Introduction to WebGL

CMSC 161: Interactive Computer Graphics

2nd Semester 2014-2015

Institute of Computer Science

University of the Philippines – Los Baños

Lecture by James Carlo Plaras

What is WebGL?

Web Graphics **L**ibrary

New standard for 3D graphics on the Web

What is WebGL?

Web Graphics **L**ibrary

HTML 5 family of technologies

What is WebGL?

Web Graphics **L**ibrary

Client based rendering using the client’s graphics hardware

Definitions

Rendering

process of generating image from a scene/model

Definitions

Software-based rendering

rendering that uses CPU

Definitions

Hardware-based rendering

rendering that uses GPU

Definitions

Server-based rendering

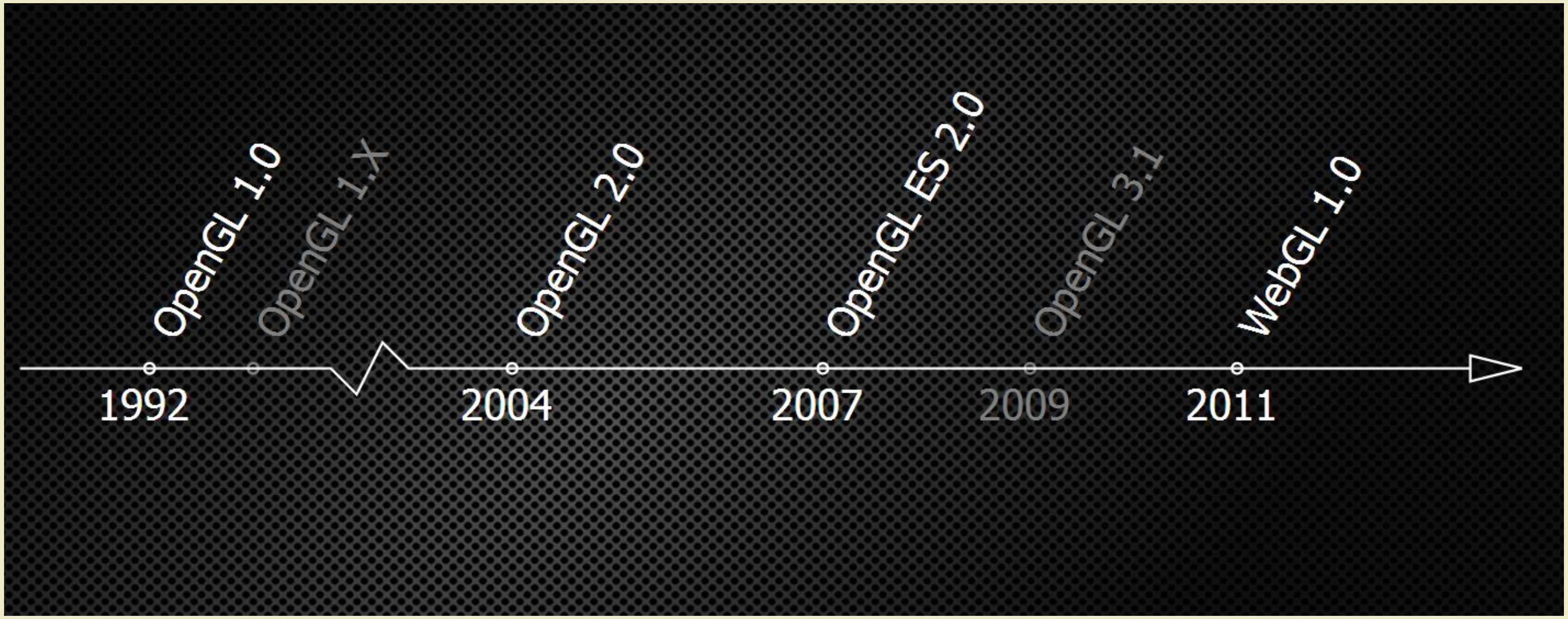
Server provides the scene that will be rendered in the client

Definitions

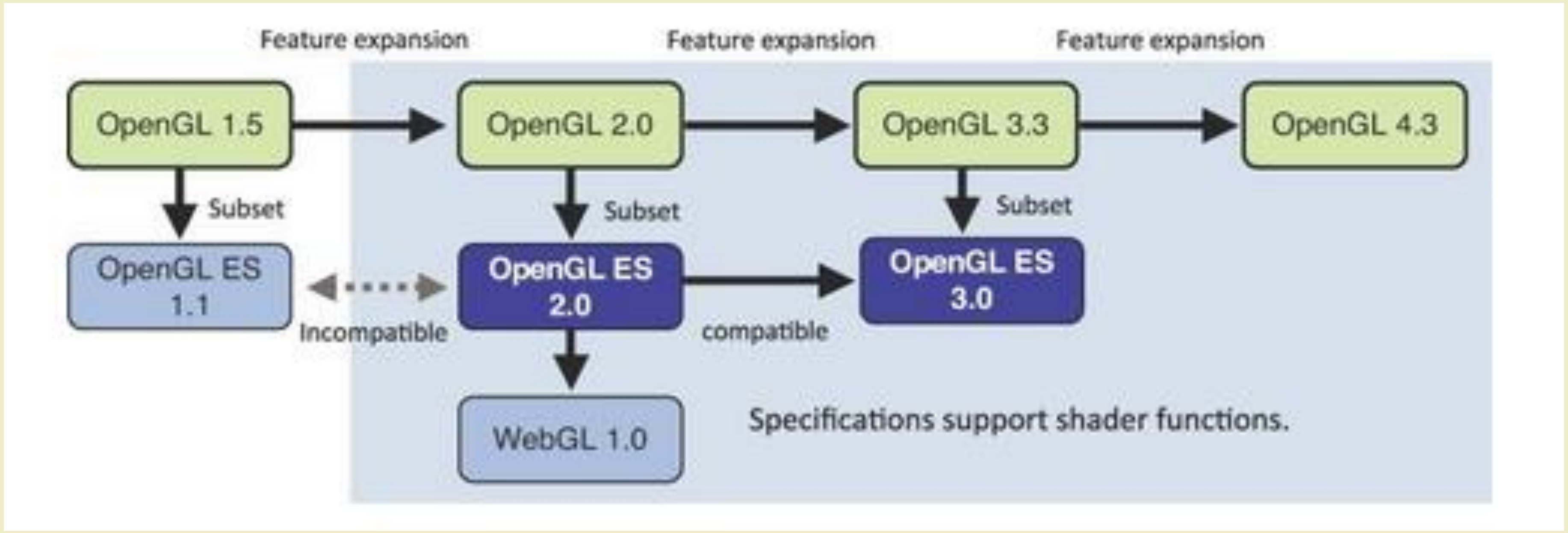
Server-based rendering

Server <u>renders</u> the scene before it is shown to the client

History of WebGL



OpenGL-OpenGL ES-WebGL Relationship



Advantages of WebGL

HTML/Javascript/Text Editor

Automatic memory management

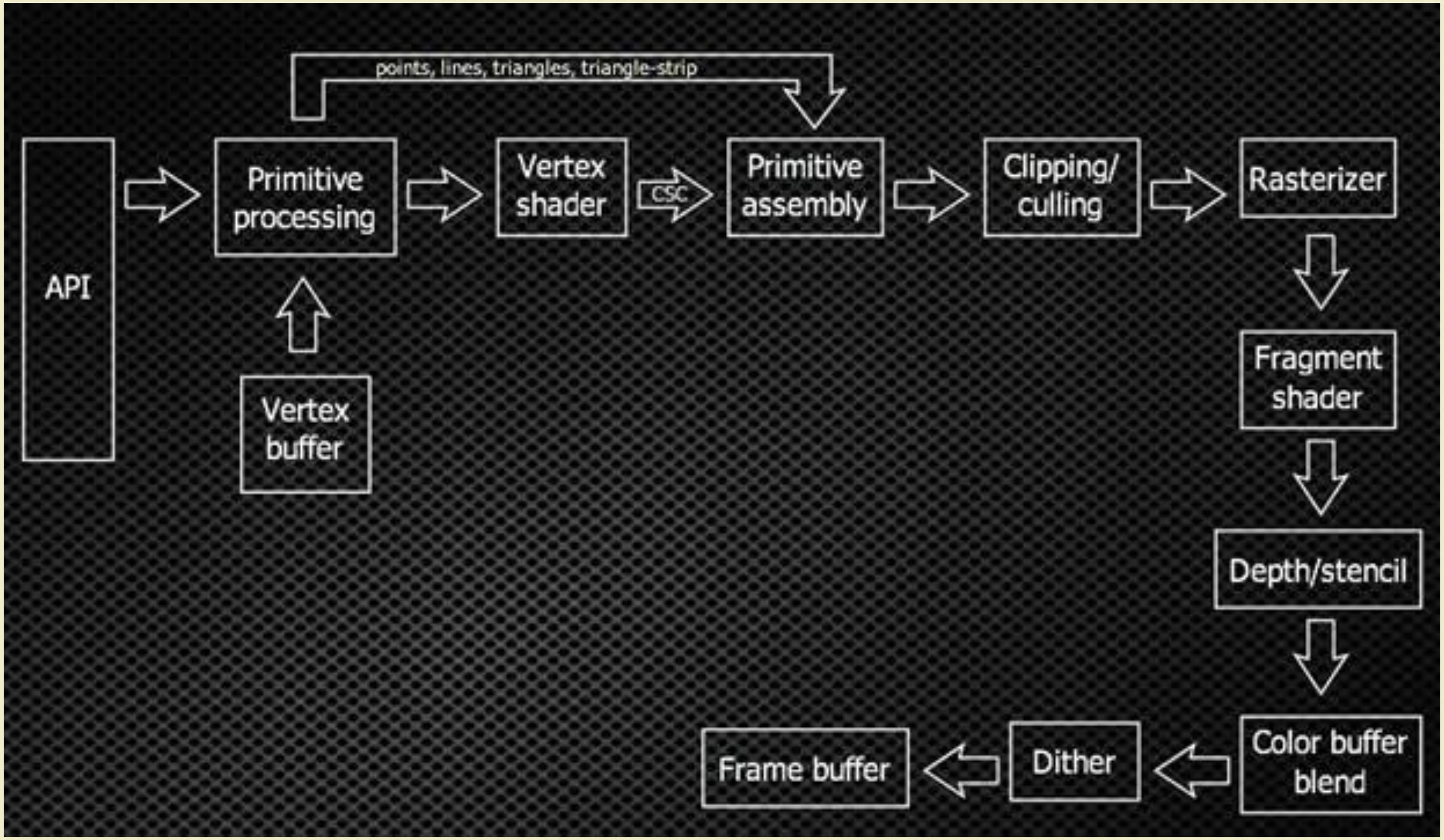
Pervasiveness (Cross Platform)

