Objectives

- **WebGL Overview**
  - **webGL introduction**
  - **A simple example using webGL**
  - **Shader Programing using GLSL**

openGL

- **OpenGL**（英语：*Open Graphics Library*，译名：**开放图形库**或者“开放式图形库”）
- 是用于渲染2D、3D矢量图形的跨语言、跨平台的应用程序编程接口（API）。
- 由近350个不同的函数调用组成，用来绘制从简单的图形比特到复杂的三维景象。而另一种程序接口系统是仅用于Microsoft Windows上的Direct3D。
- OpenGL常用于CAD、虚拟现实、科学可视化程序和电子游戏开发。
- OpenGL规范由1992年成立的OpenGL架构评审委员会（ARB）维护。

openGL

- Derivation：openGL, openGL ES, webGL

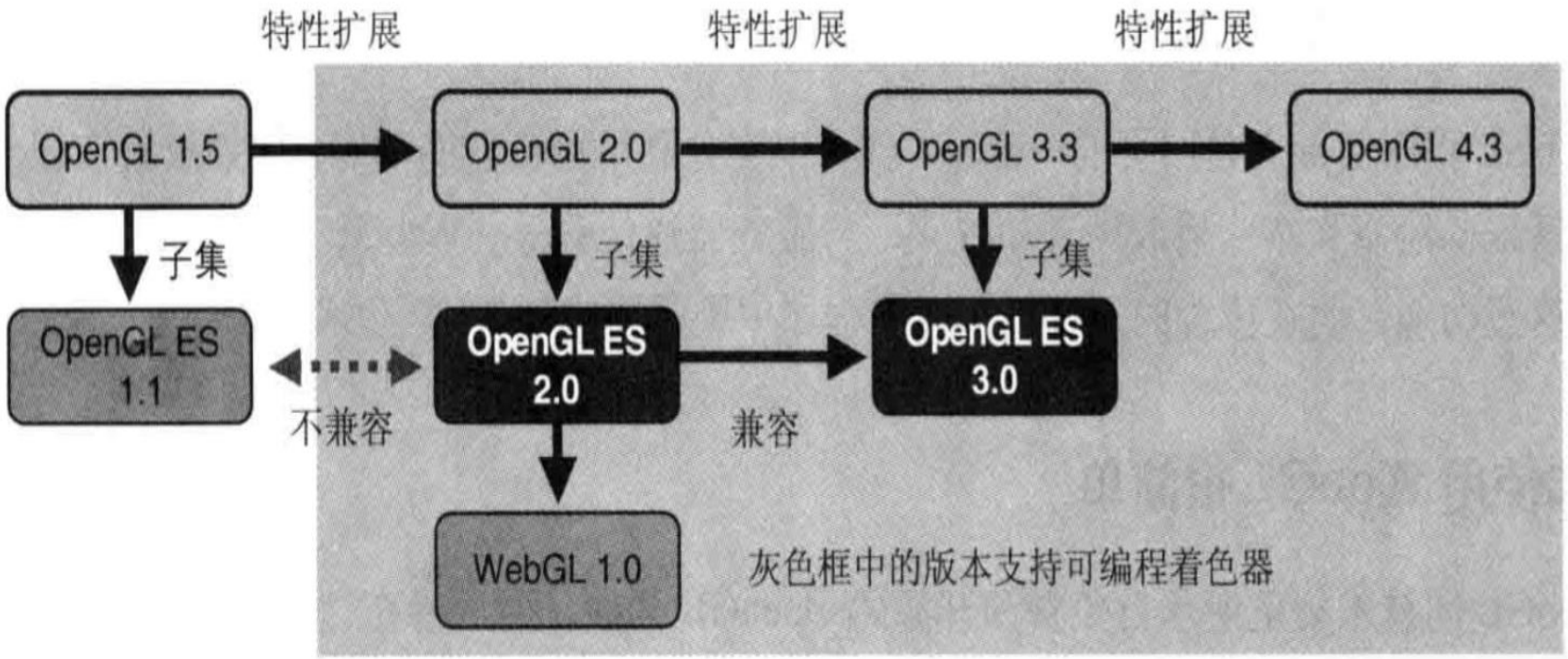


图 1.4 OpenGL、OpenGL ES 1.1//2.0/3.0 和 WebGL 之间的关系

webGL

- WebGL（Web Graphics Library）
  - 是一种3D绘图协议，这种绘图技术标准允许把JavaScript和OpenGL ES 2.0结合在一起，通过增加OpenGL ES 2.0的一个JavaScript绑定，
- Advantage：
  - webGL程序实际上是网页的一部分，可以充分利用浏览器的功能，如放置按钮，弹出对话框，播放声音和视频，与服务器通信等等，而在传统的三维图形应用程序中则需要你额外编写这些代码
  - WebGL可以为HTML5 Canvas提供硬件3D加速渲染，这样Web开发人员就可以借助系统显卡来在浏览器里更流畅地展示3D场景和模型了，还能创建复杂的导航和数据视觉化。
  - 显然，WebGL技术标准免去了开发网页专用渲染插件的麻烦，可被用于创建具有复杂3D结构的网站页面，甚至可以用来设计3D网页游戏等等。

