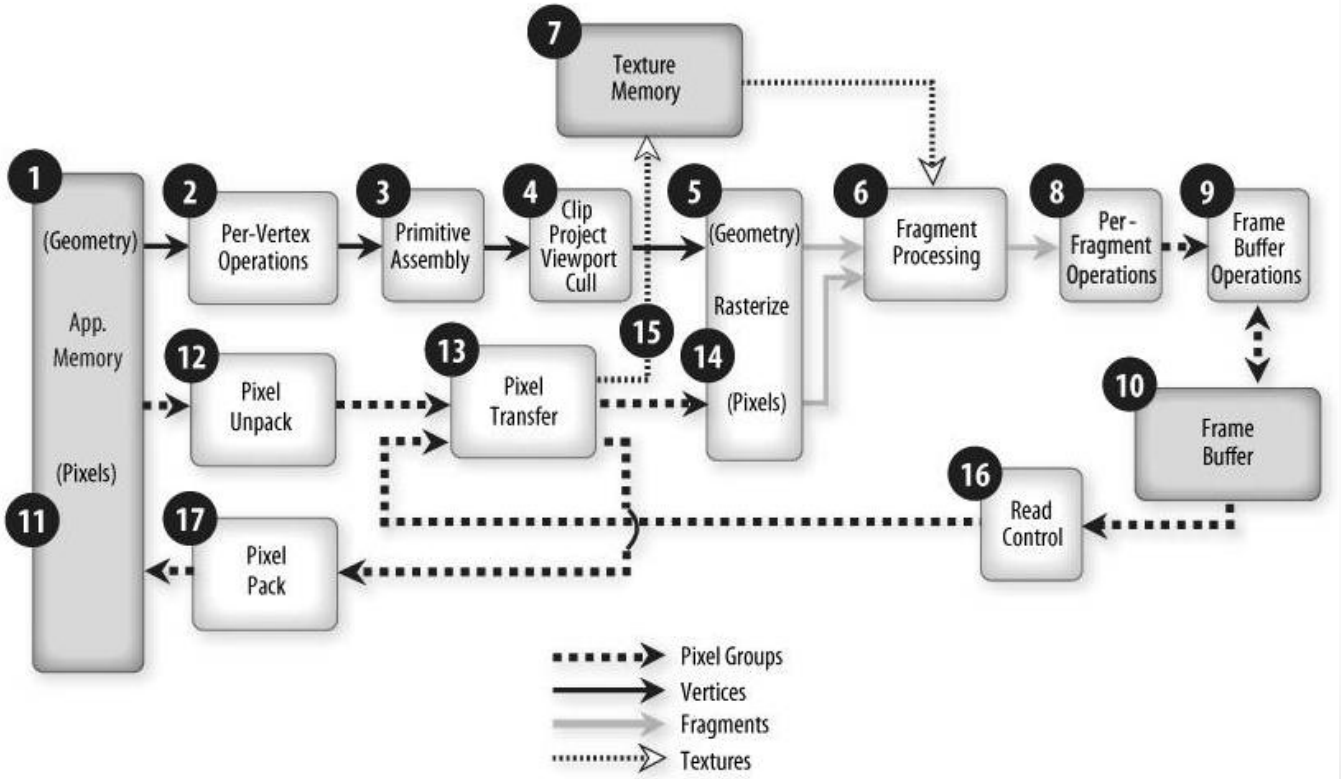
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