Performance

Zero-compilation

WEBGL PROGRAMMABLE PIPELINE

WebGL Programmable Pipeline



API and Vertex Buffer

Application Programming Interface (API)

Built in functions to communicate with the WebGL system

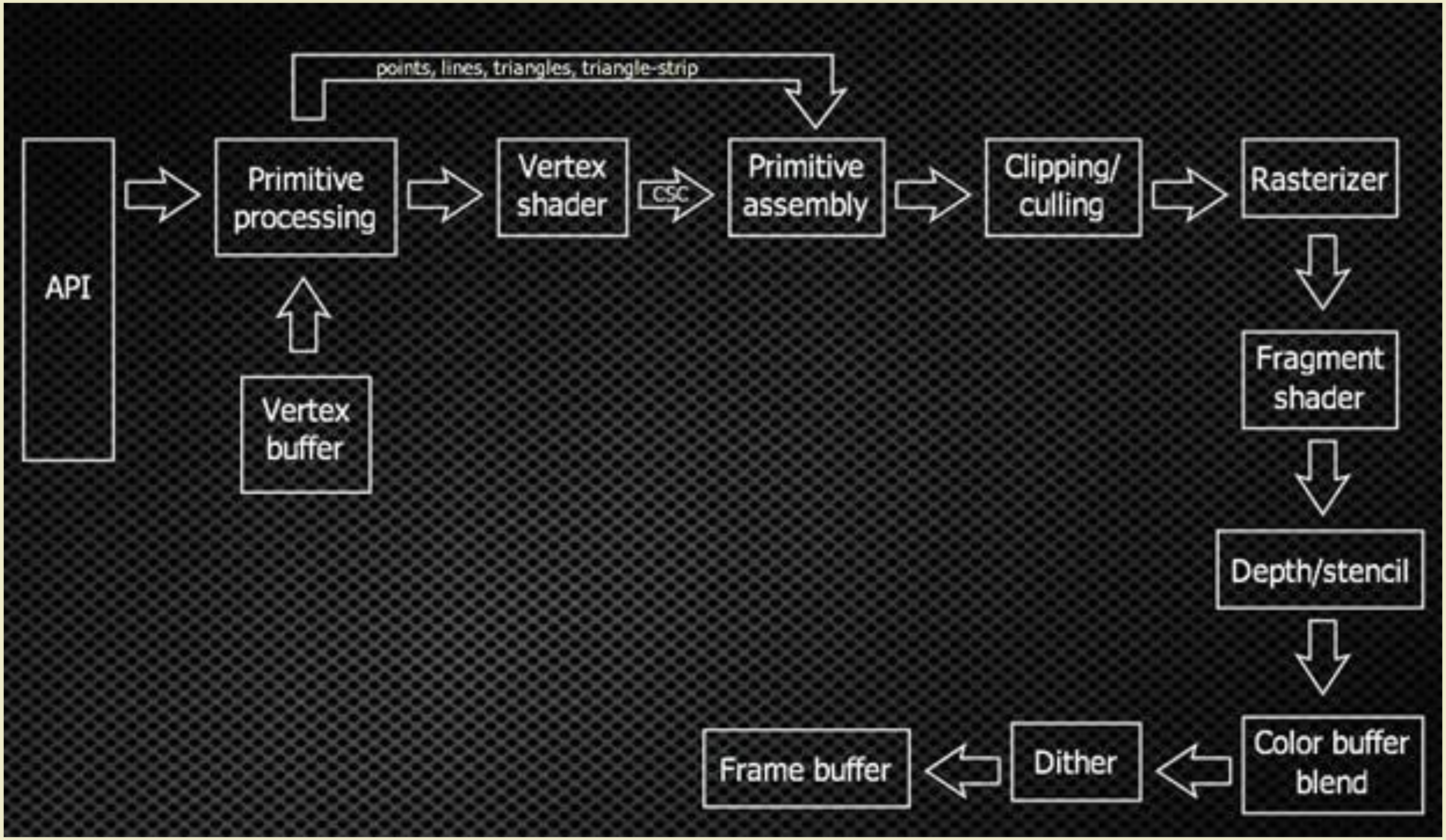
API and Vertex Buffer

Vertex Buffer

Contains the per-vertex data

Locations, vertex color, vertex size

WebGL Programmable Pipeline



Primitive Processing

In primitive processing...

Per-vertex information are passed into the

Vertex Shader

Primitive Processing

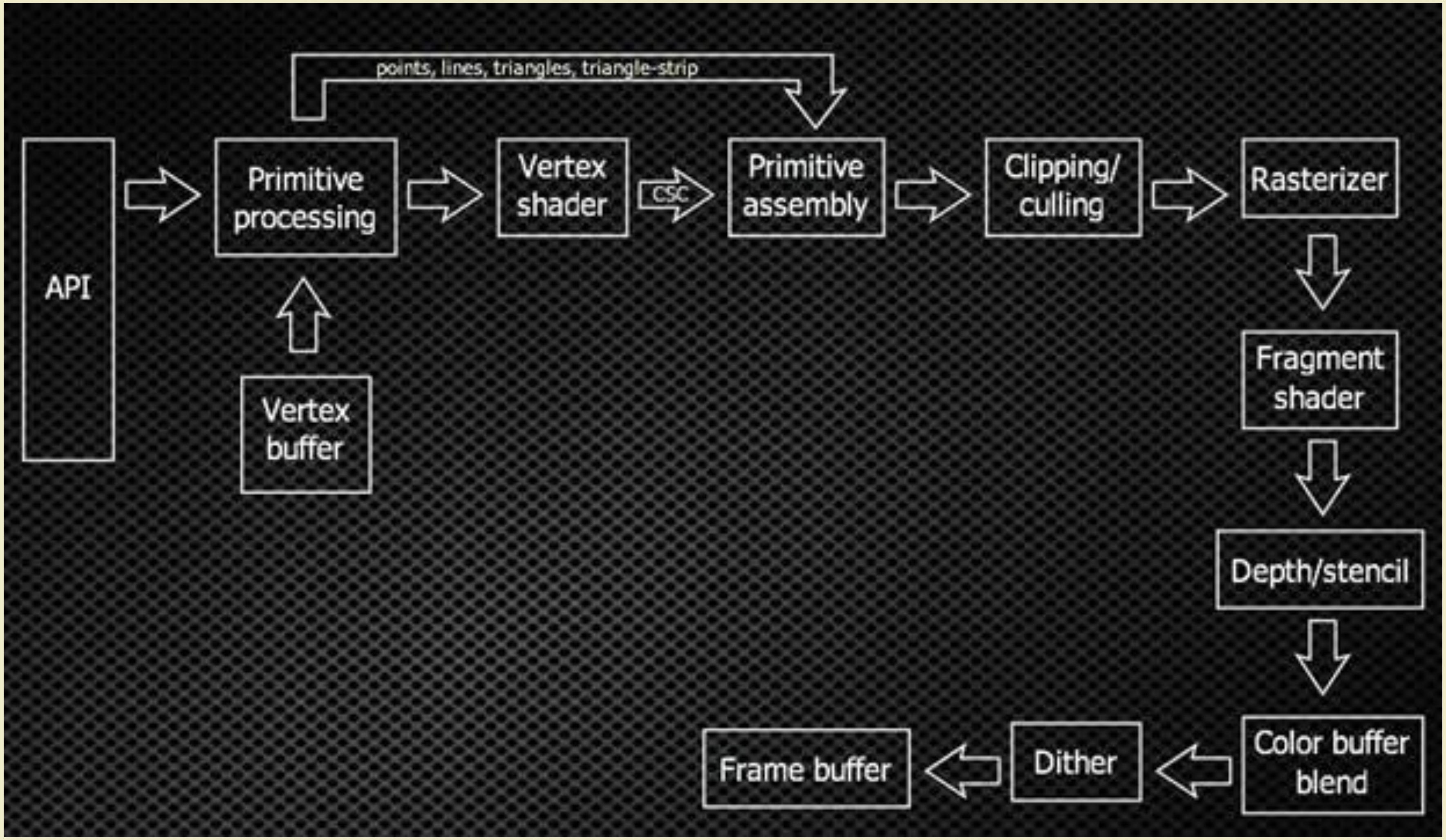
In primitive processing...

Primitives are specified using the WebGL API

and is passed to the **Primitive Assembly**

POINTS, LINES, TRIANGLES, TRIANGLE_FAN,...

WebGL Programmable Pipeline



What is a Shader?

Programs designed to run on the graphics processor (GPU)

What is a Shader?

WebGl uses **GLSL ES** for shader programs

OpenGL Shading Language Embedded Systems

WebGL Shaders

Written in GLSL

C-like code that runs in the GPU

WebGL Shaders

Utilizes three types of variables

Uniforms, Attributes, Varyings

Vertex Shader

Executed for every vertex to

Describe the traits of a vertex

(position, size, etc.)

