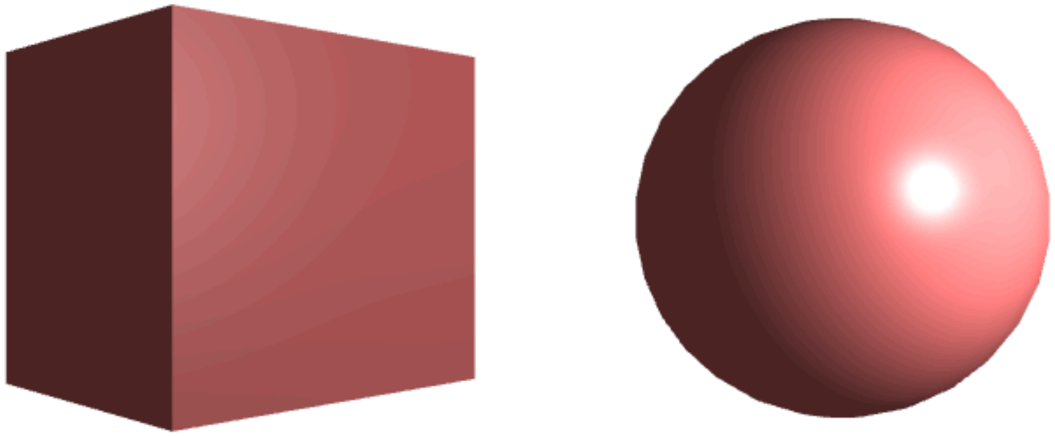


Computer Graphics

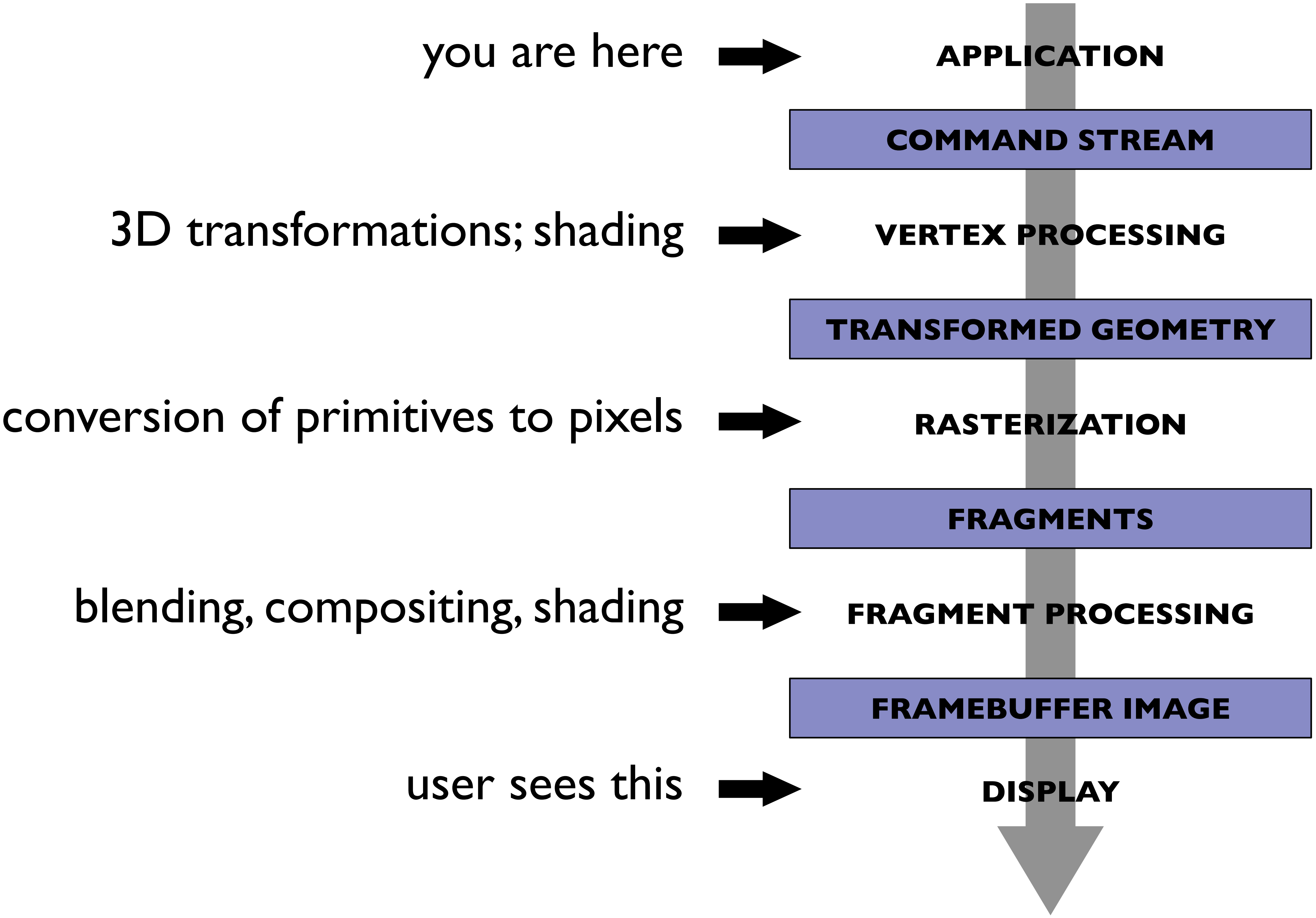
Lecture 24

OpenGL Lab: Data Plumbing

Announcements

- HW2 grades are out
- FP group formation due today; proposal due Friday
    - if you still don't have a group, hang around after class
- Midterm out Friday, due Tuesday at the start of class.
- Tuesday is a "lab day" again, bring a laptop if you can (we'll use Julia on Monday)
- A2 artifacts are posted and voting is open! Vote by Monday night and we'll showcase the winners on Tuesday.

Graphics Pipeline: Overview



you are here → APPLICATION

COMMAND STREAM

3D transformations; shading → VERTEX PROCESSING

TRANSFORMED GEOMETRY

conversion of primitives to pixels → RASTERIZATION

FRAGMENTS

blending, compositing, shading → FRAGMENT PROCESSING

FRAMEBUFFER IMAGE

user sees this → DISPLAY

OpenGL: Your job, conceptually