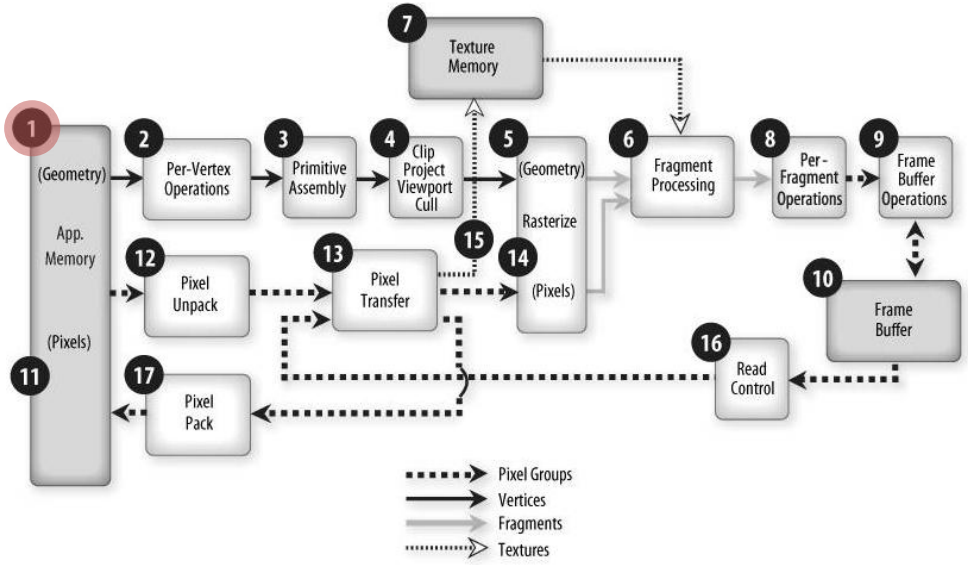
CG Basics II – Programmable Pipeline

Bachelor's Degree in Video Game Design and Development

Fixed Pipeline

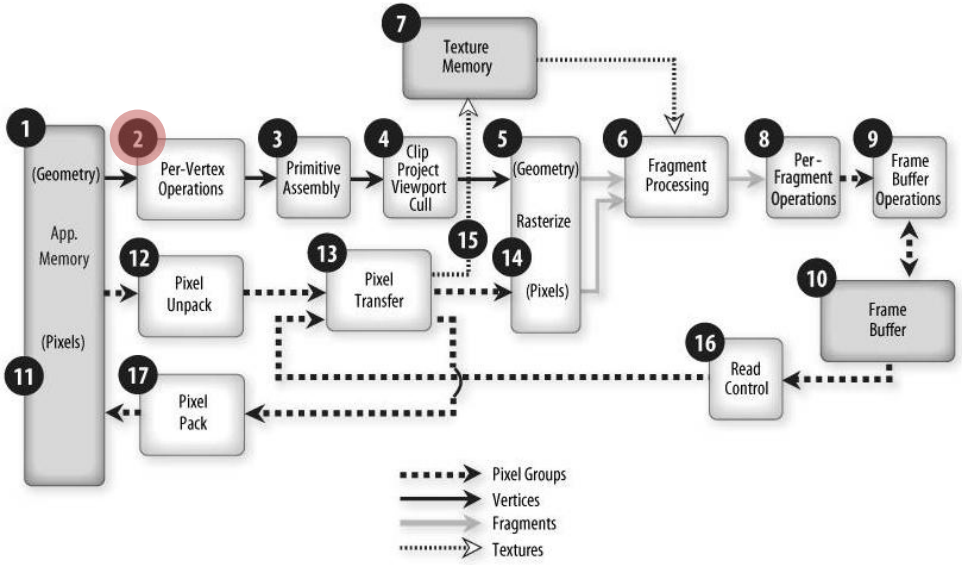


Fixed Pipeline



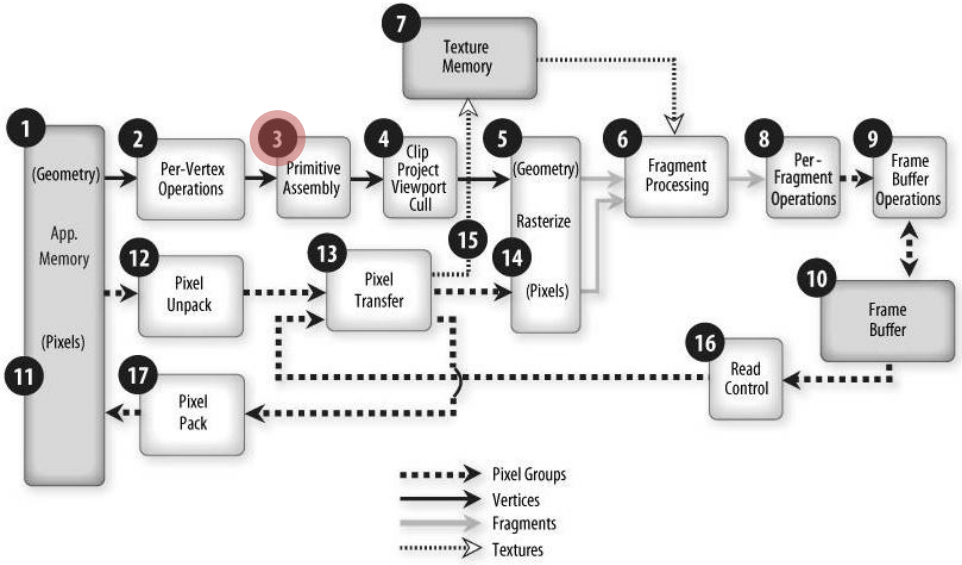
- **Geometry loading**
  - Send the primitives to draw (points, lines, polygons)
  - In WebGL we will store them directly on the GPU by using GPU buffers

Fixed Pipeline



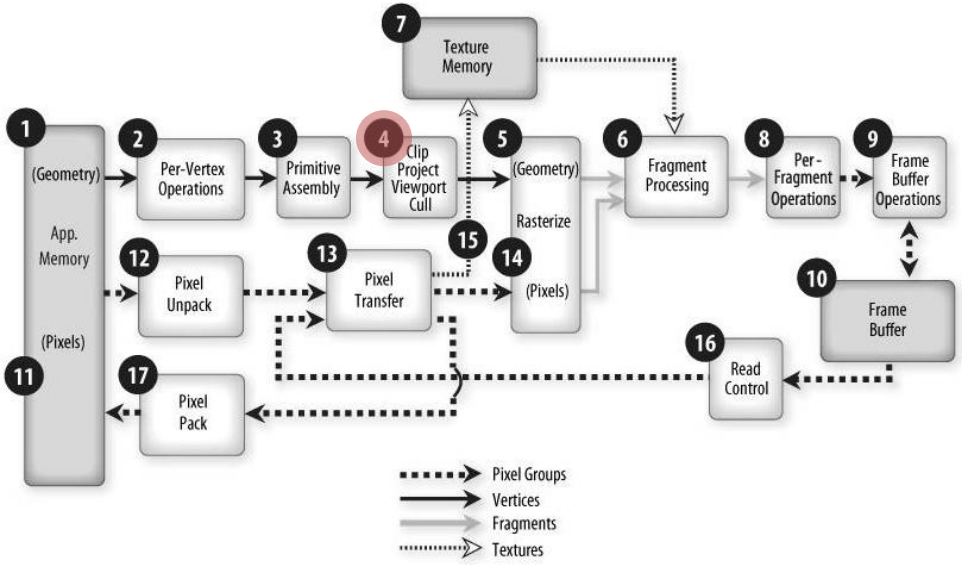
- **Per-vertex operations**
  - Vertex are transformed (modelview and projection)
  - Normals are transformed and illumination is computed for each vertex
  - Texture coordinates are generated automatically
  - Texture coordinates are transformed (texture matrix)

Fixed Pipeline



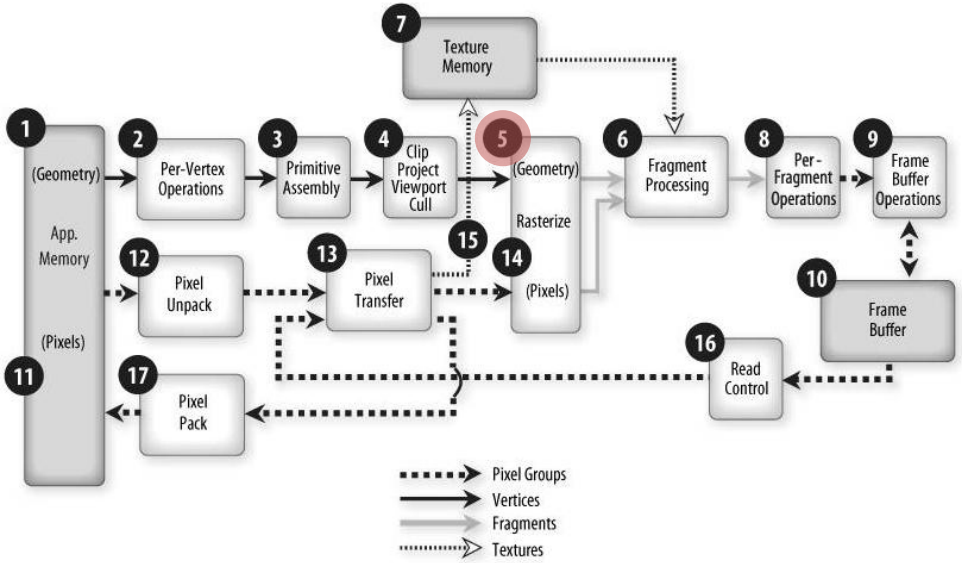
- **Primitive assembly**
  - Vertices are assembled to form primitives
  - Each primitive (GL_POINT, GL_LINES, GL_POLYGON) requires a different clipping

Fixed Pipeline



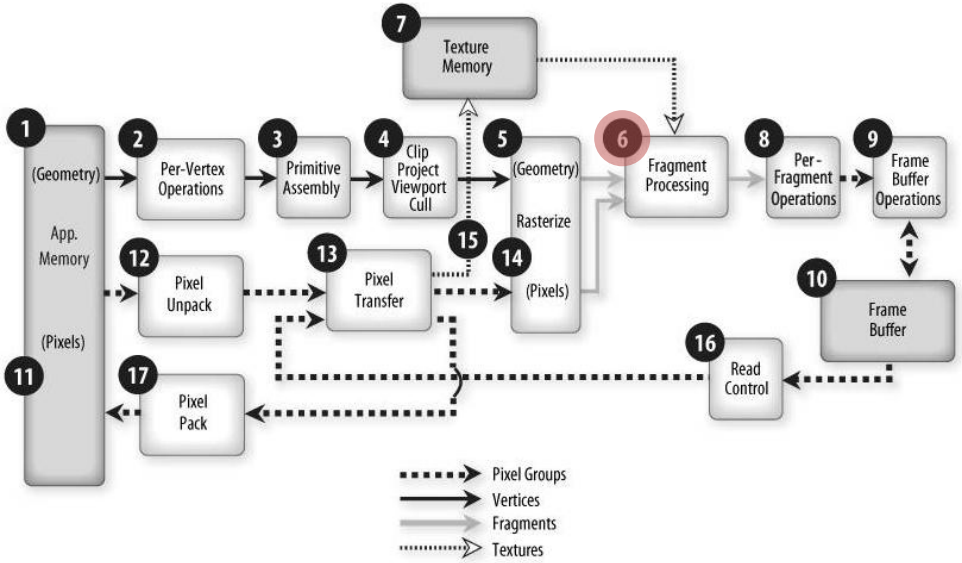
- **Primitive processing**
  - Frustum clipping is performed
  - Perspective division: (x, y, z) divided by w
  - Viewport and depth transform → window coordinates
  - Backface culling

