## **Computer Graphics 1**

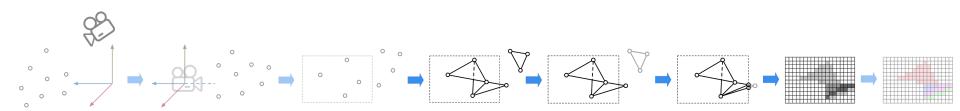
#### **Tutorial 6 Rasterization II**

Summer Semester 2021 Ludwig-Maximilians-Universität München

#### **Tutorial 6: Rasterization II**

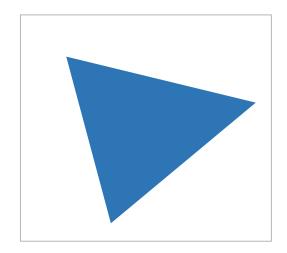
#### • Drawing Sampling

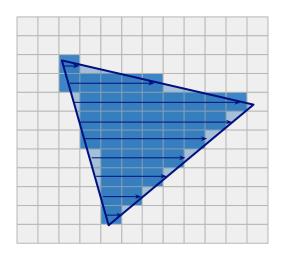
- Issues with Bresenham algorithm
- Point-in-triangle assertion
- Anti-aliasing
- Modern Rasterization Rendering Pipeline
  - Shader language and shader programs
  - OpenGL Shading Language (GLSL)
  - Vertex Shader
  - Fragment Shader
  - Debugging Shaders

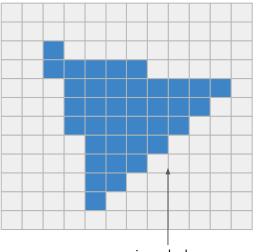


#### **Issues with Bresenham and Scan Line Algorithms**

- Performance: Desire parallelized execution for all pixels but drawing a line from left to right is sequential
- Aliasing: scan converted objects exhibit discretization artifacts (staircase effect)





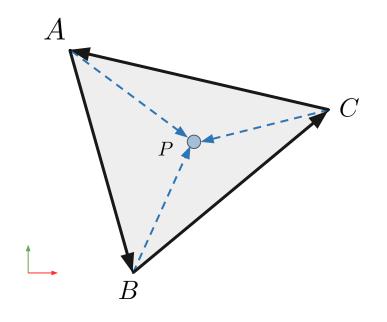




#### **Point-in-Triangle Assertion**

Basic idea: If P is always on the left of all edges

P is on the left side of AB:  $\langle \overrightarrow{AB} \times \overrightarrow{AP}, (0, 0, 1, 0)^{\top} \rangle$ P is on the left side of BC:  $\langle \overrightarrow{BC} \times \overrightarrow{BP}, (0, 0, 1, 0)^{\top} \rangle$ P is on the left side of CA:  $\langle \overrightarrow{CA} \times \overrightarrow{CP}, (0, 0, 1, 0)^{\top} \rangle$  $\Rightarrow$  P is inside triangle ABC



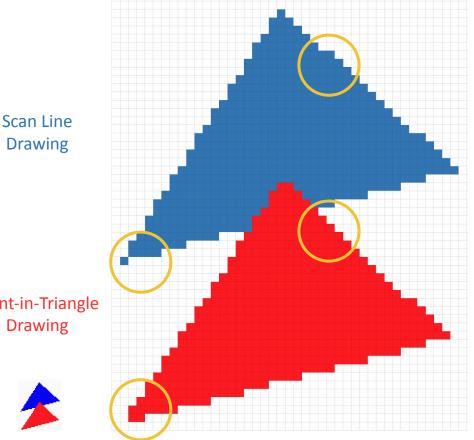
Alternative to scan line algorithm for triangle drawing:

For all pixels in the AABB of a given ABC, if a pixel is inside the triangle, then draw the pixel.

Point-in-triangle assertion is implemented on the GPU as fixed, specialized function. The GPU executes this test for all pixels parallelly and efficiently. Testing point-in-triangle is the most practical and efficient approach to draw a triangle.

#### Scan line vs. Point-in-Triangle Assertion based Drawing

- Scanline algorithm embeds numeric issue inside the algorithm design: when should the coordinates of a vertex position be numerically rounded (i.e. which pixel to initiate the drawing)?
- Point-in-triangle assertion is a boolean assertion to check if pixel center is inside the triangle, and can be easily optimized and executed in parallel

