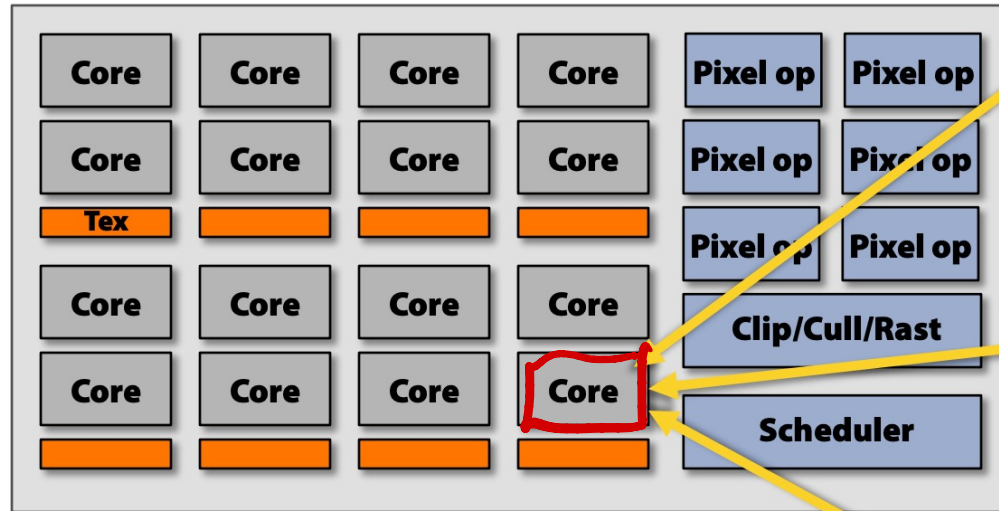
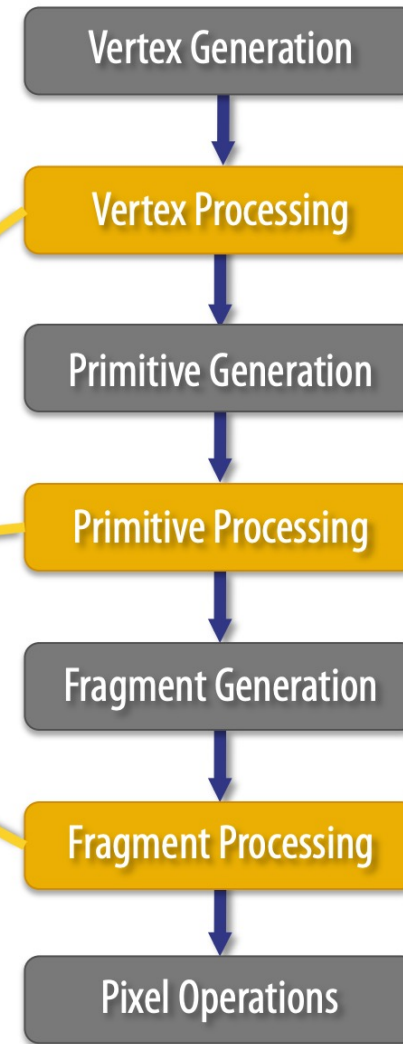


18 –g|sl

Recall: GPU architecture



NVIDIA GeForce 8800
("unified shading" GPU)



vertex processing & fragment processing

Aside: WebGL vs WebGL2 vs WEbGPU vs ...
Slightly different GLSL versions ...

- WebGL ~ = OpenGL ES2, WebGL2 ~ = OpenGL ES3

- <https://webgl2fundamentals.org/webgl/lessons/webgl1-to-webgl2.html>

- <https://www.khronos.org/webgl/>

- https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf

- <https://www.khronos.org/files/webgl20-reference-guide.pdf>

- <https://www.w3.org/community/gpu/> *web GPU*

GLSL (GL Shader Language)

WebGL 'cheat sheet'

- Vertex shader + fragment shader
- C-like language (akin to C++, C#, Java, etc)

Types [4.1]

A shader can aggregate these using arrays and structures to build more complex types. There are no pointer types.