Fixed Pipeline



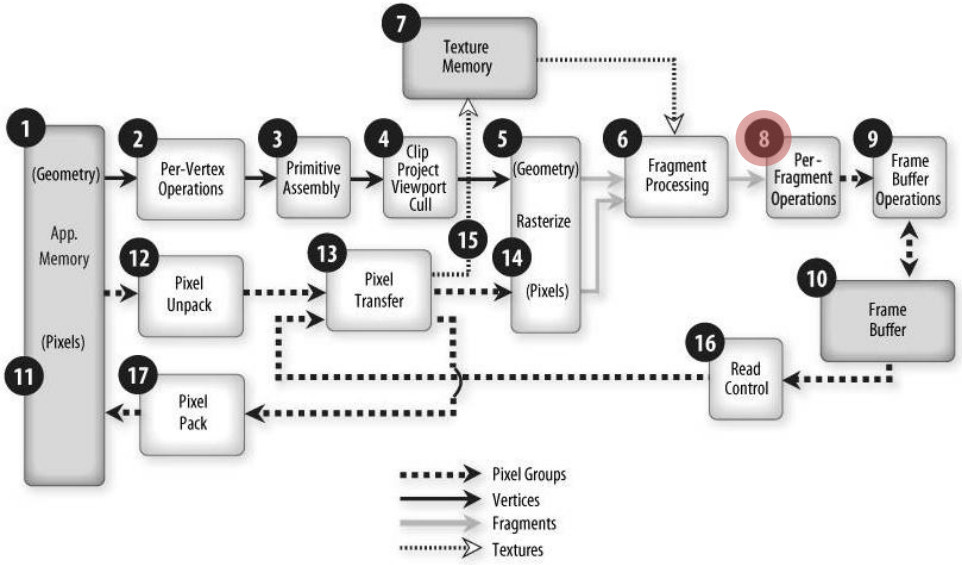
- **Rasterization**
  - **The fragments for each primitive are generated**
  - **Each fragment has different attributes**
    - **Window coordinates (x, y, z, w)**
    - **Primary color (interpolated if Gouraud shading is applied)**
    - **Secondary color (interpolated if Gouraud shading is applied)**
    - **Texture coordinates (interpolated)**

Fixed Pipeline



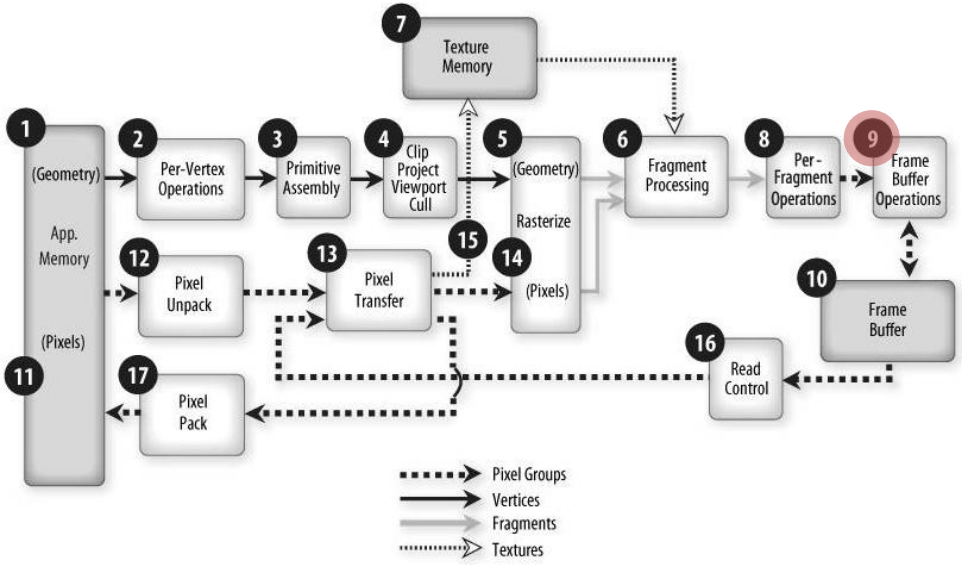
- **Fragment processing (“Shading”)**
  - **Primary and secondary colors combination**
  - **Texture mapping**
  - **Fog**

Fixed Pipeline



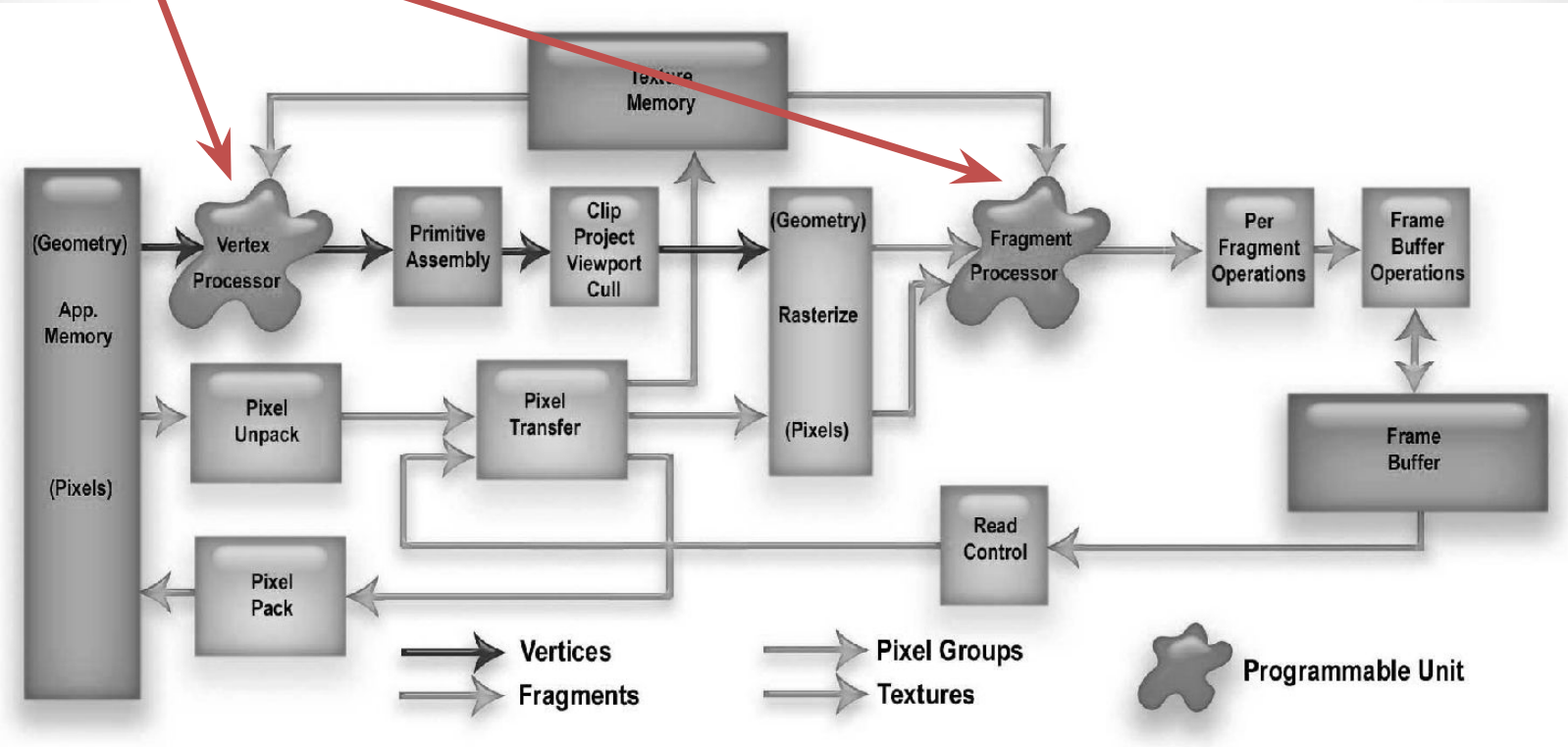
- **Per-fragment operations**
  - Pixel ownership
  - Scissor test
  - Alpha test
  - Stencil test
  - Depth test
  - Blending
  - Dithering
  - Logical ops

Fixed Pipeline



- **Frame buffer operations**
  - Update of the buffers selected with glDrawBuffers
  - Affected by glColorMask, glDepthMask…

