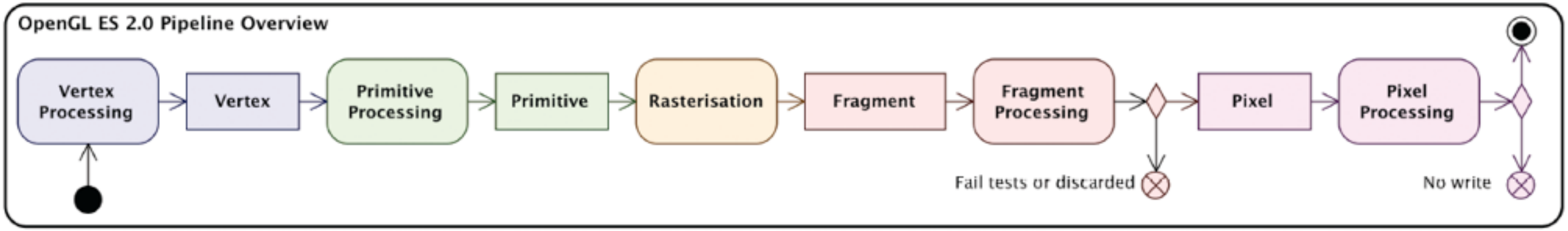
Computer Graphics

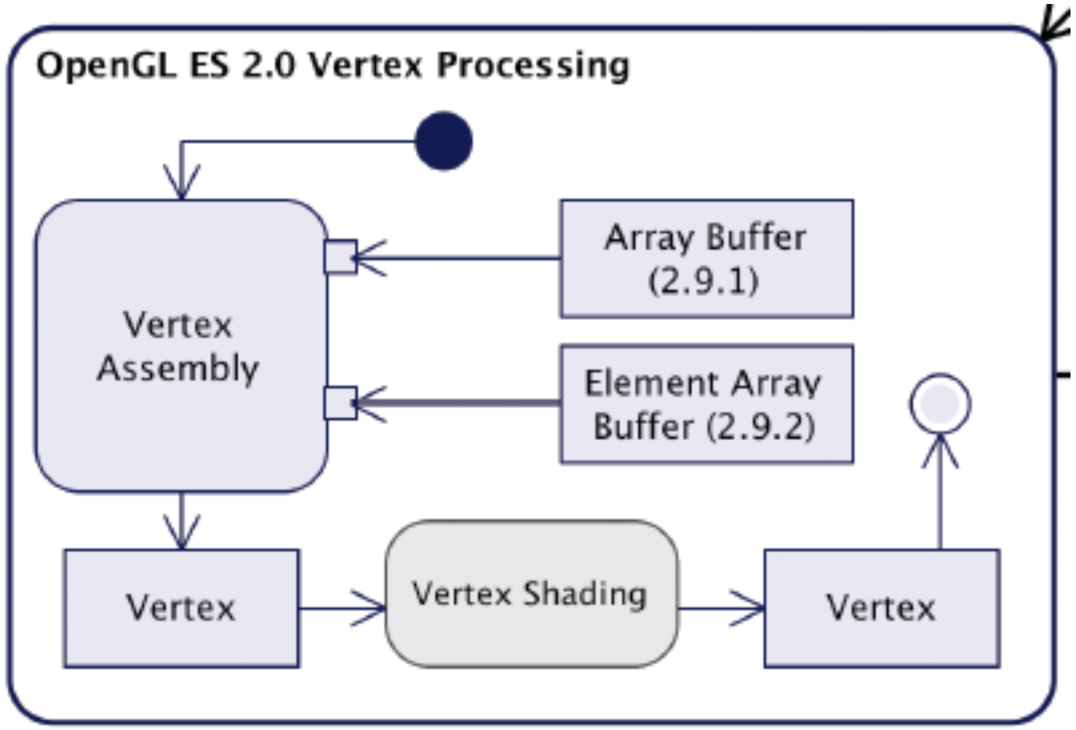
spring, 2013

OpenGL ES Pipeline

- OpenGL Pipeline Map by openglinsights.com (cross-referenced with GLES 2.0 spec)