Vertex Shader

Executed for every vertex to

Compute vertex related operations

(normal vectors, up-vectors,

transformations)

Vertex Shader

Executed for every vertex to

Pass results of vertex related operations

to the **primitive assembly**

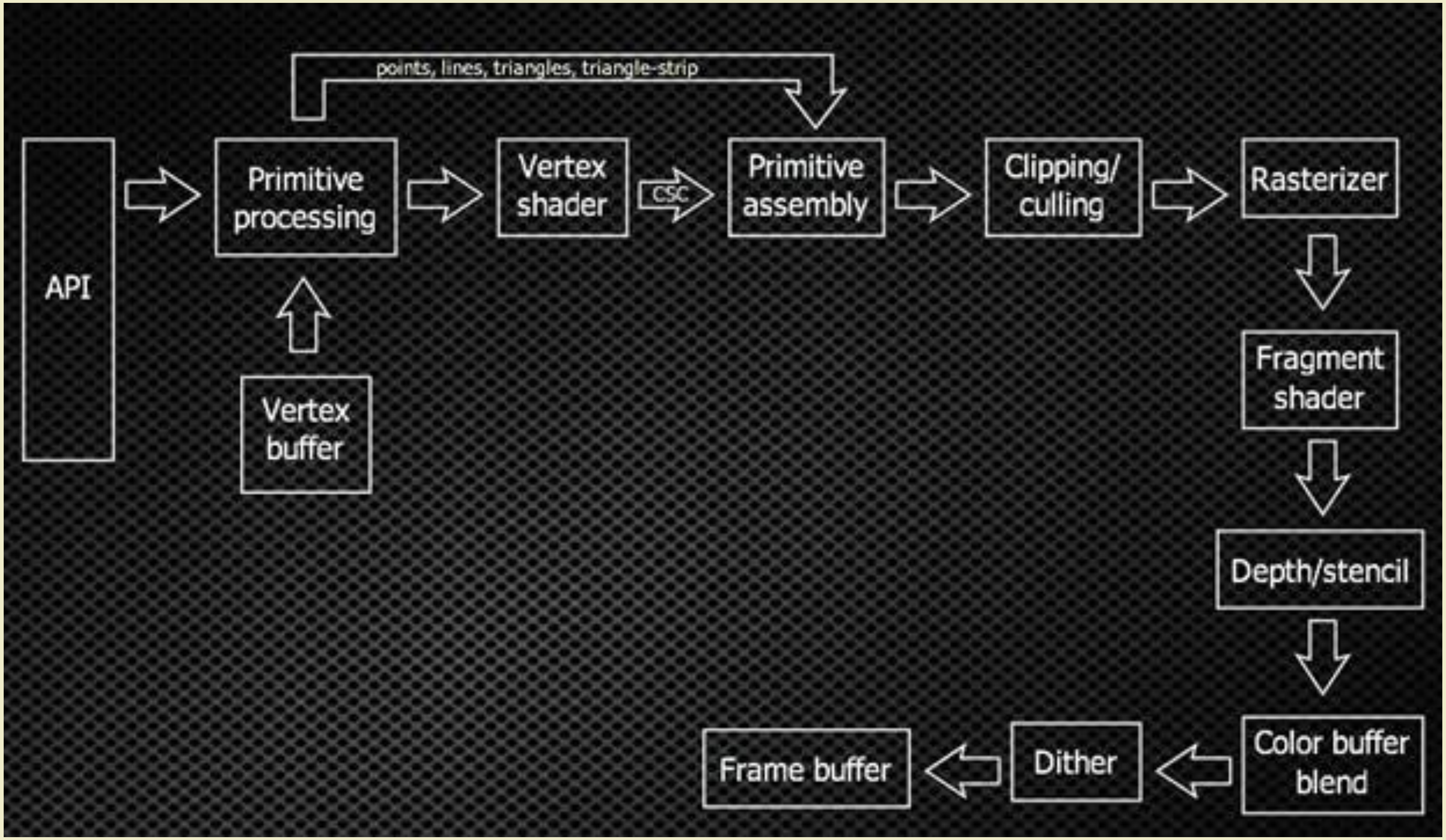
Vertex Shader Sample

```
void main() {
      gl_Position = vec4(0.0,0.0,0.0,1.0);
}
```

Vertex Shader Sample

```
attribute vec3 aPosition;

attribute vec3 aNormal;

uniform mat4 uModel;
uniform mat4 uView;
uniform mat4 uProjection;
uniform mat4 uNormal;
uniform vec3 uMaterialDiffuse;
uniform vec3 uLightDiffuse;
uniform vec3 uLightDirection;

varying vec4 vColor;
void main() {
     gl_Position = uProjection * uView * uModel * vec4(aPosition,1.0);

     vec3 corrected_aNormal = vec3(uNormal * vec4(aNormal,1.0));
     vec3 normalized_aNormal = normalize(corrected_aNormal);
     vec3 normalized_uLightDirection = normalize(uLightDirection);

     float lambertCoefficient = max(dot(-normalized_uLightDirection,normalized_aNormal),0.0);
     vec3 diffuseColor =  uLightDiffuse * uMaterialDiffuse * lambertCoefficient;

     vColor = vec4(diffuseColor,1.0);
}
```