(send geometry)

- Send buffers full of data to GPU up front.
- Tell GL how to interpret them (triangles, ...)

(write vertex shader)

- GL executes custom-written **vertex shader** program on each vertex (to determine its location in **clip space**)

= normalized device coordinates

- GL **rasterizes** primitives into pixel-shaped **fragments**

(write fragment shader)

- GL executes custom-written **fragment shader** program on each fragment to determine its color.
- GL writes fragment colors to framebuffer pixels; neat things appear on your screen.

Terminology, so far

- Clipping
- Rasterization
- Interpolation
- Fragment
- Shader

WebGL: Your Jobs

- Send geometry
- Write a vertex shader
- Write a fragment shader

WebGL: Your Jobs

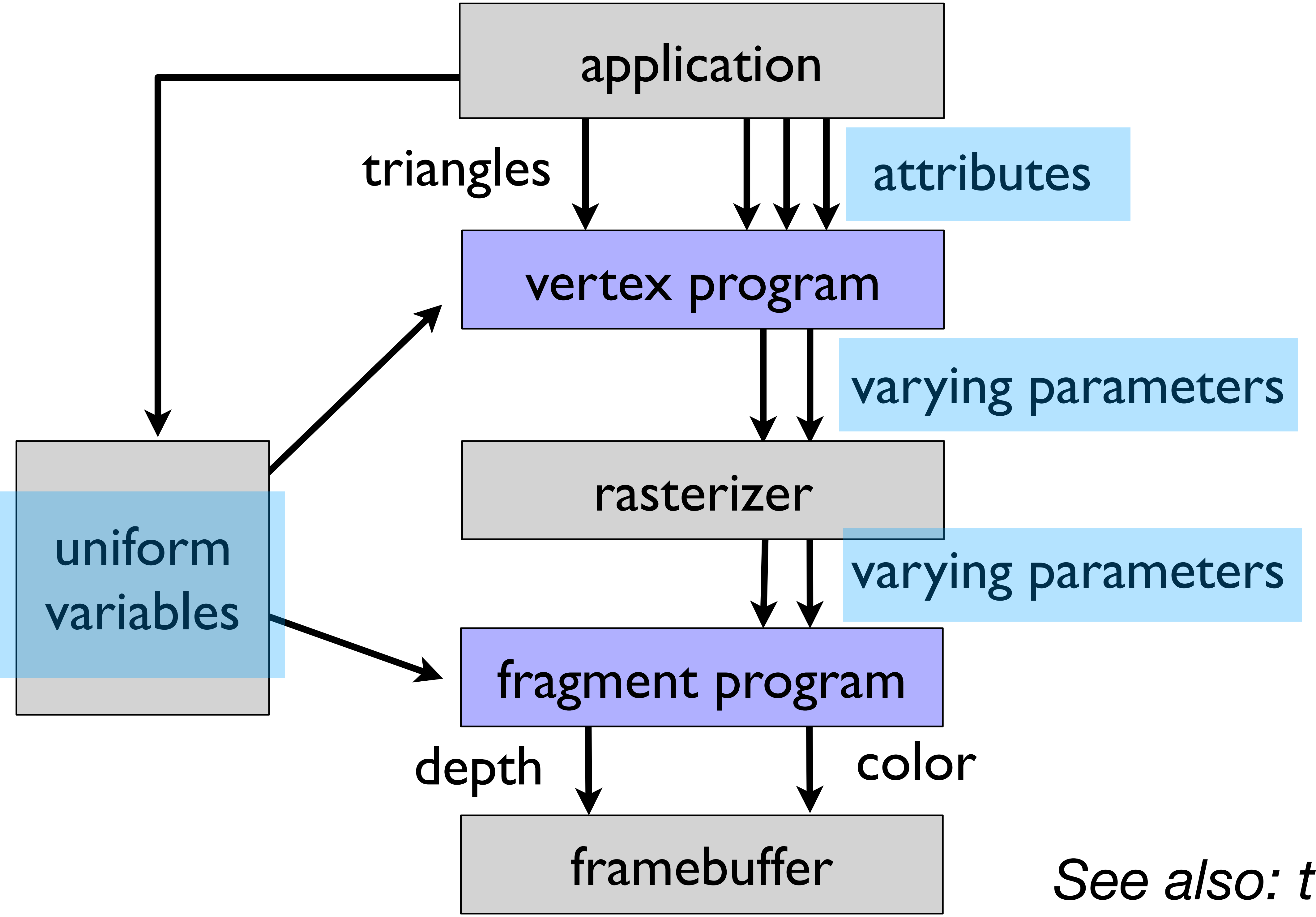
- Send geometry by calling `gl` functions
- Write a vertex shader
- Write a fragment shader

