frame buffer
401	gl_FragCoord	Specify a number of vertices and primitives in an OpenGL application	The window relative coordinates, *x, y, z* and *1/w* should be accessible for each fragment
402	gl_FrontFacing	Using **glFrontFace**, specify a number of front and back-facing primitives in an OpenGL application	The value should be true if the fragment belongs to a front-facing primitive, and false otherwise
403	gl_PointCoord		
601	gl_FragColor	Specify a range of colours, *x*, in a fragment shader	The colour, *x*, should be passed through the OpenGL pipeline and ultimately be written to the frame buffer as the colour for the fragment
602	gl_FragData	Specify a range of values, *x*, in a fragment shader	The offscreen buffers, specified by **glDrawBuffers**, should be modified, with **gl_FragData[0]** updating the first buffer, and so on up to **GL_MAX_DRAW_BUFFER***S*
604	discard	Create a number of polygons in an OpenGL application and assign the fragments a range of **gl_Frag** data in a fragment shader. Use the **discard** keyword to discard a number of fragments	Fragments that the shader executes with the **discard** keyword should not be passed through the OpenGL pipeline and no update of the frame buffer contents should be performed Fragments that the shader executes without the **discard** keyword should be passed through the OpenGL pipeline and update the frame buffer as normal

Test Case ID	Function/Variable Under Test	Description	Expected Result
701	attribute	Using **glVertexAttrib** and **glBindAttribLocation**, specify a number of frequently changing attribute variables, up to **GL_MAX_VERTEX_ATTRIBS**, in an OpenGL application, and read them in a vertex shader	All variables should be correctly passed from the OpenGL application to the vertex shader
702	attribute	Using vertex arrays, specify a number of frequently changing attribute variables, up to **GL_MAX_VERTEX_ATTRIBS**, in an OpenGL application, and read them in a vertex shader	All variables should be correctly passed from the OpenGL application to the vertex shader
801	uniform	Using **glUniform** and **glGetUniformLocation**, specify a number of infrequently changing uniform variables, up to **GL_MAX_VERTEX_UNIFORM_COMPONENTS**, in an OpenGL application and read them in a vertex and fragment shader	All variables should be correctly passed from the OpenGL application to both the vertex and the fragment shader
901	varying	Specify a number of varying variables, up to **GL_MAX_VARYING_FLOATS**, in a vertex shader and read the variables in a fragment shader	All variables passed from the vertex shader should be accessible from the fragment shader
902	varying	Attempt to read a number of varying variables in a fragment shader that have not been passed from a vertex shader	The compiler should fail, as varying variables must be passed in from the vertex shader
903	invariant	Declare a number of shader output variables with the **invariant** qualifier in different shaders	The output values from the same expression in different shaders should be the same
904	invariant	Attempt to declare non-output variables as invariant	The shader should fail to compile
905	invariant	Use **#pragma STDGL invariant(all)** to force all output variables to be invariant	The output values from the same expression in different shaders should be the same
1002	sampler2D	Load a range of 2D textures into each of the available texture units in an OpenGL application, and access these textures within a shader	Each of the textures should be accessible from within a shader using the built-in texture functions
1003	sampler3D	Load a range of 2D textures into each of the available texture units in an OpenGL application, and access these textures within a shader	Each of the textures should be accessible from within a shader using the built-in texture functions