WebGL Programmable Pipeline



Primitive Assembly

Creates triangles or lines out of the vertices

Primitive Assembly

Determined by the mode of drawing passed

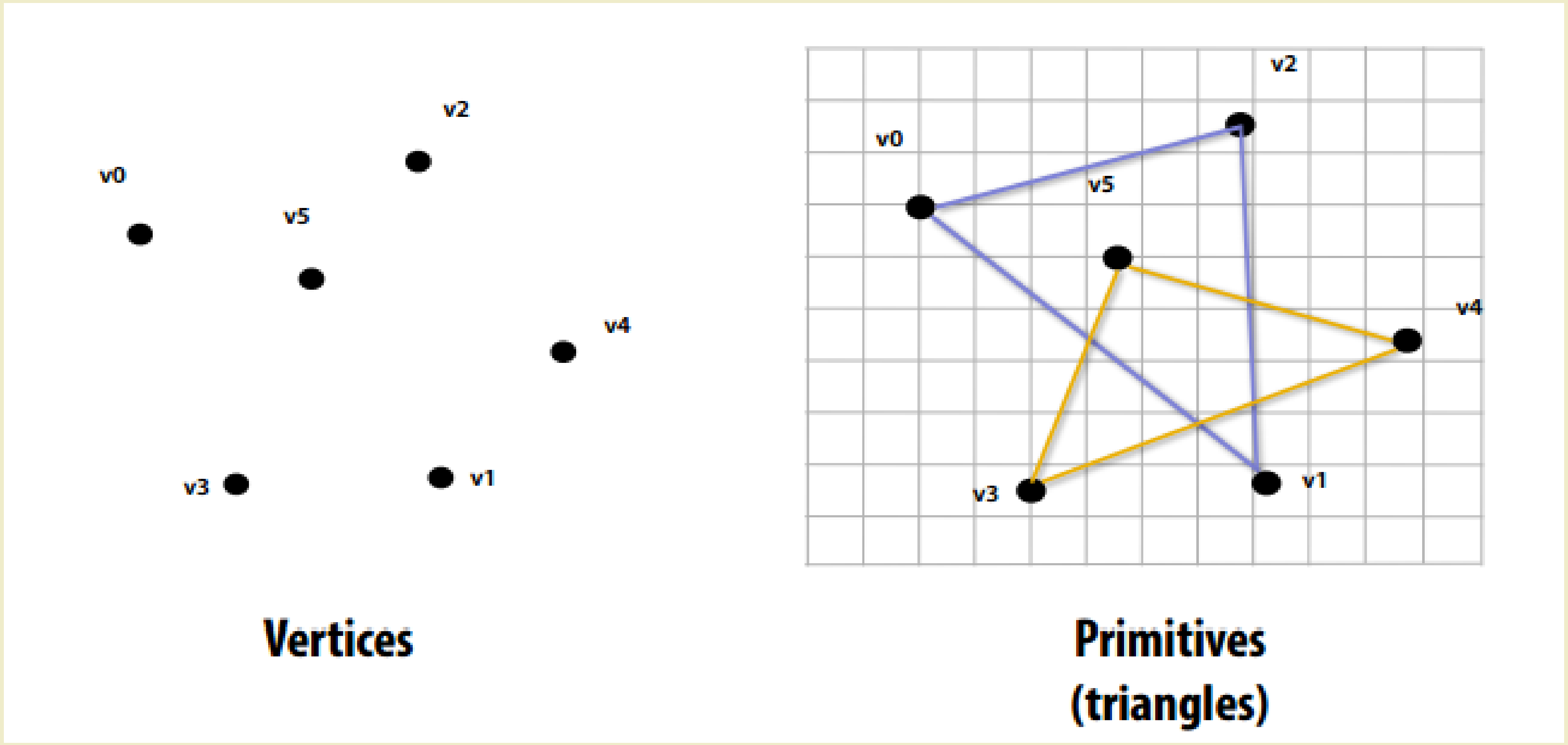
at primitive processing stage

gl.POINTS,

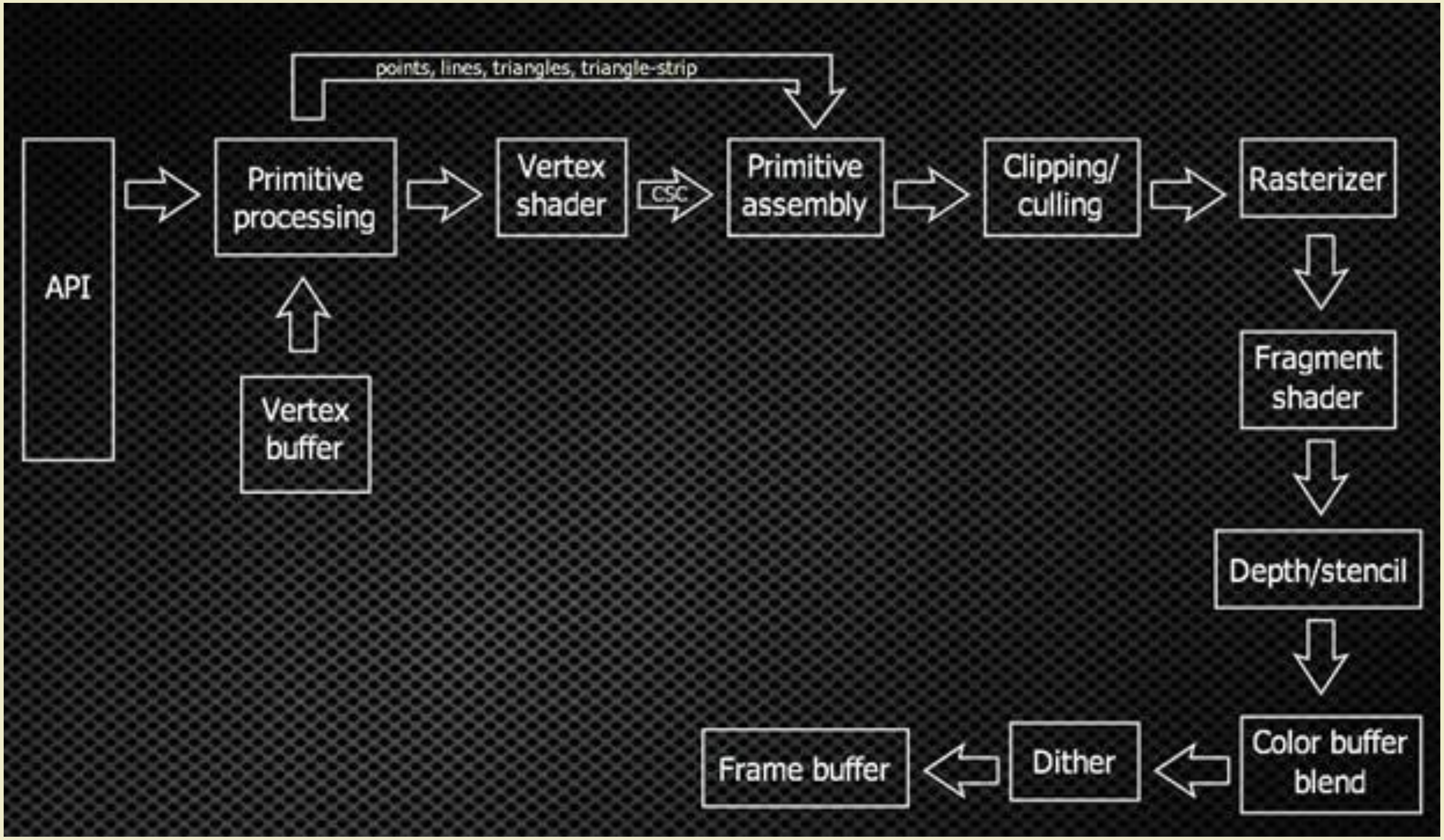
gl.LINE_STRIP, gl.LINE_LOOP, gl.LINES,

gl.TRIANGLE_STRIP, gl.TRIANGLE_FAN, gl.TRIANGLES

Primitive Assembly



WebGL Programmable Pipeline



Clipping/Culling

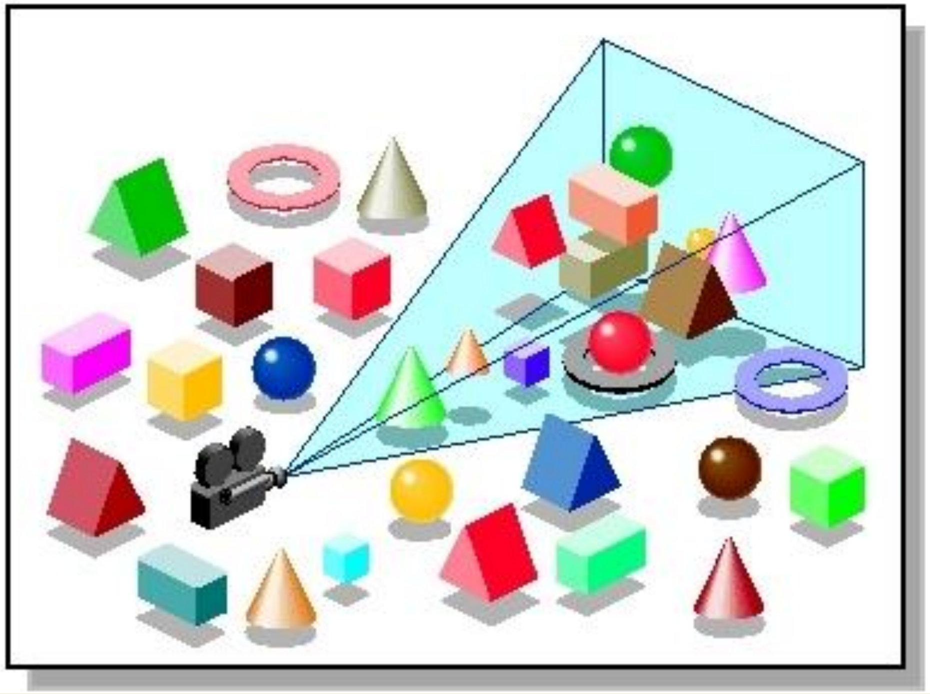
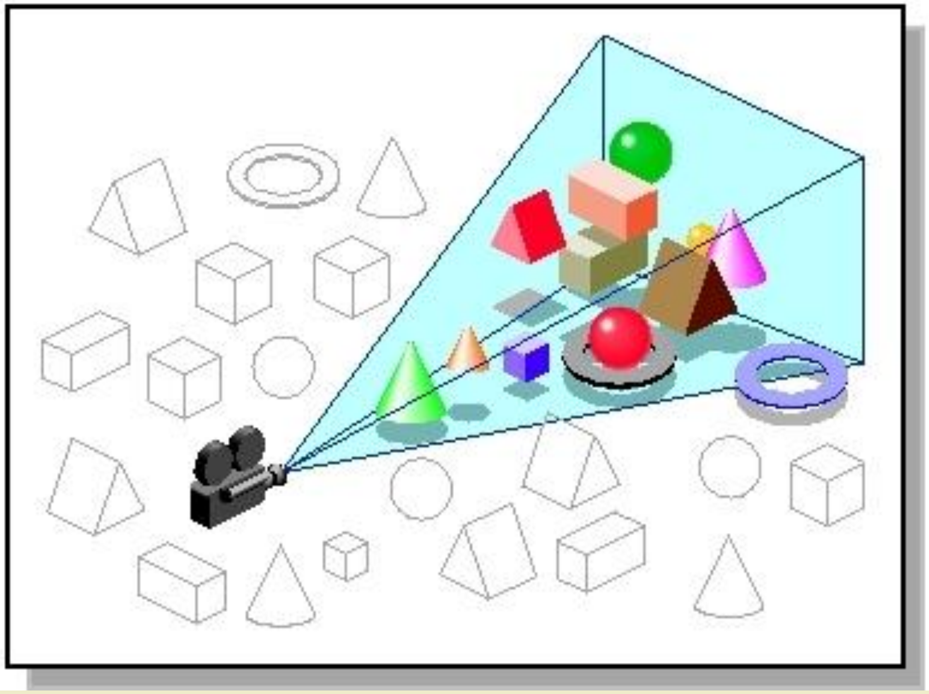
Clipping