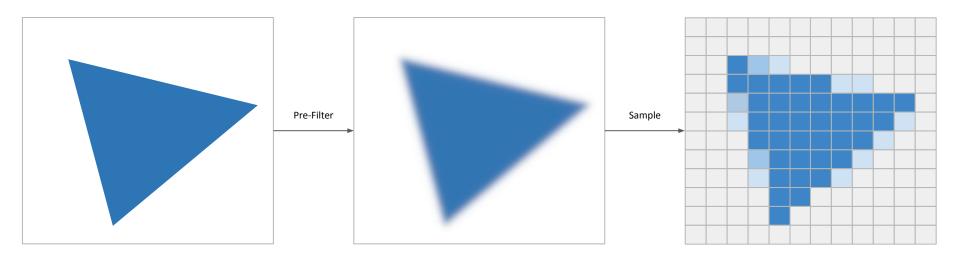


Point-in-Triangle Drawing

\*Corner case: if a pixel center is exactly at the edge of the triangle: decide yourself in the implementation

### **Aliasing and Antialiasing**

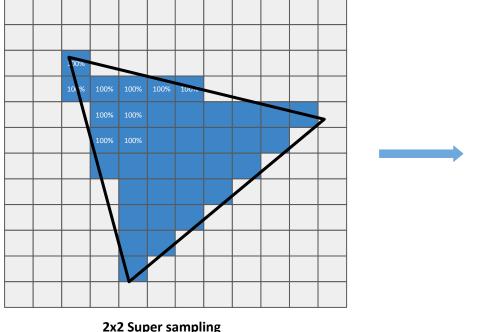
- Both scan line algorithm, and point-in-triangle assertion based drawing introduces line *aliasing* issue
- How to reduce aliasing issue?
  - Higher resolution display (therefore higher frame buffer) i.e. + €€€
    - Disadvantage: adds more computation cost to software, and needs high resolution on hardware
  - Antialiasing

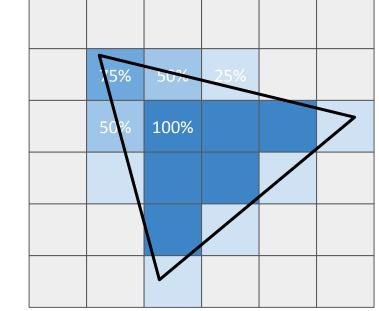


## Multi-Sample Anti-aliasing (MSAA)

Multi sample antialiasing (MSAA): Sampling high resolution samples then render in a lower resolution

MSAA computes the coverage of a triangle area on a pixel





Averaging down

### **Antialiasing Today**

The antialiasing methods that appear in many video games:

- Fast Approximate Antialiasing (FXAA, 2009)
- Temporal Antialiasing (TXAA, 2012)
- Deep Learning Super Sampling (DLSS 2.0, 2020)



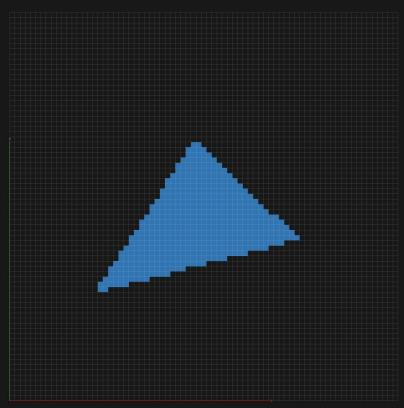
#### **Breakout: Implement Point-in-Triangle Assertion**

Enter folder demos/06-raster2/1-draw (live demo)

Look for TODO comments in the main.ts

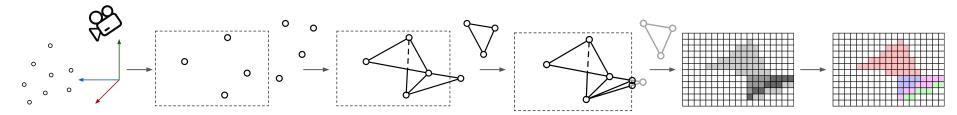
1. Implement the drawTriangle function for the point-in-triangle assertion based drawing of a given triangle.

2. Modify vertex positions of the triangle, answer these questions:How efficient is the point-in-triangle assertion?Does the shape of the triangle impact the performance?



