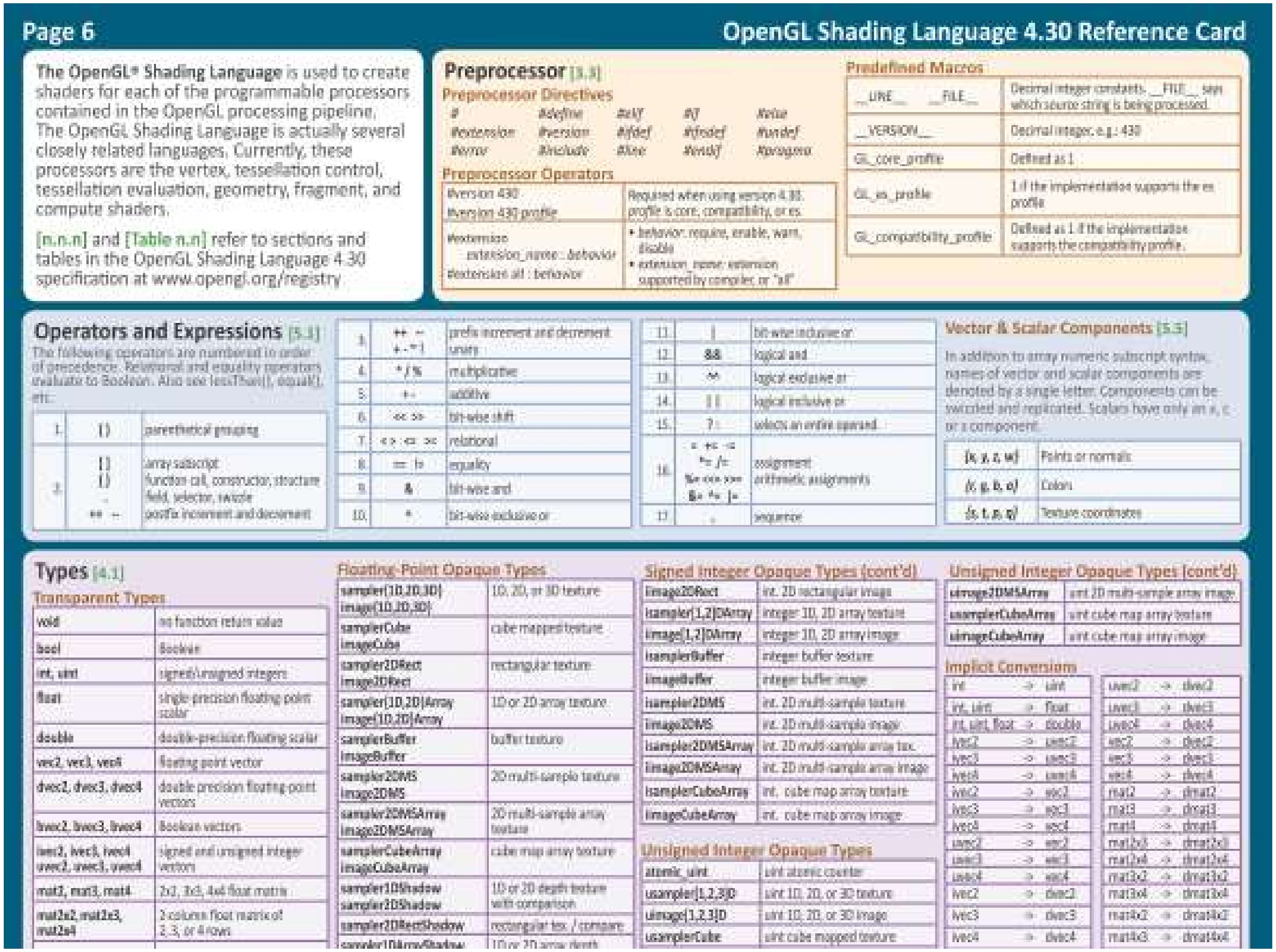
Shader

GLSL Syntax

Hochschule Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

OpenGL Reference Card Page 6ff



Page 6

OpenGL Shading Language 4.30 Reference Card

The OpenGL® Shading Language is used to create shaders for each of the programmable processors contained in the OpenGL processing pipeline. The OpenGL Shading Language is actually several closely related languages. Currently, these processors are the vertex, tessellation control, tessellation evaluation, geometry, fragment, and compute shaders.