Primitives that lie outside the viewing volume is disregarded

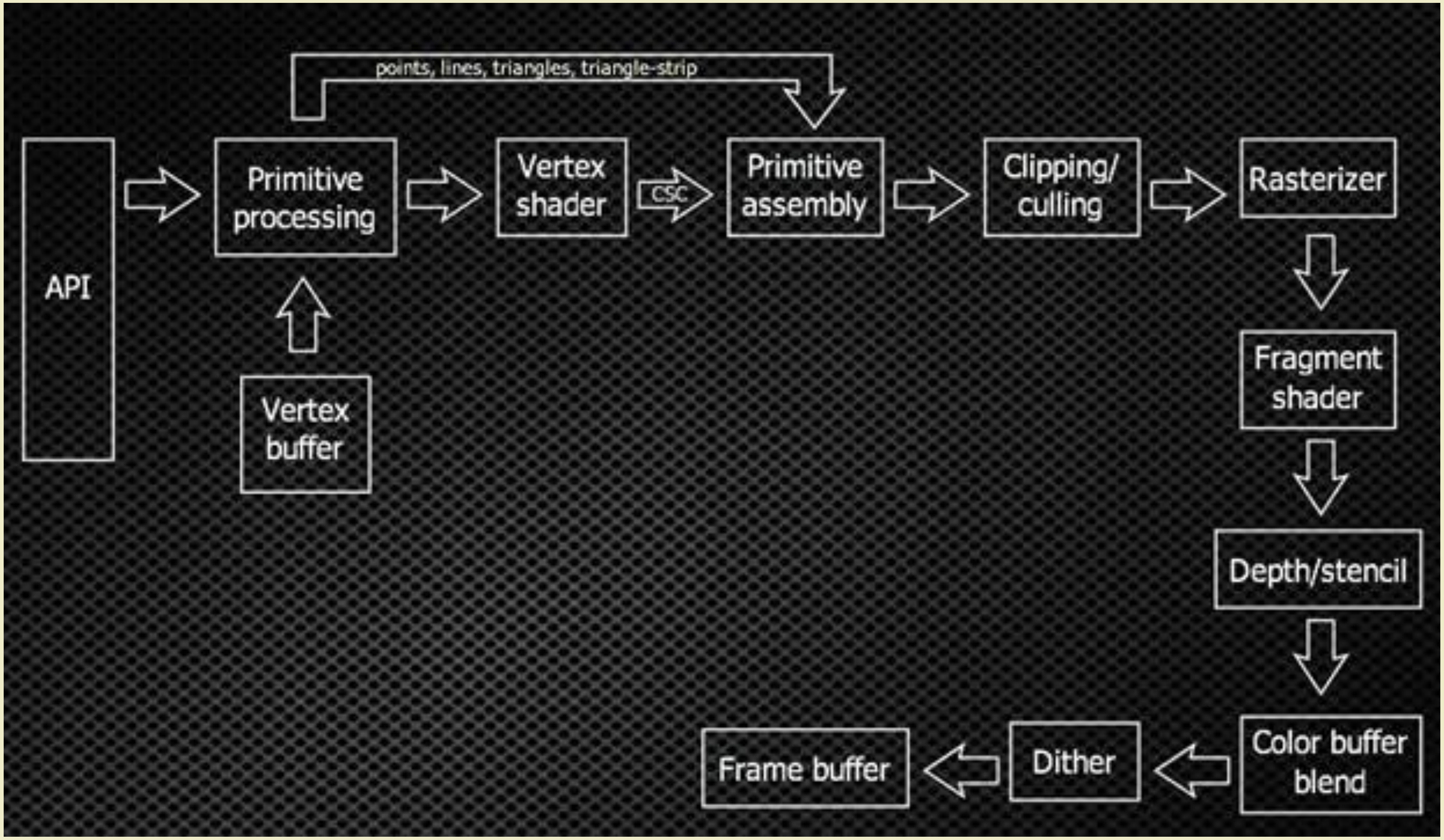
Clipping/Culling

Culling

Removal of back faced primitives

WebGL Programmable Pipeline

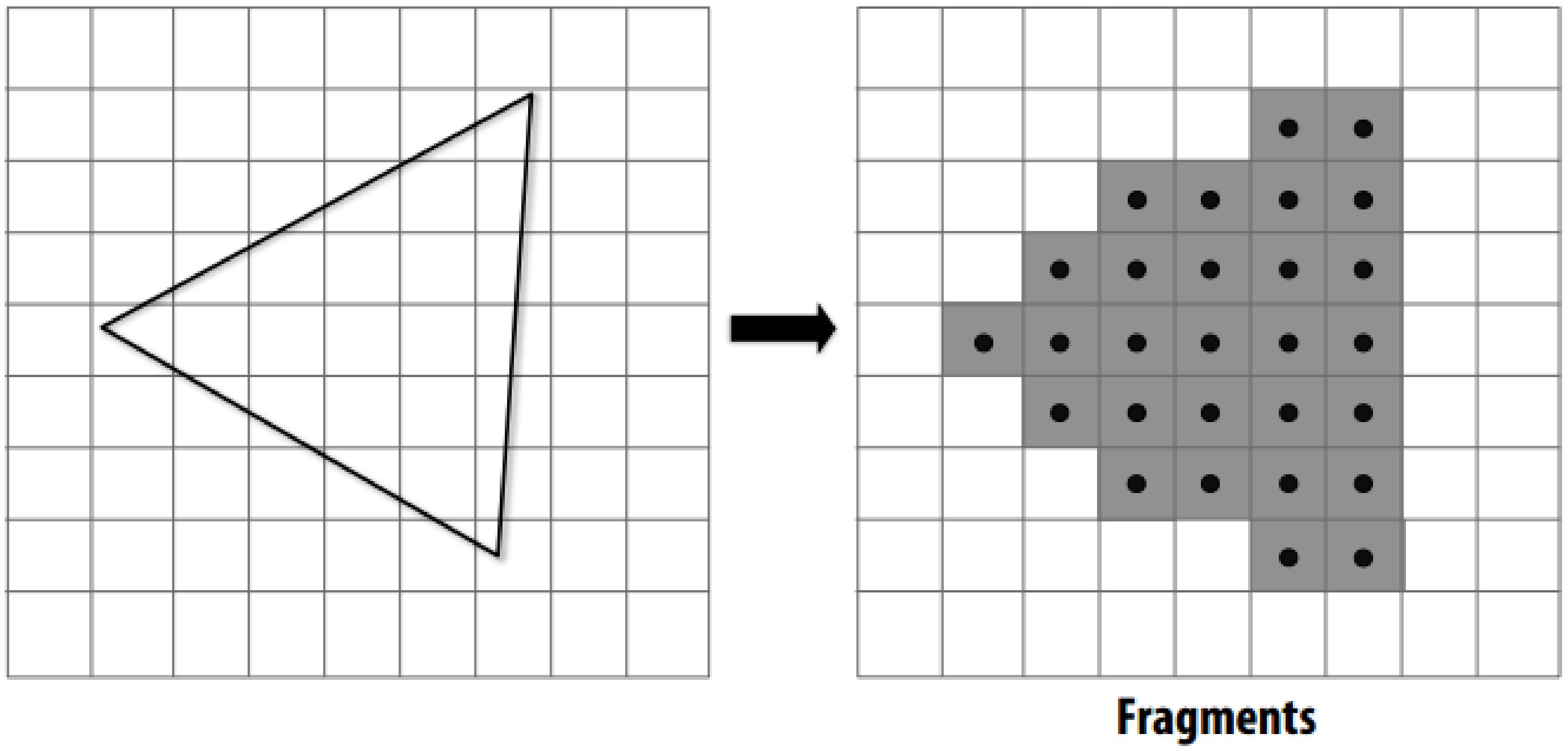


Rasterization

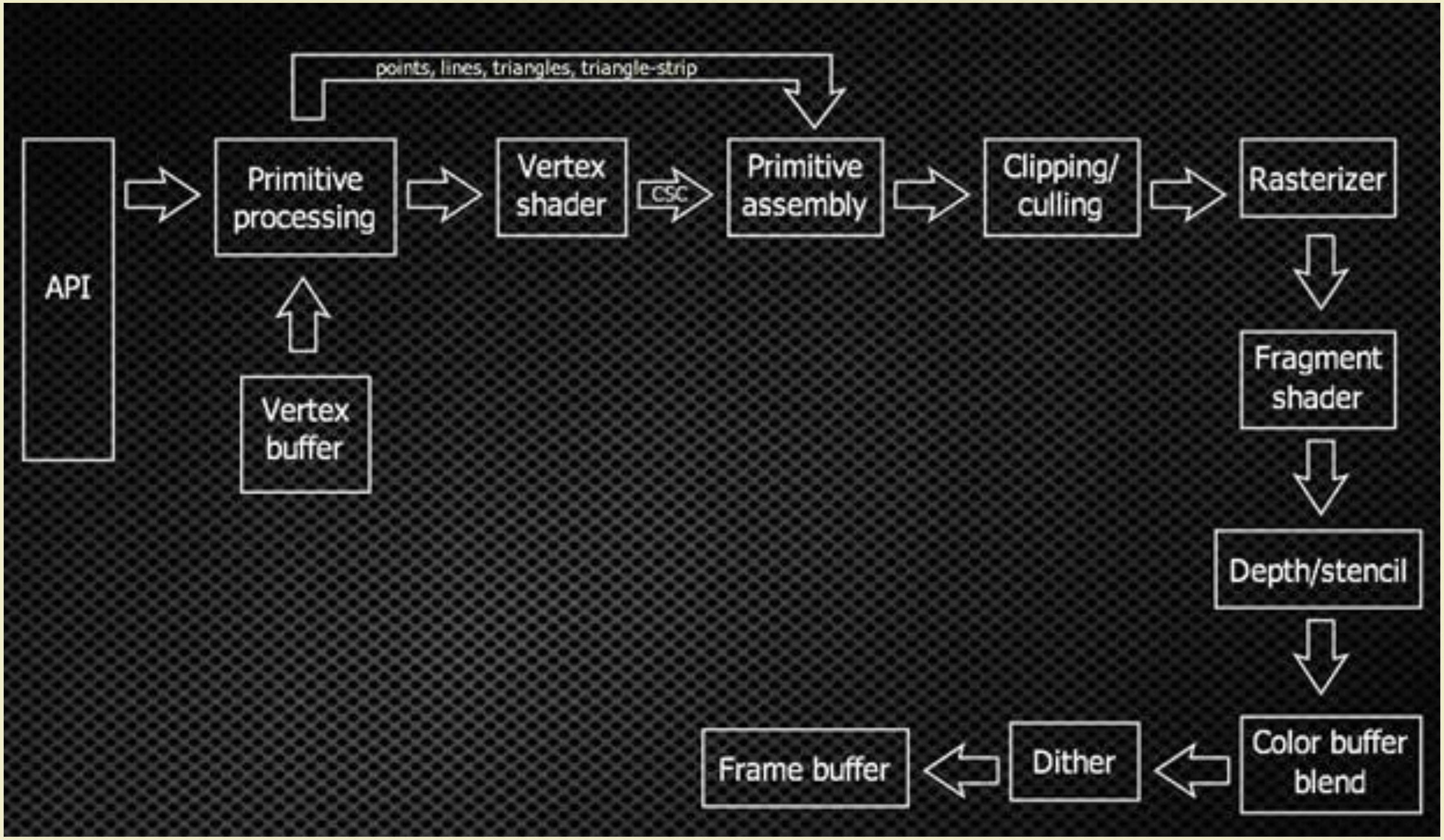
Primitives are broken down into **fragments**