WebGL program structure

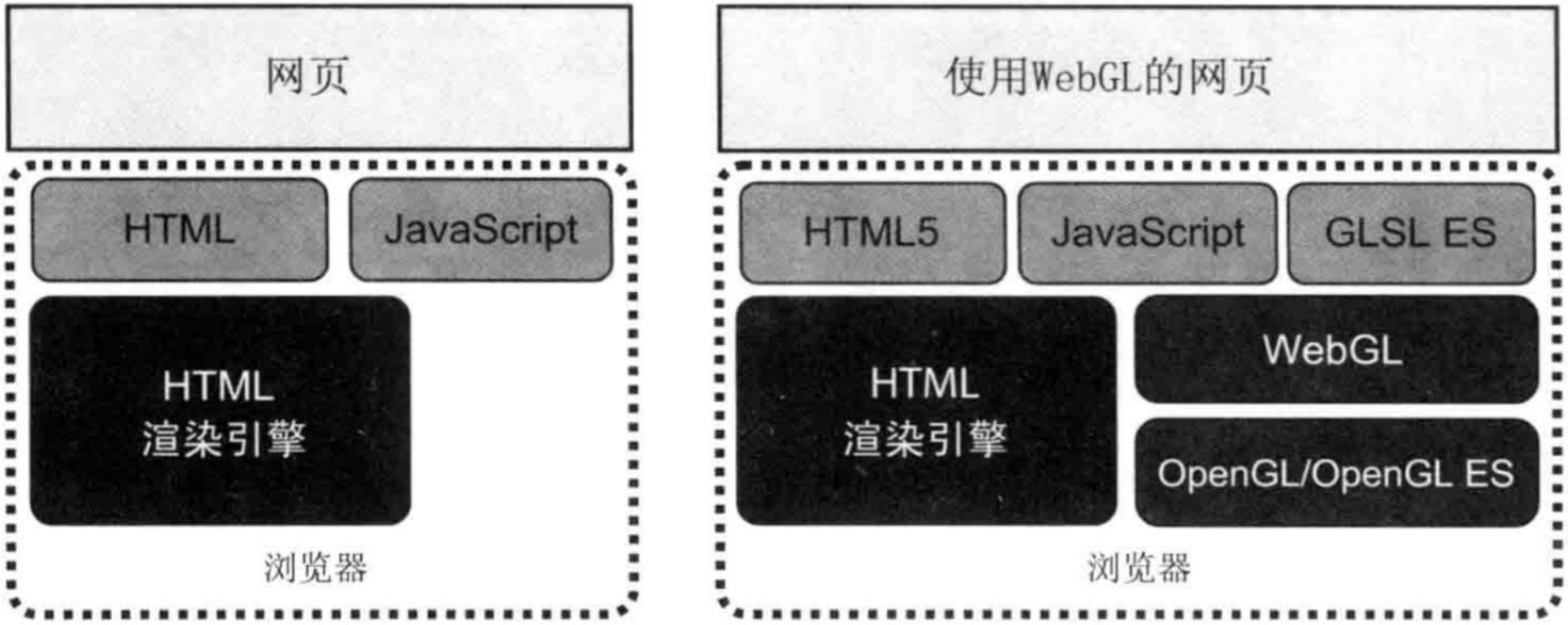
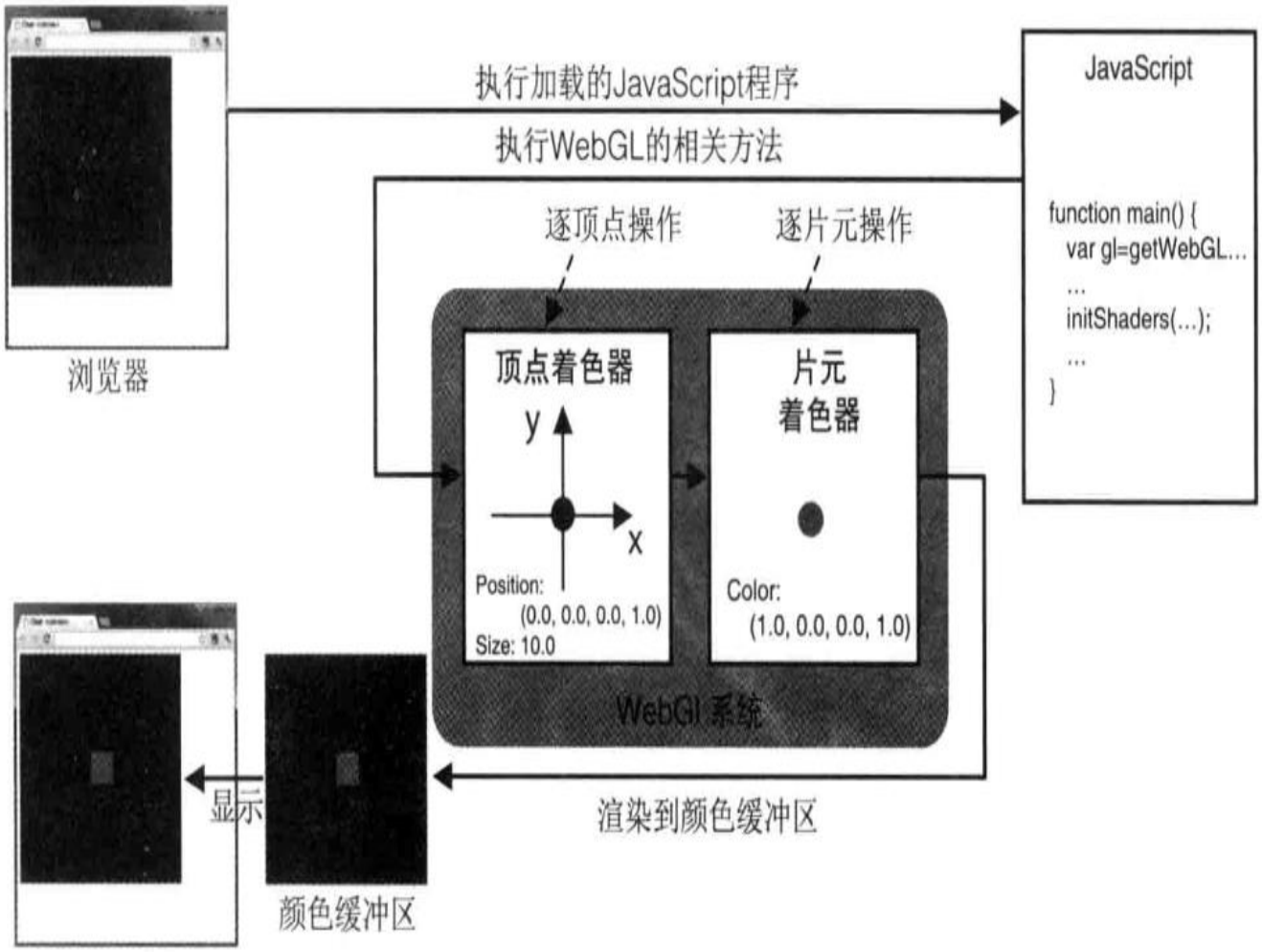


图 1.5 传统的动态网页（左侧）和 WebGL 网页（右侧）的软件结构

webGL的程序结构和执行框架



Three Languages

- HTML：
  - 超文本标记语言，标准通用标记语言下的一个应用。超文本标记语言的结构包括“头”部分（英语：Head）、和“主体”部分（英语：Body）。网页静态
- JAVASCRIPT：
  - 一种直译式脚本语言，解释器被称为JavaScript引擎，为浏览器的一部分，广泛用于客户端的脚本语言，用来给HTML网页增加动态功能。
- GLSL:
  - GLSL：OpenGL着色语言（OpenGL Shading Language）是OpenGL中用以进行着色编程的语言，开发人员用之编写短小自定义程序-着色器程序shader。
  - （着色器程序在图形卡的GPU（Graphic Processor Unit图形处理单元）上执行的，代替了固定的渲染管线的一部分，使渲染管线具有可编程性）
  - GLSL是使用C语言为基础的高阶着色语言，比较简单，从而避免了采用汇编语言或硬件规格语言编程的复杂性。

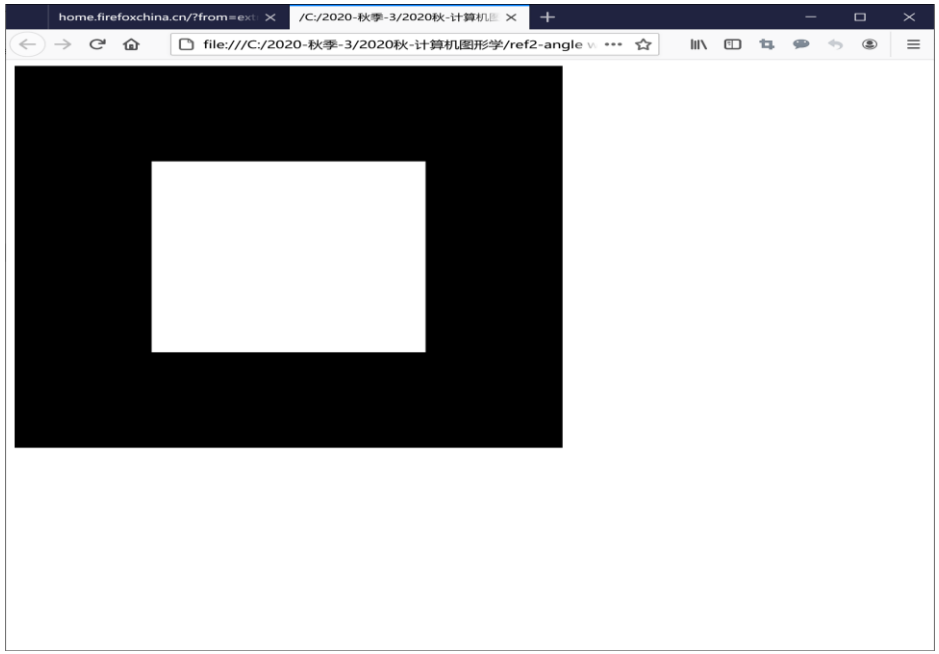
Objectives

- WebGL Overview
  - webGL introduction
  - A simple example using webGL
  - Shader Programing using GLSL

simpleAsquare

2020-秋季-3 › 2020秋-计算机图形学 › ref2-angle webgl › Chap1-simpleAsquare

名称	修改日期	类型	大小
square.html	2017/9/20 17:13	HTML 文件	2 KB
square.js	2020/9/19 17:57	JavaScript 文件	4 KB