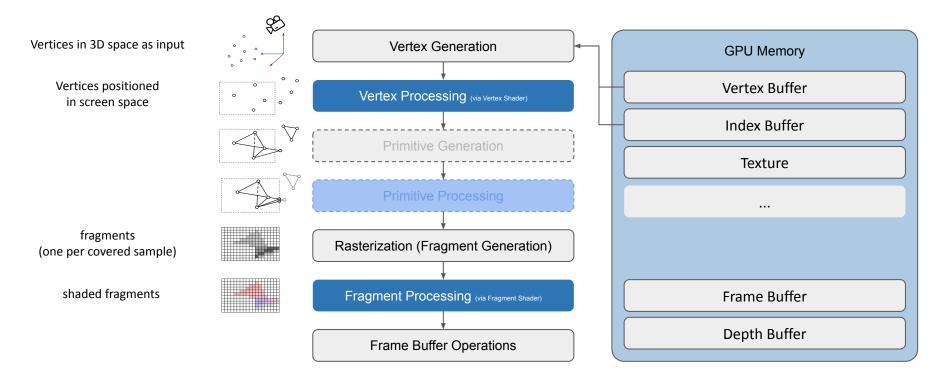
#### **Tutorial 6: Rasterization II**

- Drawing Sampling
  - Issues with Bresenham algorithm
  - Point-in-triangle Assertion
  - $\circ$  Anti-aliasing
- Modern Rasterization Rendering Pipeline
  - Shader language and shader programs
  - OpenGL Shading Language (GLSL)
  - Vertex Shader
  - Fragment Shader
  - Debugging Shaders



#### Modern Rasterization Rendering Pipeline (on GPU)

The pipeline can be executed for multiple *pass*es, and one rendering *pass* means: 1) create a frame buffer, 2) specify one or more buffers as output, and 3) render content from an output buffer



#### **OpenGL** *Deprecated*!

OpenGL is a standardized set of APIs that describes the previous rasterization rendering pipeline on a GPU.

Advantage:

• Cross platform

Disadvantages:

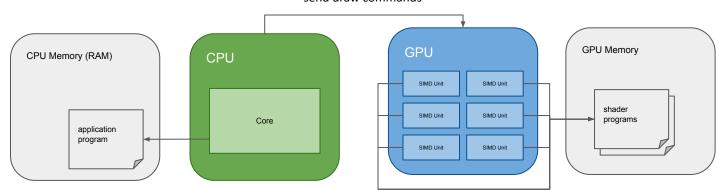
- Compatibility: different versions have different set of APIs or different API behaviors
- State-machine programming model, C-style and not easy to use
- Debugging is (or was) non-trivial

Fore more, see http://docs.gl/. We will not discuss OpenGL in detail. Instead...



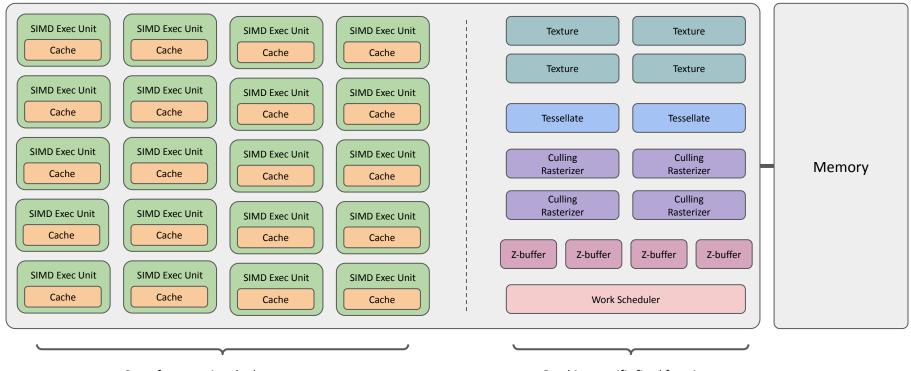
### **Shader Program and Shading Language**

- Shader is a small program that runs on GPU instead of CPU
- Shader programs are written in language similar to C but with restrictions, called *shading language*
- To run a shader program (on GPU), similar to CPU programs, one must:
  - 1. create shaders for compilation
  - 2. compile shaders for execution
  - 3. link shader programs together and the application
  - 4. use shader program when necessary



send draw commands

#### **Executing Shaders on a Multi-core Processor (GPU)**

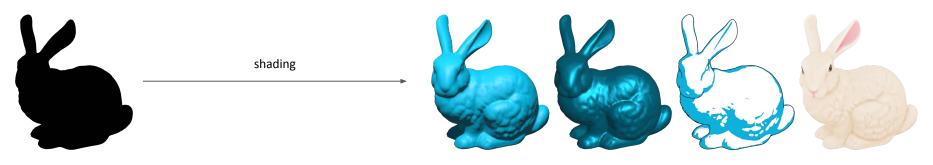


Cores for executing shader programs (programmable), in parallel

Graphics-specific fixed functions (non-programmable) and compute resources

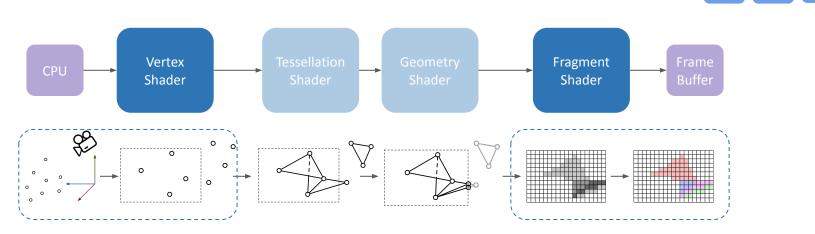
### Why A Language?