[n.n.n] and [Table n.n] refer to sections and tables in the OpenGL Shading Language 4.30 specification at www.opengl.org/registry

Preprocessor [3.3]

Preprocessor Directives

#	#define	#elif	#if	#else
#extension	#version	#ifdef	#ifndef	#undef
#error	#include	#line	#endif	#pragma

Preprocessor Operators

#version 430 #version 430 profile	Required when using version 4.30. profile is core, compatibility, or es.
#extension extension_name : behavior #extension all : behavior	• behavior: require, enable, warn, disable • extension_name: extension supported by compiler, or "all"

Predefined Macros

__LINE__ __FILE__	Decimal integer constants. __FILE__ says which source string is being processed.
__VERSION__	Decimal integer, e.g.: 430
GL_core_profile	Defined as 1
GL_es_profile	1 if the implementation supports the es profile
GL_compatibility_profile	Defined as 1 if the implementation supports the compatibility profile.

Operators and Expressions [5.1]

The following operators are numbered in order of precedence. Relational and equality operators evaluate to Boolean. Also see lessThan(), equal(), etc.

1.	( )	parenthetical grouping
2.	[ ] ( ) . ++ --	array subscript function call, constructor, structure field, selector, swizzle postfix increment and decrement
3.	++ -- + - ~ !	prefix increment and decrement unary
4.	* / %	multiplicative
5.	+ -	additive
6.	<< >>	bit-wise shift
7.	< > <= >=	relational
8.	== !=	equality
9.	&	bit-wise and
10.	^	bit-wise exclusive or
11.	|	bit-wise inclusive or
12.	&&	logical and
13.	^^	logical exclusive or
14.	||	logical inclusive or
15.	? :	selects an entire operand
16.	= += -= *= /= %= <<= >>= &= ^= |=	assignment arithmetic assignments
17.	,	sequence

Vector & Scalar Components [5.5]

In addition to array numeric subscript syntax, names of vector and scalar components are denoted by a single letter. Components can be swizzled and replicated. Scalars have only an x, r, or s component.

{x, y, z, w}	Points or normals
{r, g, b, a}	Colors
{s, t, p, q}	Texture coordinates

Types [4.1]

Transparent Types

void	no function return value
bool	Boolean
int, uint	signed/unsigned integers
float	single-precision floating-point scalar
double	double-precision floating scalar
vec2, vec3, vec4	floating point vector
dvec2, dvec3, dvec4	double precision floating-point vectors
bvec2, bvec3, bvec4	Boolean vectors
ivec2, ivec3, ivec4 uvec2, uvec3, uvec4	signed and unsigned integer vectors
mat2, mat3, mat4	2x2, 3x3, 4x4 float matrix
mat2x2, mat2x3, mat2x4	2-column float matrix of 2, 3, or 4 rows

Floating-Point Opaque Types

sampler{1D,2D,3D}
image{1D,2D,3D}
	1D, 2D, or 3D texture

samplerCube imageCube	cube mapped texture
sampler2DRect image2DRect	rectangular texture
sampler{1D,2D}Array image{1D,2D}Array	1D or 2D array texture
samplerBuffer imageBuffer	buffer texture
sampler2DMS image2DMS	2D multi-sample texture
sampler2DMSArray image2DMSArray	2D multi-sample array texture
samplerCubeArray imageCubeArray	cube map array texture
sampler1DShadow sampler2DShadow	1D or 2D depth texture with comparison
sampler2DRectShadow	rectangular tex. / compare
sampler1DArrayShadow	1D or 2D array depth

