Computer Graphics (COMP0027) 2022/23

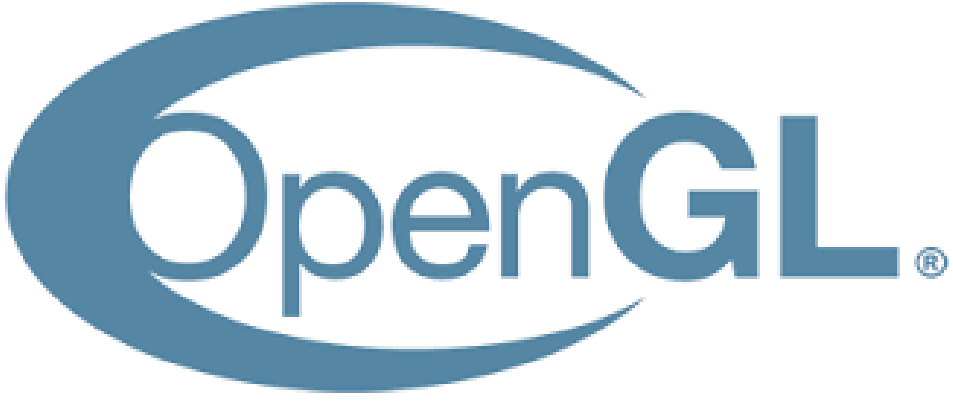
OpenGL

Tobias Ritschel

UCL

Overview

- OpenGL idea
- Pipeline overview
- Individual stages
- Example code
- Deprecated OpenGL



OpenGL: Hard or easy?

- Old-fashioned (deprecated) GL is didactical
- Easy to explain
- But
  - This is not what is out there
  - This is not what we will do here
  - We will try to see how it is in 2017

OpenGL Philosophy

- Platform and language-independent
- Rendering-only
- Aims to be real-time
- Supports Graphics Hardware (GPUs)
- State system
- Client-server system
- Extendable (OpenGL Extensions)