- High-level, domain-specific language to describe *shading behavior* 
  - Better utilize GPU and can customize
  - In ancient times: assembly on GPUs
  - e.g. GLSL in OpenGL, HLSL in DirectX
- Shading is a *local* behavior for a specific material
- A rasterizer turns geometries into pixels via sampling but does not include the process of how to figure out what is the "correct" color of a pixel, e.g. different shading behavior



## **OpenGL ES Shading Language (GLSL ES)**

- GLSL ES (shortly GLSL) enables programmable stages of graphics pipeline computing using GPU in WebGL
- Different shader stages
  - vertex shader
  - tessellation shader
  - geometry shader
  - fragment shader
  - compute shader
  - o ...



#### WebGL Shader Support

- Safari doesn't work with WebGL2 (why Apple? why?)
- Use Chrome, or Firefox
- . . . WebGL Report × + 🖈 🔹 👗 🏠 webglreport.com/?v=2 WebGL Report WebGL 2 WebGL 1 ✓ This browser supports WebGL 2 Platform: MacInte Vertex Shader Transform Feedback Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/537.36 (KHTML, like Gecko Browser User Agent: Chrome/91 0 4472 57 Safari/537 36 ax Interleaved Components Context Name: webal2 lax Vertex Uniform Vectors: 1024 ax Separate Attribs: GL Version: WebGL 2.0 (OpenGL ES 3.0 Chromium) lax Vertex Texture Image lax Separate Components: CISIES 300 (O ER CI SI ER 3 ax Varying Vector • • • • • • a webgireport.com 90 C est float precision Textures Max Vertex Uniform WebGL Report lax Texture Size: lax Vertex Uniform Blocks Max Cube Map Texture Size lax Vertex Output av Combined Texture Image Units: nisotropy WebGL 2 lax 3D Texture Size Max Array Texture Layers: × This browser does not support WebGL 2 ax Texture LOD Bias: Check out Get WebGL, or try installing the latest version of Chrome, or Firefox. Rasterizer liased Line Width Range Uniform Buffers liased Point Size Range: Platform: MacIntel Uniform Buffer Bindings Browser User Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_6) AppleWebKit/605.1.15 (KHTML, like Gecko) Max Uniform Block Size: Version/14.0.3 Safari/605.1.15 iform Buffer Offset Alignment: Fragment Shader ax Combined Vertex Uniform lax Combined Fragment Uniform mponents ft Extensions" option, and relaunch oat/int precision: Framebuffer lax Fragr Max Draw Buffers: Aax Color Attachmer

16384

16384

2048

2048

16

80

#### **Basic GLSL Concepts**

• types: int, float, vec2, vec3, vec4, mat2, mat3, mat4, sampler2D, struct, array

vec4 position; // position.x, position.yz (result in vec2) to access components vec4 normal; // x, y, z, w for coordinates vec4 uv; // s, t, p, q for texture coordinates vec4 color; // r, g, b, a for color channels

• *quantifiers*: in, out, inout, uniform

in vec4 color; // an input of a shader out vec4 colors; // an output of a shader, "varyings" before WebGL 2.0 uniform float ka, kd, ks, p; // constant (but not compile-time constant)

• *function*: a code block maps a list of parameters to a list of return values

float rand(vec2 co){ // generates a random number
 return fract(sin(dot(co.xy ,vec2(12.9898,78.233))) \* 43758.5453);
}

• *flow control*: if/else/for statements (in almost every-language)

See a summarized reference note: https://www.khronos.org/files/webgl20-reference-guide.pdf, page 6-8