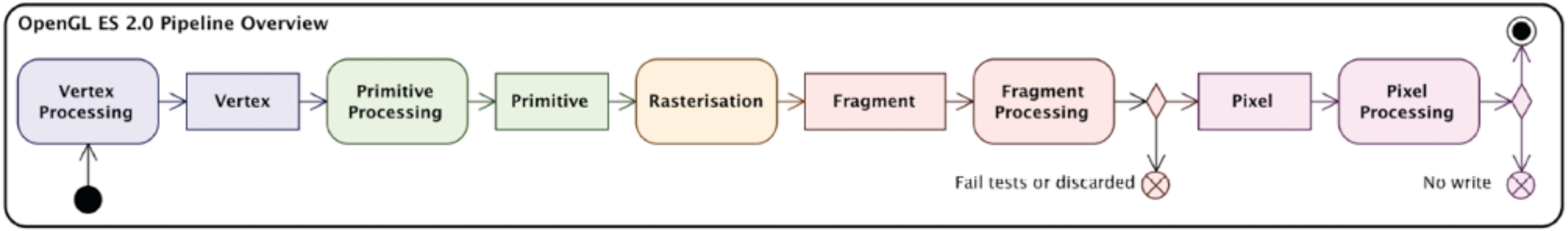


Vertex Processing

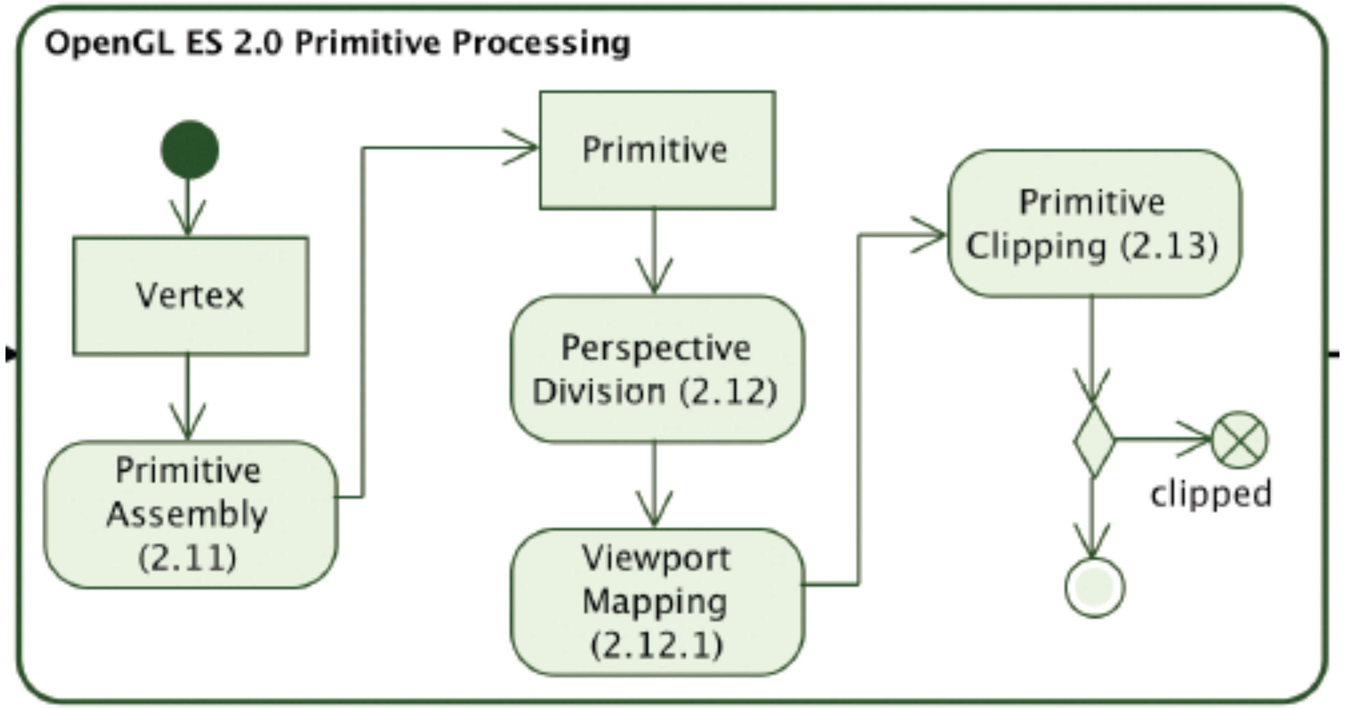


What are done in Vertex Shading?

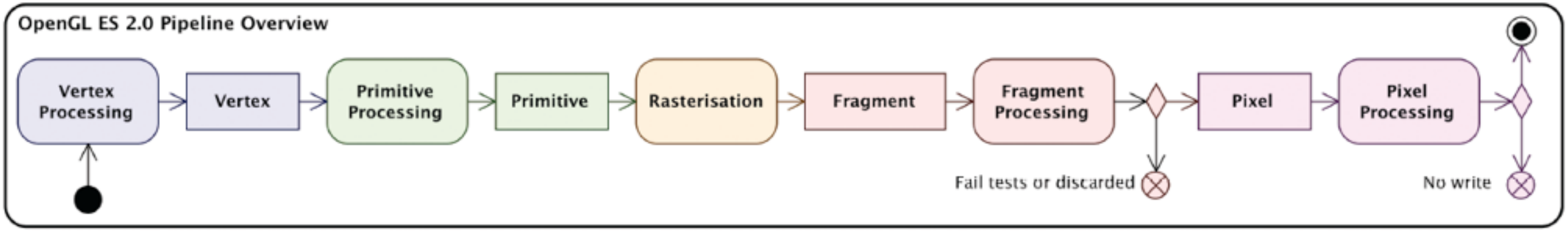
- Transformations (Object coords --> Clip coords)
- Shading (for Gouraud shading)
- Texture mapping (displacement mapping, etc.)
- ...And more



