OpenGL ES Shading Language 1.0 Quick Reference Card - Page 4

Built-In Inputs, Outputs, and Constants [7]

Shader programs use Special Variables to communicate with fixed-function parts of the pipeline. Output Special Variables may be read back after writing. Input Special Variables are read-only. All Special Variables have global scope.

Vertex Shader Special Variables [7.1]

Outputs:

Variable	Description	Units or coordinate system
highp vec4 gl_Position;	transformed vertex position	clip coordinates
mediump float gl_PointSize;	transformed point size (point rasterization only)	pixels

Fragment Shader Special Variables [7.2]

Fragment shaders may write to **gl_FragColor** or to one or more elements of **gl_FragData[]**, but not both.
The size of the **gl_FragData** array is given by the built-in constant **gl_MaxDrawBuffers**.

Inputs:

Variable	Description	Units or coordinate system
mediump vec4 gl_FragCoord;	fragment position within frame buffer	window coordinates
bool gl_FrontFacing;	fragment belongs to a front-facing primitive	Boolean
mediump vec2 gl_PointCoord;	fragment position within a point (point rasterization only)	0.0 to 1.0 for each component

Outputs:

Variable	Description	Units or coordinate system
mediump vec4 gl_FragColor;	fragment color	RGBA color
mediump vec4 gl_FragData[*n*]	fragment color for color attachment *n*	RGBA color

Built-In Constants With Minimum Values [7.4]

Built-in Constant	Minimum value
const mediump int gl_MaxVertexAttribs	8
const mediump int gl_MaxVertexUniformVectors	128
const mediump int gl_MaxVaryingVectors	8
const mediump int gl_MaxVertexTextureImageUnits	0
const mediump int gl_MaxCombinedTextureImageUnits	8
const mediump int gl_MaxTextureImageUnits	8
const mediump int gl_MaxFragmentUniformVectors	16
const mediump int gl_MaxDrawBuffers	1

Built-In Uniform State [7.5]

Specifies depth range in window coordinates. If an implementation does not support highp precision in the fragment language, and state is listed as highp, then that state will only be available as mediump in the fragment language.

```
struct gl_DepthRangeParameters {
    highp float near;    // n
    highp float far;     // f
    highp float diff;    // f - n
};

uniform gl_DepthRangeParameters  gl_DepthRange;
```

Built-In Functions

Angle & Trigonometry Functions [8.1]

Component-wise operation. Parameters specified as *angle* are assumed to be in units of radians. T is float, vec2, vec3, vec4.

T **radians**(T *degrees*)	degrees to radians
T **degrees**(T *radians*)	radians to degrees
T **sin**(T *angle*)	sine
T **cos**(T *angle*)	cosine
T **tan**(T *angle*)	tangent
T **asin**(T *x*)	arc sine
T **acos**(T *x*)	arc cosine
T **atan**(T *y*, T *x*) T **atan**(T *y_over_x*)	arc tangent

Exponential Functions [8.2]

Component-wise operation. T is float, vec2, vec3, vec4.

T **pow**(T *x*, T *y*)	x^y
T **exp**(T *x*)	e^x
T **log**(T *x*)	ln
T **exp2**(T *x*)	2^x
T **log2**(T *x*)	$\log_2$
T **sqrt**(T *x*)	square root
T **inversesqrt**(T *x*)	inverse square root

Common Functions [8.3]

Component-wise operation. T is float, vec2, vec3, vec4.

T **abs**(T *x*)	absolute value
T **sign**(T *x*)	returns -1.0, 0.0, or 1.0
T **floor**(T *x*)	nearest integer <= *x*
T **ceil**(T *x*)	nearest integer >= *x*
T **fract**(T *x*)	*x* - **floor**(*x*)
T **mod**(T *x*, T *y*) T **mod**(T *x*, float *y*)	modulus
T **min**(T *x*, T *y*) T **min**(T *x*, float *y*)	minimum value
T **max**(T *x*, T *y*) T **max**(T *x*, float *y*)	maximum value
T **clamp**(T *x*, T *minVal*, T *maxVal*) T **clamp**(T *x*, float *minVal*, float *maxVal*)	**min**(**max**(*x*, **minVal**), **maxVal**)
T **mix**(T *x*, T *y*, T *a*) T **mix**(T *x*, T *y*, float *a*)	linear blend of *x* and *y*
T **step**(T *edge*, T *x*) T **step**(float *edge*, T *x*)	0.0 if *x* < *edge*, else 1.0
T **smoothstep**(T *edge0*, T *edge1*, T *x*) T **smoothstep**(float *edge0*, float *edge1*, T *x*)	clip and smooth

Geometric Functions [8.4]