Program. Pipeline



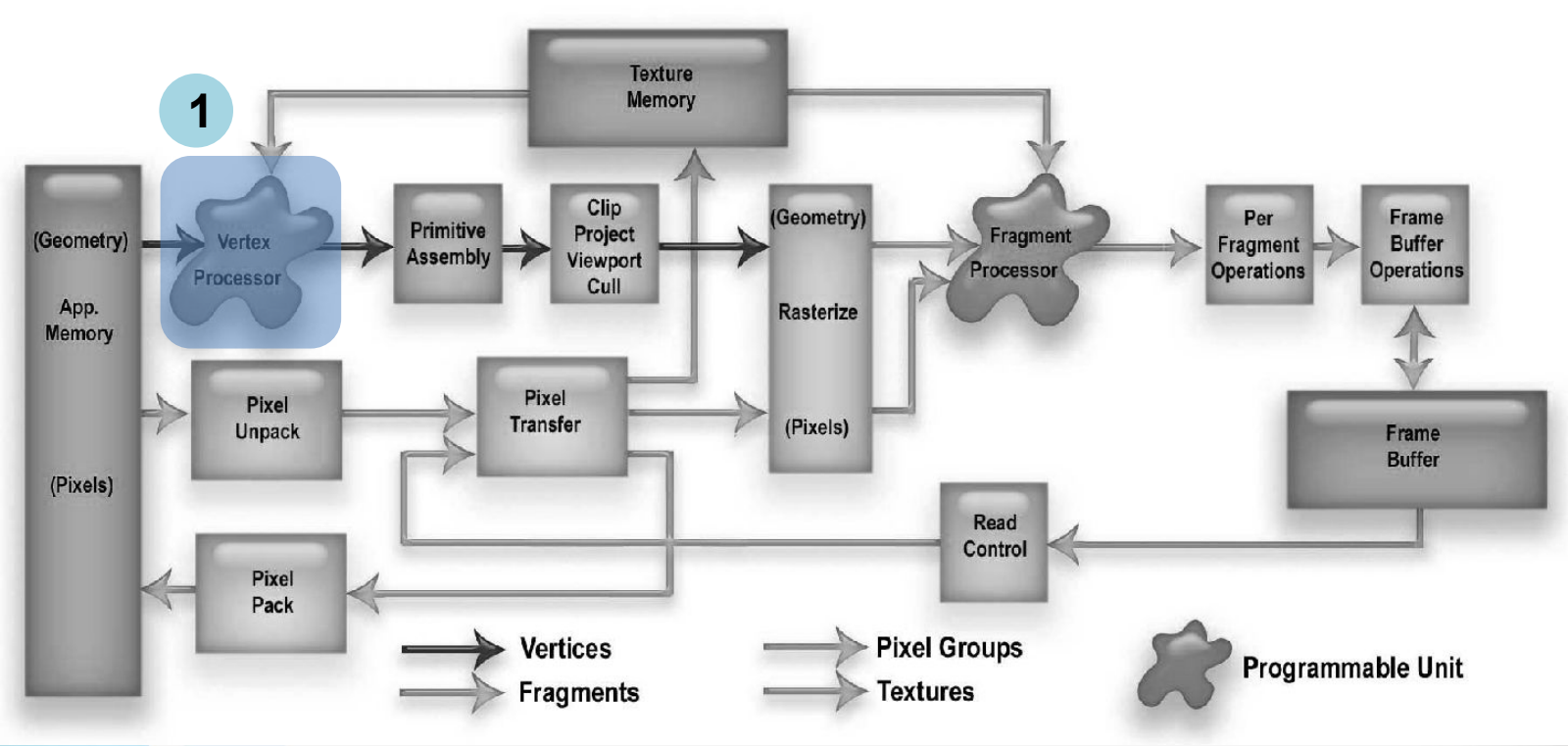
Program. Pipeline

- Vertex processor
  - Part of the GPU that executes a program for each vertex
- Fragment processor
  - Part of the GPU that executes a program for each fragment
- Shader
  - Source code of a program to be executed on the GPU
    - Vertex shader, fragment shader, geometry shader…
- Program
  - Executable file of a shader
    - Vertex program, fragment program, geometry program…

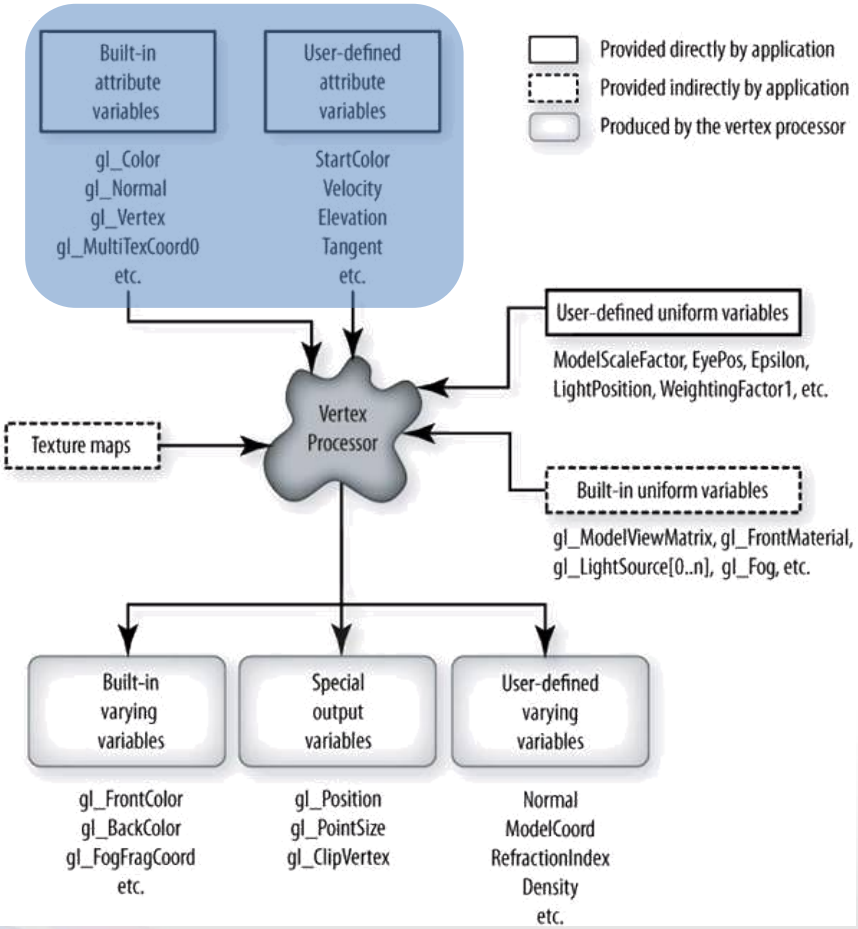
Program. Pipeline

- By enabling a vertex program, we tell the GPU to execute that program instead of the fixed per-vertex operations
- By enabling a fragment program, we tell the GPU to execute that program instead of the fixed per-fragment operations
- To do this, even the simpler vertex/fragment programs must reproduce the fixed OpenGL/WebGL functionalities