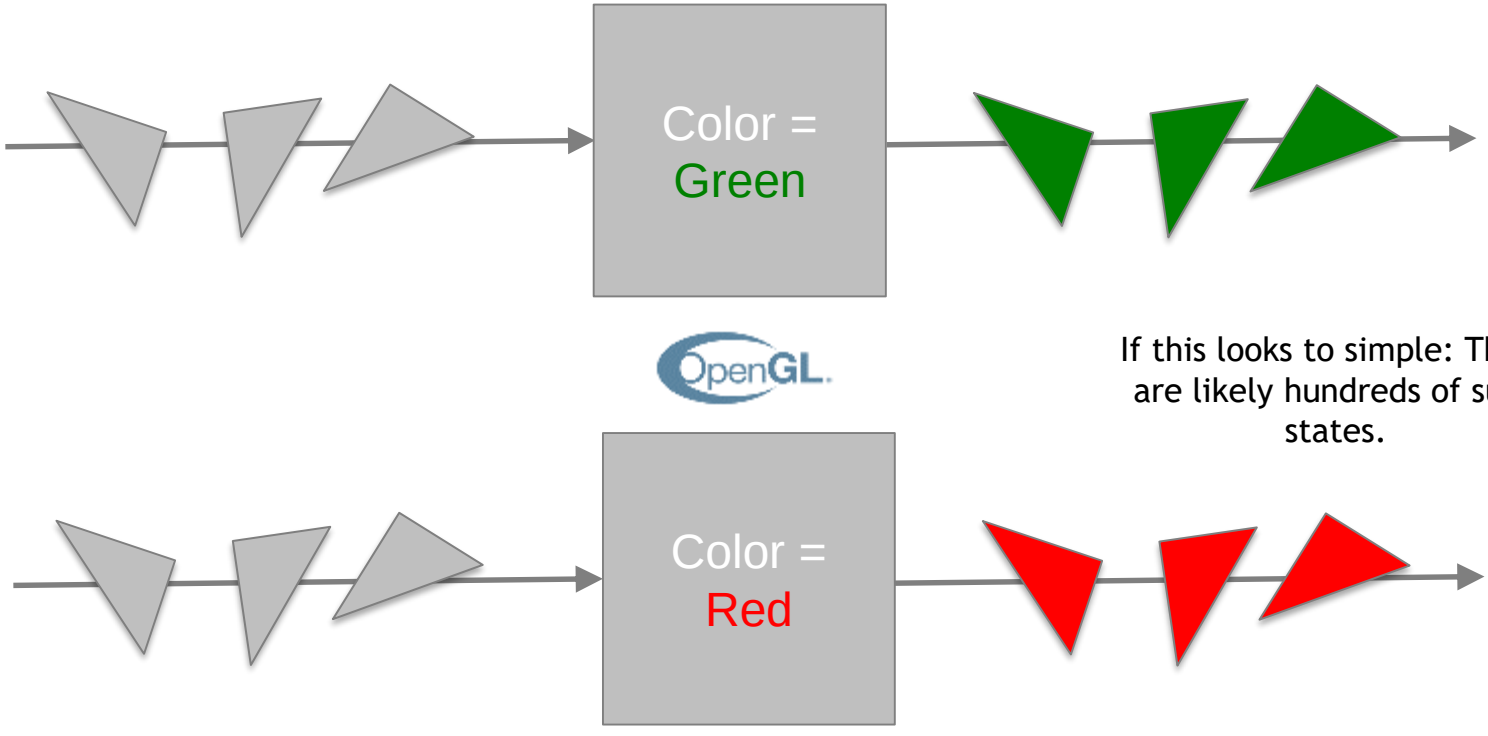
OpenGL

- OpenGL is
  - .. a specification to create images in a frame buffer
  - .. an API to access that mechanism
  - .. well specified

OpenGL

- OpenGL is not
  - .. a window system
  - .. a user interface
  - .. a display mechanism
  - .. a library
  - .. modeling cameras, materials or lights

What is a state system?

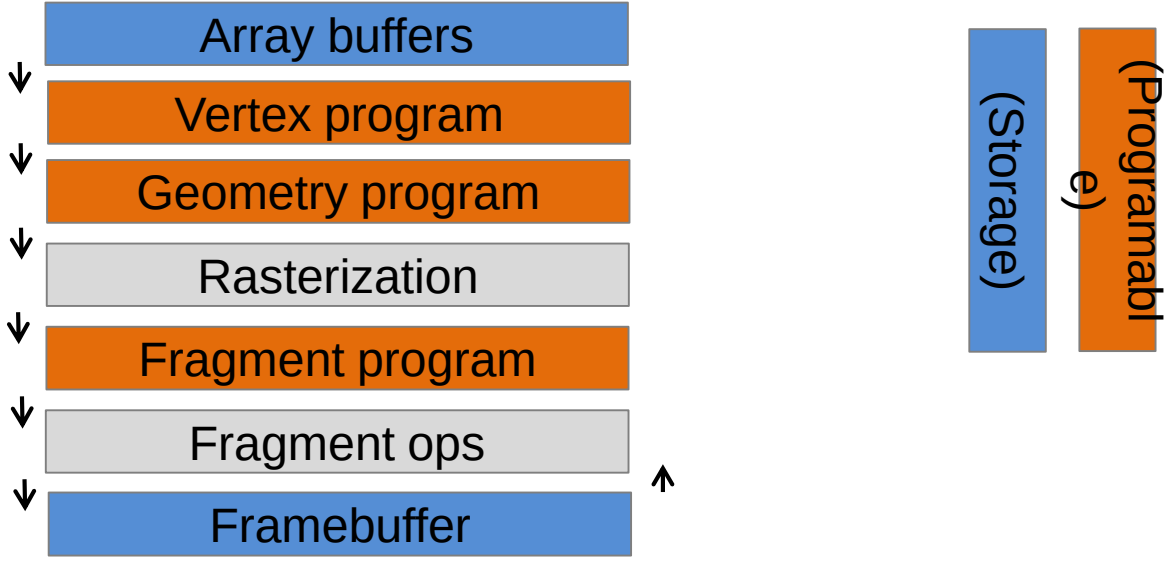


If this looks to simple: There are likely hundreds of such states.