	nading Language	3.07	<b>API</b>	Quick Re	eteren	ce Guide								Page	
The OpenGL® ESS	hading Language is two clo	selv-		Preproce	essor 13.4	ų									
related languages v	which are used to create sha	ders for		Preprocess	or Directiv	res									
	ment processors contained			The number s		be immediately p		or follow							
WebGL 2.0 is based	nd OpenGL ES processing pi t on OpenGL ES 3.0.	peanes.		r reff	Adefine Rendd	Pundef Persor	#if Porpan	10	#[def Acctension	#(ed Fine	d i	Nelse			
				frameles of Par	Hannessar Dir	ections.									
[n.n.n] and [Table i	n.n] refer to sections and ta hading Language 3.0 specifi	Dies			on 300 es" mu ting version 1		nt line of a	shader p	program writte	n in GLSL D	Sversion 3.0	D. If omitted, I	he shader w	il be treats	
www.khronos.org/	nating Language 3.0 specifi realistru/eles/	cation a	5	<ul> <li>Restern</li> </ul>	sion extensio	in nome : behavie	ior, where i	betevio	or can be requ	ire, enable,	wirn, or dis	bir; and whe	ne extension	nony is th	
www.ononos.org/	region y/ pro/			extensio	on supported I	by the compiler for, offil - enable	a out effective	in cheste	v and minute	a Mala de	and a				
			-	Apragre	a debugijor	, off - enable o	ir disable c	compilie	ig shaders w	ith debug	information	(default off	1		
Types [4.1]				Predefined	Macros										
A shader can aggregate more complex types. Th	these using arrays and structure wre are no pointer types.	IS TO DUIN	2	_LINE_		Decimal Integer of	contact the	at is one	more than the	number of	proceeding re	wlines in the o	uniet source	e stating	
Basic Types				_FLE Decimal integer constant that says which source string number is currently being processed.											
	no function return value or empty parameter list			VERSION Decimal integer, e.g.: 300											
	Boolean			GL_ES		Defined and set t	to integer 1	if runnie	ig on an Opend	L 45 Shade	g Language.				
int, uint	signed, unsigned integer							-		-			_	-	
foit	foating scalar		-11	Qualifier	re		_		Parame	ter Qual	fiers [4.4	1			
			-11							Parameter Qualifiers [4.4] Input values are copied in at function call time, output values are copied out at function return time.					
			-11		rations may	be preceded by	one stora	101	copied cs	t at functi	on return t	ime.			
	dated interest webst		-11	qualifier.		none (Default) same as its									
wer2 wer3 wer4			-11		(Default) local read/write memory, or input parameter Comple-time constant, or read-only function			in I	or function	paramieters p	essed into a fu	nction			
			-11	const	parameter					or function	pirimiteri -	arried back or	e of a fury to	n but	
	242, 343, 464 TOLE MATH	_	-11	in controld in	Unkage int	to a shader from a p	previous sta	ice .	out	not initialize	d for use whe	n passed in		10000	
			-11	centroid in out	-				inout 1	or function	paramieters p	essed both int	a and out of a	function	
			-11	centroid out		it of a shader to a su									
matiki2, matiki3, matiki4	4x2, 4x3, 4x4 Toat matrix		-	122	Value does	not change across the	e primitive b	iene .	Precisio	n and Pr	ecision Q	ualifiers [4	1.5]		
Floating Point Samp	pler Types (opaque)			uniform	shader, Dor	uniforms form the Im erGLES, and the appl	ikadi bebilik ikation	623	Any floati	ng point, in	tager, or sa	mpler decla	ation can h	ave the	
sampler2D, sampler3D access a 2D or 3D texture			The following interconlation recalifiers for shader network					type preceded by one of these precision qualifiers:							
sample/Cabe access cube mapped texture			and inputs may procede in, centroid in, out, or centroid out.					highp Satisfies minimum requirements for the vertex language.							
samplerCubeShadow access cube map depth texture with comparison			sentroid out. smooth Peripective carried interpolation					mediump	Ronge-	and precision the	is between th	at provided I	ly lowp		
sampler205hadow access 2D depth testure with comparison			Bat	To interaction							can be less the	to burther mediane but -1			
sampler20Array access 2D array texture		1		fowp Range and precision can be less than mediump, but still represents all color values for any color channel.											
	access 2D array depth texture with co	maarlum		Interface Bi	locks [4.3.	7] ions can be grou	and later		Ranger -	dancin	s for ore-t-	ion qualifier	Indian	e noint's	
			-	interface bloc	its, for exam	uple:	1 (111 Direct	-arred	condrop of		19 Magnitu	de .	ktiege	Range	
	pler Types (opaque)									FP Kange	Range	FP Precision			
kampler20, kampler20 access an integer 20 or 30 texture		-11	mat4 Mode/NewProjectionMatrix; uniform mat3 NormalMatrix; // restatement of qualifier					r highp			") Relative 2"				
IsamplerCube access integer cube mapped texture		4	foat Deformation;					mediump	1-2".2"	(2*,2*)	Relative 21	[-25,26-1]	11.2*-1		
isampler20Array access integer 20 array texture			Layout Qualifiers [4.3.8]					lown	4.2.25	01.2		62.24			
Unrigged Integer St	ampler Types (opaque)			Invostilou	out-qualifier	ri block-declarati	101								
	access unsigned integer 2D or 3D text			layout(loy	out-qualifier	in/out/uniform			A precisio subseque		nt establish t, and same	ies a default aler declarat	precision o	wannier fo	
	access unsigned integer cube mapped		1	layout(loy decise	out-qualifier ation	r) kn/out/uniform				n highp in					
	access unsigned integer 20 annu text		-11												
		- 1	Input Layout Qualifiers [4.3.8.1] Invariant Qualifiers Exu For all shader states:												
Structures and Arrays [4.1.8, 4.1.9]			location = integer constant						Ppragma STDGL invariant(all) Force all output variables to be						
Structures struct type name [ members			Output Layout Qualifiers [4.3.8.2]								Qualify a previously declared variable				
struct-nome	1) optional variable declaration,	// optional variable declaration,			For all shader stages: location = integer constant					Invariant centroid out yet3 Calor:			Qualify as part of a variable declaration		
	// optionally an array		-11			ualifiers (4.3.8.	21		vec31	alor;	40				
Amays feat foo[3]; Structures	blocks, and structure members can b	1975		Layout qualify	er identifiers	for uniform bloc	cks:		-						
Only 1-dim	rensional arrays supported.			shared, pack	ued, std140,	(row, column)_m	ajor				ation [4.]		here must fo	dow a	
		_		-	_		-	-	strict ord	er. This ord	ler is either				
Operators and	Expressions		2	< > ci 31	original				inat	iant, interg	volation, st	proge, precis	ion		
Operators [5.1] Numbered in order of precedence. The			1	aa ja	equality			1.1	storp	pe, porper	eter, precis	ion			
			9	4	bt-wise and			t-t							
Boolean. To compare ve such as lessThan(), equi	ictors component-wise, use func all. etc. 18.71	Domi	22.		bit-was each			L-1	1.						
	escription	Assec	11.	65	bit-wise indu logical and	NUM OF			Manager -		A 10 13	_			
	aventhetical grouping	8/8	11	M	logical exclusion	live or		1	Vector Co In addition t	o array cu	meric subsi	ript syntax,	sames of w	sctor	
	tay subscript		14	11	logical indus								reponents a	an be	
1 () pa	ection rail & methyday douture	L-R			selection (5	elects an entire ope	eget		swizzled and	repricates	1.0.8 0.01	xx, pcs.zy s that represen	-	and a	
1 () pa (1 ar (1 b)			15.	2:	Use mix() to components	select indvidual	L	L-II				s that represen s that represen			
1 () pa () ar 2 () 5 ++ - p()	eld or method selector, swizzler ostfia increment and decrement					an market p.1									
1. () pa 1. () pa 2. () for 1. () pa 1. ()	eld or method selector, switzler ostfle increment and decrement who increment and decrement	R-L			assignment								hotors month	100	
1 () pr 1 () pr 2 () fo + - pr 1 + - pr	eld or method selector, swizzler ostfia increment and decrement	R-L	15.	+1 + 1 h					(LCAR)	os unen sci	send rectar	that represent	tenture coord	white	
1. () p 1. () f 2. () f 4 p 4 s / m	eld or method selector, salizater oxfile increment and decrement refle increment and decrement sary		15.	+1 -1 *1 /1 %4 -00 -300	assignment anthmetic as	usignments		Lik	(LLAG)	ow when an	eang sectors	that represent	texture coord	white	
1 () µ 1 () µ 2 () 6 ++ - 0 1 ++ - 1 0 4 + 5 / m 5 + - 4	eld or method selector, selcoler oxfle increment and decrement who increment and decrement sary ultiplicative	£-8	15.	+1 + 1 h		signments			(1.6 p. 4)	Je when an	anna regau	that represent	texture coord	w/rs	