Vertex Shader



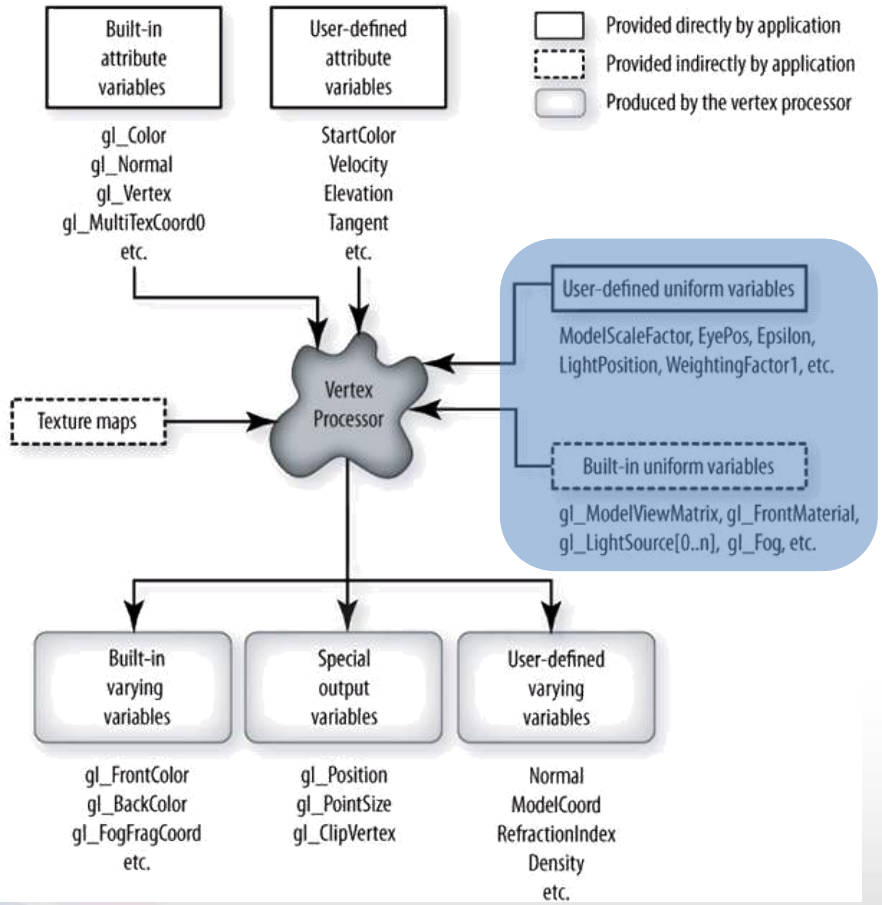
Vertex Shader



Vertex Shader

- ***Attribute* variables**
  - Represent the attributes of a vertex
  - They can be different for each vertex of a primitive
  - Built-in attributes: predefined attribs. (we must declare them in WebGL!!)
    - Sent from the application by using glColor, glNormal…
    - Access from within the shader with gl_Color, gl_Normal…
  - User-defined attributes: we must declare them
    - Sent from the application by using glVertexAttrib, and linked with a name by using glGetAttribLocation
    - Access from within the shader with a user-defined name: speed, etc.

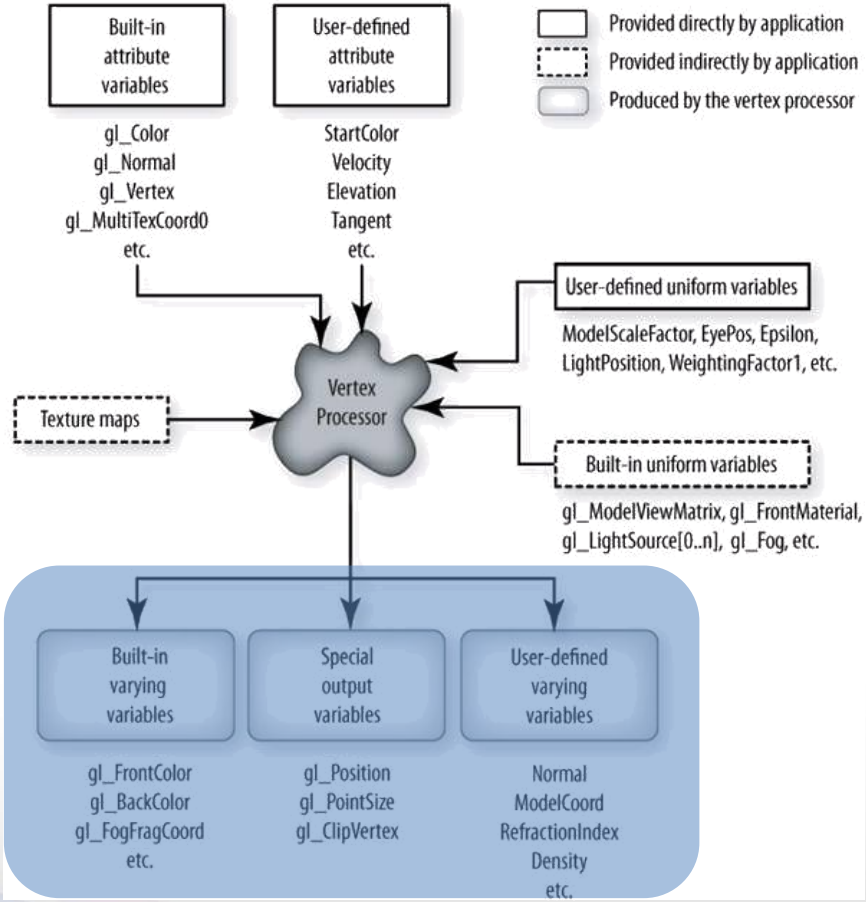
Vertex Shader



Vertex Shaders

- *Uniform* variables
  - Variables that normally have the same value for the vertices of a primitive
  - Built-in variables: OpenGL/WebGL specific variables
    - Access from within the shader with gl_ModelViewMatrix, gl_LightSource[0..n], …
    - They must be declared in WebGL!!
  - User-defined variables: we must declare them
    - Sent from the application by using glUniform, and linked with a name by using glGetUniformLocation
    - Access from within the shader with a user-defined name: eyePos, etc.

Vertex Shader



Vertex Shader

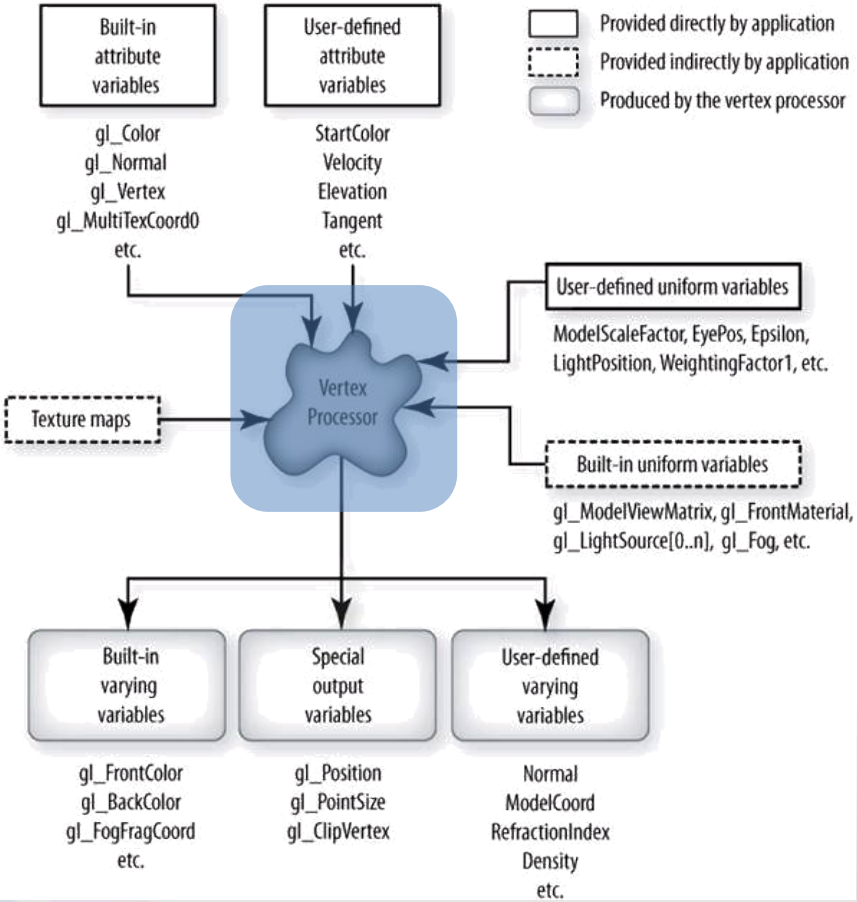
- *Varying* variables
  - Variables sent from the vertex program to the fragment program
  - Output variables for the vertex program
  - Input variables for the fragment program and computed by interpolation
  - Built-in variables: OpenGL/WebGL specific variables
    - For instance gl_FrontColor, …
    - They must be declared in WebGL!!
  - User-defined variables: we must declare them
    - Normal, Refraction…