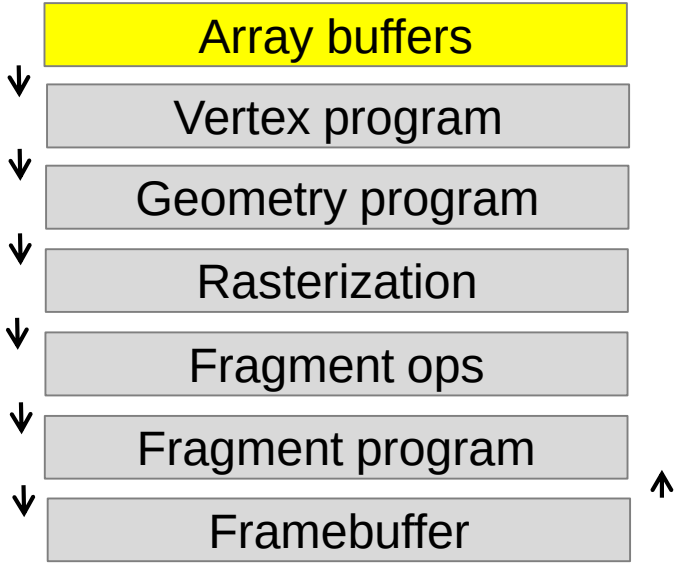
How to program OpenGL

- We will here use WebGL in JavaScript
- A subset of the specification
  - More restricted (-)
  - More modern (+)
  - Runs on all devices (+)
  - Easy to run and compile (+)

OpenGL Pipeline



The OpenGL Pipeline


Array buffers
Vertex program
Geometry program
Rasterization
Fragment ops
Fragment program
Framebuffer

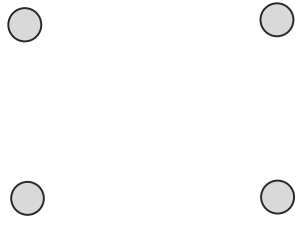
Array Buffers

- Just arrays of memory on the GPU
- It will take batches of data from those arrays and feed them into the pipeline
- Instead of using `new()` or `malloc()` we use gl functions like `bufferData()` to manage these.
- For now, assume they define vertices of a polygonal mesh
- Don’t upload every frame. That is slow.

Array Buffer: Example 1

```
{0, 0, 2, 0, 2, 1, 0, 1}
{x, y, x, y, x, y, x, y}
```

Vertex 0 Vertex 1 Vertex 2 Vertex 3



Array Buffer: Example 2

```
{0, 0,   2, 0,   2, 1,   0, 1}
{x, y,   x, y,   x, y,   x, y}

{0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0}
{r, g, b, r, g, b, r, g, b, r, g, b}
```

Color 0 Color 1 Color 2 Color 3

Array Buffer Example Uses

- Positions
- Colors
- Normals
- (Multiple) u, v texture coords
- Motion flow
- Tangent spaces for bump mapping
- Fully flexible .. no semantic!

Array Buffer Code

```
var positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(
 gl.ARRAY_BUFFER,
 new Float32Array(getVertices()),
 gl.STATIC_DRAW);
```

Element Arrays

- Special type of arrays
- They are used to compose vertices into polygonal primitives
- Only triangles in modern GL

Element Arrays: Example

```
{0, 0,   2, 0,   2, 1,   0, 1}
{x, y,   x, y,   x, y,   x, y}

{0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0}
{r, g, b, r, g, b, r, g, b, r, g, b}

{0, 1, 2, 1, 2, 3}
{a, b, c, a, b, c}
```

Element array: Code

```
var indexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
gl.bufferData(
 gl.ELEMENT_ARRAY_BUFFER,
 new Uint16Array(getIndices()),
 gl.STATIC_DRAW);
```