WebGL: Your Jobs

- Send geometry by calling `gl` functions
- Write a vertex shader
- Write a fragment shader

in **GLSL**, the GL shader language

WebGL Data Plumbing: Overview



See also: today's lecture notes

WebGL: Hello, Triangle!

- Send geometry by calling `gl` functions
- Write a vertex shader
- Write a fragment shader

in **GLSL**, the GL shader language

WebGL: Hello, Triangle!

- Send geometry by calling `gl` functions
- Write a vertex shader
- Write a fragment shader

in **GLSL**, the GL shader language

WebGL: Hello, Triangle!

- Send geometry by calling `gl` functions
- Write a vertex shader
- Write a fragment shader

in **GLSL**, the GL shader language

A first pass at the lab code...

WebGL: Hello, Triangle!

- Send geometry by calling `gl` functions
- Write a vertex shader
- Write a fragment shader

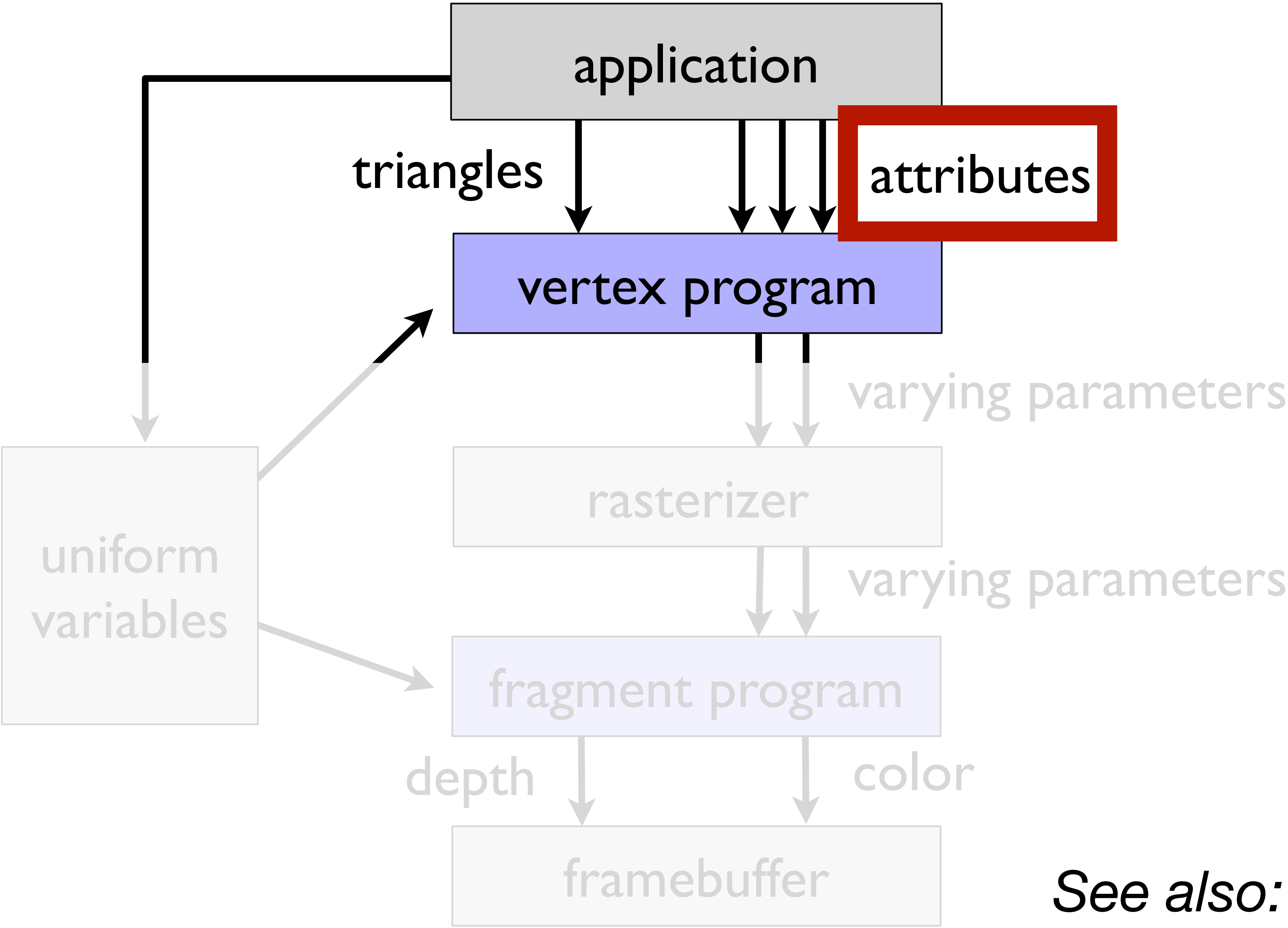
in **GLSL**, the GL shader language

A first pass at the lab code...

okay so we saw some unfamiliar words in there:

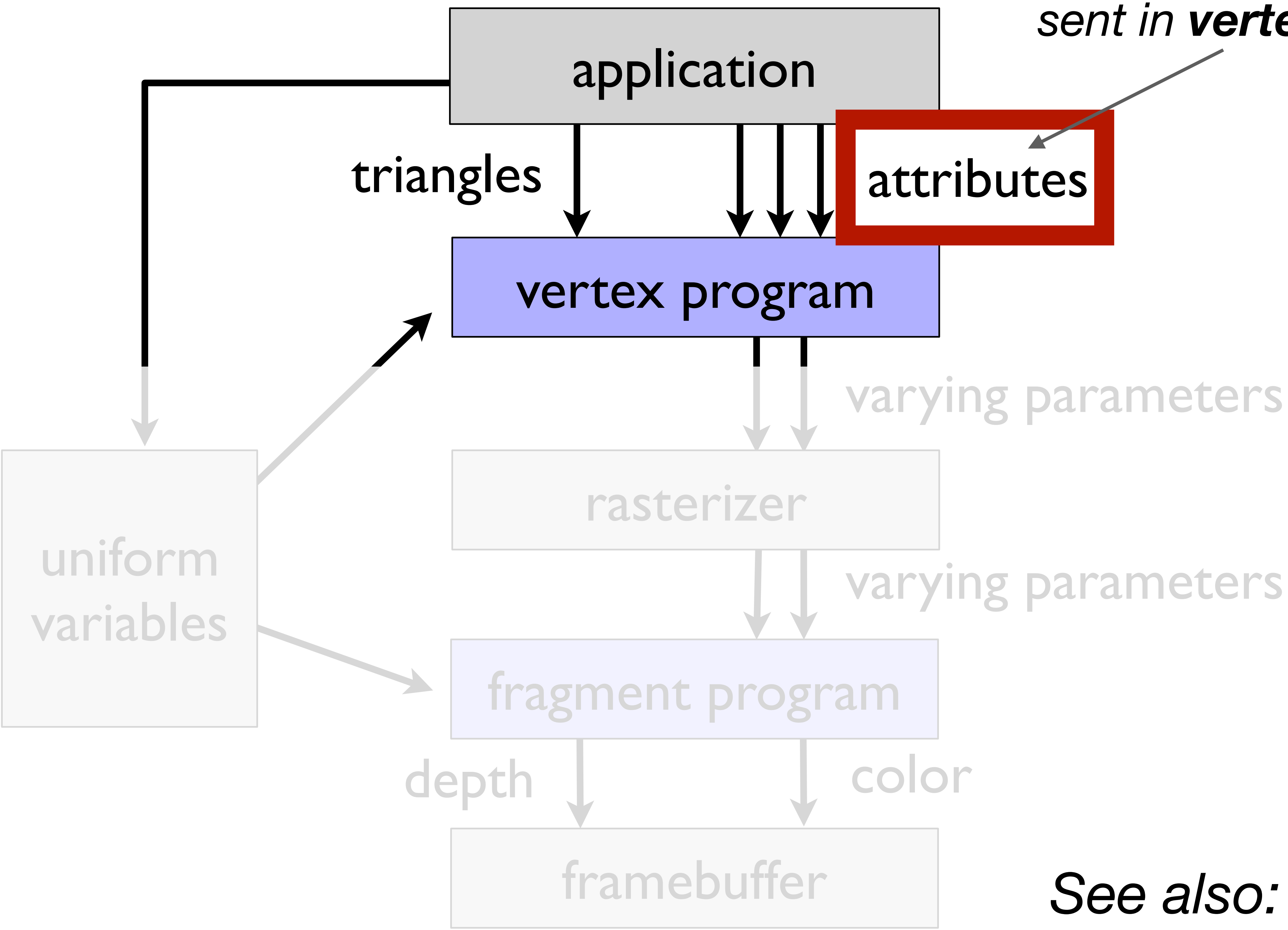
buffer
attribute

WebGL Data Plumbing



See also: today's lecture notes

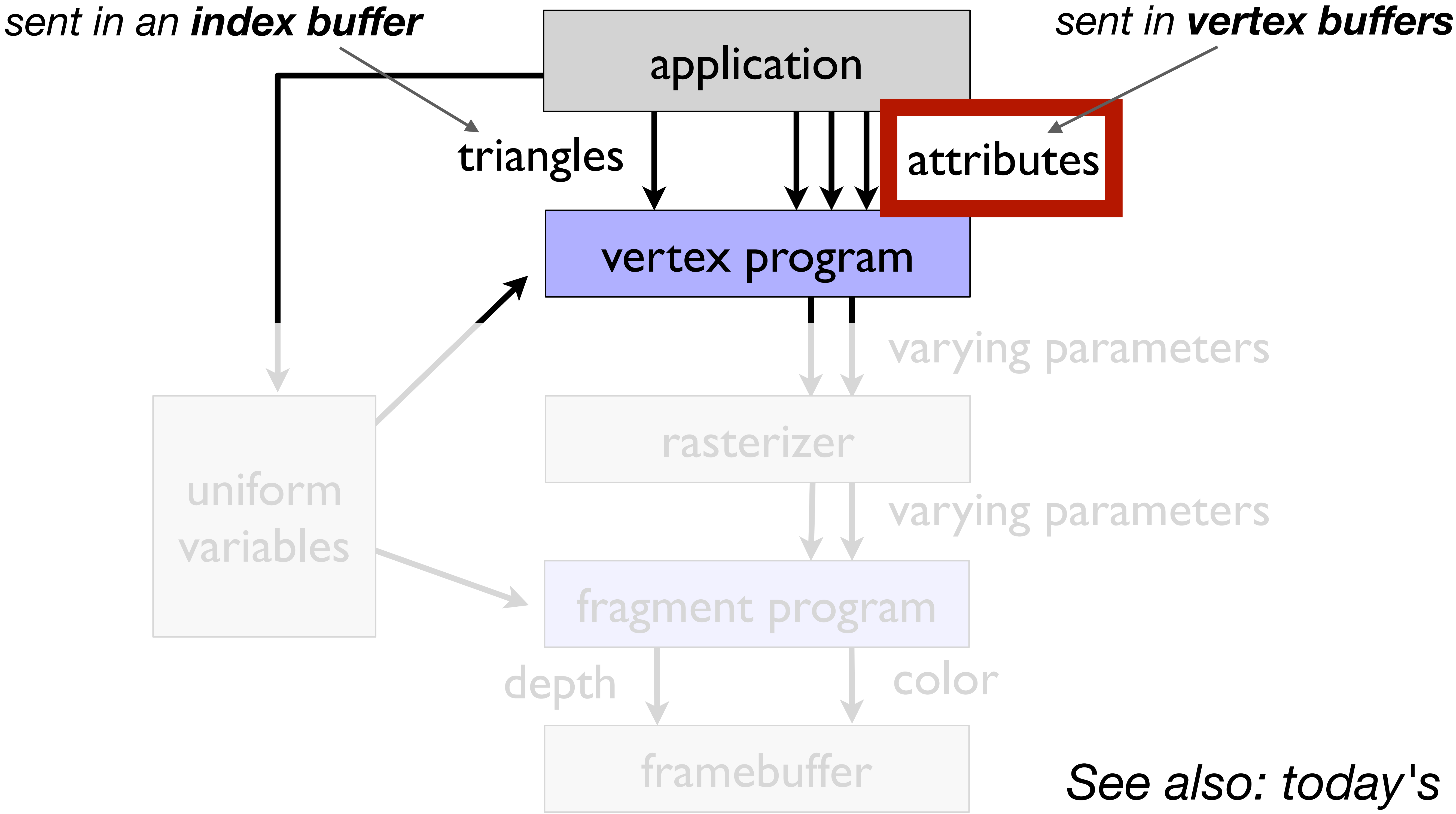
WebGL Data Plumbing



*sent in **vertex buffers***

See also: today's lecture notes

WebGL Data Plumbing



See also: today's lecture notes

WebGL: Hello, Triangle!

- Send geometry by calling `gl` functions
- Write a vertex shader
- Write a fragment shader

in **GLSL**, the GL shader language

WebGL: Hello, Triangle!

- Send geometry by calling `gl` functions

- Write a vertex shader
- Write a fragment shader

in **GLSL**, the GL shader language

WebGL: Hello, Triangle!

- Send geometry by calling `gl` functions

- Write a vertex shader
- Write a fragment shader

in **GLSL**, the GL shader language

A first look at the shader code...

Shader Responsibilities

The **vertex shader's job** is to:

- assign a value to **`gl_Position`**, which specifies the vertex's position
- assign values to any **varying** parameters needed later

The **fragment shader's job** is to:

- assign a value to **`gl_FragColor`***, which specifies the fragment's color

*deprecated in webgl2 (which uses GLSL 3.0), but not in webgl1

GLSL - GL Shader Language

- A C-like mini-language
- Basic program looks like:

```
// some declarations

void main() {
    // main program
}
```

- Built-in types for small vectors/matrices (e.g., `vec3`, `mat4`)

Task 1: Turn the triangle black

- Change the fragment shader's source code to set the triangle color to black instead of white.