Vertex Shader

- ***Special output* variables**
  - Variables that must compute the vertex program
  - They are built-in variables
  - They begin with gl_ and they are already defined (no need of defining them)
  - At least, the vertex program must compute *gl_Position*
    - Vertex coords. in the clipping coords. System
    - Normally computed as follows:

$$ProjMatrix * MVMatrix * vertexCoords$$

Vertex Shader



Vertex Shader

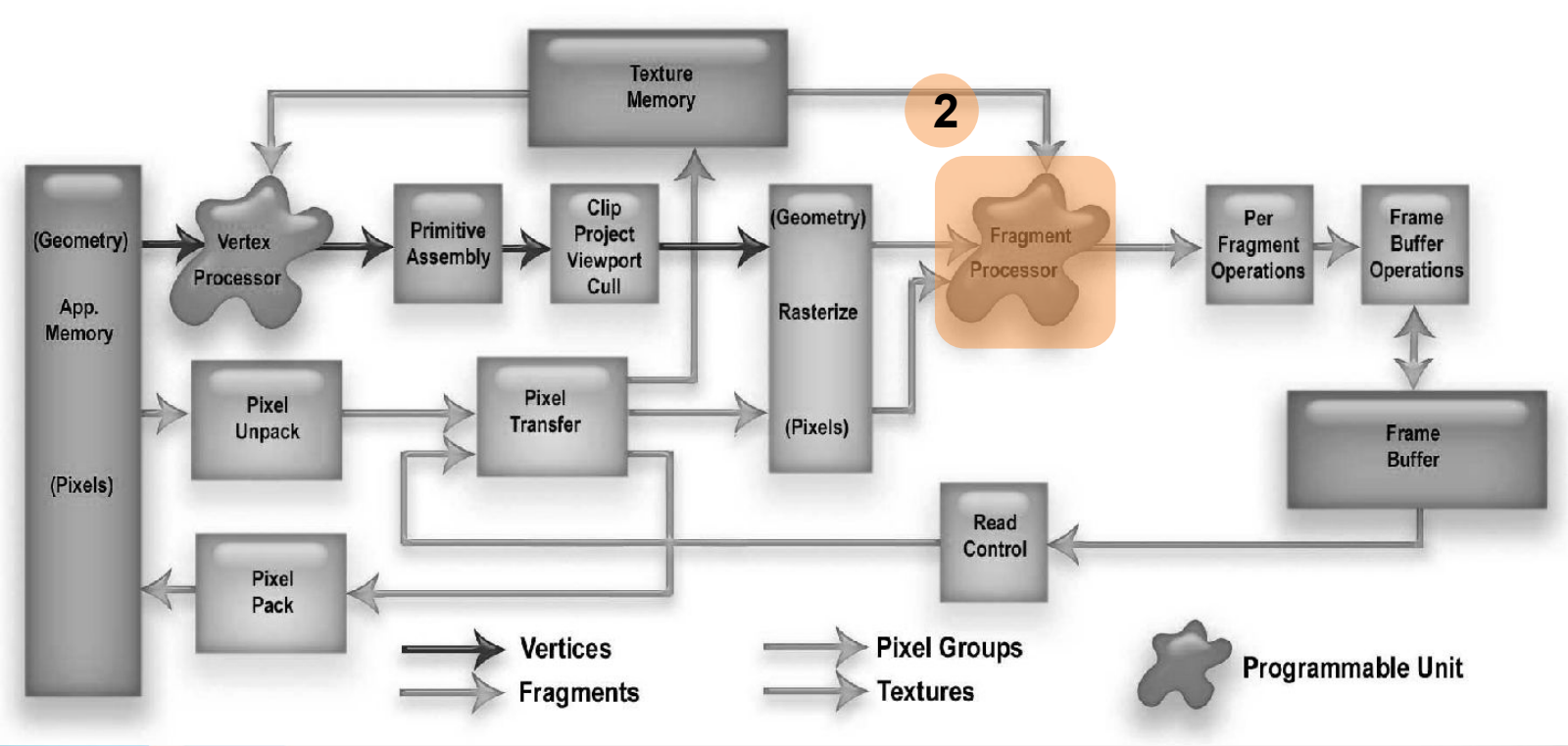
- A vertex program is executed for each vertex sent to render
- The usual tasks of the vertex program are:
  - Transform the vertex coords. (object space → clipping space)
  - Transform and normalize the normal of the vertex (eye space)
  - Compute the lighting of the vertex
  - Generate the texture coords. of the vertex
  - Transform the texture coords.

Vertex Shader

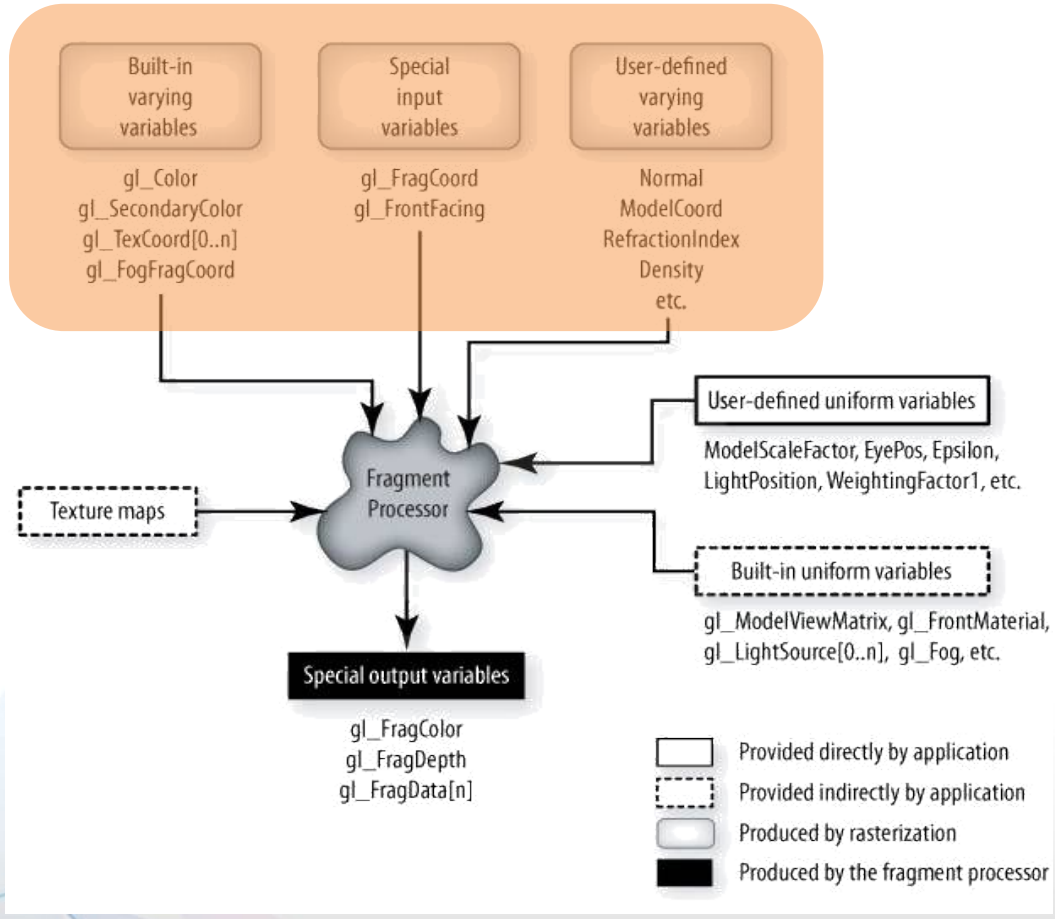
- **Example (WebGL)**

```
attribute vec3 aVertexPosition;
attribute vec4 aVertexColor;

uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;

varying vec4 vColor;

void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    vColor = aVertexColor;
}
```