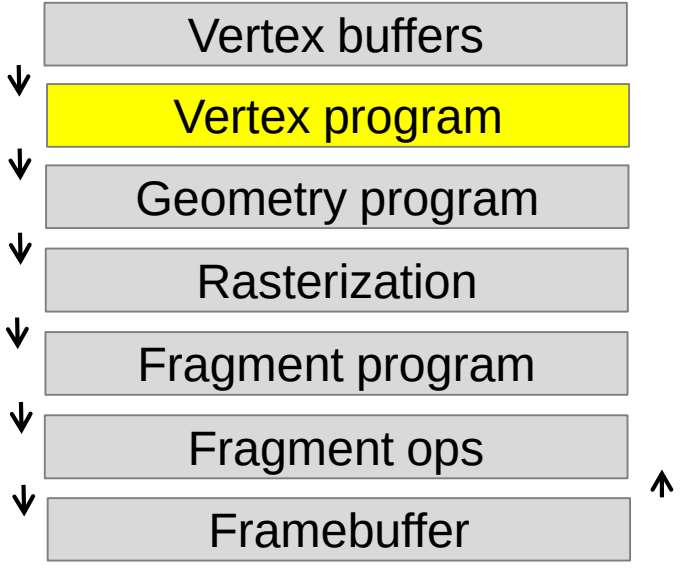
DEMO

Definitions and changing a few array values

Issuing a draw call

- Issue draw call with an element array buffers

```
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, tri);
gl.drawElements(gl.TRIANGLES,
 tri.numItems,
 gl.UNSIGNED_SHORT,
 0);
```


Vertex buffers
Vertex program
Geometry program
Rasterization
Fragment program
Fragment ops
Framebuffer

Vertex program

- Also called “shader”
- There are so many things that can happen to a vertex, that this needs a full programming language: GLSL
- Executed
  - at every vertex
  - in parallel

Vertex program

- Input:
  - Vertices from array buffers now called **attributes**
  - Some **uniforms** that are the same for all vertices
- Output:
  - The clip space **coordinate** of that vertex (before division with w)
  - Every VS output you plan to use per-pixel called **varyings**

What array goes into which attribute?

- Everything possible
- Programmable mapping
- Example for 6 positions of cube:

```
   gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
   var location = gl.getAttribLocation(shaderProgram, "position”);
gl.enableVertexAttribArray(location);
gl.vertexAttribPointer(location, 6, gl.FLOAT, false,  0,   0);
```

Vertex program: Example

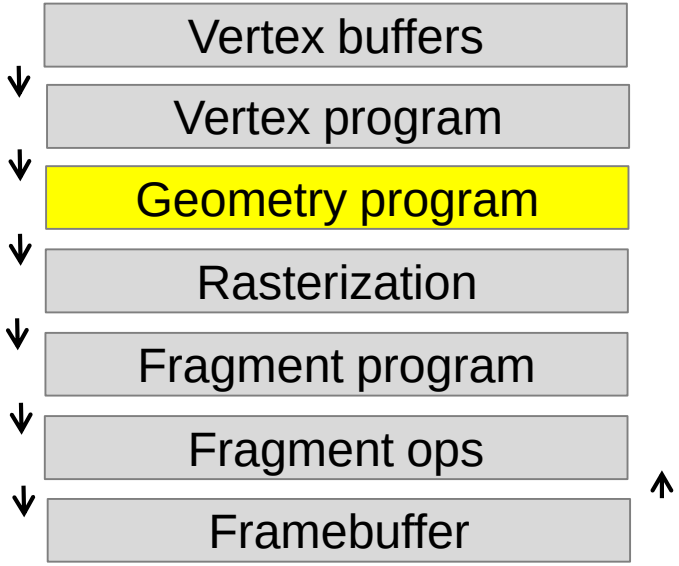
- Input:
  - attribute `position`
  - uniform `direction`
  - uniform matrix
- Output:
  - Clip-space coordinate
  - varying `color`

```
uniform vec3 direction;
uniform mat4 matrix;
attribute vec3 position;
varying color;

void main() {
 gl_Vertex = matrix * position
 color = vec3(dot(position, direction);
}
```

DEMO

Code-walk VS, adding a bit of animation via uniform


Vertex buffers
Vertex program
Geometry program
Rasterization
Fragment program
Fragment ops
Framebuffer

Geometry programs

- So far have processed single **vertices**, not **primitives**
- Remember: Use element array to turn vertices into primitives
- **Geometry** programs can change entire primitives
- Not yet in WebGL implementations. So no demo.