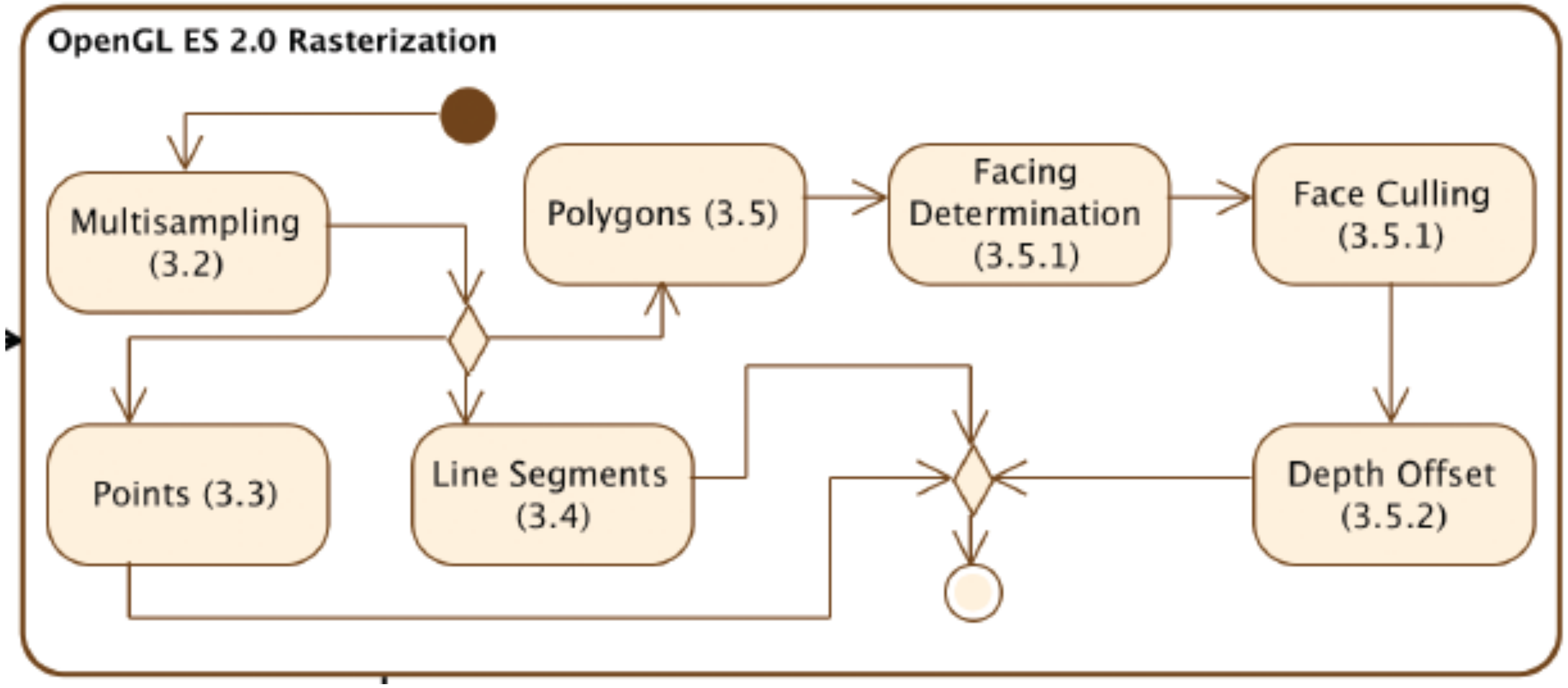
Primitive Processing


OpenGL ES 2.0 Primitive Processing
Vertex
Primitive Assembly (2.11)
Primitive
Perspective Division (2.12)
Viewport Mapping (2.12.1)
Primitive Clipping (2.13)
clipped

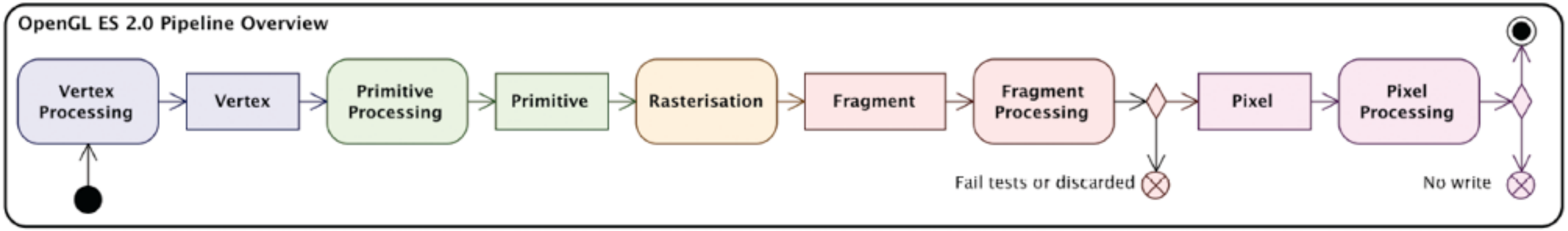
Error:
Primitive Clipping should be before Perspective Division!



Rasterization



Linear interpolation for varying variables (position, color, texcoords, normals, etc.)