Signed Integer Opaque Types (cont'd)

iimage2DRect	int. 2D rectangular image
isampler[1,2]DArray	integer 1D, 2D array texture
iimage[1,2]DArray	integer 1D, 2D array image
isamplerBuffer	integer buffer texture
iimageBuffer	integer buffer image
isampler2DMS	int. 2D multi-sample texture
iimage2DMS	int. 2D multi-sample image
isampler2DMSArray	int. 2D multi-sample array tex.
iimage2DMSArray	int. 2D multi-sample array image
isamplerCubeArray	int. cube map array texture
iimageCubeArray	int. cube map array image

Unsigned Integer Opaque Types

atomic_uint	uint atomic counter
usampler[1,2,3]D	uint 1D, 2D, or 3D texture
uimage[1,2,3]D	uint 1D, 2D, or 3D image
usamplerCube	uint cube mapped texture

Unsigned Integer Opaque Types (cont'd)

uimage2DMSArray	uint 2D multi-sample array image
usamplerCubeArray	uint cube map array texture
uimageCubeArray	uint cube map array image

Implicit Conversions

int	->	uint	uvec2	->	dvec2
int, uint	->	float	uvec3	->	dvec3
int, uint, float	->	double	uvec4	->	dvec4
ivec2	->	uvec2	vec2	->	dvec2
ivec3	->	uvec3	vec3	->	dvec3
ivec4	->	uvec4	vec4	->	dvec4
ivec2	->	vec2	mat2	->	dmat2
ivec3	->	vec3	mat3	->	dmat3
ivec4	->	vec4	mat4	->	dmat4
uvec2	->	vec2	mat2x3	->	dmat2x3
uvec3	->	vec3	mat2x4	->	dmat2x4
uvec4	->	vec4	mat3x2	->	dmat3x2
ivec2	->	dvec2	mat3x4	->	dmat3x4
ivec3	->	dvec3	mat4x2	->	dmat4x2
ivec4	->	dvec4	mat4x3	->	dmat4x3

see www.opengl.org/sdk/docs/reference_card/opengl44-quick-reference-card.pdf

GLSL Syntax Overview

- GLSL is like C without
  - Pointers
  - Recursion
  - Dynamic memory allocation
- GLSL is like C with
  - Built-in vector, matrix and sampler types
  - Constructors
  - A math library
  - Input and output qualifiers

GLSL Syntax Overview

- GLSL has a preprocessor

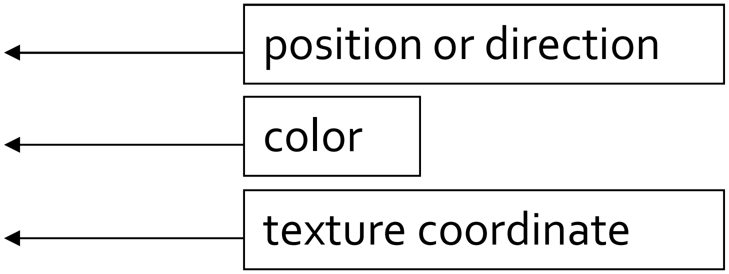
```
#version 330
#ifdef FAST_EXACT_METHOD
  FastExact();
#else
  SlowApproximate();
#endif
```

- All shaders have main()

```
void main() {
  ...
}
```

Vectors

- Scalar types: `float`, `int`, `uint` , and `bool`
- Vectors are also built-in types:
  - `vec2`, `vec3`, and `vec4`
  - Also `ivec*`, `uvec*`, and `bvec*`
- Access components three ways:
  - `.x, .y, .z, .w`
  - `.r, .g, .b, .a`
  - `.s, .t, .p, .q`



position or direction

color

texture coordinate

Vectors

- Vectors have constructors

```
vec3 xyz = vec3(1.0, 2.0, 3.0);

vec3 xyz = vec3(1.0); // [1.0, 1.0, 1.0]

~~vec3 xyz = (vec3)1.0;~~ // error

vec3 xyz = vec3(vec2(1.0, 2.0), 3.0);
```