Geometry program

- Input:
  - All attributes (e.g. 3) that form a primitive
- Output:
  - One or multiple new primitives

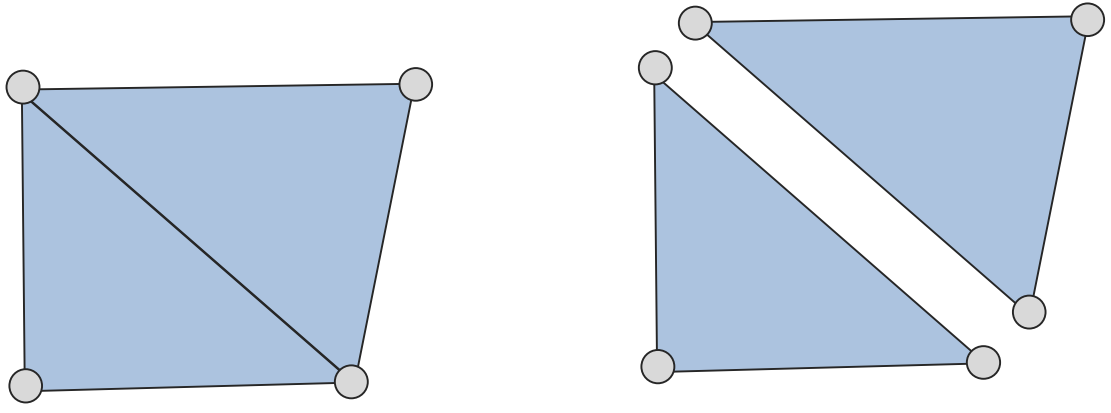
Geometry program: Example

- Input:
  - Tri positions
  - Tri normals
  - A magic jump
- Output:
  - The same, just changed

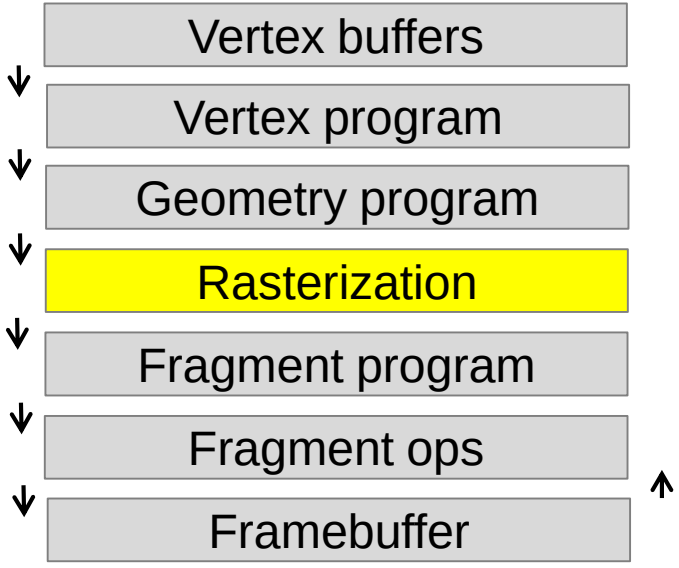
```
in vec3 ip[3]; in vec3 in[3];
out vec3 op[3]; out vec3 on[3];
uniform float jump;

void main() {
 for(int i = 0; i<3; i++) {
  op[i] = ip[i] + jump * in[i];
 }
 on = in;
}
```

Geometry program: Example result



Quiz: Why is this so interesting, could a VS do that?