Basic Types

<code>void</code>	no function return value or empty parameter list
<code>bool</code>	Boolean
<code>int</code>	signed integer
<code>float</code>	floating scalar
<code>vec2, vec3, vec4</code>	n-component floating point vector
<code>bvec2, bvec3, bvec4</code>	Boolean vector
<code>ivec2, ivec3, ivec4</code>	signed integer vector
<code>mat2, mat3, mat4</code>	2x2, 3x3, 4x4 float matrix
<code>sampler2D</code>	access a 2D texture
<code>samplerCube</code>	access cube mapped texture

Structures and Arrays [4.1.8, 4.1.9]

Structures	<pre>struct type-name { members } struct-name[]; // optional variable declaration, // optionally an array</pre>
Arrays	<pre>float foo[3]; * structures and blocks can be arrays * only 1-dimensional arrays supported * structure members can be arrays</pre>

Operators and Functions

- Common operators

+ - * / *= +=

float x
vec3 v

v * x is valid

- Common and unique functions

needed for shading.

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)	arc tangent
T atan(T y_over_x)	

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 $\leq x$
T ceil(T x)	nearest integer $\geq x$
T fract(T x)	$x - \text{floor}(x)$
T mod(T x, T y)	modulus
T mod(T x, float y)	
T min(T x, T y)	minimum value
T min(T x, float y)	
T max(T x, T y)	maximum value
T max(T x, float y)	
T clamp(T x, T minVal, T maxVal)	
T clamp(T x, float minVal, float maxVal)	$\min(\max(x, \text{minVal}), \text{maxVal})$
T mix(T x, T y, T a)	linear blend of x and y
T mix(T x, T y, float a)	
T step(T edge, T x)	0.0 if $x < \text{edge}$, else 1.0
T step(float edge, T x)	
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 $\text{dot}(Nref, I) < 0$, else $-N$
T reflect(T I, T N)	reflection direction $I - 2 * \text{dot}(N, I) * N$
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)	$x \leq y$
bvec greaterThan(T x, T y)	$x > y$
bvec greaterThanEqual(T x, T y)	$x \geq y$
bvec equal(T x, T y)	$x == y$
bvec equal(bvec x, bvec y)	
bvec notEqual(T x, T y)	$x != y$
bvec notEqual(bvec x, bvec 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)

Swizzling

vec3 color
vec4 cl = vec4 [color, 1.0]