SimpleASquare Files

- HTML Files
  - square.html(your codes here!)
- JS Files
  - **`square.js`** (your codes here!)
  - **`../Common/webgl-utils.js:`**
    - Standard utilities for setting up WebGL context in Common directory on website
  - **`../Common/initShaders.js:`**
    - contains JS and WebGL code for reading, compiling and linking the shaders
  - **`../Common/MV.js:`** Angel's matrix-vector package

编写程序Five steps

square.html

- Describe page (HTML file)
  - *Request webGL Canvas*
  - *read in necessary files*
- Define shaders (HTML file)
  - *could be done with a separate file (browser dependent)*

square.js

-Compute or specify data (JS file)

- Send data to GPU (JS file)

- Render (JS file)

Step1:Describe page (HTML file)

- *read in necessary files（公共库文件等）*
- *Create canvas :网页“画布”*
  - canvas只能绘制2D图,结合webGL,才可以绘制3D图形

```
<!Standard utilities for setting up WebGL context in Common directory on website>
<script type="text/javascript" src="../Common/webgl-utils.js"></script>

<!contains JS and WebGL code for reading, compiling and linking the shaders>
<script type="text/javascript" src="../Common/initShaders.js"></script>

<!Angel matrix-vector package>
<script type="text/javascript" src="../Common/MV.js"></script>

<!the application file, 需要自己编写的js脚本代码>
<script type="text/javascript" src="square.js"></script>
</head>
```

```
<body>
    <canvas id="gl-canvas" width="512" height="512">
    Oops ... your browser doesn't support the HTML5 canvas element
    </canvas>
</body>
```

Step2: Specify data（JS file)

- *configure webGL and load shaders*

```
/* onload 的init函数相当于主程序,执行代码的入口,
onload: determines where to start execution determines where to start execution
配置环境等参数，canvas,data,buffer,shader varibles,并调用render绘图*/
window.onload = function init(){
    //获取画布ID，创建GL环境。canvas gets WebGL context from HTML file
    var canvas = document.getElementById( "gl-canvas" );
    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    //配置WEBGL的绘图环境（设置一些初始参数：视口，背景色）Configure WebGL
    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 0.0, 0.0, 0.0, 1.0 );

    //装载shader，并使用程序容器
    //Load shaders
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );
```

- *Specify Object data*

```
//定义场景中的景物顶点数据，Four Vertices,vertices use vec2 type in MV.js
var vertices = [
    vec2( -0.5, -0.5 ),
    vec2(  -0.5,  0.5 ),
    vec2(  0.5, 0.5 ),
    vec2( 0.5, -0.5)
];
```

Step3:Send data to GPU(JS file)

//采用缓冲区发送数据给GPU

1）创建顶点缓存对象VBO(顶点缓存对象)

```
var buffer=gl.createBuffer() ;
```

2）绑定顶点缓存对象VBO,作为当前缓冲区

```
gl.bindBuffer(gl.ARRAY_BUFFER,buffer);
```

3）将顶点数据points写入VBO

```
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices),gl.STATIC_DRAW);
```

4）定义顶点属性变量vPosition用以获取shader顶点变量vPosition的索引

```
var vPosition=gl.getAttribLocation(program,"vPosition");
```

5）将当前缓冲区对象分配给（匹配）顶点属性变量vPosition

```
gl.vertexAttribPointer(vPosition,-,-,false,0,0);
```

6）开启（使能）顶点属性变量vPosition

```
gl.enableVertexAttribArray(vPosition)
```

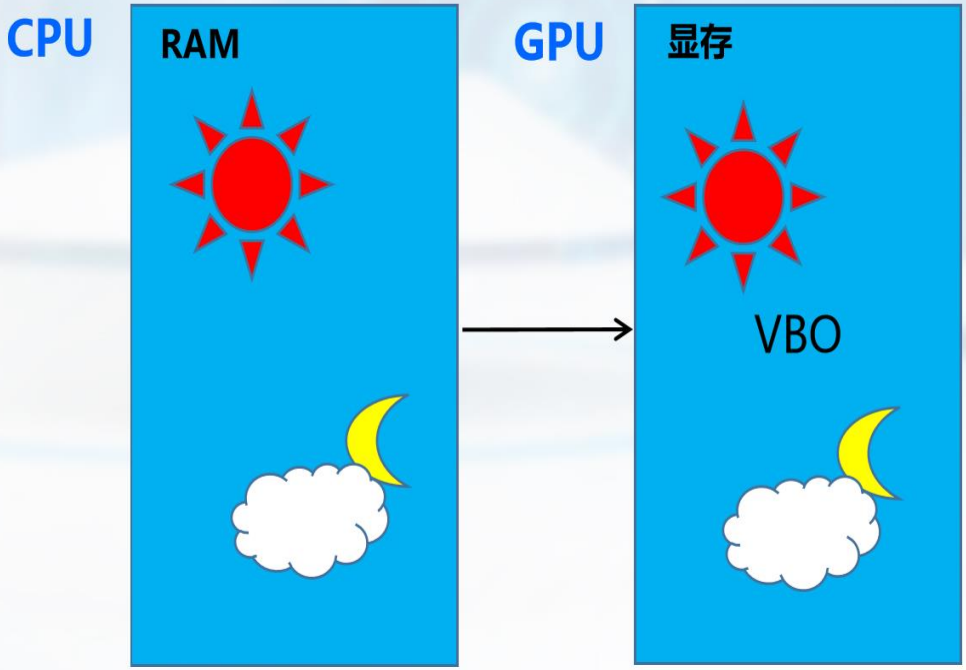
//MV.Js里的函数***flatten(); to convert JS array to an array of float32's***

VBO(顶点缓冲区对象）

VBO

VBO(Vertex Buffer Object)顶点缓冲区对象，主要用来存储顶点的各种信息。

好处：模型的顶点信息放进VBO，这样每次画模型时， 数据不用再从CPU的势力范围内存里取，而是直接从GPU的显存里取，从而提高效率。



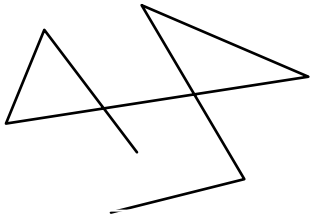
Step4: Render(JS file)

```
        render(); //调用自己写的渲染函数(绘制函数)
};

function render() {
   //清屏，用clearcolor设置的颜色填充颜色缓存（帧缓存）
   gl.clear( gl.COLOR_BUFFER_BIT );

  //用发送给GPU的数据-四个顶点绘制两个三角形-三角扇图元
  gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 );    // 0, 1 , 2, 3
}
```