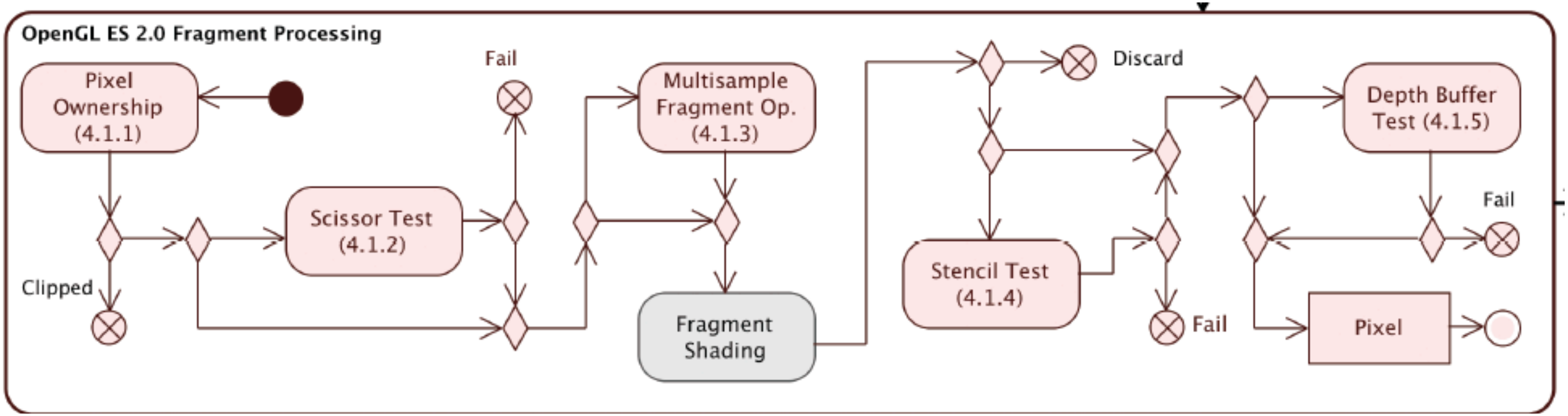


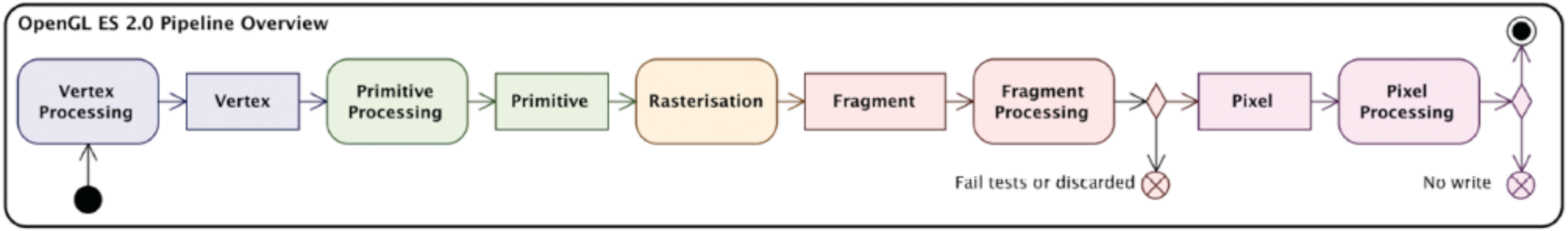
Fragment Processing

Error:
Fragment Shading is before Pixel Ownership

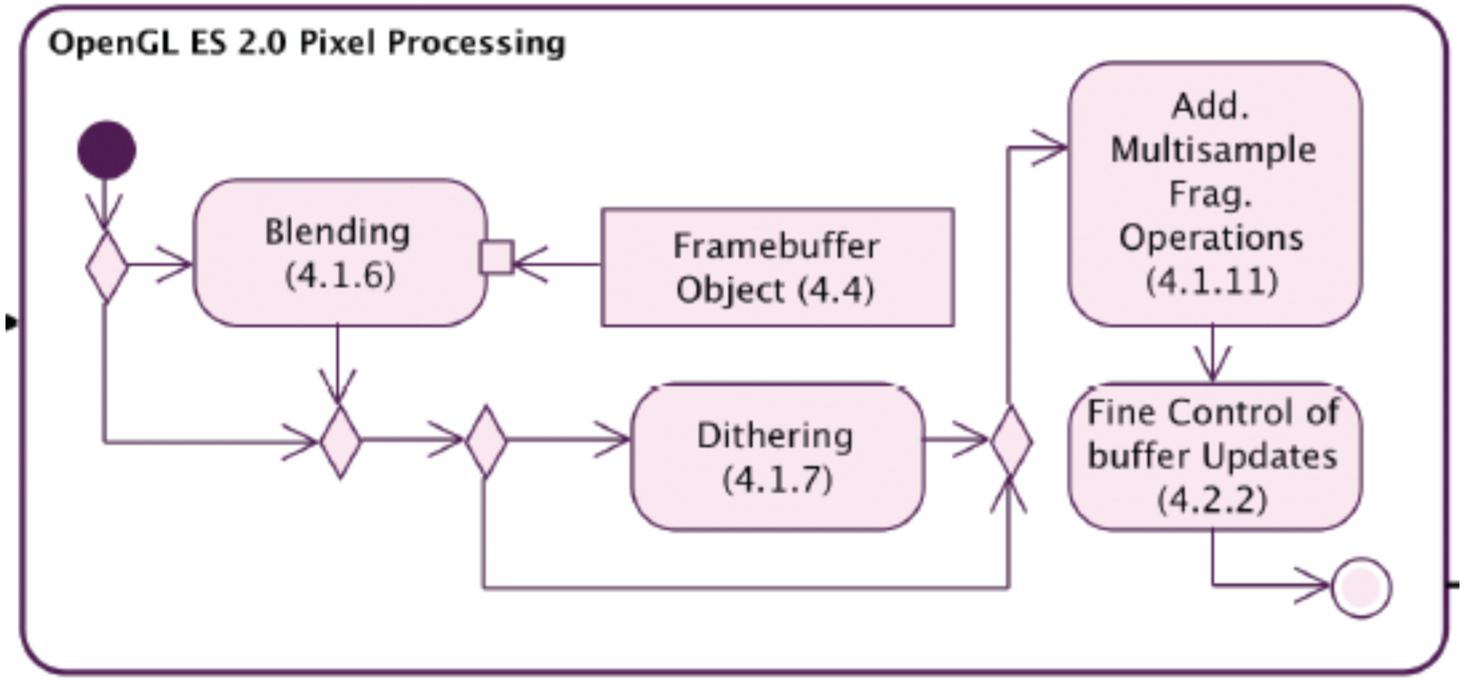
What are done in fragment shading?

