Aggregate Operations and Constructors

Matrix Constructor Examples [5.4.2]

```
mat2(float)                    // init diagonal
mat2(vec2, vec2);              // column-major order
mat2(float, float,
     float, float);            // column-major order
```

Structure Constructor Example [5.4.3]

```
struct light {
    float intensity;
    vec3 pos;
};
light lightVar = light(3.0, vec3(1.0, 2.0, 3.0));
```

Matrix Components [5.6]

Access components of a matrix with array subscripting syntax. For example:

```
mat4 m;          // m represents a matrix
m[1] = vec4(2.0); // sets second column to all 2.0
m[0][0] = 1.0;   // sets upper left element to 1.0
m[2][3] = 2.0;   // sets 4th element of 3rd column to 2.0
```

Examples of operations on matrices and vectors:

```
m = f * m;       // scalar * matrix component-wise
v = f * v;       // scalar * vector component-wise
v = v * v;       // vector * vector component-wise
m = m +/- m;     // matrix component-wise +/-
m = m * m;       // linear algebraic multiply
m = v * m;       // row vector * matrix linear algebraic multiply
m = m * v;       // matrix * column vector linear algebraic multiply
f = dot(v, v);   // vector dot product
v = cross(v, v); // vector cross product
m = matrixCompMult(m, m);   // component-wise multiply
```

(more examples ↗)

Structure Operations [5.7]

Select structure fields using the period (.) operator. Valid operators are:

.	field selector
== !=	equality
=	assignment

Array Operations [5.7]

Array elements are accessed using the array subscript operator "[ ]". For example:

```
diffuseColor += lightIntensity[3] * NdotL;
```

The size of an array can be determined using the **.length()** operator. For example:

```
for (i = 0; i < a.length(); i++)
    a[i] = 0.0;
```

Statements and Structure

Iteration and Jumps [6]

Entry	void main()

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

Jump	break, continue, return discard // Fragment shader only

Selection	if ( ) { }
	if ( ) { } else { }
	switch ( ) { break, case }

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]

Inputs:

```
int     gl_VertexID;     // integer index
int     gl_InstanceID;   // instance number
```

Outputs:

```
out gl_PerVertex {
    vec4    gl_Position;     // transformed vertex position in clip coordinates
    float   gl_PointSize;    // transformed point size in pixels (point rasterization only)
};
```

Fragment Shader Special Variables [7.2]

Inputs:

```
highp vec4    gl_FragCoord;    // fragment position within frame buffer
bool          gl_FrontFacing;  // fragment belongs to a front-facing primitive
mediump vec2  gl_PointCoord;   // 0.0 to 1.0 for each component
```

Outputs:

```
highp float   gl_FragDepth;    // depth range
```

Built-In Constants With Minimum Values [7.3]

Built-in Constant	Minimum value
const mediump int gl_MaxVertexAttribs	16
const mediump int gl_MaxVertexUniformVectors	256
const mediump int gl_MaxVertexOutputVectors	16
const mediump int gl_MaxFragmentInputVectors	15
const mediump int gl_MaxVertexTextureImageUnits	16
const mediump int gl_MaxCombinedTextureImageUnits	32
const mediump int gl_MaxTextureImageUnits	16
const mediump int gl_MaxFragmentUniformVectors	224
const mediump int gl_MaxDrawBuffers	4
const mediump int gl_MinProgramTexelOffset	-8
const mediump int gl_MaxProgramTexelOffset	7

Built-In Uniform State [7.4]

As an aid to accessing OpenGL ES processing state, the following uniform variables are built into the OpenGL ES Shading Language.

```
struct gl_DepthRangeParameters {
    float near;    // n
    float far;     // f
    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
T **sinh** (T *x*);	hyperbolic sine
T **cosh** (T *x*);	hyperbolic cosine
T **tanh** (T *x*);	hyperbolic tangent
T **asinh** (T *x*);	arc hyperbolic sine; inverse of sinh
T **acosh** (T *x*);	arc hyperbolic cosine; non-negative inverse of cosh
T **atanh** (T *x*);	arc hyperbolic tangent; inverse of tanh

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 and vec*n*, TI is int and ivec*n*, TU is uint and uvec*n*, and TB is bool and bvec*n*, where *n* is 2, 3, or 4.

T **abs**(T *x*); TI **abs**(TI *x*);	absolute value
T **sign**(T *x*); TI **sign**(TI *x*);	returns -1.0, 0.0, or 1.0
T **floor**(T *x*);	nearest integer <= *x*
T **trunc** (T *x*);	nearest integer a such that |*a*| <= |*x*|
T **round** (T *x*);	round to nearest integer
T **roundEven** (T *x*);	round to nearest integer
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*); T **modf**(T *x*, out T *i*);	modulus
T **min**(T *x*, T *y*); TI **min**(TI *x*, TI *y*); TU **min**(TU *x*, TU *y*); T **min**(T *x*, float *y*); TI **min**(TI *x*, int *y*); TU **min**(TU *x*, uint *y*);	minimum value
T **max**(T *x*, T *y*); TI **max**(TI *x*, TI *y*); TU **max**(TU *x*, TU *y*); T **max**(T *x*, float *y*); TI **max**(TI *x*, int *y*); TU **max**(TU *x*, uint *y*);	maximum value
T **clamp**(T *x*, T *minVal*, T *maxVal*); TI **clamp**(V *x*, TI *minVal*, TI *maxVal*); TU **clamp**(TU *x*, TU *minVal*, TU *maxVal*); T **clamp**(T *x*, float *minVal*, float *maxVal*); TI **clamp**(TI *x*, int *minVal*, int *maxVal*); TU **clamp**(TU *x*, uint *minVal*, uint *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 **mix**(T *x*, T *y*, TB *a*);	Selects vector source for each returned component
T **step**(T *edge*, T *x*); T **step**(float *edge*, T *x*);	0.0 if *x* < *edge*, else 1.0

(more Common Functions ↗)