Appendix C
OpenGL ES Shading Language

GLSL ES Language Reference

This appendix provides a reference for the GLSL ES language used to program shaders for WebGL. For further details on GLSL ES, see the official GLSL ES specification, available at. `www.khronos.org/registry/gles/specs/2.0/GLSL_ES_Specification_1.0.17.pdf`.

Data types

GLSL ES has its own set of data types, completely separate from JavaScript. Table C-1 lists these data types.

Table C-1 Data types

Name	Description

`void`	Special type used when a function has no return value or to indicate an empty parameter list.
	Example:
	`void myFunc(void) {`
	`  ...`
	`}`

`int`	Signed integer value.
	Example:
	`int myInt1 = 14;`
	`int myInt2 = -7;`

`float`	Single-precision floating-point value. Example: `float myFloat = 3.14159;`
`bool`	Boolean true/false value. Example: `bool myBool = true;`
`vec2, vec3, vec4`	Vectors with two, three, or four floating-point components. Example: `vec3 myVec = vec3(1.0, 2.5, 0.25);`

Name	Description
`bvec2, bvec3, bvec4`	Vectors with two, three, or four Boolean components. Example: `bvec3 myBoolVec = bvec3(true, false, false);`
`ivec2, ivec3, ivec4`	Vectors with two, three, or four integer components. Example: `ivec3 myIntVec = ivec3(12, -4, 84);`

`mat2, mat3, mat4`	2x2, 3x3, and 4x4 floating-point matrices.
	Example:
	`mat2 myMat = mat2(`
	`  1.0, 0.5,`
	`  2.7, 1.5`
	`);`

`sampler2D`	2D texture that can be accessed with the `texture2D` functions.
	Example:
	`sampler2D uTexture;`
	`vec4 color = texture2D(uTexture, texCoords);`

`samplerCube`	Cube-mapped texture that can be accessed with `textureCube` functions.
	Example:
	`samplerCube uSkyMap;`
	`vec4 color = textureCube(uSkyMap, texCoords);`

Built-in functions

The following tables describe the built-in GLSL ES functions. The functions are grouped in tables based on the area of the functionality they provide.

Return values and parameters declared with the generic `genType` type can take `float`, `vec2`, `vec3`, or `vec4` values. All `genType` arguments used in a function call must be of the same type. The type of the argument determines the type of the return value.

Math functions

Table C-2 lists the basic math functions available in GLSL ES.

Table C-2 Built-in math functions

Function	Return value

`genType pow(`
`  genType x,`
`  genType n`
`)`
	The value of x raised to the nth power.
	Example:
	`vec2 result = pow(vec2(2.0, 3.0), vec2(4.0, 5.0));`
	`// result == vec2(16.0, 243.0)`

`genType exp(`
`  genType n`
`)`
	The constant *e* (2.718...) raised to the nth power.
	Example:
	`vec2 result = exp(vec2(2.0, 3.0));`
	`// result == vec2(7.39, 20.09)`

`genType exp2(`
`  genType n`
`)`
	2 raised to the nth power.
	Example:
	`vec2 result = exp2(vec2(2.0, 3.0));`
	`// result == vec2(4.0, 8.0)`

`genType log(`
`  genType x`
`)`
	The natural logarithm of x.
	Example:
	`vec2 result = log(vec2(4.0, 8.0));`
	`// result == vec2(1.39, 2.08)`

`genType log2(`
`  genType x`
`)`
	The base-2 logarithm of x.
	Example:
	`vec2 result = log2(vec2(4.0, 8.0));`
	`// result == vec2(2.0, 3.0)`

`genType sqrt(`
`  genType x`
`)`
	The square root of x.
	Example:
	`vec2 result = sqrt(vec2(9.0, 25.0));`
	`// result = vec2(3.0, 5.0)`

`genType inversesqrt(`
`  genType x`
`)`
	The inverse square root of x, that is, `1 / sqrt(x)`.
	Example:
	`vec2 result = inversesqrt(vec2(9.0, 25.0));`
	`// result = vec2(0.33..., 0.04)`

`genType abs(`
`  genType x`
`)`
	The absolute value of x.
	Example:
	`vec2 result = abs(vec2(-4.0, 8.0));`
	`// result = vec2(4.0, 8.0)`