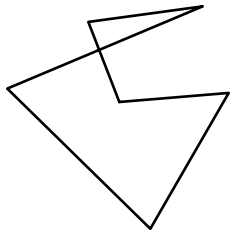
WebGLPrimitives图元

- webGL面图元：点，线，面

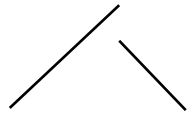
GL_POINTS



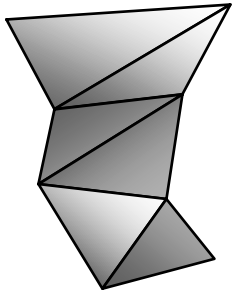
GL_LINE_STRIP



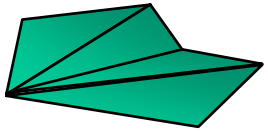
GL_LINE_LOOP



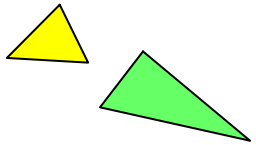
GL_LINES



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN



GL_TRIANGLES

Triangularization三角化

- webGL只提供三角形的面图元
- 其它Convex polygon凸面体表面可表示为多个三角形面组合：

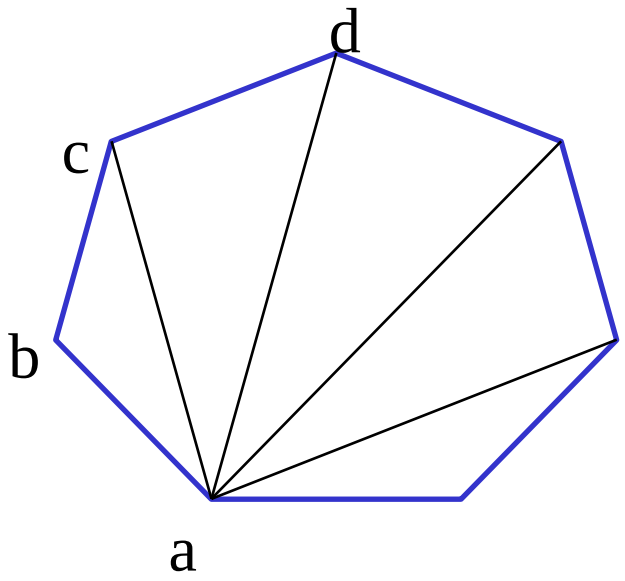
➢三角扇**GL_TRIANGLE_FAN**

➢三角形集合**GL_TRIANGLES**

➢三角带**GL_TRIANGLE_STRIP**

例如：画一个矩形：abcd四个顶点，由abc，acd两个三角形构成三角扇表示

```
gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 );
```



GL_TRIANGLE_FAN

Start with abc,
remove b, then acd,
….

step5.Define shaders (HTML file)


```
//顶点着色器代码
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
void main()
{
    gl_Position = vPosition;
}
</script>

//片元着色器代码
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
void main()
{
    gl_FragColor = vec4( 1.0, 1.0, 1.0, 1.0 );
}
</script>
```

Note:We assign names to the shaders that we can use in the JS file

Shader着色器

- 用GLSL语言编写的着色器程序shader
- 是在GPU上解释执行的程序，是可编程流水线中的处理模块
  - ✓Fragment shader:逐顶点操作，对每个顶点都执行的相同代码
  - ✓Fragment shader:逐片元操作，对每个片元执行的相同代码

Character：Data flow model& shader render

- Application应用程序:只将顶点属性数据等打包发送给GPU,并告知画什么图元。在CPU上执行。
- Shader着色器:实现图形处理的具体操作代码。在GPU上执行。

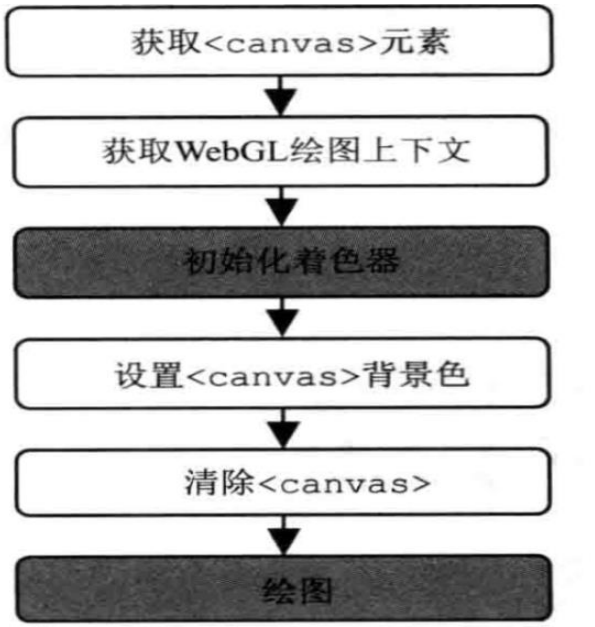
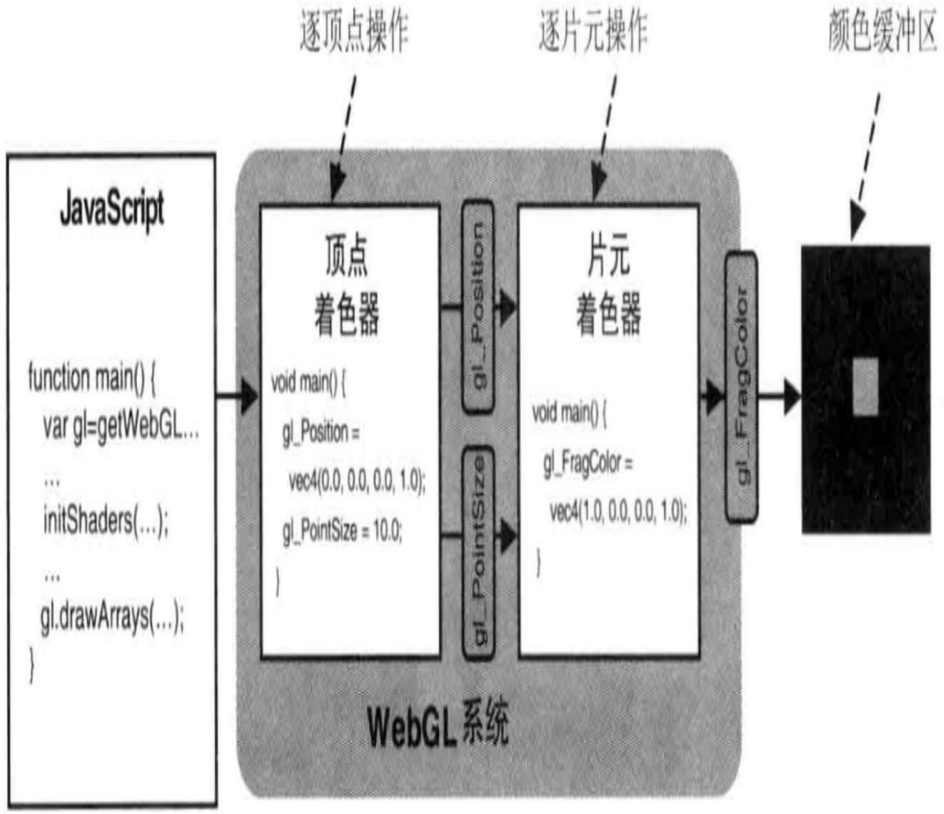