- Address components of vectors

vec3 color; float x = color.g

vec2 svalue = color.xy

vec4 → .rgba
 .kzyw

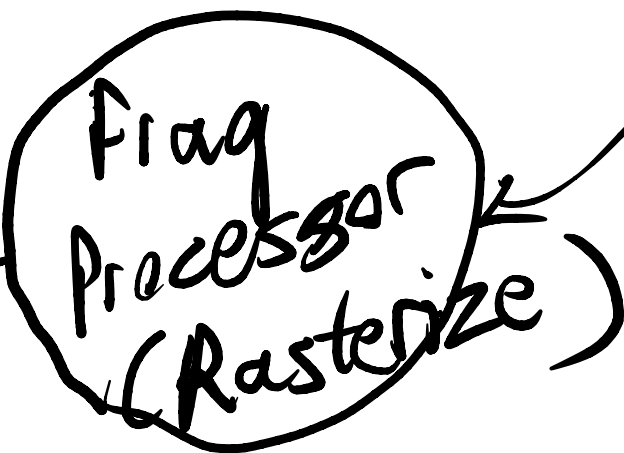
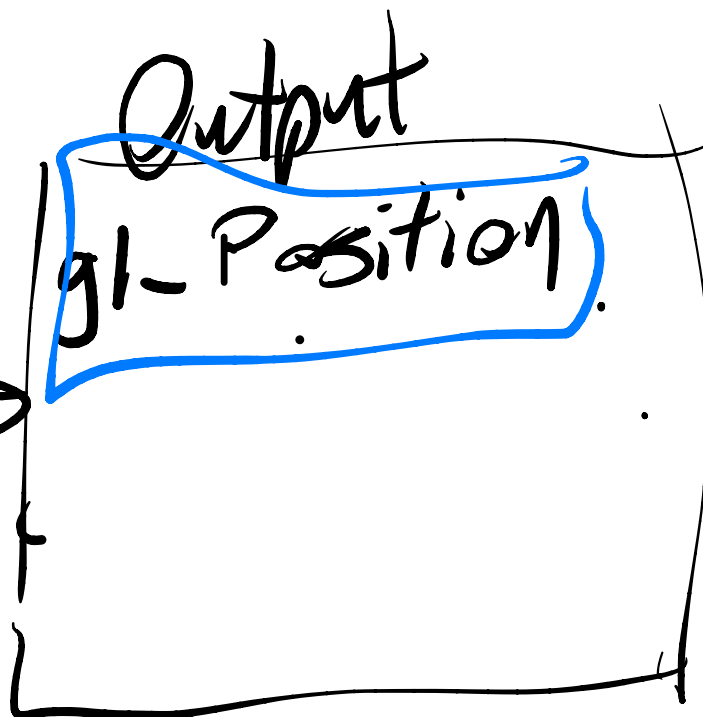
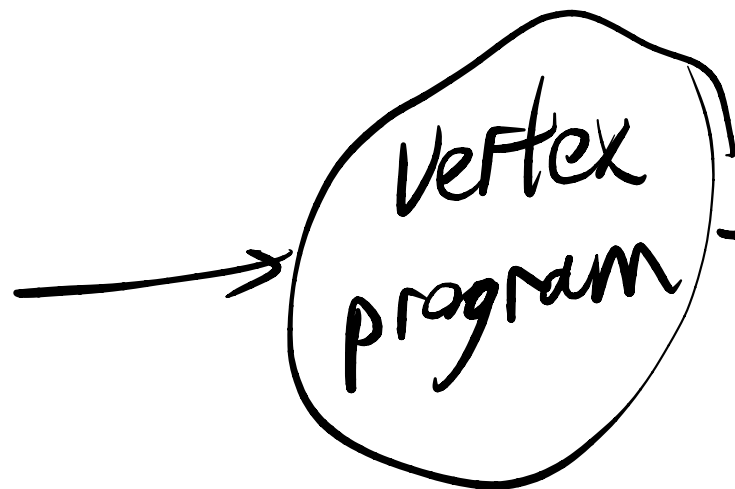
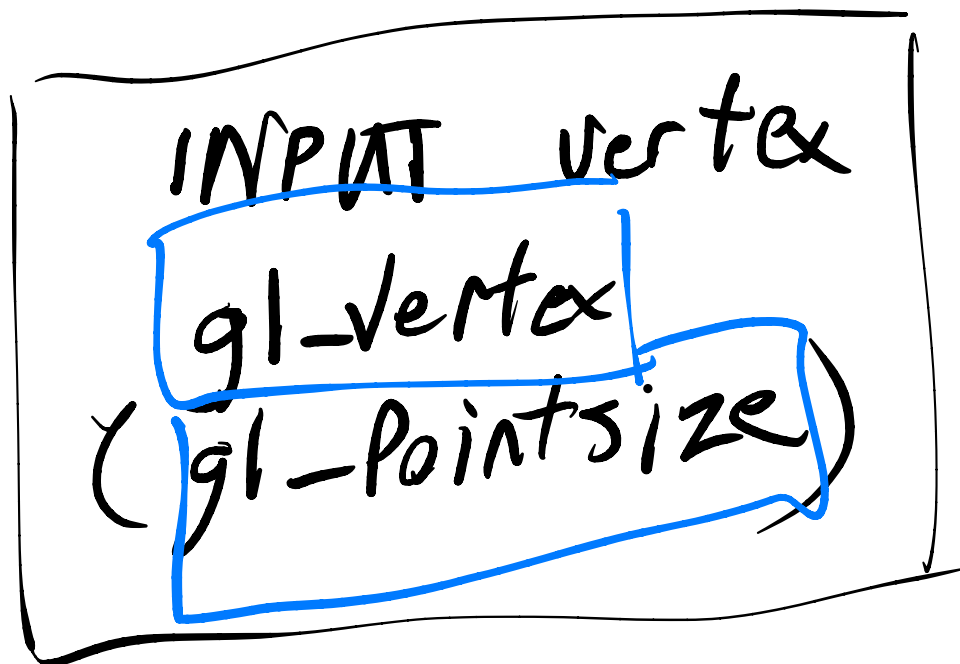
vec3 = color.rg