- *Note*: colors are `vec4`s; the 4th channel is transparency ("alpha"):

    - 0.0 is fully transparent, 1.0 is fully opaque

Shader Responsibilities

The **vertex shader's job** is to:

- assign a value to **`gl_Position`**, which specifies the vertex's position
- assign values to any **varying** parameters needed later

The **fragment shader's job** is to:

- assign a value to **`gl_FragColor`***, which specifies the fragment's color

*deprecated in webgl2 (which uses GLSL 3.0), but not in webgl1

Shader Responsibilities

The **vertex shader's job** is to:

- assign a value to **`gl_Position`**, which specifies the vertex's position
- assign values to any **varying** parameters needed later

Lab code so far:

```
gl_Position = vec4(Position, 1.0)
```

The **fragment shader's job** is to:

- assign a value to **`gl_FragColor`***, which specifies the fragment's color

*deprecated in webgl2 (which uses GLSL 3.0), but not in webgl1

Shader Responsibilities

The **vertex shader's job** is to:

- assign a value to **`gl_Position`**, which specifies the vertex's position
- assign values to any **varying** parameters needed later

Lab code so far:

```
gl_Position = vec4(Position, 1.0)
```

The **fragment shader's job** is to:

- assign a value to **`gl_FragColor`**[*], which specifies the fragment's color

Lab code so far:

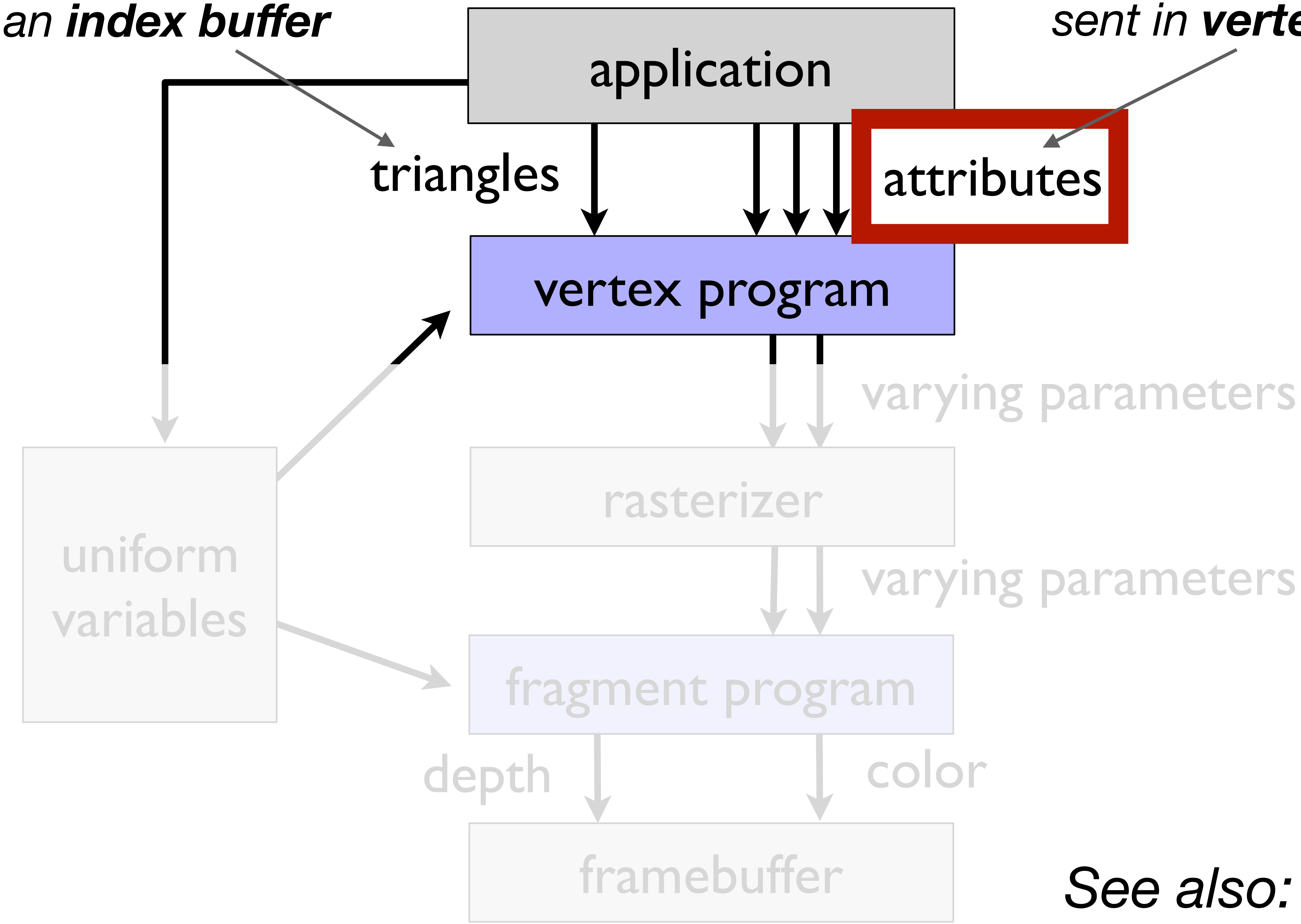
```
gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0)
```