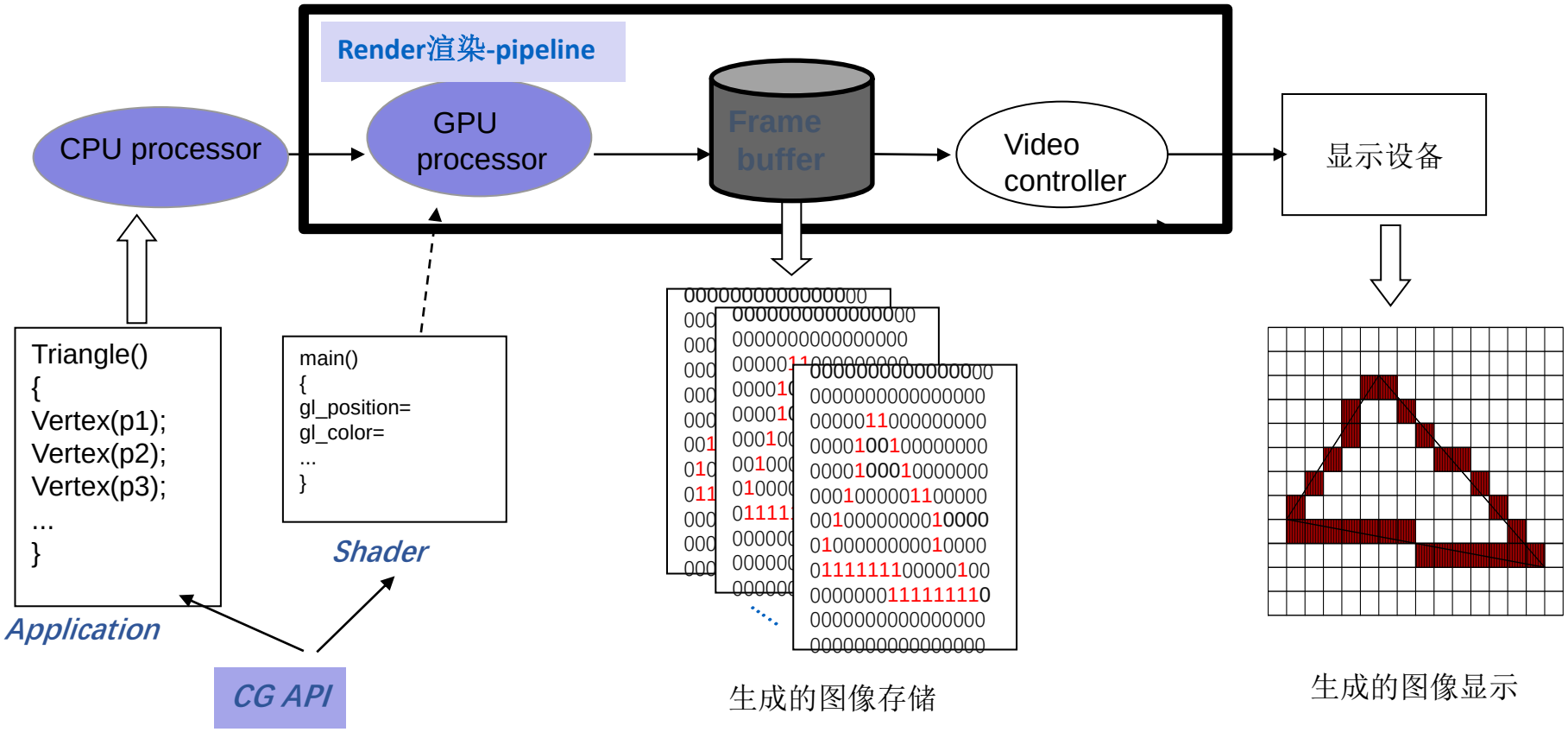
图 2.13 WebGL 程序的执行流程



Render sketch map

Performance is achieved by using GPU rather than CPU

- *Application's job is to send data to GPU, GPU does all rendering,*
- Control GPU through programs called shaders



Objectives

- **WebGL Overview**
  - **webGL introduction**
  - **A simple example using webGL: square**
  - **Shader Programming using GLSL**

着色器语言SL（Shading Language）

- Cook的着色树
- Pixar的Renderman
- openGL的着色器语言GLSL
  - 跨平台性(各种桌面操作系统，甚至移动平台OS)
  - openGL3.1开始全面支持可编程管线编程
    - Totally shader-based
      - ***No default shaders***
      - ***Each application must provide both a vertex and a fragment shader***
  - 着色器代码shader以文本文件形式存在，CPU中不编译，在程序运行时由GPU的显卡驱动对它进行翻译。

着色器编程语言GLSL

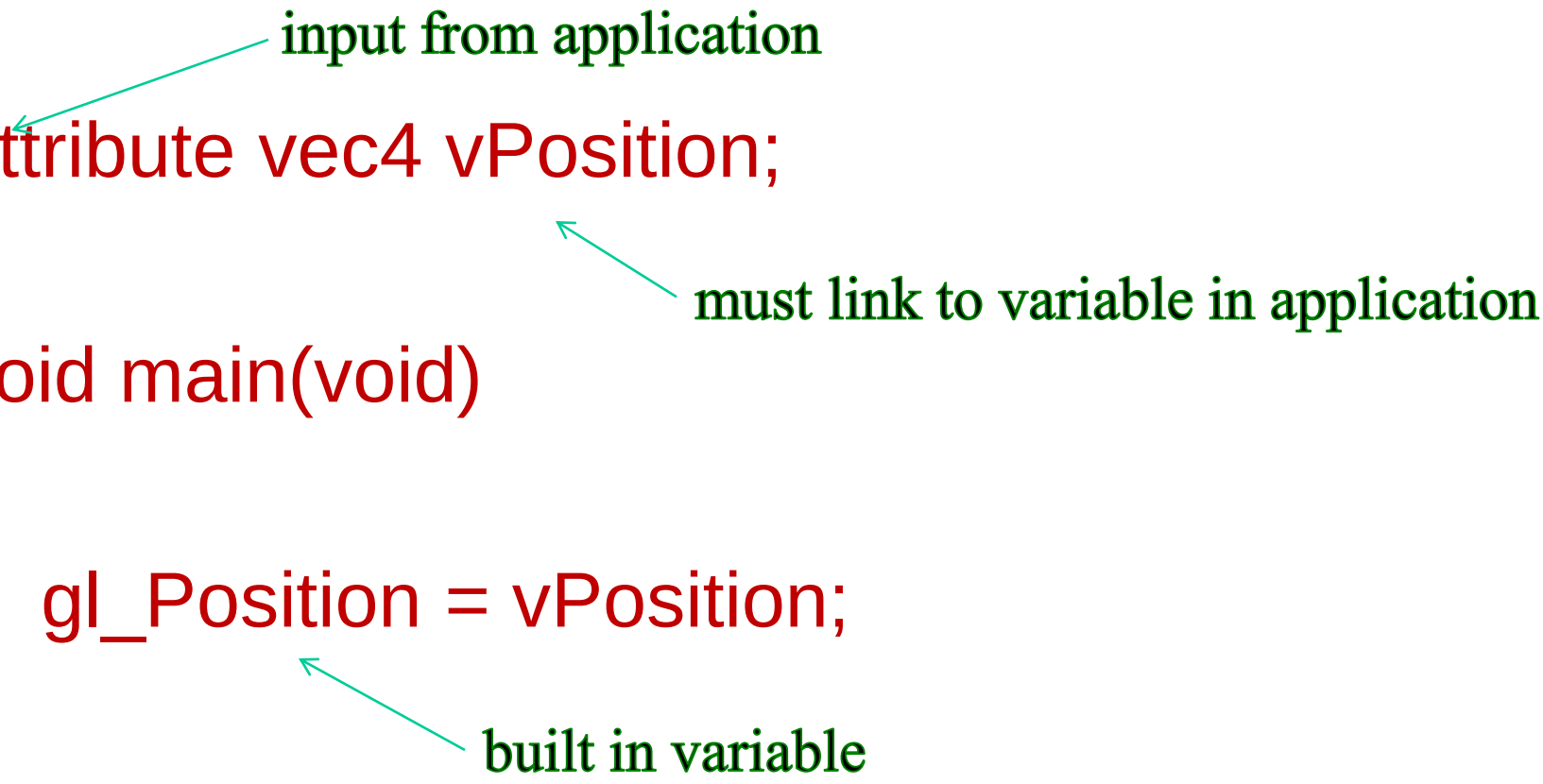
- OpenGL Shading Language: GLSL
- C-like with
  - Matrix and vector types (2, 3, 4 dimensional)
  - Overloaded operators
  - C++ like constructors
- Similar to Nvidia's Cg and Microsoft HLSL
- Code sent to shaders as source code
- As of OpenGL 3.1, application must provide shaders

1.Trivial pass-through Shaders

- These are trivial pass-through shaders (do nothing)