Syntax is very picky

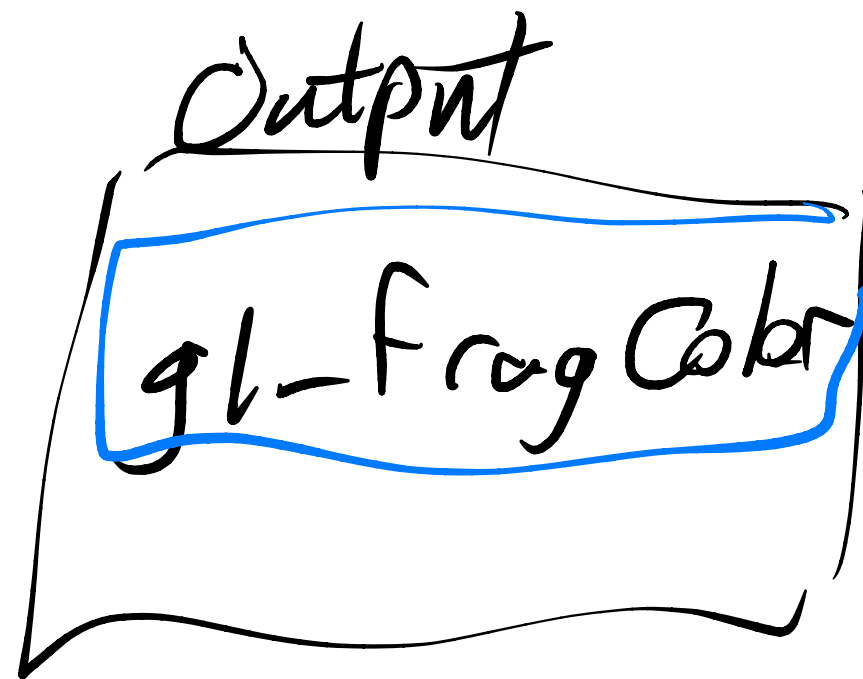
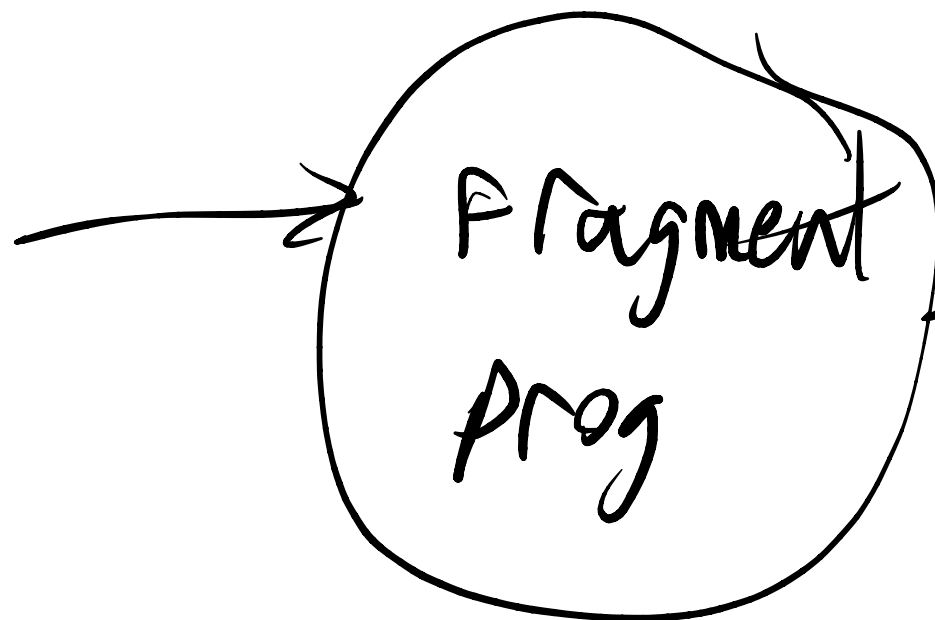
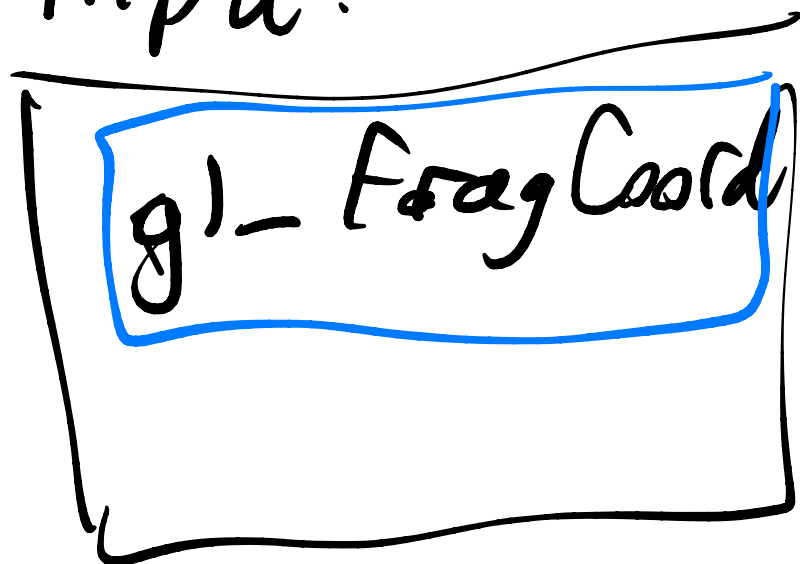
eg. ; after every statement