Fragment Shader



Fragment Shader



Fragment Shader

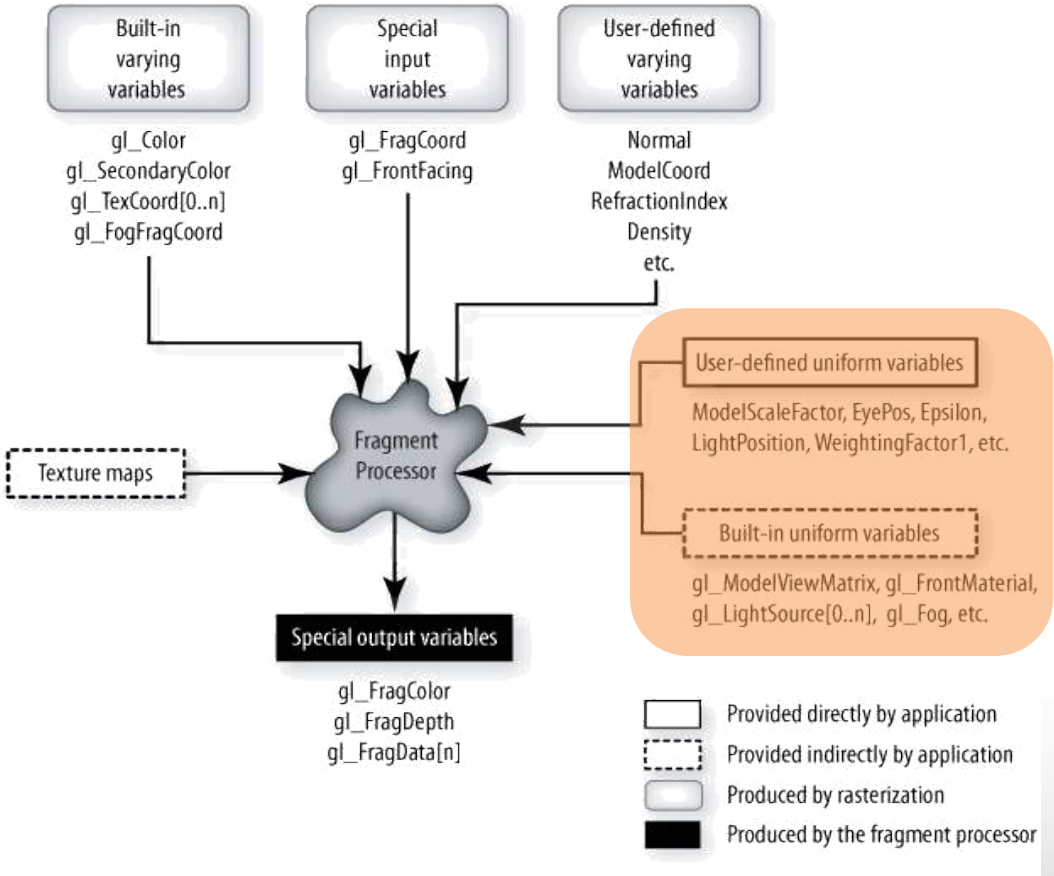
- *Varying* variables
  - Variables computed in the vertex shader and sent from the vertex program to the fragment program
  - Their values for each fragment are computed interpolating the values at each vertex of a primitive
  - Built-in variables:
    - varying vec4 glColor;
    - varying vec4 glTexCoord[];
    - …
  - They must be declared in WebGL!!

Fragment Shader

- ***Special input* variables**
  - **Computed automatically by OpenGL/WebGL**
  - **They can be read in the fragment shader**

```
vec4 gl_FragCoord;      // fragment coords in window space

bool gl_FrontFacing;    // true if the fragment belongs to a front face
```

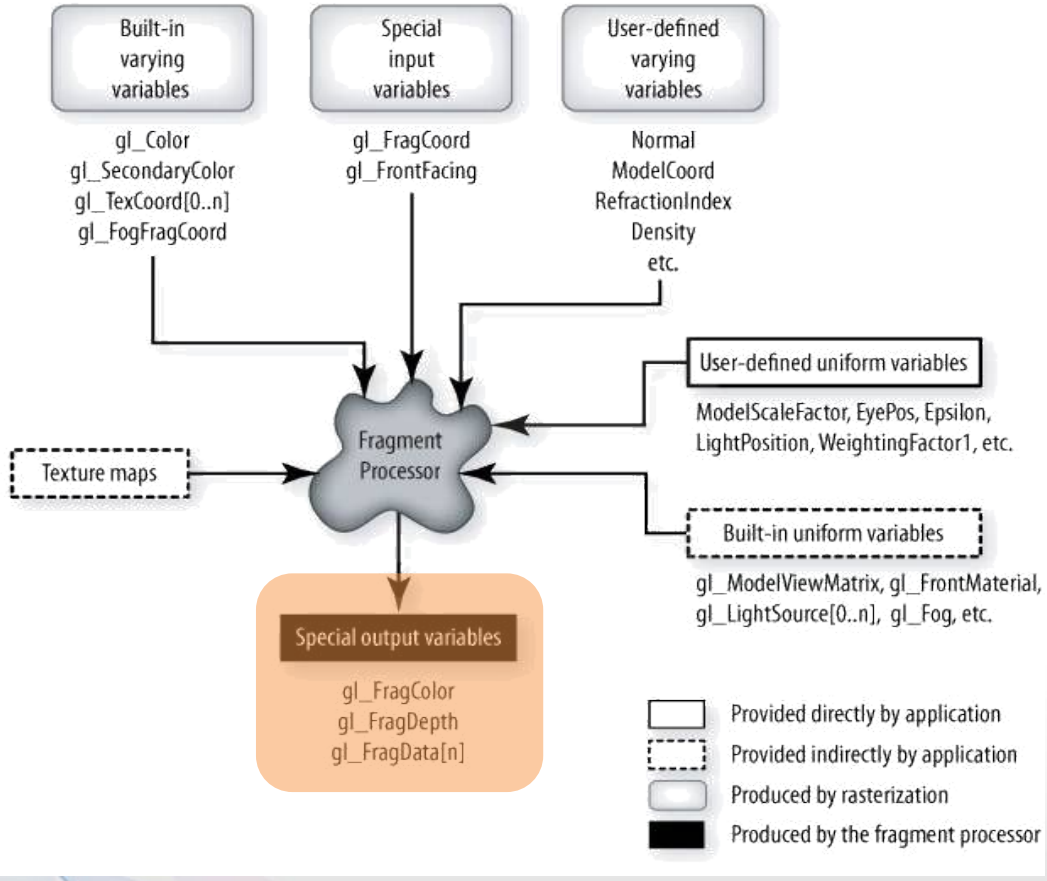

Fragment Shader



Vertex Shaders

- *Uniform* variables
  - Variables that normally have the same value for the vertices of a primitive
  - Built-in variables: OpenGL/WebGL specific variables
    - Access from within the shader with gl_ModelViewMatrix, gl_LightSource[0..n], …
    - They are the same for both vertex and fragment shaders!
    - They must be declared in WebGL!!
  - User-defined variables: we must declare them
    - Sent from the application by using glUniform, and linked with a name by using glGetUniformLocation
    - Access from within the shader with a user-defined name: eyePos, etc.