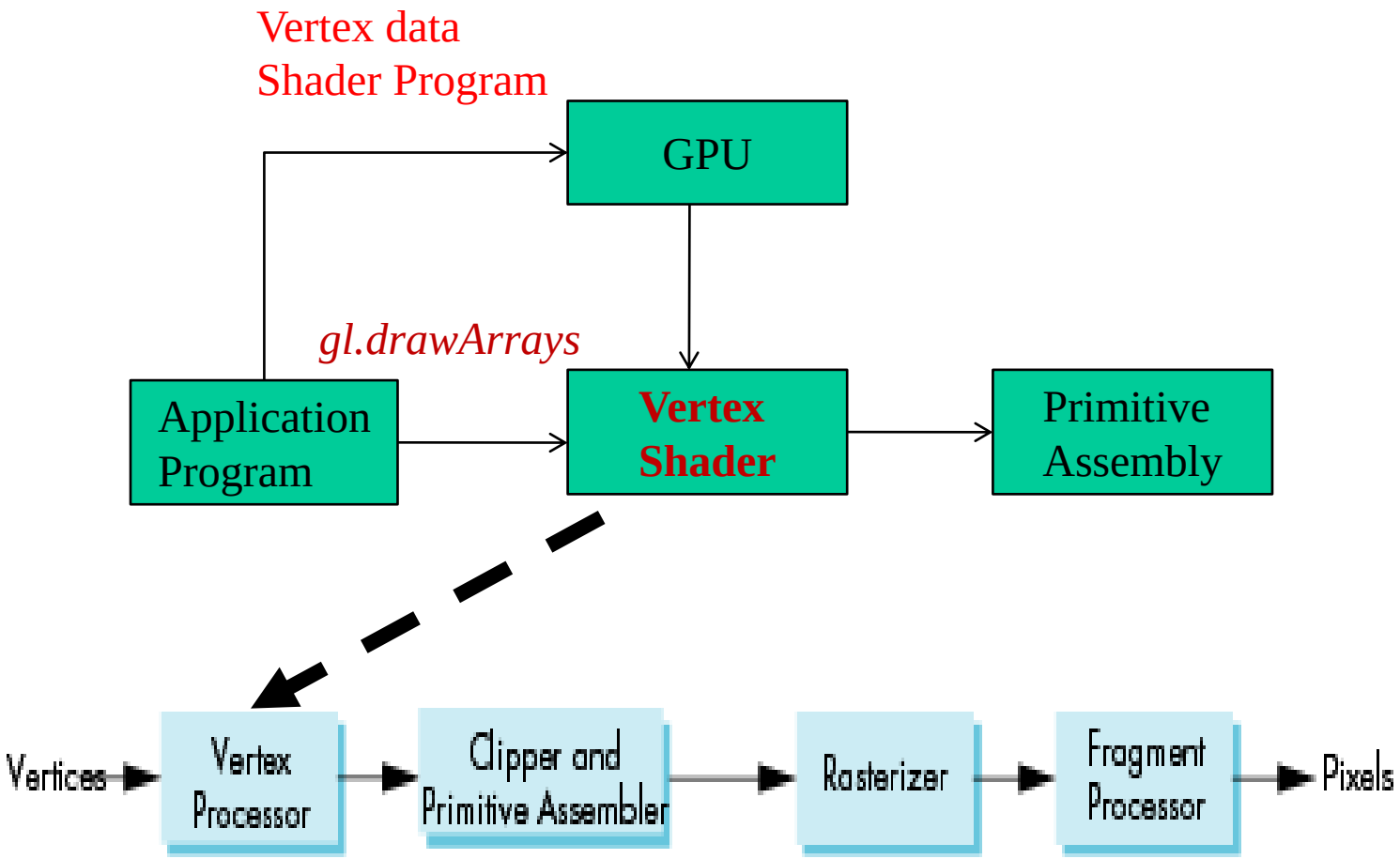
- which set the two required built-in variables
  - gl_Position
  - gl_FragColor

Simple Vertex Shader



Execution Model

- Vertex shader

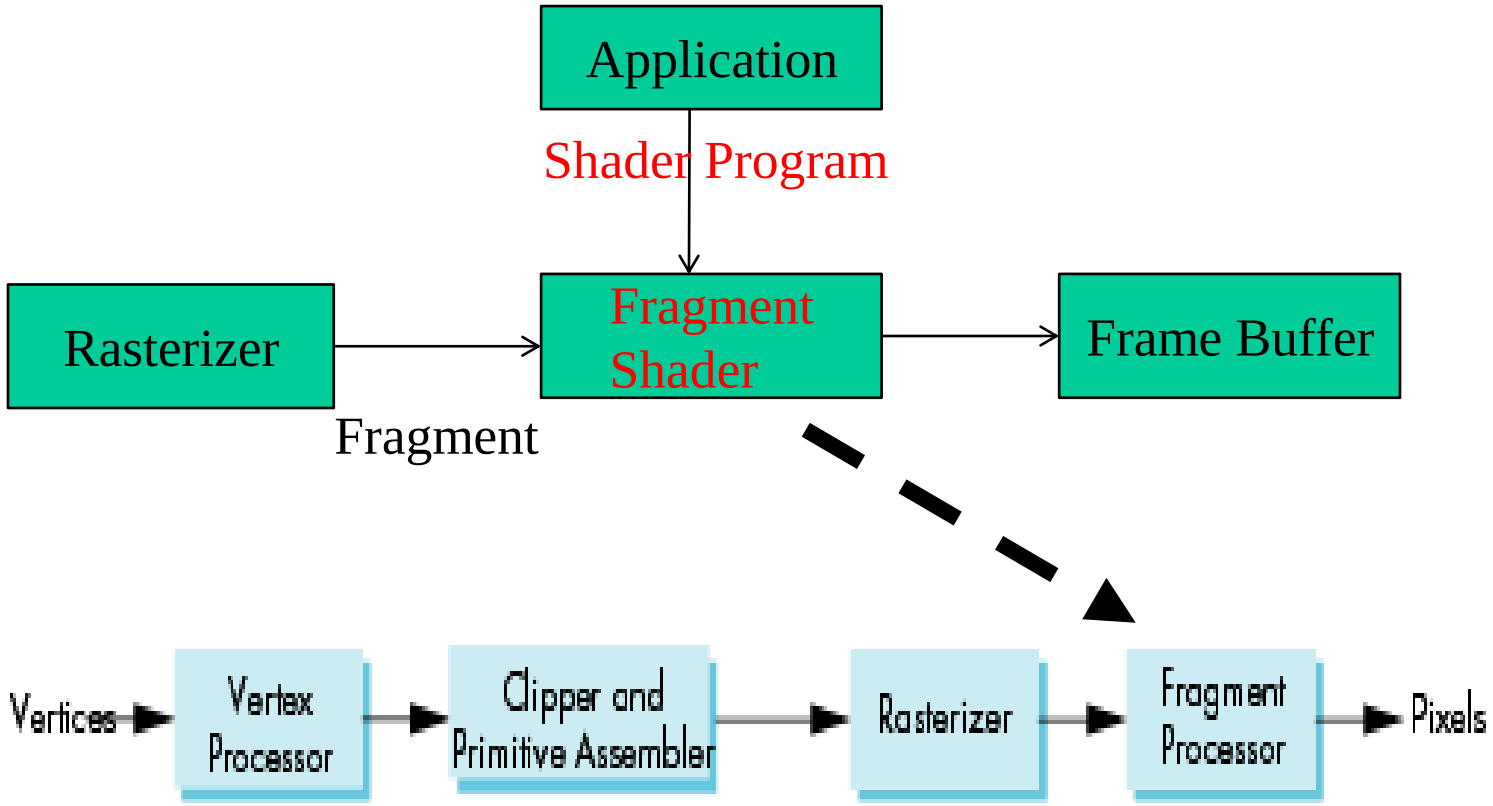


Simple Fragment Program

```
precision mediump float;
void main(void)
{
   gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);

}
```