Test Case ID	Function/Variable Under Test	Description	Expected Result
1004	samplerCube	Load a range of cube map textures into each of the available texture units in an OpenGL application, and access these textures within a shader	Each of the textures should be accessible from within a shader using the built-in texture functions
1101	gl_MaxCombinedTextureImageUnits	Check the value of the constant in a vertex and a fragment shader	Constant should be accessible and have a minimum value of **2**
1102	gl_MaxVaryingVectors	Check the value of the constant in a vertex and a fragment shader	Constant should be accessible and have a minimum value of **8**
1103	gl_MaxDrawBuffers	Check the value of the constant in a vertex and a fragment shader	Constant should be accessible and have a minimum value of **1**
1104	gl_MaxTextureImageUnits	Check the value of the constant in a vertex and a fragment shader	Constant should be accessible and have a minimum value of **2**
1105	gl_MaxFragmentUniformVectors	Check the value of the constant in a vertex and a fragment shader	Constant should be accessible and have a minimum value of **16**
1106	gl_MaxVertexAttribs	Check the value of the constant in a vertex and a fragment shader	Constant should be accessible and have a minimum value of **8s**
1107	gl_MaxVertexTextureImageUnits	Check the value of the constant in a vertex and a fragment shader	Constant should be accessible and have a minimum value of **0**
1108	gl_MaxVertexUniformVectors	Check the value of the constant in a vertex and a fragment shader	Constant should be accessible and have a minimum value of **128**
1301	sin	Pass a range of values, *x*, to the function	The sine of *x* should be returned
1302	cos	Pass a range of values, *x*, to the function	The cosine *x* should be returned
1303	tan	Pass a range of values, *x*, to the function	The tangent of *x* should be returned
1304	asin	Pass a range of values, x, to the function	An angle whose sine is *x* should be returned
1305	acos	Pass a range of values, x, to the function	An angle whose cosine is *x* should be returned
1306	atan	Pass a range of values, x and y, to the function	An angle whose tangent is *y/x* should be returned
1307	radians	Pass a range of values, *degrees*, to the function	pi / 180 * *degrees* should be returned
1308	degrees	Pass a range of values, *radians*, to the function	180 / pi * *radians* should be returned

Test Case ID	Function/Variable Under Test	Description	Expected Result
1401	pow	Pass a range of values, *x* and *y*, to the function	*x* raised to the *y* power should be returned
1402	exp	Pass a range of values, *x*, to the function	The natural exponentiation of *x* should be returned
1403	log	Pass a range of values, *x*, to the function	The natural logarithm of *x* should be returned
1404	exp2	Pass a range of values, *x*, to the function	2 raised to the *x* power should be returned
1405	log2	Pass a range of values, *x*, to the function	The base 2 log of *x* should be returned
1406	sqrt	Pass a range of values, *x*, to the function	The positive square root of *x* should be returned
1407	inversesqrt	Pass a range of values, *x*, to the function	The reciprocal of the positive square root of *x* should be returned
1501	abs	Pass a range of values, *x*, to the function	The absolute value of *x* should be returned
1502	ceil	Pass a range of values, *x*, to the function	The nearest integer that is greater than or equal to *x* should be returned
1503	clamp	Pass a range of values, *x*, *minVal* and *maxVal* to the function	***min*(*max*(*x, minVal),maxVal)* should be returned
1504	floor	Pass a range of values, *x*, to the function	The nearest integer that is less than or equal to *x* should be returned
1505	fract	Pass a range of values, *x*, to the function	*x* – ***floor***(*x*) should be returned
1506	max	Pass a range of values, *x and y*, to the function	If *x* is less than *y*, *y* should be returned, otherwise *x* should be returned
1507	min	Pass a range of values, *x and y*, to the function	If *y* is less than x, *y* should be returned, otherwise *x* should be returned
1508	mix	Pass a range of values, *x*, *y*, and *a* to the function	The linear blend of *x* and *y* using the floating-point value *a* should be returned. *(x * (1.0) – a + y * a)*
1509	mod	Pass a range of values, *x* and *y* to the function	The modulus of *x* using *y* should be returned *(x – y * **floor** (x/y))*
1510	sign	Pass a range of values, *x*, to the function	If *x* is greater than 0, 1.0 should be returned If *x* is equal to 0, 0.0 should be returned If *x* is less than 0, -1.0 should be returned

