

# WEBGL

*New Technology*

# WEBGL

---

- Javascript API
- based on OpenGL ES 2.0
- 3D Graphics in `<canvas>`
- GPU
- Low-level
- WebGL Libraries (three.js,...)

# INstantiate

.....



# INSTANTIATE WEBGL

---

- <canvas>
- Provide fallback

```
|  
  
window.onload = function () {  
    canvas = document.getElementById("canvas");  
  
    // Try to grab the standard context. If it fails, fallback to experimental.  
    var gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl")  
  
    // Only continue if WebGL is available and working  
    if (gl) {  
        // Set clear color to blue, fully opaque  
        gl.clearColor(0.2, 0.4, 0.67, 1.0);  
        // Clear the color as well as the depth buffer.  
        gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);  
    }  
};
```

# THE RENDERING PIPELINE

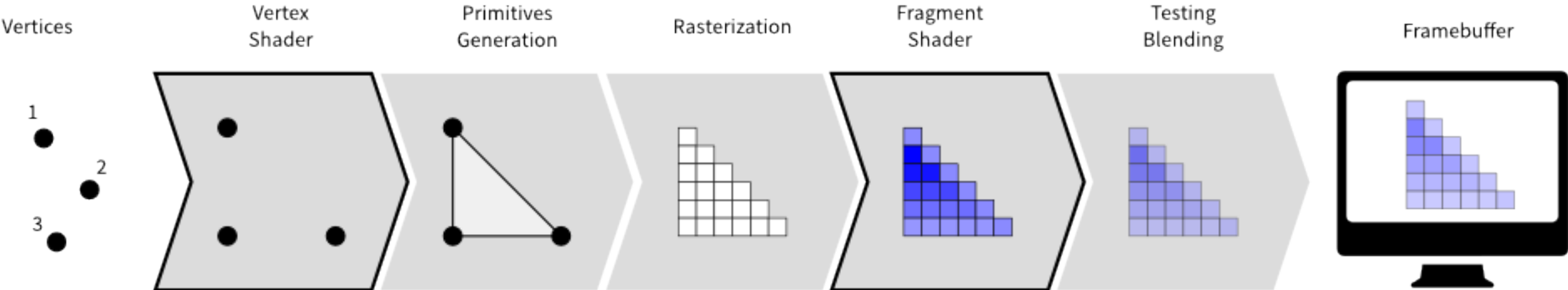
---

*from code to 3D*



# THE RENDERING PIPELINE

.....



# VERTICES

---

- Vertex array
- Attributes
- Created on CPU
- On GPU = **vertex buffers**
- Index array
- limit data transfer

```
// Create buffers
var boxVertices = [
  // X, Y, Z    // COLOR: R, G, B
  // Top
  -1.0, 1.0, -1.0,    0.5, 0.5, 0.5,
  -1.0, 1.0, 1.0,     0.5, 0.5, 0.5,
  1.0, 1.0, 1.0,      0.5, 0.5, 0.5,
  1.0, 1.0, -1.0,     0.5, 0.5, 0.5,

  // Left
  -1.0, 1.0, 1.0,     0.75, 0.25, 0.5,
  -1.0, -1.0, 1.0,    0.75, 0.25, 0.5,
  -1.0, -1.0, -1.0,   0.75, 0.25, 0.5,
  -1.0, 1.0, -1.0,    0.75, 0.25, 0.5,

  ...
]

// Create Buffer on GPU
var BoxVertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, BoxVertexBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(boxVertices), gl.STATIC_DRAW);
```

# VERTEX SHADER

---

- Use default or own vertex shaders
- Calculates position,...

```
attribute vec2 vertPosition;  
attribute vec3 vertColor;  
  
varying vec3 fragColor;  
  
void main() {  
    fragColor = vertColor;  
  
    gl_Position = vec4(vertPosition, 0.0, 1.0);  
}
```



# FRAGMENT SHADER

---

- After rasterizer (pixels)
- GLSL
- Outputs color, lighting, texture mapping

```
precision mediump float;
```

```
varying vec2 fragTexCoord;  
uniform sampler2D sampler;
```

```
void main() {  
    gl_FragColor = texture2D(sampler, fragTexCoord);  
}
```

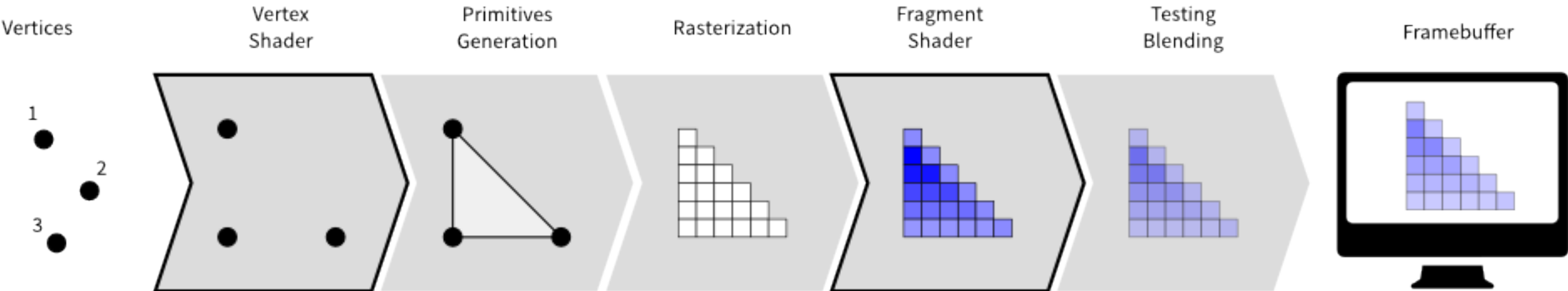
# FRAME BUFFER

---

- Final destination
- Screen

# THE RENDERING PIPELINE

.....



# SHADERS

---



# SHADERS

---

- Vertex shader and Fragment shader
- Functions
- Linked into program
- Typically several programs per app
- GLSL (Graphics Library Shader Language)

# VERTEX SHADERS

---

- clip-space coordinates
- Called once per vertex
- `gl_Position`
- Needs data
  - Attributes (pulled from buffers)
  - Uniforms (data that stays the same)
  - Textures (data from pixels)

# FRAGMENT SHADERS

---

- Provides color to pixel
- Called once per pixel
- `gl_FragColor`
- Needs data
  - Uniforms (data that stays the same)
  - Textures (data from pixels)
  - Varyings (data passed from the vertex shader)

# CODE

.....  
*example*