Execution Model

- Fragment shader

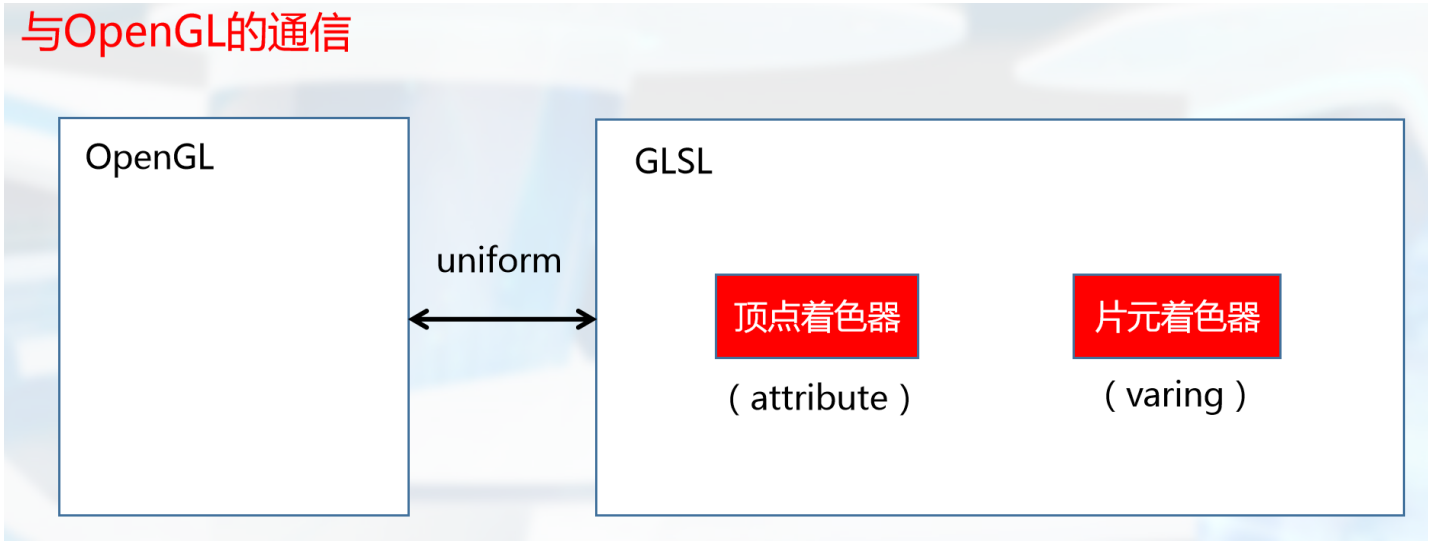


Simple Fragment Program

```
//In GLSL for WebGL we must specify desired precision in fragment shaders
//Precision Declaration
precision mediump float;

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

2. Qualifiers限定符

- GLSL has many of the same qualifiers such as **`const`** as C/C++
- Qualifiers(后面详解)高版本用in,out,inout
  - Attribute
  - Varying
  - Uniform



Our Naming Convention(命名约定)

- Attributes variables passed to vertex shader have names beginning with v (v Position, vColor) in both the application and the shader
  - can have the same name, but note that these are different entities with the same name
- Variable variables(Varying Qualified) begin with f (fColor) in both shaders
  - *must* have same name
- Uniform variables are unadorned（命名不限）and in both the application and the shader
  - can have the same name

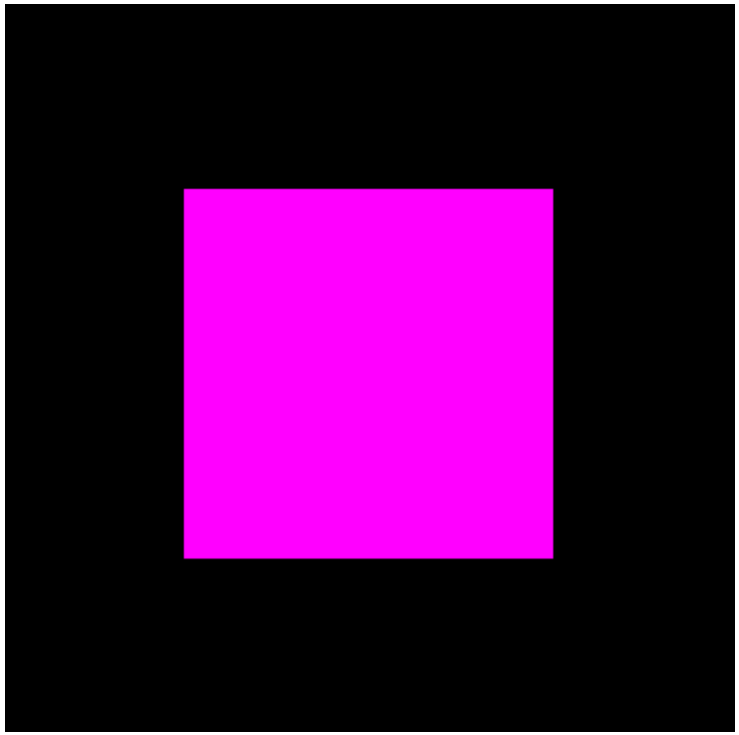
Setting Colors

- 图形颜色直接指定，可用以下三种编程方式：
  - Fragment color: can alter via fragment shader code~Example1
  - Application color: pass to vertex shader as a uniform variable or as a vertex attribute~Example2
  - Vertex shader color: pass to fragment shader as varying variable~Example3

Example1: set color in fragment shader

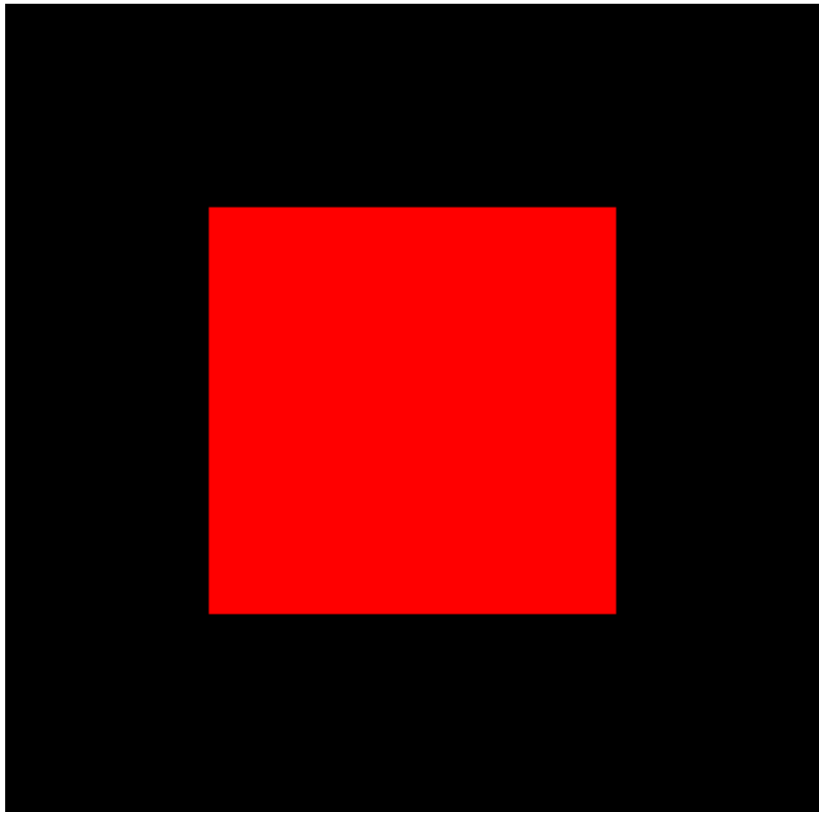
- 直接在片元着色器中对图形赋予颜色

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



Example2:Sending a Uniform Variable

APP中定义了一种颜色，作为uniform变量传递给片元着色器，则该图元用单色绘制



Example2:Sending a Uniform Variable

- APP中定义了一种颜色并传给片元着色器

Application

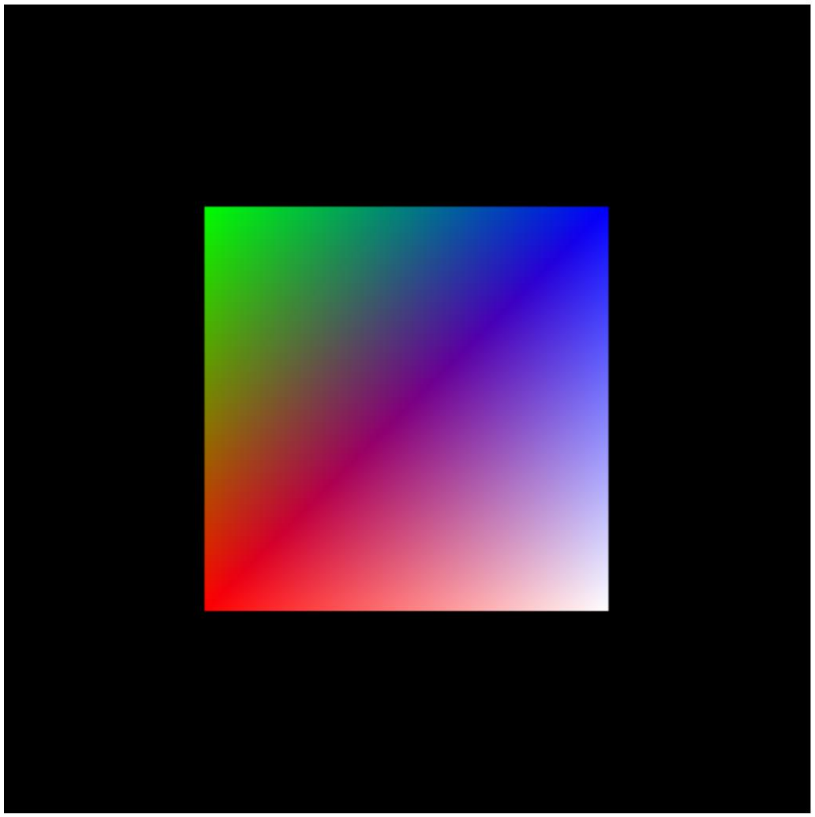
```
// 传递uniform常量给片元着色器，即该图元的每个片元颜色相同
var color = vec4(1.0, 0.0, 0.0, 1.0);
colorLoc = gl.getUniformLocation( program, "color" );
gl.uniform4fv( colorLoc, color);
```

Fragment Shader