Info flow

CPU → vertex → fragment



Input.



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 <i>n</i>	RGBA color

Information Flow

Step through an Example

- <https://webglsfundamentals.org/webgl/lessons/resources/webgl-state-diagram.html#no-help>

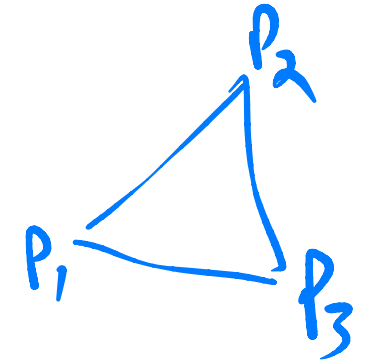
Simple Vertex Shader

```
attribute vec4 position;  
attribute vec3 normal;  
attribute vec2 texcoord;  
uniform mat4 projection;  
uniform mat4 modelView;  
varying vec3 v_normal;  
varying vec2 v_texcoord;  
void main() {  
    gl_Position = projection * modelView * position;  
    v_normal = mat3(modelView) * normal;  
    v_texcoord = texcoord;  
}
```

input defined per vertex

inputs per MVA
(same for each vertex)

outputs out of
in to frag



vertex } interpolated
by primitive
processor

Simple Fragment Shader

```
precision highp float;
varying vec3 v_normal;
varying vec2 v_texcoord;
uniform sampler2D diffuse; ←
uniform sampler2D decal; ←
uniform vec4 diffuseMult; ←
uniform vec3 lightDir; ←
void main() {
    vec3 normal = normalize(v_normal);
    float light = dot(normal, lightDir) * 0.5 + 0.5;
    vec4 color = texture2D(diffuse, v_texcoord) * diffuseMult;
    vec4 decalColor = texture2D(decal, v_texcoord);
    decalColor.rgb *= decalColor.a;
    color = color * (1.0 - decalColor.a) + decalColor;
    gl_FragColor = vec4(color.rgb * light, color.a);
}
```


A bit more ...

https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

<https://webglfundamentals.org/webgl/lessons/webgl-how-it-works.html>

<https://webglfundamentals.org/webgl/lessons/resources/webgl-state-diagram.html#no-help>

<https://threejs.org/docs/#api/en/materials/ShaderMaterial>