Fragments <-> Pixels

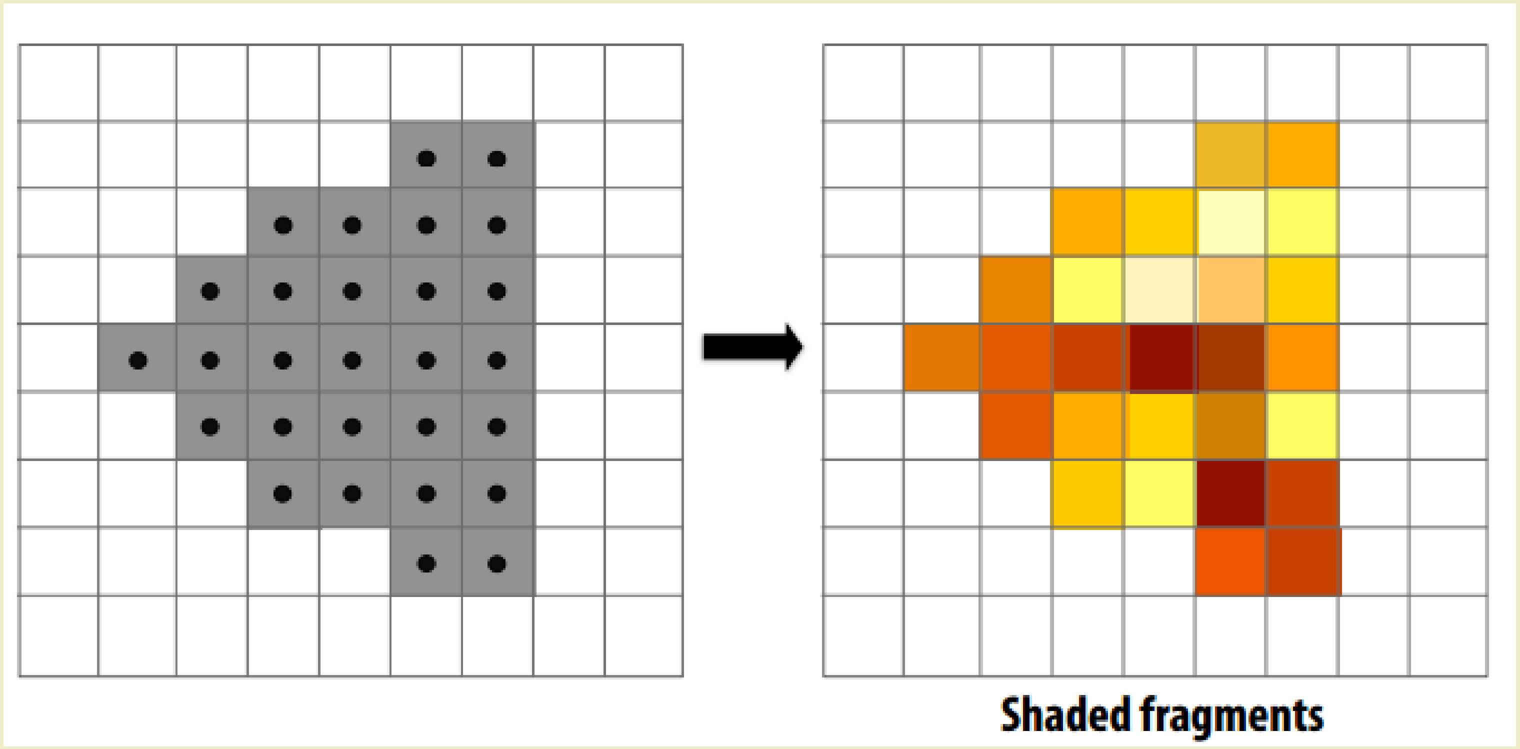
Rasterization



WebGL Programmable Pipeline



Fragment Shader



Fragment Shader

Executed per fragment

Color to be displayed at each fragment

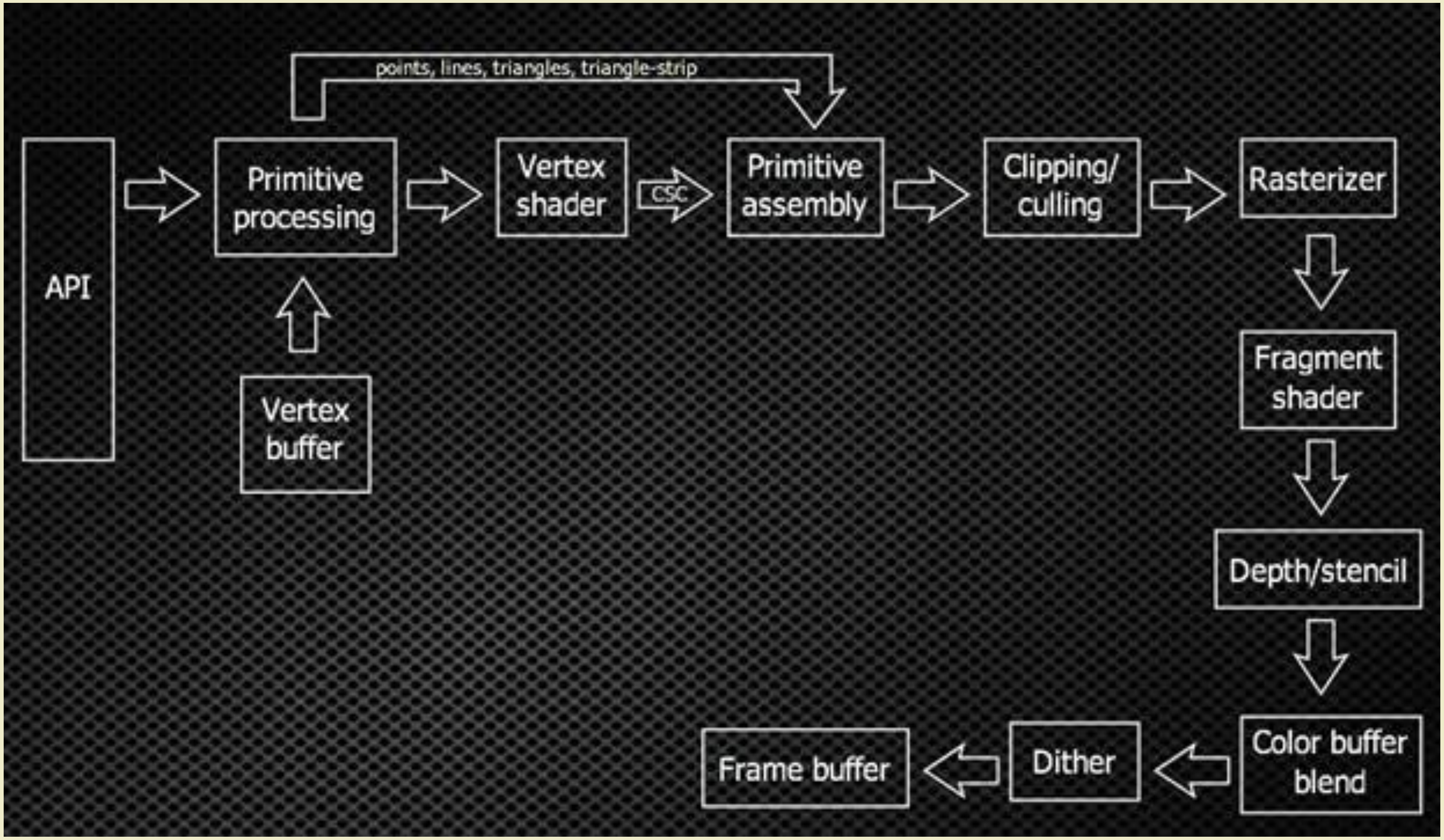
Fragment Shader Sample

```
void main() {
        gl_FragColor = vec4(0.0,1.0,0.0,1.0);
}
```

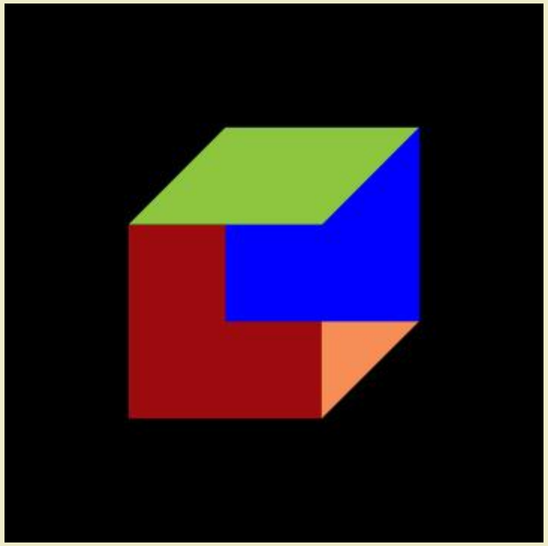
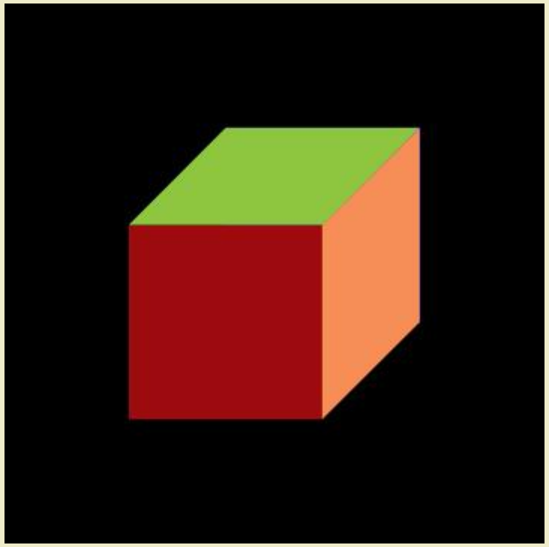
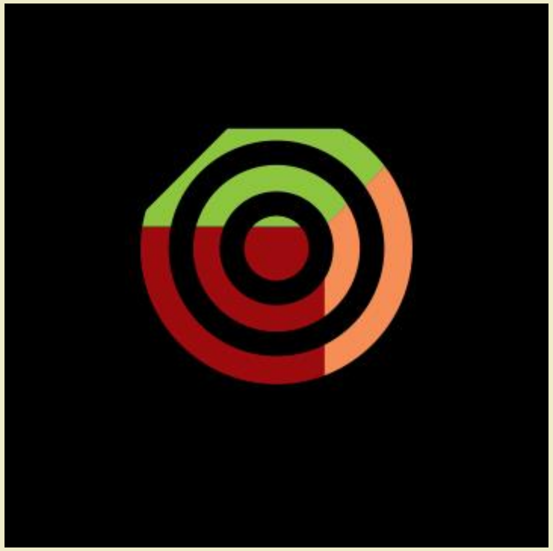
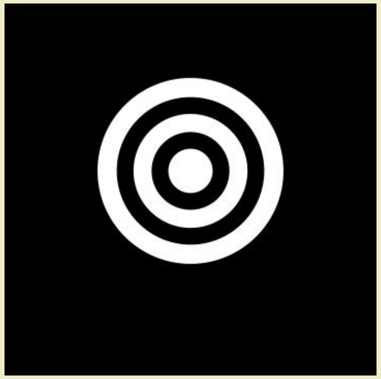
Fragment Shader Sample

```
precision mediump float;
uniform vec3 uLightDirection;
uniform vec3 uEyePosition;
uniform vec3 uMaterialAmbient;
uniform vec3 uLightAmbient;
uniform vec3 uMaterialDiffuse;
uniform vec3 uLightDiffuse;
uniform vec3 uMaterialSpecular;
uniform vec3 uLightSpecular;
uniform float uShininess;
uniform bool uEnableAmbient;
uniform bool uEnableDiffuse;
uniform bool uEnableSpecular;
varying vec3 vNormal;
varying vec3 vPosition;
void main() {

                      vec3 ambientColor = vec3(0.0,0.0,0.0);
                      vec3 diffuseColor = vec3(0.0,0.0,0.0);
                      vec3 specularColor = vec3(0.0,0.0,0.0);
                      vec3 normalized_aNormal = normalize(vNormal);
                      vec3 normalized_uLightDirection = normalize(uLightDirection);
                      vec3 eyeDirection = uEyePosition - vPosition;
                      vec3 normalized_eyeDirection = normalize(eyeDirection);
                      vec3 reflectDirection = reflect(normalized_uLightDirection,normalized_aNormal);
                      vec3 normalized_reflectDirection = normalize(reflectDirection);
                      //ambient
                      if(uEnableAmbient) {
                      ambientColor = uLightAmbient * uMaterialAmbient;
                      }
                      //diffuse
                      if(uEnableDiffuse) {

                      float lambertCoefficient = max(dot(-normalized_uLightDirection,normalized_aNormal),0.0);
                      diffuseColor =  uLightDiffuse * uMaterialDiffuse * lambertCoefficient;
                      }
                      //specular
                      if(uEnableSpecular) {
                      float specularCoefficient = max(dot(normalized_reflectDirection,normalized_eyeDirection),0.0);
                      specularCoefficient = pow(specularCoefficient,uShininess);
                      specularColor =  uLightSpecular * uMaterialSpecular * specularCoefficient;
                      //specularColor = vec3(1.0,1.0,1.0) * specularCoefficient;
                      }
                      vec4 finalColor = vec4(ambientColor+diffuseColor+specularColor,1.0);
                      gl_FragColor = finalColor;
}
```