Test Case ID	Function/Variable Under Test	Description	Expected Result
1511	smoothstep	Pass a range of values, *edge0, edge1* and *x* to the function	If *x* <= *edge0*, 0 should be returned. If *x* >= *edge1*, 1.0 should be returned If *edge0* < *x* < *edge1*, smooth Hermite interpolation between 0 and 1 should be performed and returned If *edge0* >= *edge1*, results are undefined
1512	step	Pass a range of values, *edge* and *x,* to the function	If *x* < *edge*, 0 should be returned, otherwise 1.0 should be returned
1602	cross	Pass a range of 3-component vectors, *x* and *y*, to the function	The cross product of *x* and *y* should be returned
1603	distance	Pass a range of values, *p0* and *p1* to the function	The distance between *p0* and *p1* should be returned (**length**(p0-p1))
1604	dot	Pass a range of values, *x* and *y*, to the function	The dot product of *x* and *y* should be returned
1605	faceforward	Pass a range of values, *N*, *I* and *Nref* to the function	If **dot**(*Nref,I*) < 0.0, *N* should be returned, otherwise –*N* should be returned
1606	length	Pass a range of values, *x*, to the function	The length of vector *x* should be returned If *x* is a float, the returned value should be the same as the absolute value
1607	normalise	Pass a range of values, *x*, to the function	A vector in the same direction as *x*, of length 1, should be returned If *x* is a float, the returned value should always be 1
1608	reflect	Pass a range of values, *I* and *N* to the function	For the incident vector *I* and surface orientation *N*, the reflection direction should be returned *N* must already be normalised (I – 2.0 * **dot**(N, I) * N)

1609	refract	Pass a range of values, *I*, *N* and *eta* to the function	For the incident vector *I*, surface normal *N*, and ratio of indices to refraction *eta*, the refraction vector should be returned
			k = 1.0 – eta * eta * (1.0 – **dot**(N, I) * **dot**(N, I))
			if (k < 0.0)
			result = 0.0
			else
			result = eta * I – (eta * **dot**(N, I) * **sqrt**(k)) * N

Test Case ID	Function/Variable Under Test	Description	Expected Result
1701	dFdx	In a fragment shader, pass a range of values, *p*, to the function	The derivative in the x-direction for the input argument *p* should be returned
1702	dFdy	In a fragment shader, pass a range of values, *p*, to the function	The derivative in the x-direction for the input argument *p* should be returned
1703	fwidth	In a fragment shader, pass a range of values, *p*, to the function	The sum of the absolute derivative in x and y for the input argument *p* should be returned *result* = ***abs*(*dFdx*(*p*)) + *abs*(*dFdy*(*p*))**
1801	matrixCompMult	Pass a range of matrices, *x* and *y*, to the function	The component-wise multiplication of the two matrices should be returned
1901	all	Pass a range of vectors, *x*, to the function	If all of the components of *x* are true, true should be returned
1902	any	Pass a range of vectors, *x*, to the function	If any of the components of *x* are true, true should be returned
1903	equal	Pass a range of vectors, *x* and *y*, to the function	The component-wise compare of *x* == *y* should be returned
1904	greaterThan	Pass a range of vectors, *x* and *y*, to the function	The component-wise compare of *x* > *y* should be returned
1905	greaterThanEqual	Pass a range of vectors, *x* and *y*, to the function	The component-wise compare of *x* >= *y* should be returned
1906	lessThan	Pass a range of vectors, *x* and *y*, to the function	The component-wise compare of *x* < *y* should be returned
1907	lessThanEqual	Pass a range of vectors, *x* and *y*, to the function	The component-wise compare of *x* <= *y* should be returned
1908	not	Pass a range of vectors, *x*, to the function	The component-wise logical complement of *x* should be returned
1909	notEqual	Pass a range of vectors, *x* and *y*, to the function	The component-wise compare of *x* != *y* should be returned
2003	texture2D	Pass a range of values, *sampler*, *coord* and *bias*, to the function	The 2D texture currently specified by *sampler* should be accessed using texture coordinate *coord* and the corresponding vec4 should be returned
2004	texture2DProj	Pass a range of values, *sampler*, *coord* and *bias*, to the function	The 2D texture currently specified by *sampler* should be accessed using texture coordinate *coord*, with *coord.s*, *coord.t* being divided by the last component of *coord* The corresponding vec4 should be returned