#### **Attributes and Uniforms**

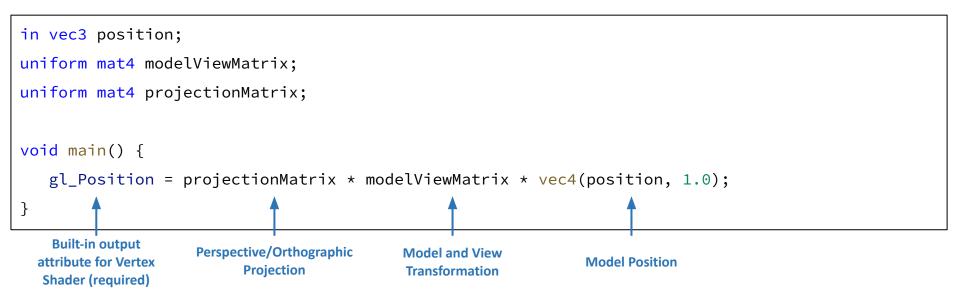
- (Vertex) attributes are user defined
  - Global variables that can be different for each vertex (e.g. normal vector)
  - Read-only, only available in Vertex Shader
  - Definable in program code
- uniforms are
  - Parameters that are the same for many/all vertices/primitives are defined, they are identified via their GLSL variable names (analogous to attributes)
  - Each variable is assigned a "location" (index)
    - compare strings more efficiently than with every change
  - Can be read in vertex and fragment shaders (read-only)

#### **Shader Programs: Vertex Shader**

- Transformation of single vertices and their attributes (e.g. normals, ...)
  - No vertex generation
  - No vertex destruction (handled by clipping)
- Calculation of all attributes that remain constant per vertex
  - $\circ$   $\,$  Saves computing time compared to the Fragment Shader  $\,$
  - e.g. lighting by vertex (old-fashioned)
- Set attributes to be interpolated per fragment
  - $\circ~$  e.g. normals for per-pixel lighting
- **gl\_Position**: *must* be written in the vertex shader.
- Determines the position of the vertices, otherwise cannot continue to the subsequent stages of the pipeline.