Swizzling

- Swizzle: select or rearrange components

```
vec4 c = vec4(0.5, 1.0, 0.8, 1.0);

vec3 rgb = c.rgb;  // [0.5, 1.0, 0.8]
     rgb = c.xyz;  // same thing! [0.5, 1.0, 0.8]
vec3 bgr = c.bgr;  // [0.8, 1.0, 0.5]

vec3 rrr = c.rrr;  // [0.5, 0.5, 0.5]

c.a = 0.5;         // [0.5, 1.0, 0.8, 0.5]
c.rb = vec2(0.0);  // [0.0, 1.0, 0.0, 0.5]

float g = rgb[1];  // 0.5, indexing, not swizzling
```

Matrices

- Matrices are built-in types:
  - Square: **`mat2`**, **`mat3`**, and **`mat4`**
  - Rectangular: **`mat`*`m`*`x`*`n`***. ***m*** columns, ***n*** rows
    - **`mat2x3`**
- Stored column major

Matrices

- Matrix Constructors

```
mat3 i = mat3(1.0); // 3x3 identity matrix

mat2 m = mat2(1.0, 2.0,  // [1.0 3.0] column major!
              3.0, 4.0); // [2.0 4.0]
```

- Accessing Elements

```
float f = m[column][row]; // m some 3x3 matrix

float x = m[0].x; // x component of first column

vec2 yz = m[1].yz; //yz components of second column
```

Vectors and Matrices

- Matrix and vector operations are easy and fast:

```
vec3 xyz = // ...

vec3 v0 = 2.0 * xyz;        // scale
vec3 v1 =  v0 + xyz;        // component-wise
vec3 v2 =  v0 * xyz;        // component-wise

mat3 m  = mat3(v0, v1, v2); // give columns
mat3 m2 = mat3(2.0);        // diagonal all 2’s

mat3 m3   = 3.0 * m;        // scale a matrix
mat3 mm2  = m * m2;         // matrix * matrix
vec3 xyz2 = mm2 * xyz;      // matrix * vector
```

Built-in Functions

- Selected Trigonometry Functions

```
float s = sin(theta);
float c = cos(theta);
float t = tan(theta);

float as = asin(theta);

vec3 angles = vec3(/* ... */);
vec3 vs = sin(angles); //vector version
```

Built-in Functions

- Exponential Functions

```
float xToTheY = pow(x, y);
float eToTheX = exp(x);
float twoToTheX = exp2(x);

float l = log(x);   // ln
float l2 = log2(x); // log2

float s = sqrt(x);
float is = inversesqrt(x); // single GPU instr.
```

Built-in Functions

- Selected Common Functions

```
float ax = abs(x);  // absolute value
float sx = sign(x); // -1.0, 0.0, 1.0

float m0 = min(x, y); // minimum value
float m1 = max(x, y); // maximum value
float c  = clamp(x, 0.0, 1.0);

// many others:  floor(), ceil(),
// step(), smoothstep(), …
```

Built-in Functions

- Rewrite with one function call

```
float minimum = // ...
float maximum = // ...
float x = // ...

float f = min(max(x, minimum), maximum);

float f = clamp(x, minimum, maximum);
```

Built-in Functions

- Rewrite this without the **`if`** statement

```
float x = // ...
float f;

if (x > 0.0) {
  f = 2.0;
}
else {
  f = -2.0;
}

f = 2.0 * sign(x);
```

Built-in Functions

- Rewrite this without the `if` statement

```
float root1 = // ...
float root2 = // ...

if (root1 < root2) {
  return vec3(0.0, 0.0, root1);
}
else {
  return vec3(0.0, 0.0, root2);
}

return vec3(0.0, 0.0, min(root1, root2));
```