Test Case ID	Function/Variable Under Test	Description	Expected Result
2005	texture3D	Pass a range of values, *sampler*, *coord* and *bias*, to the function	The 3D texture currently specified by *sampler* should be accessed using texture coordinate *coord* and the corresponding vec4 should be returned
2006	texture3DProj	Pass a range of values, *sampler*, *coord* and *bias*, to the function	The 3D texture currently specified by *sampler* should be accessed using texture coordinate *coord* divided by *coord.q* The corresponding vec4 should be returned
2007	textureCube	Pass a range of values, *sampler*, *coord* and *bias*, to the function	The cube map texture currently specified by *sampler* should be accessed using texture coordinate *coord*, with the direction of *coord* being used to select the face in which to do a two-dimensional texture lookup The corresponding vec4 should be returned
2103	texture2DLod	In a vertex shader, pass a range of values, *sampler*, *coord* and *lod*, to the function	The 2D texture currently specified by *sampler* should be accessed using texture coordinate *coord* with level-of-detail *lod*, and the corresponding vec4 should be returned
2104	texture2DProjLod	In a vertex shader, pass a range of values, *sampler*, *coord* and *lod*, to the function	The 2D texture currently specified by *sampler* should be accessed using texture coordinate *coord*, with *coord.s*, *coord.t* being divided by the last component of *coord*, and with level-of-detail *lod* The corresponding vec4 should be returned
2105	texture3DLod	In a vertex shader, pass a range of values, *sampler*, *coord* and *lod*, to the function	The 3D texture currently specified by *sampler* should be accessed using texture coordinate *coord* with level-of-detail *lod*, and the corresponding vec4 should be returned
2106	texture3DProjLod	In a vertex shader, pass a range of values, *sampler*, *coord* and *lod*, to the function	The 3D texture currently specified by *sampler* should be accessed using texture coordinate *coord* divided by *coord.q* and with level-of-detail *lod* The corresponding vec4 should be returned

Test Case ID	Function/Variable Under Test	Description	Expected Result
2107	textureCubeLod	Pass a range of values, *sampler*, *coord* and *lod*, to the function	The cube map texture currently specified by *sampler* should be accessed with level-of-detail *lod* and using texture coordinate *coord*, with the direction of *coord* being used to select the face in which to do a two-dimensional texture lookup The corresponding vec4 should be returned
2201	noise1	Pass a range of values, *x*, to the function	A 1D noise value based on *x* should be returned in the range [-1, 1], covering at least [-0.6, 0.6] with a Gaussian-like distribution, an overall average of 0, and the result should be repeatable for the same input value.
2202	noise2	Pass a range of values, *x*, to the function	A 2D noise value based on *x* should be returned in the range [-1, 1], covering at least [-0.6, 0.6] with a Gaussian-like distribution, an overall average of 0, and the result should be repeatable for the same input value.
2203	noise3	Pass a range of values, *x*, to the function	A 3D noise value based on *x* should be returned in the range [-1, 1], covering at least [-0.6, 0.6] with a Gaussian-like distribution, an overall average of 0, and the result should be repeatable for the same input value.
2204	noise4	Pass a range of values, *x*, to the function	A 4D noise value based on *x* should be returned in the range [-1, 1], covering at least [-0.6, 0.6] with a Gaussian-like distribution, an overall average of 0, and the result should be repeatable for the same input value.
2301	in	Pass a number of variables to a function using the **in** qualifier	The data should be passed to the function by value, the variable can be written to, but should not be copied back out
2302	const in	Pass a number of variables to a function using the **const in** qualifier	The data should be passed to the function by value, the variable should not be able to be written to, and should not be copied back out to the caller
2303	out	Specify a number of parameters of a function using the **out** qualifier	The variable should be undefined at entry to the function, it should be readable and writeable, and it should be copied back out to the caller

Test Case ID	Function/Variable Under Test	Description	Expected Result
2304	inout	Pass a number of variables to a function using the **inout** qualifier	The data should be passed to the function by value, it may be modified within the function, and should be passed back out to the caller
2305	return	Functions can be declared as void, or can return a value. Create a number of functions using void and all other data types	Functions should return data of the specified data type
2036	n/a	Attempt to call a function recursively	The compiler should fail, as functions may not be called recursively, either directly or indirectly
2401	n/a	Convert data between all valid data types	Data should be converted correctly to the new data type
2411	Initialising Data!		
2501	n/a	Pass a range of variables and data types from an OpenGL application to a shader, as well as to built-in and user-defined functions	Variables should retain their precision, and results should be calculated to the required level of precision
2502	precision	The default precision in a vertex shader is highp Declare a range of variables and perform a number of operations on them	All variables should have and retain high precision
2503	precision	The default precision in a fragment shader is mediump Declare a range of variables and perform a number of operations on them	All variables should have and retain at least medium precision
2504	precision	Use the precision statement **precision *precision-qualifier type*** to establish a default precision qualifier for *int* or *float* variables	All variables declared in the current scope should have and retain at least *precision-qualifier* precision
2505	lowp	Declare a number of variables using the **lowp** precision-qualifier and perform a range of operations on the variables	Variables should maintain at least this level of precision - 2^{-8}
2506	medp	Declare a number of variables using the **medp** precision-qualifier and perform a range of operations on the variables	Variables should maintain at least this level of precision - 2^{-10}
2507	highp	Declare a number of variables using the **highp** precision-qualifier and perform a range of operations on the variables	Variables should maintain at least this level of precision - 2^{-16}