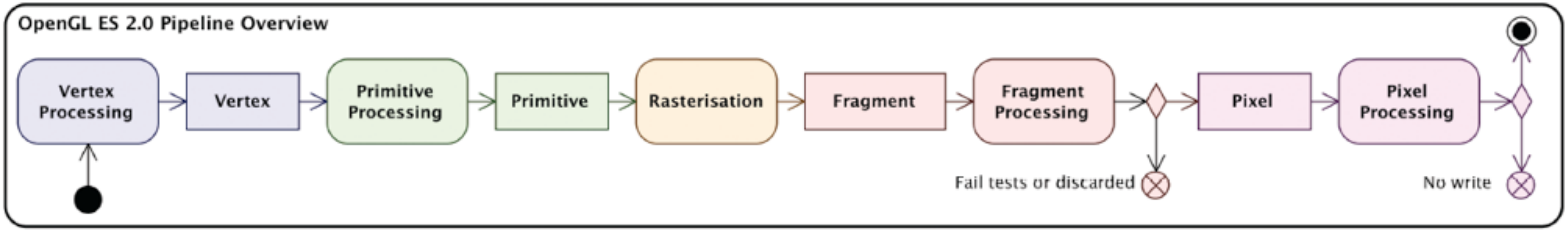
- Shading (for Phong shading)
- Texture mapping
- ...And more





Pixel Processing


OpenGL ES 2.0 Pixel Processing
Blending (4.1.6)
Framebuffer Object (4.4)
Add. Multisample Frag. Operations (4.1.11)
Dithering (4.1.7)
Fine Control of buffer Updates (4.2.2)



OpenGL ES

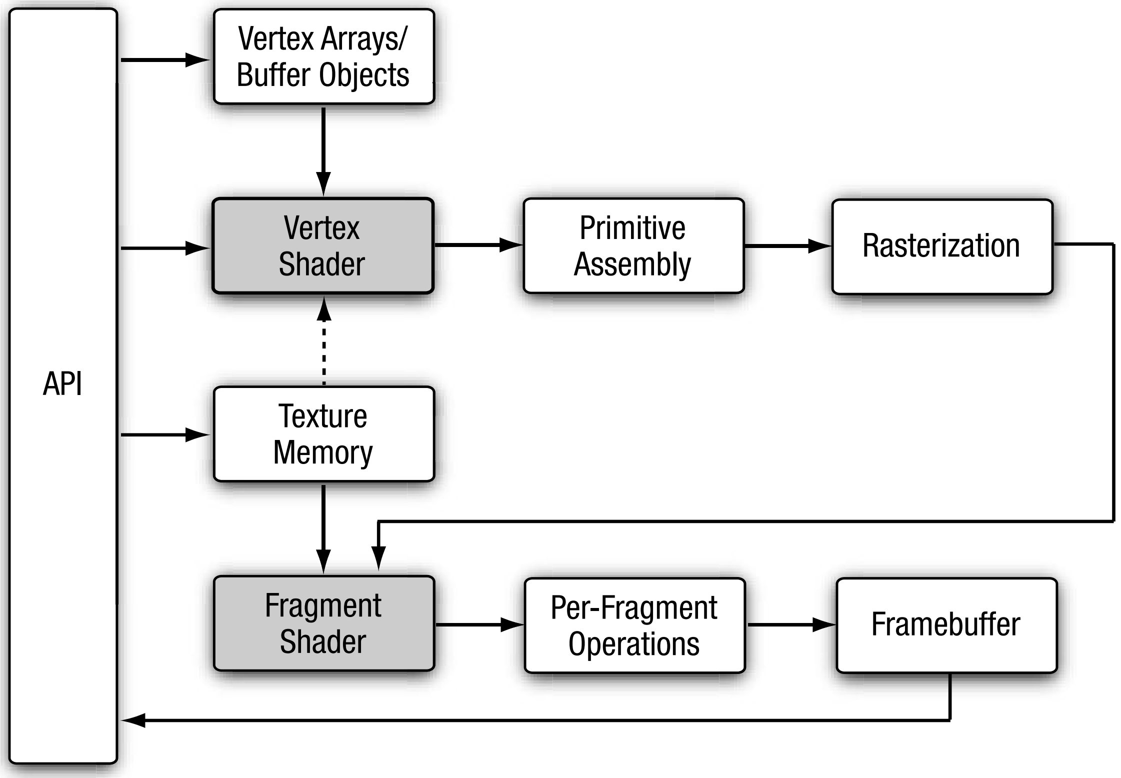
- Interactive graphics API for mobile devices
- Addresses hardware constraints such as
  - limited processing capabilities
  - limited memory availability
  - low memory bandwidth
  - sensitivity to power consumption
  - lack of floating-point hardware

Design Criteria

- Removal of redundancy
- Compatible with OpenGL --> Tricky for windows layer (EGL)
- New features for mobile HW (e.g. precision qualifiers)
- Minimum set of features for image quality
- Quality control -- conformance test