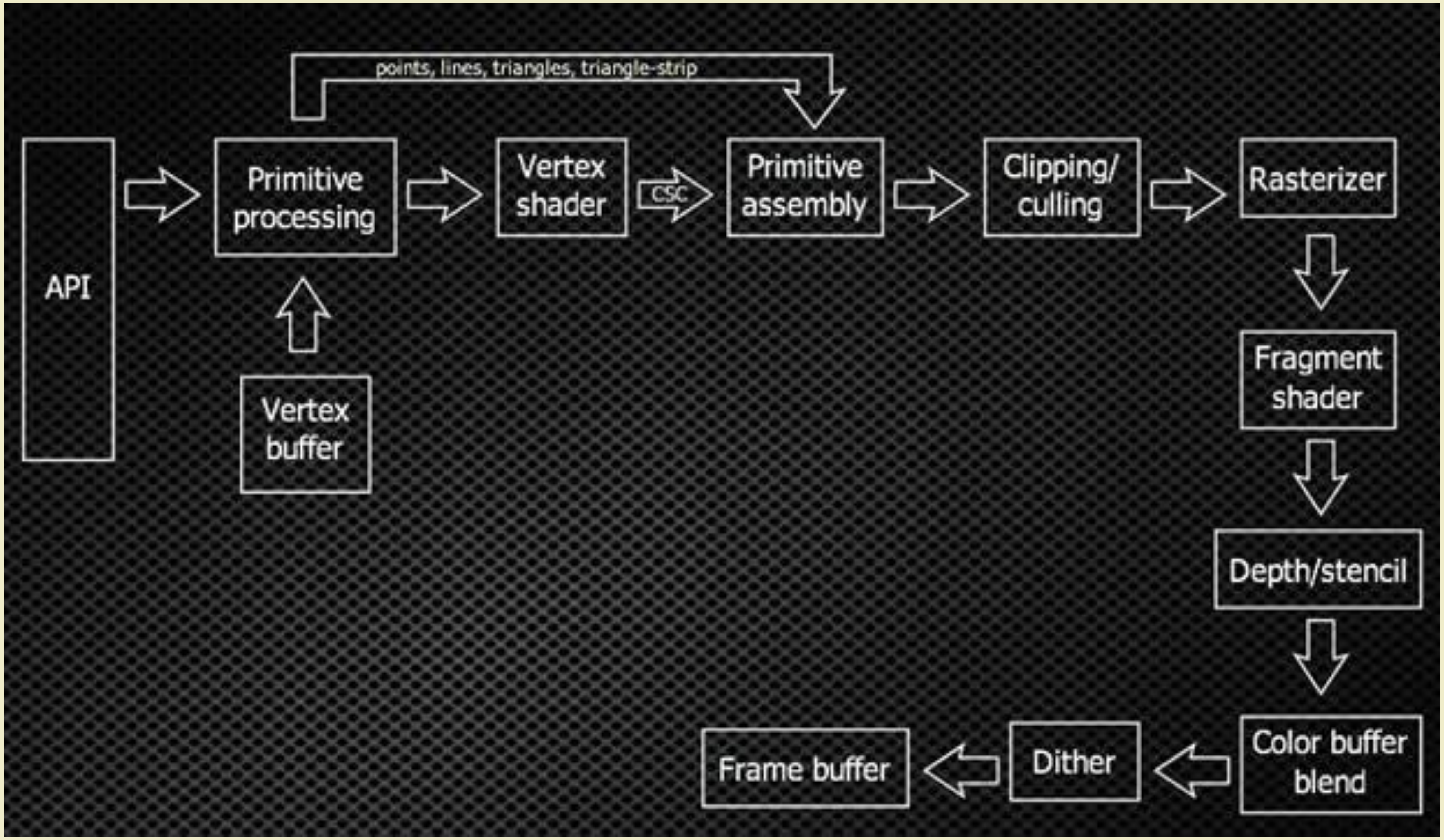
WebGL Programmable Pipeline



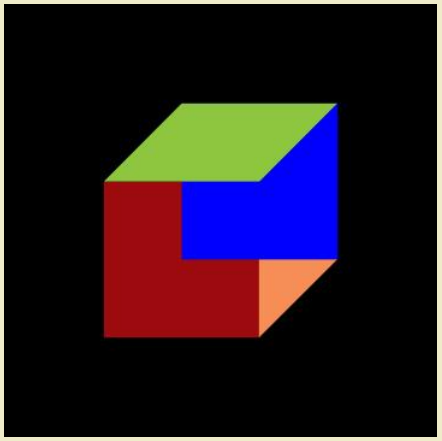
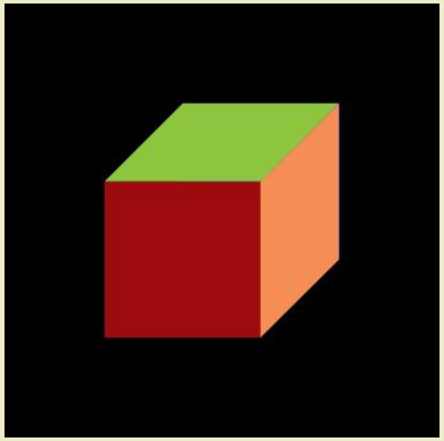
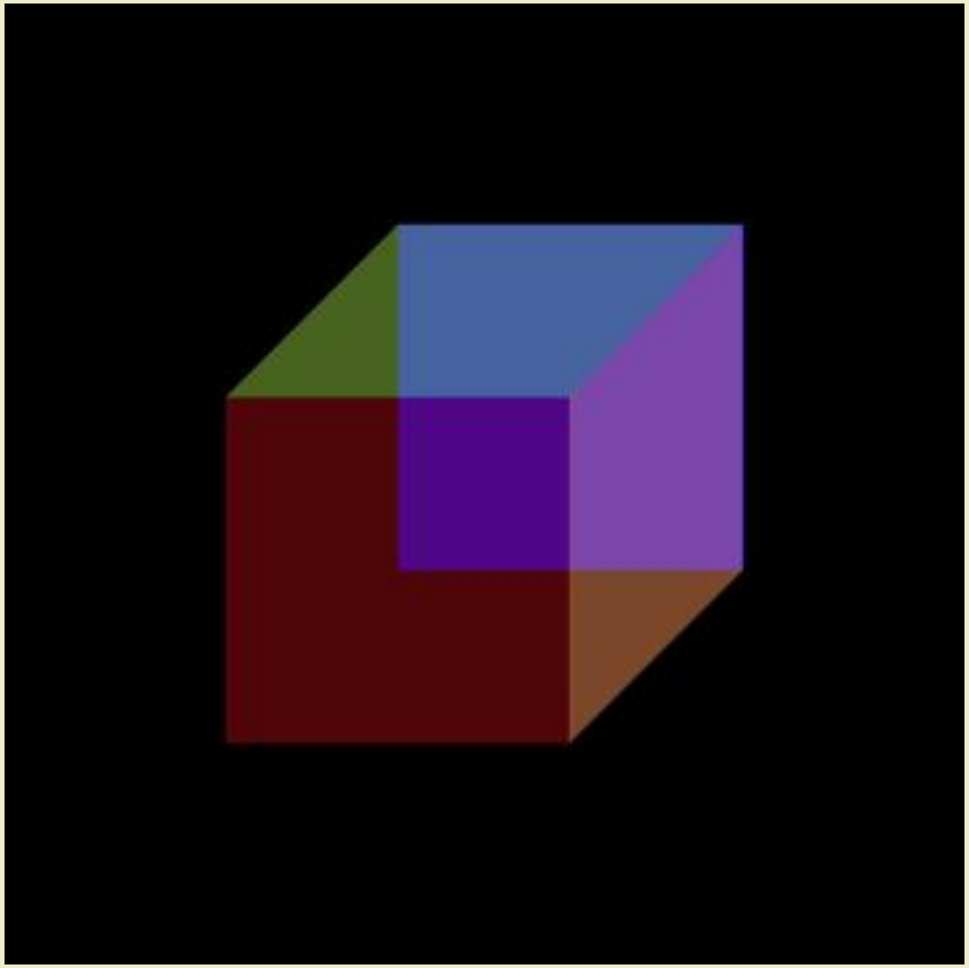
Depth/Stencil Test