Test Case ID	Function/Variable Under Test	Description	Expected Result
2600		OpenGL ES Invariance	
2700	#define		
2701	#undef		
2702	#if		
2703	#ifdef		
2704	#ifndef		
2705	#else		
2706	#elif		
2707	#endif		
2708	#error		
2709	#pragma		
2710	#extensions		
2711	defined		
2720	__LINE__	Place the __**LINE**__ macro in a number of different places in a range of shaders	
2721	__FILE__		
2722	__VERSION__		
2723	GL_ES		
2800	n/a	Use the reserved but unused keywords in a shader	The shader should fail to compile, producing a syntax error
2900	()	Parenthetical Grouping	
2901	[]		
2902	()		
2903	.		
2904	++		
2905	--		
2906	++		
2907	--		
2908	+		
2909	-		
2910	!		
2911	*		
2912	/		
2913	+		
2914	-		
2915	<		

Test Case ID	Function/Variable Under Test	Description	Expected Result
2916	>		
2917	<=		
2918	>=		
2919	==		
2920	!=		
2921	&&		
2922	^^		
2923	? :		
2924	=		
2925	+=		
2926	-=		
2927	*=		
2928	/=		
2929	,		
3000	arrays	Declare and use a number of arrays	Arrays should be created and used based on the semantics in the language specification
3010	structures	Declare and use a number of structures	Structures should be created and used based on the semantics in the language specification
3100	n/a	Read the values from built in variables with global scope in various places in the program	Variables should be visible throughout the program
3110	n/a	Declare a number of variables within a specific region of a program	The variable should only be visible in the specific region of the program
3120	n/a	Redefine a number of variables within statically nested scopes	The variable in the correct scope should be visible

4.1.5. GES_FN05 - Pre-compiled “Golden” Binary Test

Test Groups

Test Group ID	Test Group Name
100	Compiled Shader Binaries

Test Cases

Test Case ID	Function/Variable Under Test	Description	Expected Result
101	n/a	Load and use a number of pre-compiled binary shaders within an OpenGL application	Shaders should run and OpenGL output should match reference images

4.1.6. GES_SR01 - Vertex Shader Stress Test

Test Groups

Test Group ID	Test Group Name
100	Basic Stress Test

Test Cases

Test Case ID	Function/Variable Under Test	Description	Expected Result
101	n/a	For a complex scene with a lot of geometry, stress the vertex shader by using multiple textures (all texture units), multiple lights writing multiple varying outputs, using multiple user-defined attribute and uniform variables, executing multiple built-in functions, and transforming the vertices	Output should match reference images

4.1.7. GES_SR02 - Fragment Shader Stress Test

Test Groups

Test Group ID	Test Group Name
100	Basic Stress Test

Test Cases

Test Case ID	Function/Variable Under Test	Description	Expected Result
101	n/a	For a complex scene with a lot of geometry, stress the fragment shader by using multiple textures (all texture units), multiple lights, reading multiple varying inputs, using multiple user-defined attribute and uniform variables, executing multiple built-in functions, and colouring and setting FragDepth for the fragments	Output should match reference images

5. Review of this Document

5.1. Signatories

Signed off for document version:	

5.2. Change Control

Any changes to this document will result in a new version with appropriate sign off.

To do this the Signatories Section will need to be duplicated and the dates and version removed. Please leave the old details also present in the document so that the original signed version can be found.

Appendix A. Function Coverage

This section details the OpenGL Shading Language functions that will be targeted during testing.