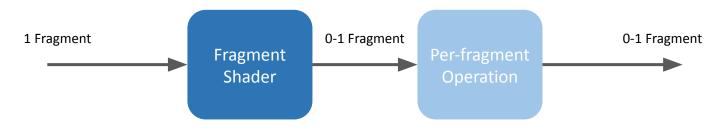


#### **Example: A Minimum Vertex Shader**

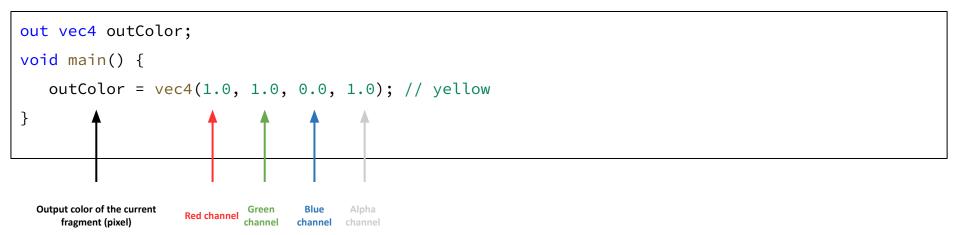


#### **Shader Programs: Fragment Shader**

- Allows calculation per result pixel that ends up in the output buffer
  - Per-pixel lighting/shading
  - $\circ$   $\;$  Sampling of data within the primitive, e.g. for
    - volume rendering
    - Implicit surfaces, glyphs
- The in attributes are interpolated (discussed later) within the primitive (can be turned off)
- Fragments can be discarded: discard
- Fragment operations: Tests, blending and etc.
- The **out** (in Fragment Shader): stores the color of a fragment.



#### **Example: A Minimum Fragment Shader**



#### Shader Support in three.js

Similar to all graphics APIs, three.js treats shader programs as string input, and supports ShaderMaterial and RawShaderMaterial for customizable shaders.

The RawShaderMaterial compiles raw shaders without any additional information. For the better collaboration with three.js internal states (e.g. transformation matrices). ShaderMaterial adds convenient default uniform and attributes.

In a vertex shader:

```
uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec3 cameraPosition;
in vec3 position; // vertex position
in vec3 normal; // vertex normal
in vec2 uv; // vertex UV
in vec4 color; // vertex color
```

In a fragment shader:

uniform mat4 viewMatrix; uniform vec3 cameraPosition;

\*There are more default uniform and attributes, see https://threejs.org/docs/index.html#api/en/renderers/webgl/WebGLProgram

#### Using ShaderMaterial in three.js

Similar to all graphics APIs, to use shader in three.js, pass shader program as a string to the material of a mesh, then three.js will do the rest of the job for us:

```
import vert from './shaders/vs.glsl';
import frag from './shaders/fs.glsl';
```

```
... // create geometry
```

```
const mesh = new Mesh(geometry, new ShaderMaterial({
  vertexColors: true, // use vertex colors the are specified in three.js
  glslVersion: GLSL3, // use the latest GLSL version (3.0)
  vertexShader: vert, // vert is a loaded string
  fragmentShader: frag, // vert is also a loaded string
}));
this.scene.add(mesh);
```

#### Example: A Minimum Vertex Shader using ShaderMaterial

in vec3 position; // provided by three.js automatically uniform mat4 modelViewMatrix; // provided by three.js automatically uniform mat4 projectionMatrix; // provided by three.js automatically

void main() {

```
gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
```

}

#### **Breakout: Getting Started with GLSL**

In folder demos/06-raster2/2-glsl (live demo), look for TODO comments in the main.ts, shaders/vs.glsl and shaders/fs.glsl, Implements the two shaders (vertex and fragment) for the tetrahedron we had worked in the previous geometry tutorial breakout.

Answer: How does the colors of the tetrahedron vertices be visualized?