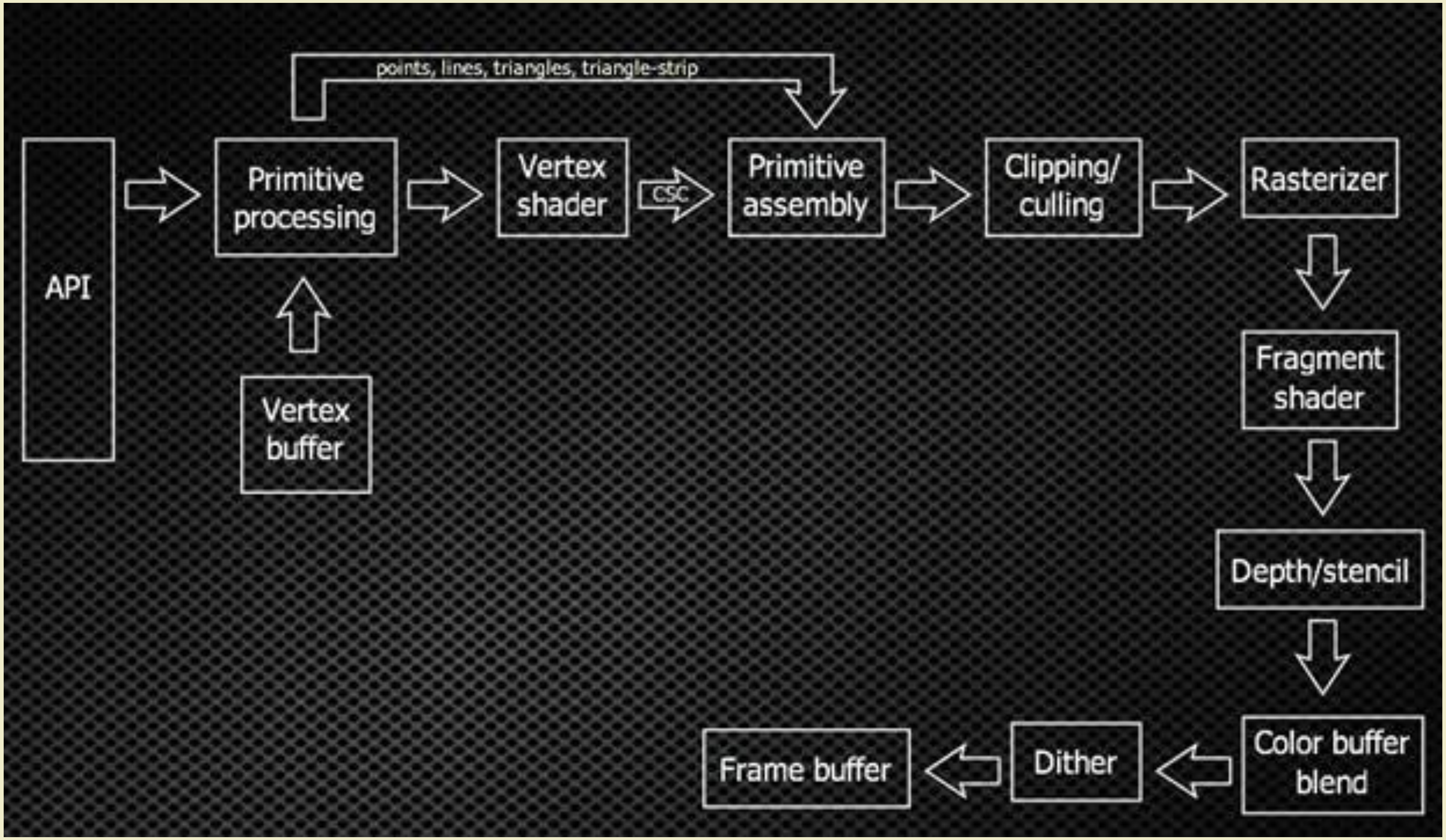
WebGL Programmable Pipeline



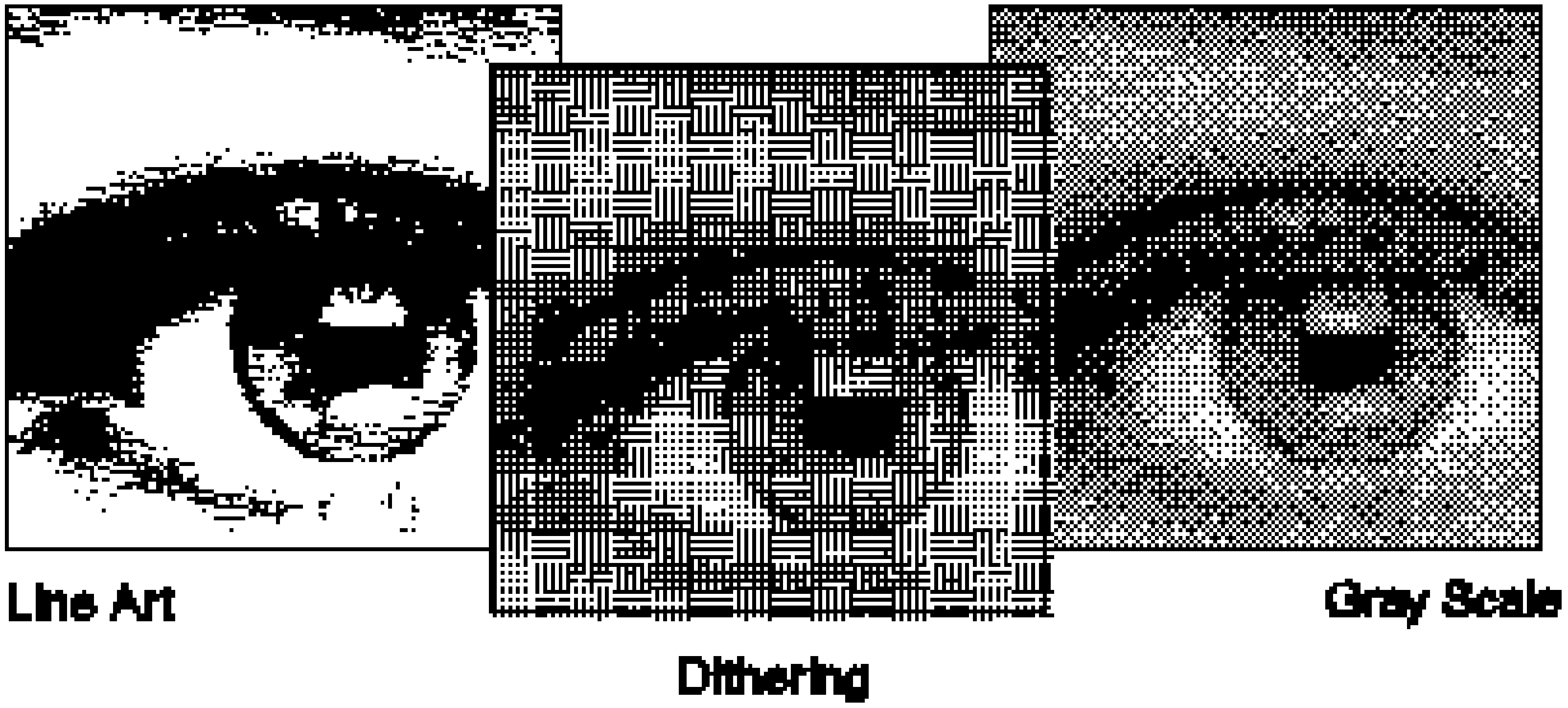
Blending

WebGL Programmable Pipeline

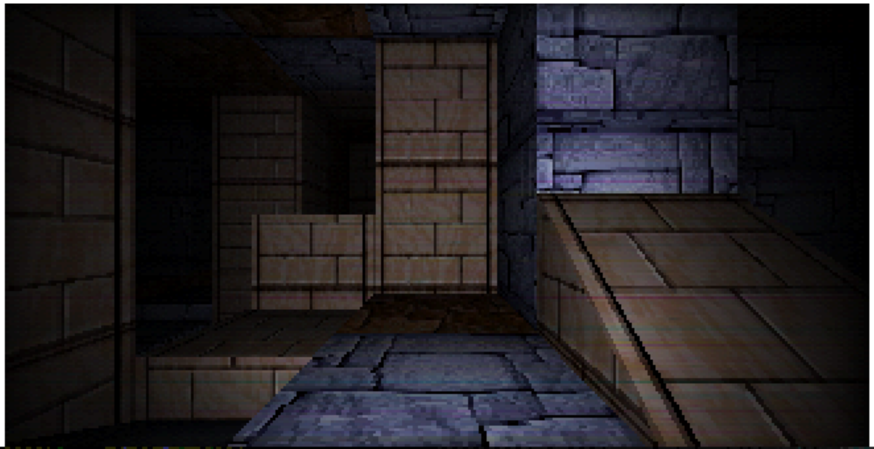


Dithering

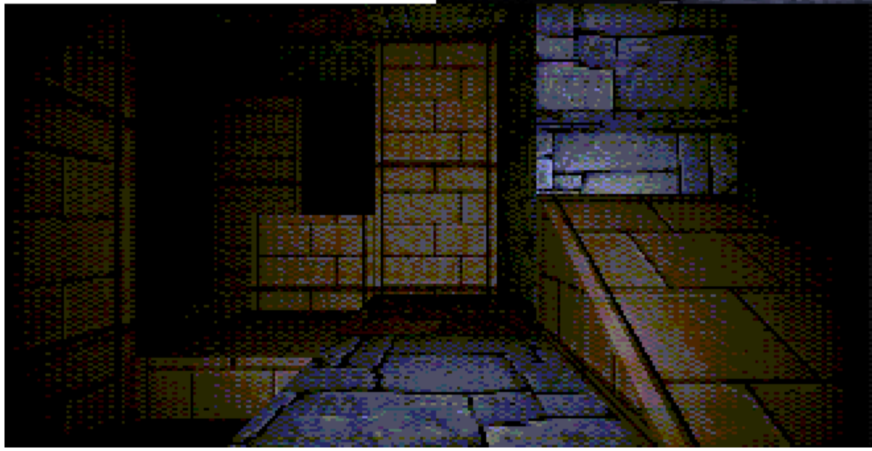


Dithering

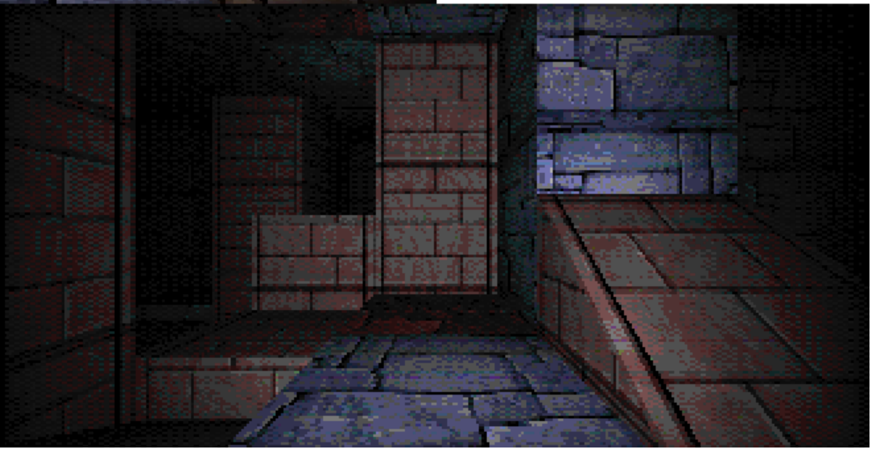
Rendered source buffer ->



RGB332



Custom HSB based palette



References

Books

- MATSUDA, K. AND LEA, R. 2013. WebGL Programming Guide. Addison-Wesley. Upper Saddle River, NJ.

Lecture Slides

- ALAMBRA, A. CMSC 161 1st Semester 2013-14 Lecture Slides

Images

- http://files.myopera.com/emoller/blog/opengl-timeline.html
- http://dev.opera.com/articles/view/raw-webgl-part1-getting-started/
- http://www.webopedia.com/FIG/DITHER.gif
- http://i5.minus.com/i75qjiyFQzVCI.jpg
- http://i.stack.imgur.com/sJDdX.png