```
// fragment shader
uniform vec4 color;
void main()
{
    gl_FragColor = color;
}
```

Example3: sending VBO

- APP中定义每个顶点颜色，然后以VBO形式发送给顶点着色器，再插值后传给片元着色器作为片元颜色。



Example3: sending VBO

Step1:在APP中指定每个顶点颜色，
然后以VBO顶点缓存对象方式发送给顶点着色器

```
//作为顶点的属性赋给每个顶点，以颜色缓存形式发送给顶点着色器var
vertextColors = [
    vec4( 1.0,0.0,0.0,1.0),
    vec4( 0.0,1.0,0.0,1.0),
    vec4( 0.0,0.0,1.0,1.0),
    vec4( 1.0,1.0,1.0,1.0)
    ];
var cBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(vertextColors),  gl.STATIC_DRAW );

var vColor = gl.getAttribLocation( program, "vColor" );
gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vColor );
```

注意:JS中变量可以和shader中的变量同名

Example3: sending VBO

step2.顶点颜色需要从顶点着色器输出，经过插值计算后，作为输入给片元着色器

```
Attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;
void main()
{
  gl_Position = vPosition;
  fColor = vColor;
}
```

Vertex Shader

```
precision mediump float;
varying vec4 fColor;
void main()
{
  gl_FragColor = fColor;
}
```

Fragment Shader

两着色器中的Varying变量必须具有相同的名字

3.Data Types

- C types:
  - int, float, bool
- Vectors:
  - float vec2, vec3, vec4
  - Also int (ivec) and boolean (bvec)
- Matrices: mat2, mat3, mat4
  - Stored by columns
  - Standard referencing m[row][column]
- C++ style constructors
  - vec3 a =vec3(1.0, 2.0, 3.0)
  - vec2 b = vec2(a)

No Pointers

- There are no pointers in GLSL
- We can use C structs which

  can be copied back from functions
- matrices and vectors are basic types, they can be passed into and output from GLSL functions, e.g.

  mat3 func(mat3 a)
- variables passed by copying

4.Operators and Functions

- Standard C functions
  - Trigonometric（三角函数）
  - Arithmetic（算术）
  - Normalize, reflect, length
- Overloading of vector and matrix types

```
mat4 a;
vec4 b, c, d;
c = b*a; // a column vector stored as a 1d array
d = a*b; // a row vector stored as a 1d array
```

Swizzling and Selection

- Can refer to array elements by element using [] or selection (.) operator with
  - x, y, z, w
  - r, g, b, a
  - s, t, p, q
  - **`a[2], a.b, a.z, a.p`** are the same
- **Swizzling** operator lets us manipulate components（多个通道）

```
vec4 a, b;
a.yz = vec2(1.0, 2.0, 3.0, 4.0);
b = a.yxzw;
```

5.Linking Shaders with Application

➢ ***../Common/initShaders.js:***

contains JS and WebGL code for reading, compiling and linking the shaders

- Create a program object
- Read shaders
- Compile shaders
- Link everything together

- Link variables in application with variables in shaders
  - *Vertex attributes*
  - *Uniform variables*

Program Object

- Container for shaders
  - Can contain multiple shaders
  - Other GLSL functions

```
var program = gl.createProgram();

gl.attachShader( program, vertShdr );
gl.attachShader( program, fragShdr );

gl.linkProgram( program );
```

Read and Compile shaders

Example: Adding a Vertex Shader

```
var vertShdr;
var vertElem =  document.getElementById( vertexShaderId );

vertShdr = gl.createShader( gl.VERTEX_SHADER );
gl.shaderSource(vertShdr, vertElem.text );//reading a shader
gl.compileShader(vertShdr );

// after program object created
gl.attachShader( program, vertShdr );
```

QQ群学习例程

- ref1:《wegGL编程指南》书及例程
- ref2: “angel教程的例程”

在 Three.js、Oak3D、PhiloGL 等一批图形库的“引诱”下，很多人放弃了基础知识，直接开始操控这些成熟的 WebGL 3D 引擎。其中有的人成功了，但是据我了解大部分的人都在初期的风光得意之后，又重新陷入了泥潭，于是不得不再次回到学习 WebGL 原生 API 的道路上；而那些一直坚持学习 WebGL 原生 API 的人，在经历了一开始的艰苦岁月，战胜了面对他人突飞猛进而自己仍在画三角形的挫败感之后，现在已经成为了 HiWebGL 社区中的中流砥柱。因此在看到这本书后，我十分愿意并有些许兴奋地向广大 WebGL 学习者推荐本书，你可以在流畅的文字描述、大量详实的图例图解中，游刃有余地在 WebGL 原生 API 中斩荆披棘，不断前进。这种感觉不再是焦躁不安，而是让我想起了上大学时的青葱岁月，也希望你能在阅读中获得不一样的新的学习体验。

——郝稼力

最大的 HTML5&WebGL 中文社区创始人，

国内第一个 WebGL 商用网站 Lao3D.com 创始人

网上参考学习资料

中文简明教程

- html教程：https://www.w3school.com.cn/html/index.asp
- js教程：https://www.w3school.com.cn/js/index.asp
- webGL教程：https://webglfundamentals.org/webgl/lessons/zh_cn/
- GL函数查询：https://developer.mozilla.org/zh-CN/
- GLSL语言基础：**https://learnopengl-cn.readthedocs.io/zh/latest/04%20Advanced%20OpenGL/08%20Advanced%20GLSL/**

官方网站

- https://www.khronos.org/webgl/
- https://www.khronos.org/registry/OpenGL-Refpages/
- https://www.khronos.org/registry/OpenGL/index_gl.php
- https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.60.html#introduction

查WebGL的函数

- https://developer.mozilla.org/zh-CN/

使用方法:右上角搜索框内输入函数名

Khronos Group

- Khronos Group团队成立于 2000 年 1 月，由包括 3Dlabs, ATI, Discreet, Evans & Sutherland, Intel, Nvidia, SGI 和 Sun Microsystems 在内的多家国际知名多媒体行业领导者创立，
- 致力于发展开放标准的应用程序接口 API，以实现在多种平台和终端设备上的富媒体创作、加速和回放。