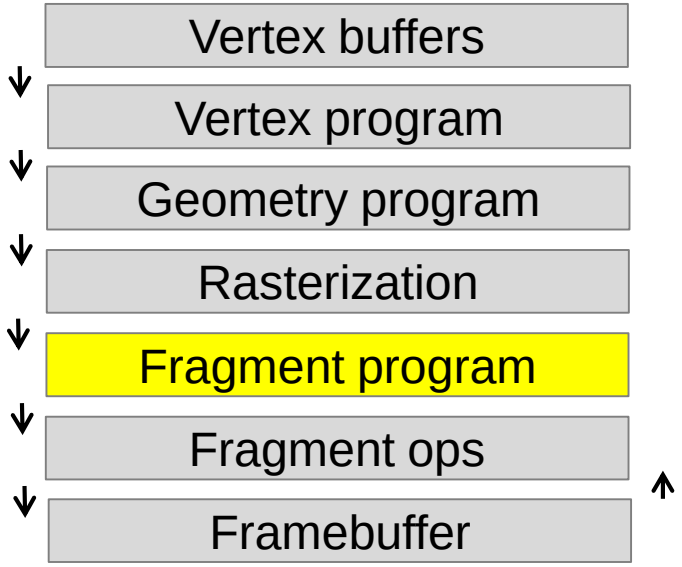

Vertex buffers
Vertex program
Geometry program
Rasterization
Fragment program
Fragment ops
Framebuffer

Rasterization

- Does what you learned
  - Clipping
  - Projective division
  - Culling
  - (perspective) Interpolation
- Can turn on and off culling and some other things

DEMO

Culling on and off (without depth buffering)


Vertex buffers
Vertex program
Geometry program
Rasterization
Fragment program
Fragment ops
Framebuffer

Fragment program

- The most important one
- Executed for every “fragment” (lots of)
- Without antialiasing a fragment is a pixel
- With anti-aliasing, multiple fragments go into a pixel

Fragment program

- Input
  - All the varyings the geometry shader outputs
  - The fragment coordinate
  - Uniforms
  - Samplers (GL name for textures)
- Output
  - A fragment color `gl_FragColor`
  - Optionally, depth `gl_FragDepth`
  - Can also `discard` fragments

Fragment program: Example

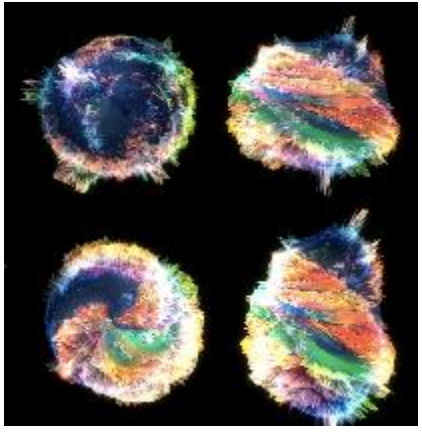
- Input:
  - varying `texCoord`
  - Sampler `texture`
- Output:
  - Color
  - Depth

```
varying vec2 texCoord;
uniform sampler2d testSampler;

void main() {
 gl_FragColor.rgb = texture(
  testSampler,
  texCoord).rgb;
 gl_FragDepth = texture(
  testSampler,
  texCoord).a;
}
```

Shader Examples



DEMO

Pixel shading code, replace with some overly simplified ones

Textures

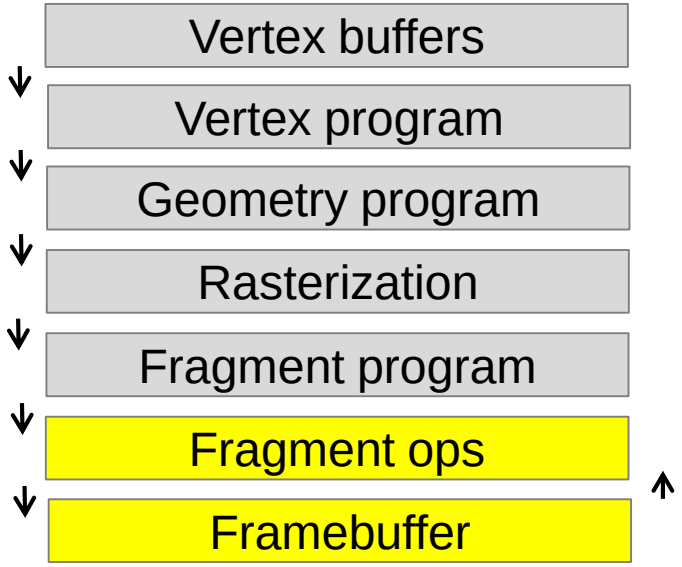
- Textures are storage objects like array buffers
- Have to allocate and fill them with specific calls
- Have the filtering modes explained in last lecture
- Read in every program stage using the function