Feature	Function/Variable
Offline Compiler	
Online Compiler	
OpenGL API Entry Points	glCreateShader glShaderSource
	glCompileShader
	glCreateProgram glAttachShader glLinkProgram glUseProgram
	glDeleteShader glDeleteProgram glDetachShader
	glGetShaderiv glGetProgramiv glGetShaderSource glGetShaderInfoLog glGetProramInfoLog glGetAttachedShaders glIsShader glIsProgram
	glVertexAttrib glVertexAttrib4Nub glVertexAttribPointer glEnableVertexAttribArray glDisableVertexAttribArray glBindAttribLocation glGetAttribLocation glGetActiveAttrib glGetVertexAttrib glGetVertexAttribPointerv
	glGetUniformLocation glUniform glUniformMatrix glGetUniform glGetActiveUniform
	glDrawBuffers
	glValidateProgram

Feature	Function/Variable
	glShaderBinaryOES glReleaseShaderCompilerOES glGetShaderPrecisionFormatOES
Vertex Shader Special Outputs	gl_Position gl_PointSize
Fragment Shader Special Inputs	gl_FragCoord gl_FrontFacing
	gl_PointCoord
Fragment Shader Special Outputs	gl_FragColor gl_FragData gl_FragDepth discard
	discard
User-Defined Attribute Variables	attribute
	glVertexAttrib* glVertexAttribPointer* glBindAttribLocation* glGetVertexAttrib* glGetVertexAttribPointer* glEnableVertexAttribArray* glDisableVertexAttribArray*
User-Defined Uniform Variables	uniform
	glUniform* glUniformMatrix* glGetUniform* glGetUniformLocation*
User-Defined Varying Variables	varying
	invariant
Texture Maps	glActiveTexture^ glBindTexture^ glTexParameter^ glTexImage^
	sampler2D sampler3D samplerCube
Built-In Constants	gl_MaxClipPlanes gl_MaxCombinedTextureImageUnits

Feature	Function/Variable
Built-In Functions	sin cos tan asin acos atan radians degrees
	pow exp log exp2 log2 sqrt inversesqrt
	abs ceil clamp floor fract max min mix mod sign smoothstep step
	cross distance dot faceforward length normalize reflect refract
	dFdx dFdy fwidth
	matrixCompMult

Feature	Function/Variable
	all any equal greaterThan greaterThanEqual lessThan lessThanEqual not notEqual
	texture2D texture2DProj texture3D texture3DProj textureCube
	texture2DLod texture2DProjLod texture3DLod texture3DProjLod textureCubeLod
	noise1 noise2 noise3 noise4
User-Defined Functions	in out inout return
Data Precision	lowp medp highp

Feature	Function/Variable	
Preprocessor	#define #undef #if #ifdef #ifndef #else #elif #endif #error #pragma #extension #version #line defined __LINE__ __FILE__ __VERSION__ GL_ES	
Reserved Words	asm class union enum typedef template this packaged goto switch default inline noinline volatile public static extern external interface flat long short double half fixed unsigned superp input output hvec2 hvec3 hvec4 dvec2 dvec3 dvec4 fvec2 fvec3 fvec4 sampler1D sampler3D sampler1DShadow sampler2DShadow sampler2DRect sampler3DRect sampler2DRectShadow sizeof cast namespace using __*	
Arithmetic and Boolean Operations	*In order of precedence. Those in brackets {} are reserved.*	
	parenthetical grouping	()
	array subscript function call, constructor structure field selector, swizzler post fix increment, decrement	[] () . ++ --
	prefix increment, decrement unary	++ -- + - {~} !
	multiplicative	* / {%}
	additive	+ -
	bit-wise shift	{<<} {>>}
	relational	< > <= >=

Feature	Function/Variable	
	equality	== !=
	bit-wise and	{&}
	bit-wise exclusive or	{^}
	bit-wise inclusive or	{|}
	logical and	&&
	logical exclusive or	^^
	logical inclusive or	||
	selection	? :
	assignment arithmetic assignments	= += -= *= /= {%=} {<<=} {>>=} {&=} {^=} {|=}
	sequence	,

** OpenGL function used to test Shading Language feature and explicitly tested in OpenGL API Entry Points*

^ OpenGL function used to test Shading Language feature but not explicitly tested