Fragment Shader

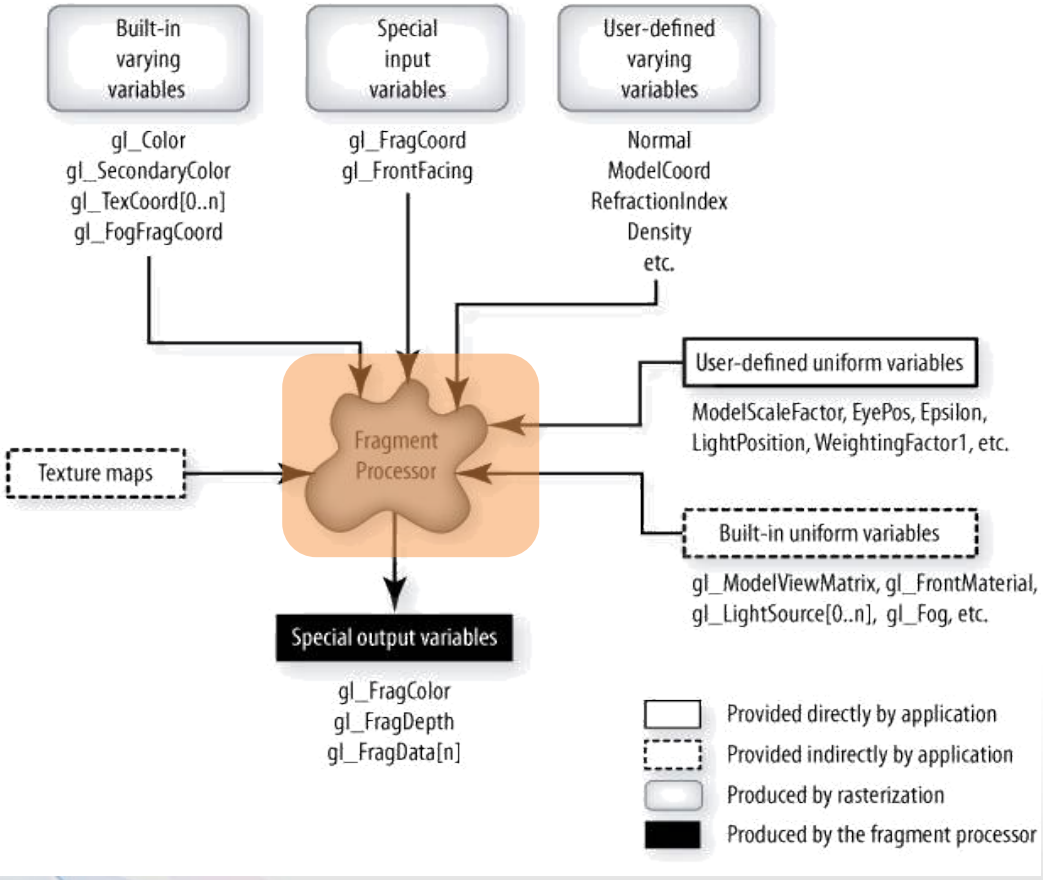


Fragment Shader

- ***Special output* variables**
  - **Variables that must compute the fragment program**

```
vec4 gl_FragColor;        // fragment color

float gl_FragDepth;       // depth of the fragment (depth-buffer)

vec4 gl_FragData[];       // used for MRT (glDrawBuffers, FBOs, …)
```

Fragment Shader



Fragment Shader

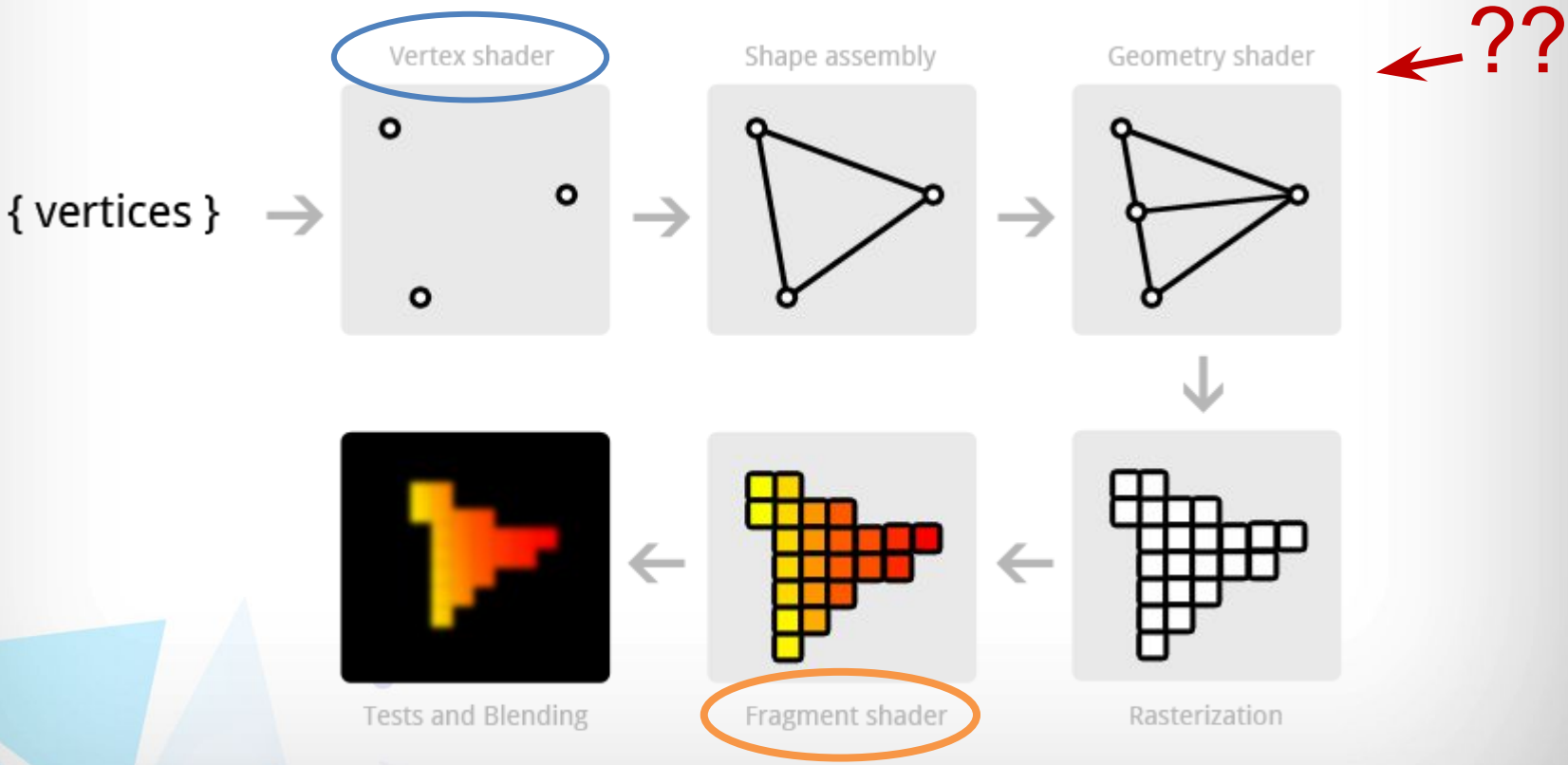
- A fragment program is executed for each fragment of each primitive
- The usual tasks of a fragment program are:
  - Texture access
  - Apply the color of the texture
  - Other effects (fog, normal mapping, etc.)
- Fragment programs can't:
  - Change fragment coordinates (but the depth of the fragment can be changed)
  - Access information of other fragments

Fragment Shader

- **Example (WebGL)**

```
precision mediump float;

varying vec2 vTextureCoord;

uniform sampler2D uSampler;

void main(void) {
    gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
}
```

To Sum Up…


??
Vertex shader
Shape assembly
Geometry shader
{ vertices }
Tests and Blending
Fragment shader
Rasterization

Questions?

www.citm.upc.edu