[*] deprecated in webgl2 (which uses GLSL 3.0), but not in webgl1

WebGL Data Plumbing

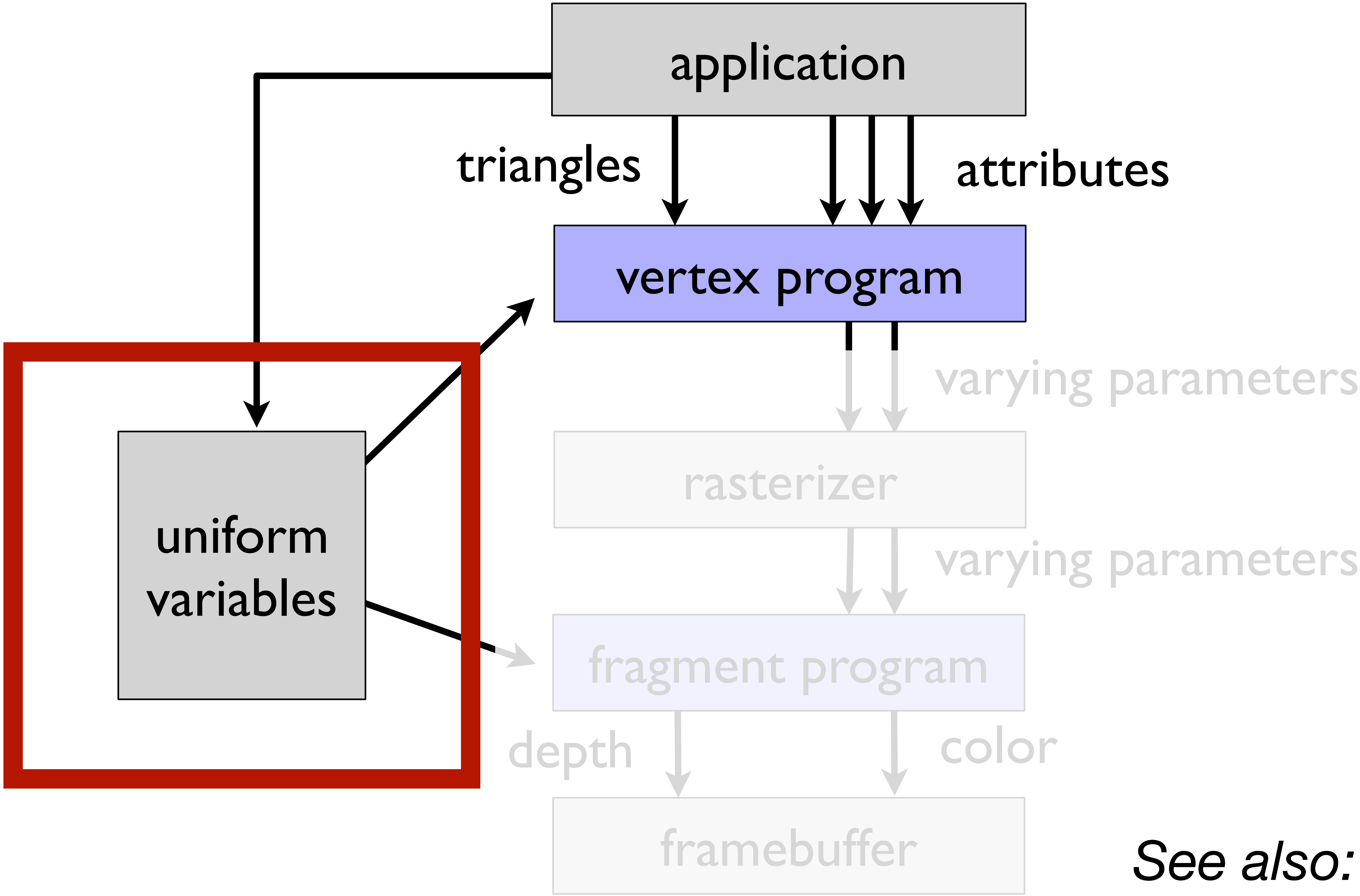
*sent in an **index buffer***

*sent in **vertex buffers***



See also: today's lecture notes

WebGL Data Plumbing



See also: today's lecture notes

GLSL - GL Shader Language

- Built-in types for small vectors/matrices (e.g., `vec3`, `mat4`). They have friendly constructors:

```
vec3 a = vec3(1.0, 1.0, 1.0)
vec4 b = vec4(a, 1.0)
```