These functions operate on vectors as vectors, not component-wise. T is float, vec2, vec3, vec4.

float **length**(T *x*)	length of vector
float **distance**(T *p0*, T *p1*)	distance between points
float **dot**(T *x*, T *y*)	dot product
vec3 **cross**(vec3 *x*, vec3 *y*)	cross product
T **normalize**(T *x*)	normalize vector to length 1
T **faceforward**(T *N*, T *I*, T *Nref*)	returns *N* if **dot**(*Nref*, *I*) < 0, else -*N*
T **reflect**(T *I*, T *N*)	reflection direction *I* - 2 * **dot**(*N*,*I*) * *N*
T **refract**(T *I*, T *N*, float *eta*)	refraction vector

Matrix Functions [8.5]

Type mat is any matrix type.

mat **matrixCompMult**(mat *x*, mat *y*)	multiply *x* by *y* component-wise

Vector Relational Functions [8.6]

Compare *x* and *y* component-wise. Sizes of input and return vectors for a particular call must match. Type bvec is bvec*n*; vec is vec*n*; ivec is ivec*n* (where *n* is 2, 3, or 4). T is the union of vec and ivec.

bvec **lessThan**(T *x*, T *y*)	x < y
bvec **lessThanEqual**(T *x*, T *y*)	x <= y
bvec **greaterThan**(T *x*, T *y*)	x > y
bvec **greaterThanEqual**(T *x*, T *y*)	x >= y
bvec **equal**(T *x*, T *y*) bvec **equal**(bvec *x*, bvec *y*)	x == y
bvec **notEqual**(T *x*, T *y*) bvec **notEqual**(bvec *x*, bvec *y*)	x!= y
bool **any**(bvec *x*)	true if any component of *x* is true
bool **all**(bvec *x*)	true if all components of *x* are true
bvec **not**(bvec *x*)	logical complement of *x*

Texture Lookup Functions [8.7]

Available only in vertex shaders.

vec4 **texture2DLod**(sampler2D *sampler*, vec2 *coord*, float *lod*)
vec4 **texture2DProjLod**(sampler2D *sampler*, vec3 *coord*, float *lod*)
vec4 **texture2DProjLod**(sampler2D *sampler*, vec4 *coord*, float *lod*)
vec4 **textureCubeLod**(samplerCube *sampler*, vec3 *coord*, float *lod*)

Available only in fragment shaders.

vec4 **texture2D**(sampler2D *sampler*, vec2 *coord*, float *bias*)
vec4 **texture2DProj**(sampler2D *sampler*, vec3 *coord*, float *bias*)
vec4 **texture2DProj**(sampler2D *sampler*, vec4 *coord*, float *bias*)
vec4 **textureCube**(samplerCube *sampler*, vec3 *coord*, float *bias*)

Available in vertex and fragment shaders.

vec4 **texture2D**(sampler2D *sampler*, vec2 *coord*)
vec4 **texture2DProj**(sampler2D *sampler*, vec3 *coord*)
vec4 **texture2DProj**(sampler2D *sampler*, vec4 *coord*)
vec4 **textureCube**(samplerCube *sampler*, vec3 *coord*)

Statements and Structure

Iteration and Jumps [6]

Function Call	call by value-return

Iteration	for (;;) { break, continue }
	while ( ) { break, continue }
	do { break, continue } while ( );

Selection	if ( ) { } if ( ) { } else { }
Jump	break, continue, return discard // Fragment shader only
Entry	void main()

Sample Program

A shader pair that applies diffuse and ambient lighting to a textured object.

Vertex Shader

```
uniform   mat4   mvp_matrix;     // model-view-projection matrix
uniform   mat3   normal_matrix;  // normal matrix
uniform   vec3   ec_light_dir;   // light direction in eye coords

attribute vec4   a_vertex;       // vertex position
attribute vec3   a_normal;       // vertex normal
attribute vec2   a_texcoord;     // texture coordinates

varying   float  v_diffuse;
varying   vec2   v_texcoord;

void main(void)
{
  // put vertex normal into eye coords
  vec3  ec_normal  = normalize(normal_matrix * a_normal);

  // emit diffuse scale factor, texcoord, and position
  v_diffuse        = max(dot(ec_light_dir, ec_normal), 0.0);
  v_texcoord       = a_texcoord;
  gl_Position      = mvp_matrix * a_vertex;
}
```

Fragment Shader

```
precision  mediump  float;

uniform    sampler2D  t_reflectance;
uniform    vec4       i_ambient;

varying    float      v_diffuse;
varying    vec2       v_texcoord;

void main (void)
{
  vec4   color  = texture2D(t_reflectance, v_texcoord);
  gl_FragColor  = color * (vec4(v_diffuse) + i_ambient);
}
```