Function	Return value

`genType sign(`
`  genType x`
`)`
	The sign of `x`.
	Example:
	`vec2 result = abs(vec2(-4.0, 8.0));`
	`// result == vec2(-1.0, 1.0)`

`genType floor(`
`  genType x`
`)`
	The nearest natural number that is smaller than or equal to `x`.
	Example:
	`vec2 result = floor(vec2(2.7, 14.0));`
	`// result == vec2(2.0, 14.0)`

`genType ceil(`
`  genType x`
`)`
	The nearest natural number that is larger than or equal to `x`.
	Example:
	`vec2 result = ceil(vec2(2.7, 14.0));`
	`// result == vec2(3.0, 14.0)`

`genType fract(`
`  genType x`
`)`
	The fraction part of `x`.
	Example:
	`vec2 result = fract(vec2(2.7, 14.0));`
	`// result == vec2(0.7, 0.0)`

`genType mod(`
`  genType x,`
`  genType y`
`)`
`genType mod(`
`  genType x,`
`  float y`
`)`
	The remainder of the division `x / y`.
	Example:
	`vec2 result = mod(vec2(5.0, 19.5), vec2(2.0, 5.0));`
	`// result == vec2(1.0, 4.5)`
	`vec2 result = mod(vec2(5.0, 19.5), 2.0);`
	`// result == vec2(1.0, 1.5)`

`genType min(`
`  genType x,`
`  genType y`
`)`
`genType min(`
`  genType x,`
`  float y`
`)`
	The smaller value of `x` and `y`.
	Example:
	`vec2 result = min(vec2(4.0, 15.5), vec2(2.5, 20.0));`
	`// result == vec2(2.5, 15.0)`
	`vec2 result = min(vec2(4.0, 15.5), 10.7);`
	`// result == vec2(4.0, 10.7)`

continued

Table C-2 continued

Name	Description

`genType max(`
`  genType x,`
`  genType y`
`)`
	The larger value of `x` and `y`.
	Example:
	`vec2 result = max(vec2(4.0, 15.5), vec2(2.5, 20.0));`
	`// result == vec2(4.0, 20.0)`

`genType max(`
`  genType x,`
`  float y`
`)`
	`vec2 result = max(vec2(4.0, 15.5), 10.7);`
	`// result == vec2(10.7, 15.5)`

Trigonometric functions

Along with the basic math functions, GLSL ES also provides the trigonometry functions listed in Table C-3.

Table C-3 Built-in trigonometric functions

Function	Return value

`genType radians(`
`  genType degrees`
`)`
	`Degrees` converted to radians.
	`radians(degrees) == degrees * π / 180`

`genType degrees(`
`  genType radians`
`)`
	`Radians` converted to degrees.
	`degrees(radians) == radians * 180 / π`

`genType sin(`
`  genType angle`
`)`
	The sine of `x`, where `x` is given in radians.

`genType cos(`
`  genType angle`
`)`
	The cosine of `x`, where `x` is given in radians.

`genType asin(`
`  genType x`
`)`
	The arc sine of `x`. Returns a value in the range [-π/2, π/2].

`genType acos(`
`  genType x`
`)`
	The arc cosine of `x`, where `x` is given in radians. Returns a value in the range [0, π].

Function	Return value

`genType atan(`
`  genType r`
`)`
	The arc tangent of `r`. Returns a value in the range [-π, π].

`genType atan(`
`  genType y,`
`  genType x`
`)`
	The arc tangent of `y/x`. Returns a value in the range [-π/2, π/2].

Vector and matrix functions

Table C-4 lists functions that make working with vectors and matrices easier.

Table C-4 Built-in vector and matrix functions

Function	Return value

`float length(`
`  genType v`
`)`
	The length of the vector `v`.
	`length(v) == sqrt(v.x*v.x + v.y*v.y + ...)`

`float distance(`
`  genType p0,`
`  genType p1`
`)`
	The distance between the points `p0` and `p1`.
	`distance(p0, p1) == length(p1 - p0)`

`float dot(`
`  genType v0,`
`  genType v1`
`)`
	The dot product of the vectors `v0` and `v1`.
	`dot(v0, v1) == v0.x*v1.x + v0.y+v1.y + ...`