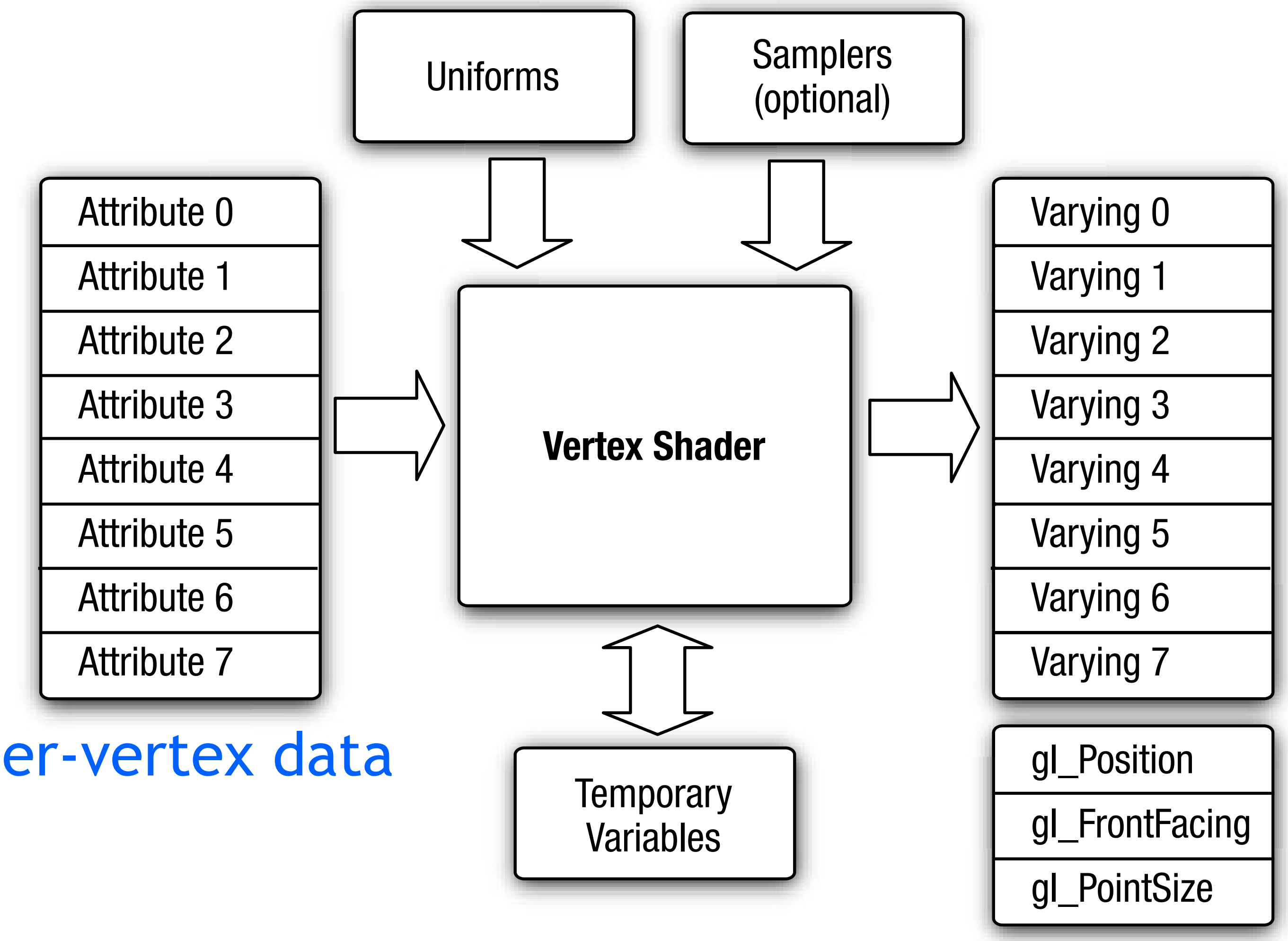
OpenGL ES 2.0

- Implements a graphics pipeline with programmable shading
- OpenGL ES 2.0 API spec. + OpenGL ES Shading Language spec.



Vertex Shader

- Implements a general purpose programmable method for operating on vertices



constant data

texture (global array)

output
- to be interpolated at the rasterization stage)
- input to the frag shader

per-vertex data

Vertex Shader (cont’d)

- Operations
  - Position transformation (object coords --> clip coords)
  - Shading (Gauroud shading)
  - Generating/transforming tex coords
  - ... and more

Example

```
// uniforms used by the vertex shader
```

P*V*M

```
uniform mat4   u_mvpMatrix; // matrix to convert P from model
                            // space to normalized device space.
```

Should be computed by the application --> requires matrix/vector library

```

// attributes input to the vertex shader
attribute vec4   a_position; // position value
attribute vec4   a_color;    // input vertex color

// varying variables - input to the fragment shader
varying vec4     v_color;    // output vertex color

void
main()
{
    v_color = a_color;
```