`texture(sampler, ivec2 texCoord)` w. `texCoord` $0..1$

`texelFetch(sampler, ivec2 texCoord)` w. `texCoord` $0..N$

DEMO

Texture allocation & turning it on in fragment program


Vertex buffers
Vertex program
Geometry program
Rasterization
Fragment program
Fragment ops
Framebuffer

Framebuffer

- How fragment colors affects the frame buffer color
- Configurable, but nor programmable
  - Depth test
  - Blending
  - Multiple render targets / render-to-texture

Depth test

- As you would expect
- Need to allocate and clear the depth buffer to use it
- *Off* by default

DEMO

Turing on and off depth test (with culling off, ideally)

Blending

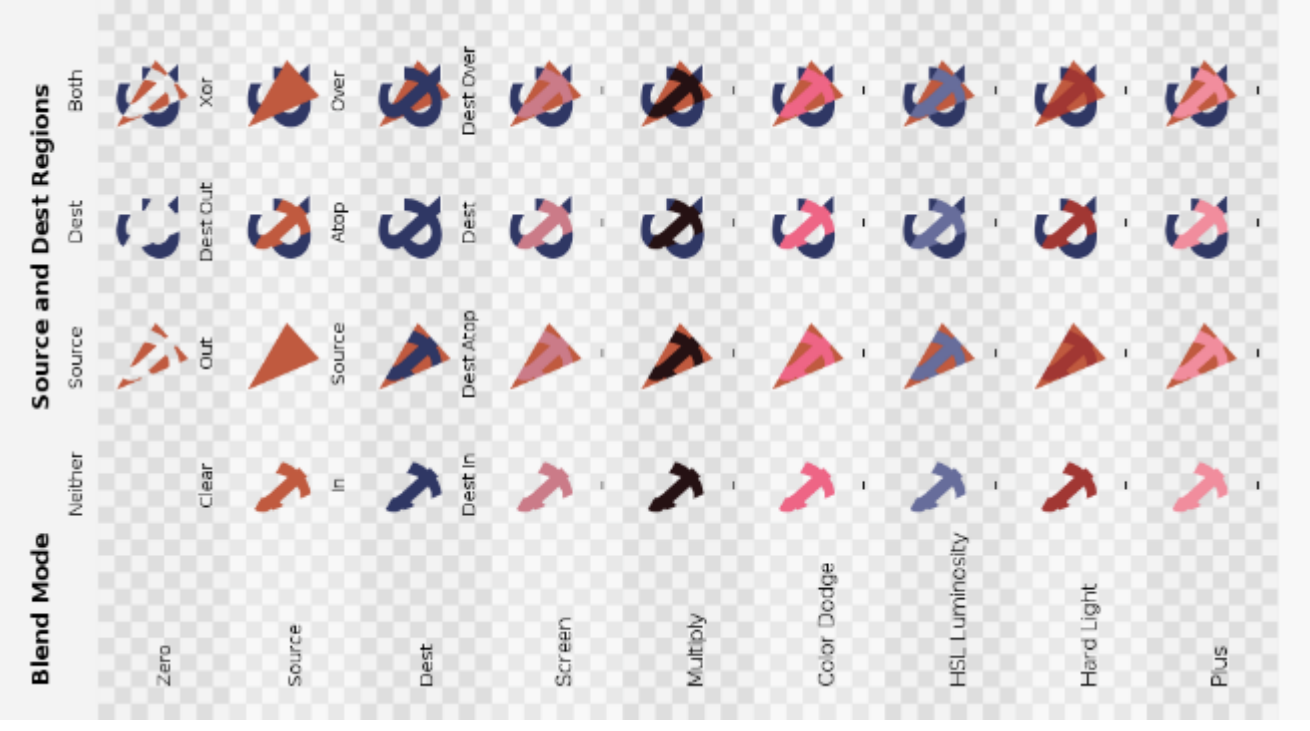
- If depth test passes, GL does not just replace the color such as we do in our coursework
- Instead, it evaluates an equation involving old and new color, including alpha