`vec3 cross(`
`  vec3 v0,`
`  vec3 v1`
`)`
	The cross-product of the vectors `v0` and `v1`.
	`cross(v0, v1) == vec3(`
	`  v0.y * v1.z – v1.y * v0.z,`
	`  v0.z * v1.x - v1.z * v0.x,`
	`  v0.x * v1.y – v1.x * v0.y`
	`)`

continued

Table C-4 continued

Function	Return value

`genType normalize(`
`  genType v`
`)`
	The vector `v` normalized to unit length.
	`normalize(v) == v / length(v)`

`genType faceforward(`
`  genType N,`
`  genType I,`
`  genType Nref`
`)`
	Flips the vector `N` if the surface normal `Nref` points away from the incidence vector `I`.
	`if (dot(Nref, I) < 0)`
	`  return N`
	`else`
	`  return -N`

`genType reflect(`
`  genType I,`
`  genType N`
`)`
	The incidence vector `I` reflected on the surface with the normal `N`.
	`reflect(I, N) == I - 2 * dot(N, I) * N`

`genType refract(`
`  genType I,`
`  genType N,`
`  float eta`
`)`
	The refraction vector for the incidence vector `I` and the surface normal `N` given the refraction index `eta`.
	The refraction vector is calculated as
	`k = 1.0 - eta * eta * (1.0 - dot(N, I) * dot(N, I))`
	`if (k < 0.0)`
	`  return genType(0.0)`
	`else`
	`  return eta * I - (eta * dot(N, I) + sqrt(k)) * N`

`mat matrixCompMult(`
`  mat m0,`
`  mat m1`
`)`
	Component-wise multiplication of matrices `m0` and `m1`, where `mat` is `mat2`, `mat3`, or `mat4`.

Vector relations

The standard relational operators, such as <, >, ==, and !=, can compare vectors only by looking at all components at once. Table C-5 lists a set of functions that perform various component-wise comparisons and evaluations of vectors.

Table C-5 Built-in vector relation functions

Function	Return value

`bvec lessThan(`
`  vec v0,`
`  vec v1`
`)`
	Component-wise evaluation of `v0 < v1`.
	`lessThan(vec2(1.0, 2.0), vec2(4.0, 2.0)) ==`
	`  bvec2(true, false)`

`bvec lessThanEqual(`
`  vec v0,`
`  vec v1`
`)`
	Component-wise evaluation of `v0 <= v1`.
	`lessThanEqual(vec2(1.0, 2.0), vec2(4.0, 2.0)) ==`
	`  bvec2(true, true)`

`bvec greaterThan(`
`  vec v0,`
`  vec v1`
`)`
	Component-wise evaluation of `v0 > v1`.
	`greaterThan(vec2(1.0, 2.0), vec2(4.0, 2.0)) ==`
	`  bvec2(false, false)`

`bvec greaterThanEqual(`
`  vec v0,`
`  vec v1`
`)`
	Component-wise evaluation of `v0 >= v1`.
	`greaterThanEqual(vec2(1.0, 2.0), vec2(4.0, 2.0)) ==`
	`  bvec2(false, true)`

`bvec equal(`
`  vec v0,`
`  vec v1`
`)`
`bvec equal(`
`  bvec v0,`
`  bvec v1`
`)`
	Component-wise evaluation of `v0 == v1`.
	`equal(vec2(1.0, 2.0), vec2(4.0, 2.0)) ==`
	`  bvec2(false, true)`
	`equal(bvec2(true, false), bvec2(false, false)) ==`
	`  bvec2(false, true)`

`bvec notEqual(`
`  vec v0,`
`  vec v1`
`)`
`bvec notEqual(`
`  bvec v0,`
`  bvec v1`
`)`
	Component-wise evaluation of `v0 != v1`.
	`notEqual(vec2(1.0, 2.0), vec2(4.0, 2.0)) ==`
	`  bvec2(true, false)`
	`notEqual(bvec2(true, false), bvec2(false, false))`
	`  == bvec2(true, false)`

continued

Table C-5 continued

Function	Return value

`bool any(`
`  bvec v`
`)`
	True if any component of vector `v` is true.
	`any(bvec3(false, false, false)) == true`
	`any(bvec3(false, true, false)) == true`

`bool all(`
`  bvec v`
`)`
	True if all components of vector `v` are true.
	`all(bvec3(false, true, false)) == false`
	`all(bvec3(true, true, true)) == true`

`bvec not(`
`  bvec v`
`)`
	Component-wise evaluation of `!v`.
	`not(bvec2(true, false)) ==`
	`  bvec2(false, true)`

Helper functions

Table C-6 lists various functions that are useful for tasks such as clamping and blending.