Built-in Functions

- Selected Geometric Functions

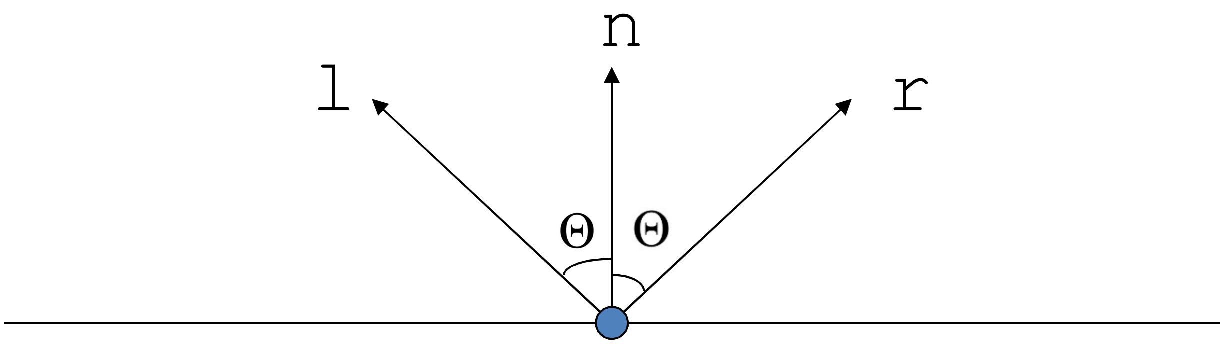
```
vec3 l = // ...
vec3 n = // ...
vec3 p = // ...
vec3 q = // ...

float  f = length(l);      // vector length
float  d = distance(p, q); // point dist.
float d2 = dot(l, n);     // dot product
vec3  v2 = cross(l, n);    // cross product
vec3  v3 = normalize(l);   // normalize
vec3  v3 = reflect(l, n);  // reflect

// also:  faceforward() and refract()
```

Built-in Functions

- `reflect(-l, n)`
  - Given `l` and `n`, find `r`
  - Angle in = angle out



Built-in Functions

- Rewrite without **`length`**

```
vec3 p = // ...
vec3 q = // ...

vec3 v = length(p - q);

vec3 v = distance(p, q);
```

Built-in Functions

- What is wrong with this code?

```
vec3 n = // ...

normalize(n);
```

Built-in Functions

- Selected Matrix Functions

```
mat4 m = // ...

mat4 t = transpose(m);
float d = determinant(m);
mat4 d = inverse(m);
```

Built-in Functions

- Selected Vector Relational Functions

```
vec3 p = vec3(1.0, 2.0, 3.0);
vec3 q = vec3(3.0, 2.0, 1.0);

bvec3 b = equal(p, q);        // (false, true, false)
bvec3 b2 = lessThan(p, q);    // (true, false, false)
bvec3 b3 = greaterThan(p, q); // (false, false, true)

bool b4 = any(b);            // true
bool b5 = all(b);            // false
```

Built-in Functions

- Rewrite this in one line of code

```
bool foo(vec3 p, vec3 q) {
  if (p.x < q.x) {
    return true;
  }
  else if (p.y < q.y) {
    return true;
  }
  else if (p.z < q.z) {
    return true;
  }
  return false;
}
return any(lessThan(p, q));
```

Samplers

- *Opaque* types for accessing textures
- Always **`uniform`**

```
// fragment shader
uniform sampler2D colorMap; // 2D texture

vec3 color = texture(colorMap, vec2(0.5, 0.5)).rgb;

vec2 size = textureSize(colorMap, 0);

// Lots of sampler types:  sampler1D,
// sampler3D, sampler2DRect, samplerCube,
// isampler*, usampler*, ...
// Lots of sampler functions: texelFetch, textureLod
```