Blending

```
x_out = x_op(x_src * x_SrcFac, x_dest * x_DestFac)
```

- `x` can be RGB or ALPHA
- `x_out` result
- `x_src` fragment program output
- `x_dest` current frame buffer content
- Can configure `x_op`, `x_src` and `x_dest`

Blending: Example



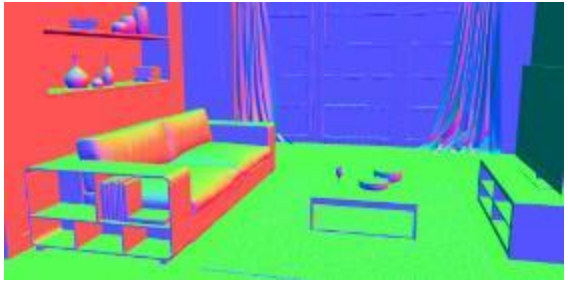
DEMO

Additive blending

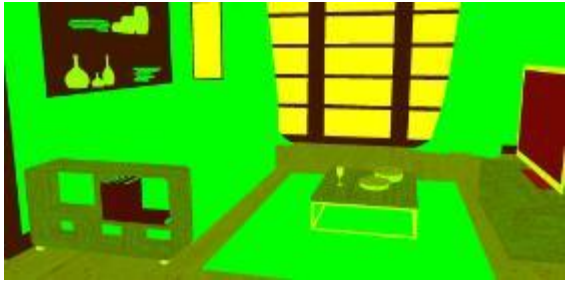
Multiple render targets

- No reason to output only a single color
- Can output multiple colors
- Each goes into its own framebuffer
- This frame buffer can the again be used as input texture in a new shader
- Example: Deferred shading
- First fill framebuffer, then only shade what you see

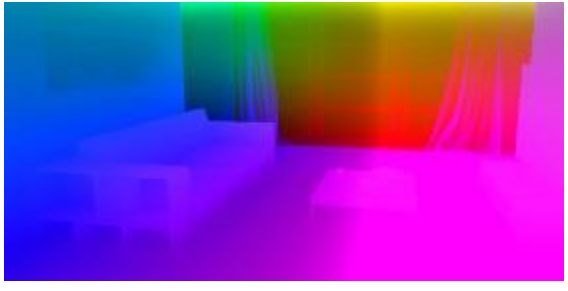
Multiple render targets



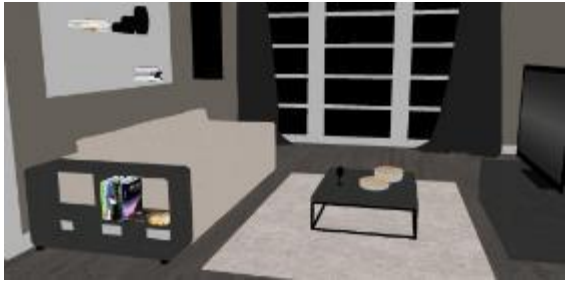
Position



Material



Normal



Reflectance

Deprecated OpenGL

- Modern GL does not do some things old GL did:
  - `glVertex` / `glColor` / …
  - `glMatrix` & Matrix stack
  - `glLight`
  - `glMaterial`
  - `GL_QUAD` / `GL_POLYGON`
- Will make many tutorials or courses out there break

Conclusion

- OpenGL is an specification
- Implements the rasterization pipeline we know
- Allows to configure many stages
- Allows to program some stages
- It turns primitives from buffers into a frame buffer using other buffers and programs