Color doesn't change

```
    gl_Position = u_mvpMatrix * a_position;
```

position transformation

```
}
```

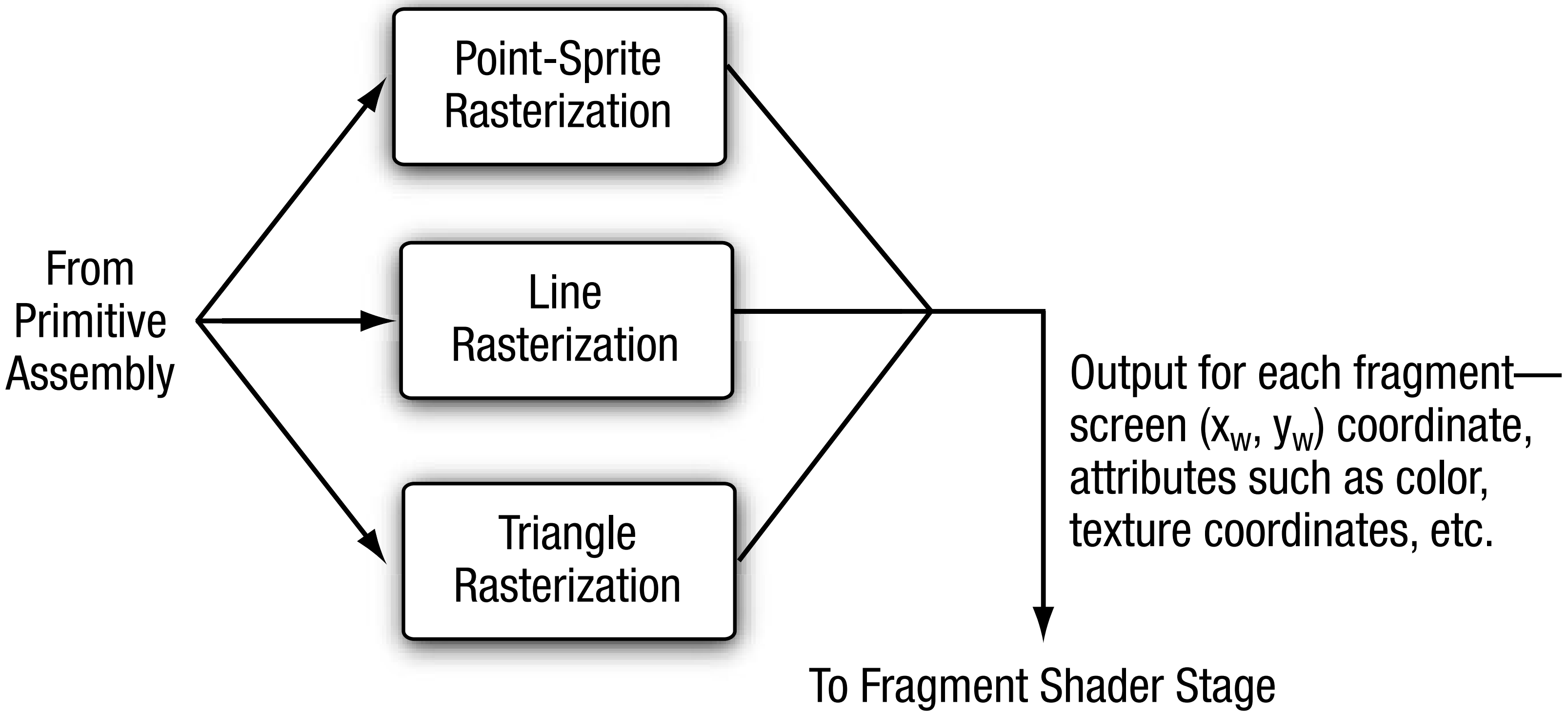
Built-in & mandatory

Primitive Assembly

- Geometric primitives: points, lines, triangles
- Clipping in eye coordinates
- error in the textbook: culling is done in rasterization stage, not here