#### **Breakout: Getting Started with GLSL**

The color propagates along: Vertex color attributes  $\rightarrow$  ShaderMaterial vertex color enabled  $\rightarrow$  VertexShader vertexColor

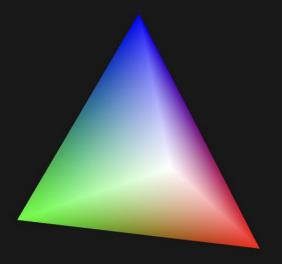
 $\rightarrow$  Fragment Shader vertexColor  $\rightarrow$  Fragment Shader outColor  $\rightarrow$  Display

#### 1. Vertex shader implementation

```
out vec3 vertexColor;
void main() {
  gl_Position = projectionMatrix * viewMatrix * modelMatrix * vec4(
      position.x, position.y, position.z, 1.0
  );
  vertexColor = color;
}
```

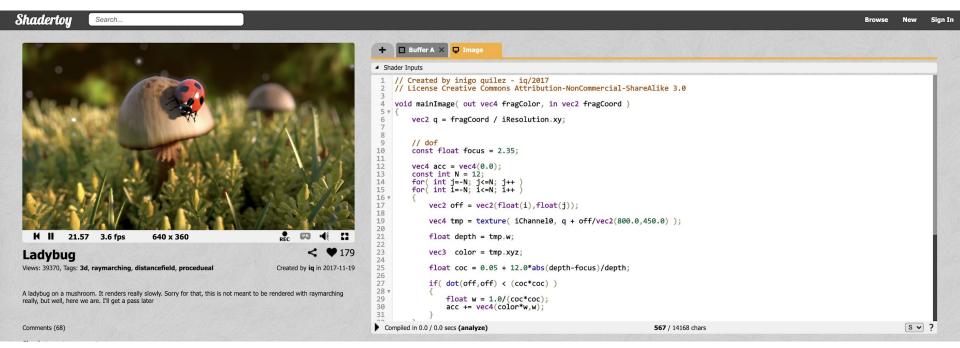
#### 2. Fragment shader implementation

```
in vec3 vertexColor;
out vec4 outputColor;
void main() {
    outputColor = vec4(vertexColor, 1.0);
}
```



#### Shaders are powerful!

- Shaders can do more than you might think, but also non-trivial to write
- ~800 lines of code:



https://www.shadertoy.com/view/4tByz3

#### **Compute Shader**

- Compute shaders allows general purpose, parallel computing on the GPU (with many many cores)
  - Examples: Physics calculations, particle systems, fluid or substance simulations
- Compute shader is located outside the rendering pipeline
  - $\circ$   $\;$  No input from inside the pipeline and no output to the pipeline
- Can read and write textures, images and shader buffers
- WebGL Support
  - No support, and will not be supported :(
  - o (Yet) very early alpha support in WebGPU and requires Chromium nightly builds

#### **Debugging Shaders**

Difficulties in order to debug shaders:

- print out values: shaders are executed on GPU but print out is on CPU
- set breakpoints: shaders are executed on GPU in parallel and unclear which and what break (sometimes)

Traditional debuggers are less used with increasing coding experience because:

- Most difficult errors in complex programs are conceptual bugs where the wrong thing is being implemented
- It is easy to waste large amounts of time stepping through variable values with without detecting such cases
- Tools are platform and hardware specific. For example: RenderDoc (no macOS support, why Apple? why?)
- $\Rightarrow$  Review the code carefully can solve almost all problems

Tool-independent, most general approach: Just render value as color then use a color picker to get the output value

#### **Breakout: Experiment with Shaders**

Enter folder demos/06-raster2/3-shaders (live demo)

Look for TODO comments in the src/shaders folder.

Render the bunny:

- 1. Using z coordinate as vertex color
- 2. Using vertex normal as vertex color
- 3. Using random value as vertex color
- 4. Adding random noise to the vertex position

•••

Be creative ;-)



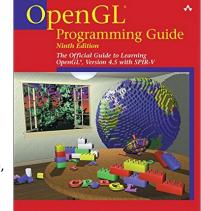
#### **Summary**

- We covered:
  - Issues with Bresenham and scan line algorithms and an alternative drawing approach that using point-in-triangle assertion
  - Aliasing and antialiasing sampling issue in drawing
  - The modern rasterization rendering pipeline and its components
  - GLSL as a programming language for writing shader programs that execute on a GPU

Check the canonical OpenGL book for the more details on a "historical" graphics standard:

To learn more about the history of shading language, check out these research papers:

- Robert L. Cook. 1984. Shade trees. SIGGRAPH Comput. Graph. 18, 3 (July 1984), 223–231.
- Pat Hanrahan and Jim Lawson. 1990. A language for shading and lighting calculations. SIGGRAPH Comput. Graph. 24, 4 (Aug. 1990), 289–298.



John Kessenich • Graham Sellers • Dave Shreiner The Khronos OpenGL ARB Working Group

# ow to create pictures like this?

**WILLING** 

-----\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Next Texture