Table C-6 Built-in helper functions

Function	Return value

`genType clamp(`
`  genType x,`
`  genType minVal,`
`  genType maxVal`
`)`
`genType clamp(`
`  genType x,`
`  float minVal,`
`  float maxVal`
`)`
	`x` clamped to the range [`minVal`,`maxVal`], that is, `max(minVal, min(maxVal, x))`.
	`clamp(`
	`  vec2(0.5, 1.7),`
	`  vec2(1.0, 1.2),`
	`  vec2(1.3, 1.5)`
	`) == vec2(1.0, 1.5)`
	`clamp(vec2(0.5, 1.7), 0.7, 1.3) == vec2(0.7, 1.3)`

Function	Return value

`genType mix(`
`  genType x,`
`  genType y,`
`  genType a`
`)`
`genType mix(`
`  genType x,`
`  genType y,`
`  float a`
`)`
	Linear interpolation between `x` and `y`.
	`mix(x, y, a) == x * (1.0 - a) + y * a`

`genType step(`
`  genType edge,`
`  genType x`
`)`
`genType step(`
`  float edge,`
`  genType x`
`)`
	0.0 for those components of `x` that are less than `edge`; 1.0 otherwise.

`genType smoothstep(`
`  genType edge0,`
`  genType edge1,`
`  genType x`
`)`
`genType smoothstep(`
`  float edge0,`
`  float edge1,`
`  genType x`
`)`
	0.0 for those components of `x` that are less than or equal to `edge0` and 1.0 for those that are greater than or equal to `edge1`. Returns a cubic Hermite interpolation when `x` lies between `edge0` and `edge1`:
	`t = clamp((x - edge0) / (edge1 - edge0), 0.0, 1.0)`
	`return t * t * (3 - 2 * t)`

Built-in variables and constants

Aside from the built-in functions, GLSL ES exposes a number of constants and variables. Table C-7 lists the constants that can be read from vertex and fragment shaders. The values of these constants depend on the OpenGL ES implementation.

Table C-7 Constants

Constant	Minimum value	Description
`int gl_MaxVertexAttribs`	8	Maximum number of vertex attributes.
`int gl_MaxVertexUniformVectors`	128	Maximum number of uniform vertex vectors.
`int gl_MaxFragmentUniformVectors`	16	Maximum number of uniform fragment vectors.
`int gl_MaxVaryingVectors`	8	Maximum number of varying vectors.
`int gl_MaxVertexTextureImageUnits`	0	Maximum number of texture image units in vertex shader.
`int gl_MaxCombinedTextureImageUnits`	8	Maximum number of texture image units in vertex shader and fragment shader combined.
`int gl_MaxTextureImageUnits`	8	Maximum number of texture units.
`int gl_MaxDrawBuffers`	1	Maximum number of output colors in the `gl_FragData` array in a fragment shader.

Table C-8 lists variables that are specific to vertex shaders and therefore cannot be used in fragment shaders.

Table C-8 Vertex shader variables

Variable	Description
`vec4 gl_Position`	The transformed and projected position of the current vertex as calculated by the vertex shader.
`float gl_PointSize`	This variable is used when drawing geometry of the type `gl.POINTS` and specifies the size of the point sprite. The default value is 1.0.

Table C-9 lists variables specific to fragment shaders. These variables cannot be used in vertex shaders.

Table C-9 Fragment shader variables

Variable	Description
`vec4 gl_FragColor`	The color value assigned to the current fragment.
`vec4 gl_FragCoord`	The coordinates of the current fragment relative to the window.
`vec4 gl_FragData[gl_MaxDrawBuffers]`	Array with the fragment data of the current fragment. Fragment shaders should not assign values to both `gl_FragCoord` and `gl_FragData`.
`vec2 gl_PointCoord`	When drawing point sprites, `gl_PointCoord` holds the two-dimensional coordinates of the fragment relative to the point. The coordinates range from 0.0 to 1.0 across the point.
`bool gl_FrontFacing`	Boolean value indicating whether the fragment is part of a front-facing primitive.