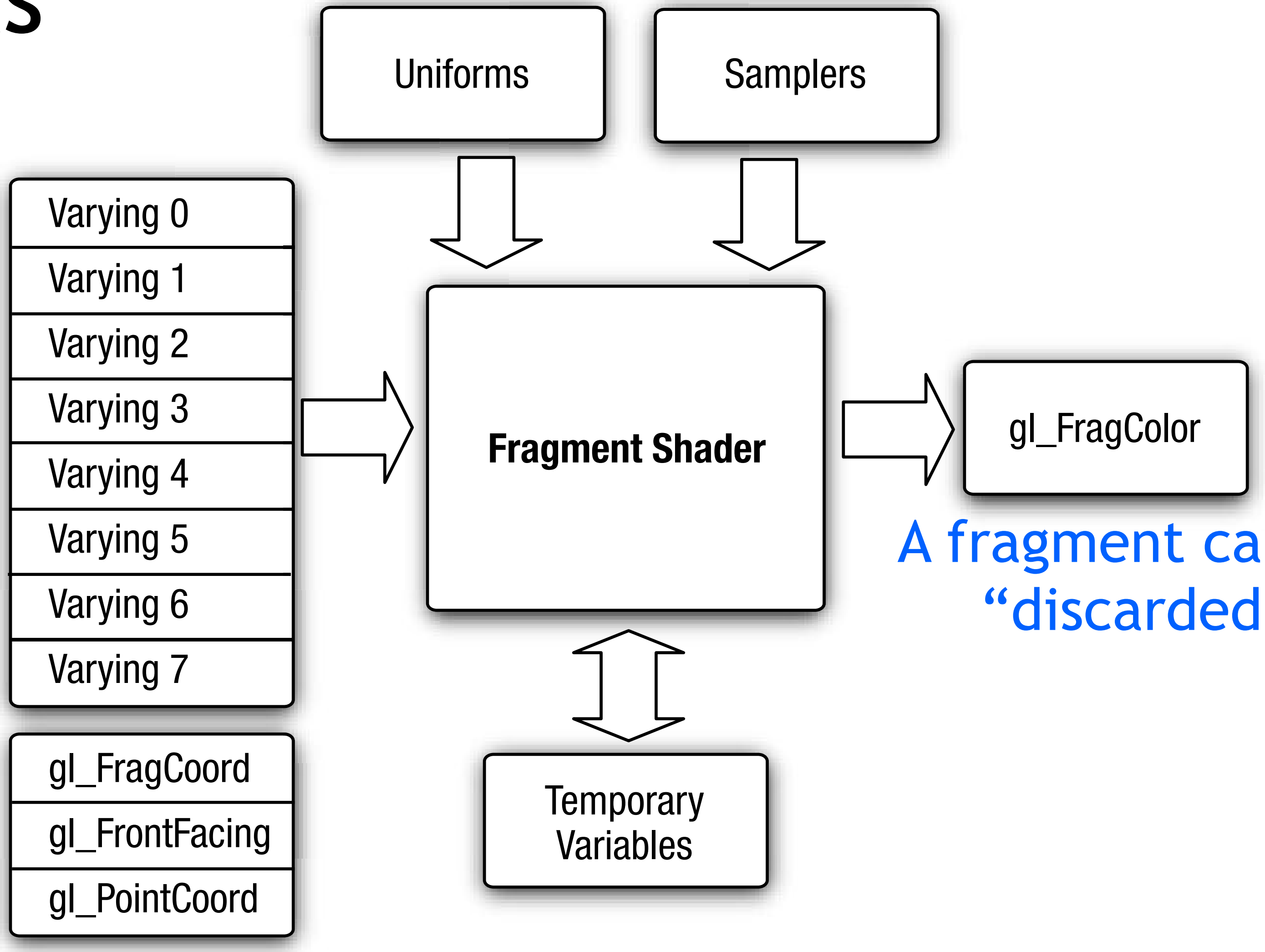
Rasterization

- Primitives converted to 2D fragments
- Face culling


From Primitive Assembly
Point-Sprite Rasterization
Line Rasterization
Triangle Rasterization
Output for each fragment—screen (x_w, y_w) coordinate, attributes such as color, texture coordinates, etc.
To Fragment Shader Stage

Fragment Shader

- Implements a general-purpose programmable method for operating on fragments



A fragment can be "discarded"

window coords (x,y,z,1/w)
true/false
tex coords for point sprites

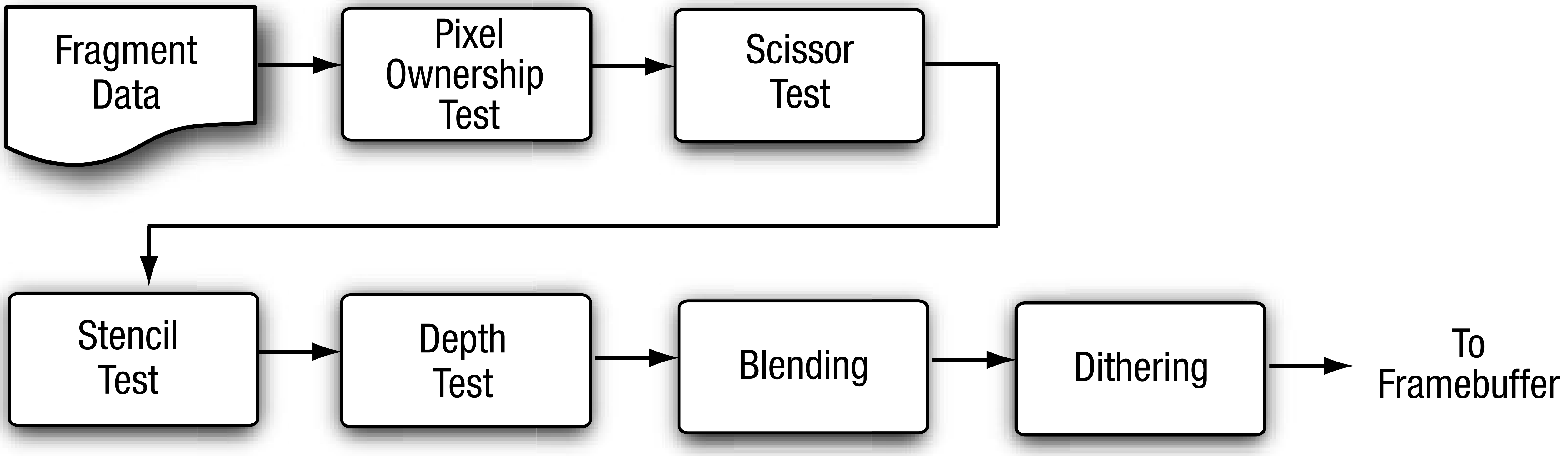
Fragment Shader (cont’d)

```
precision mediump float;
 
varying vec4   v_color;   // input vertex color from vertex shader
 
 
void
main(void)
{
    gl_FragColor = v_color;
}
```

precision qualifier

built-in

Per-Fragment Operations



Fragment Data

Pixel Ownership Test

Scissor Test

Stencil Test

Depth Test

Blending

Dithering

To Framebuffer

Per-Fragment Operations (cont’d)

- At the end, either the fragment
  - is rejected or
  - a fragment color, depth, or stencil value is written to the framebuffer at (x_w, y_w)
- Pixels can be read back from the framebuffer (but not depth & stencil)
- Alpha test & LogicOp no longer in per-fragment operation stage
  - Alpha test can be performed in the f.s.
  - LogicOp is very infrequently used

Backward Compatibility

- No backward compatibility with OpenGL ES 1.x
  - To avoid redundancy
  - Most apps do not mix programmable & fixed function pipelines
  - To reduce driver size

EGL

- An interface between OpenGL ES & the native window system
- Core functions: to create rendering context & drawing surface
  - rendering context: set of states
  - drawing surface: where the primitives are drawn

Initial Steps Using EGL

1. Query & initialize a display
2. Create a rendering surface
   - on-screen surfaces: attached to the native window system
   - off-screen surfaces: pixel buffers not displayed
3. Create a rendering context and attach it to a surface

Programming with OpenGL ES 2.0

- Libraries & header files --> platform dependent
- EGL syntax: elg* for function names, EGL* for type names
- GLES syntax: gl* for function names, GL* for type names, postfixes for argument types

Error Handling

- Can be queried by glGetError
- No other errors will be recorded until the app has queried the 1st error code using glGetError

Flush and Finish

- Client-server model --> commands are buffered
- glFlush / glFinish --> empties the command bufer
- glFlush -- asynchronous
- glFinish -- synchronous --> slow

Basic State Management

- Turned on/off by glEnagle/glDisable
- Queried by glIsEnabled