- Multiplication does matrix multiplication:

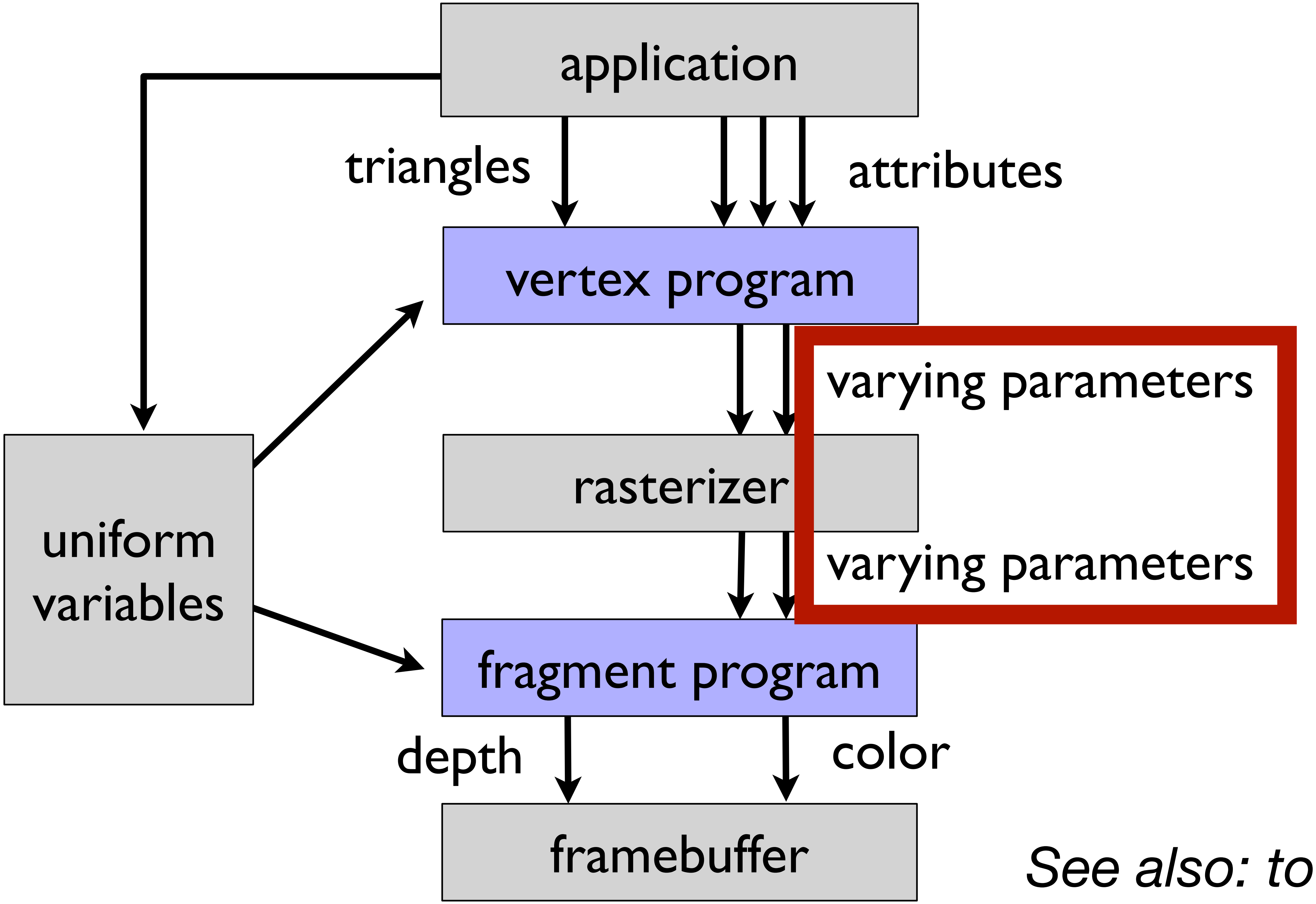
```
// GL matrices are in column-major order
mat2 A = mat2(1.0, 2.0, 3.0, 4.0);
vec2 x = vec2(1.0, 0.0);

vec2 a = A * x; // a = (1,2)
```

Task 2: Add a uniform

- Add a uniform variable called `Matrix` containing a 4x4 matrix
- In the vertex shader, multiply the `Position` attribute of the vertex by the `Matrix` to move the triangle vertices.

Terminology: data plumbing



See also: today's lecture notes

GLSL - GL Shader Language

- `varyings` are declared in both the Vertex shader and in the Fragment shader.
    - The vertex shader sets their values for each vertex, then the rasterizer **interpolates** their values for each fragment and passes to the fragment shader.
- By convention, `varying` names are usually chosen to begin with `v`, such as `vColor` or `vNormal`

Task 3: Add a varying

- Set up a `varying` parameter to set the color at each vertex
- Use the interpolated values in the fragment shader to set each fragment's color.