Samplers

- Returns **vec4**
- Coordinate access differs by sampler type

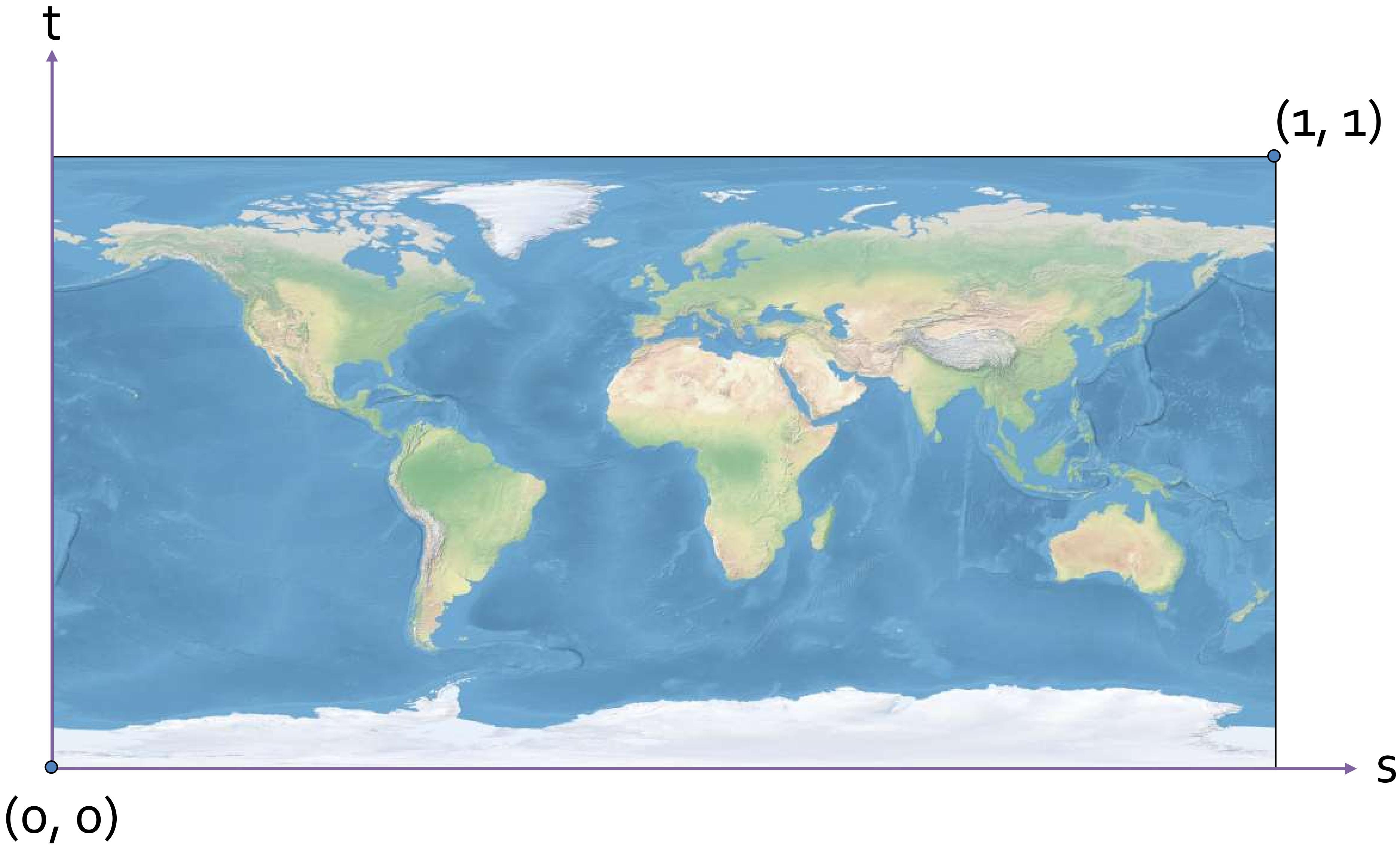
```
// fragment shader
uniform sampler2D colorMap; // 2D texture

vec3 color = texture(colorMap, vec2(0.5, 0.5)).rgb;

vec2 size = textureSize(colorMap, 0);

// Lots of sampler types:  sampler1D,
// sampler3D, sampler2DRect, samplerCube,
// isampler*, usampler*, ...
// Lots of sampler functions: texelFetch, textureLod
```

Samplers – Texture Coordinates



Images from: http://www.naturalearthdata.com/