Outlines

- GL History
- GL Architecture and Functions Formats
- A Standard Program Structure using WebGL
- Shader Programing using GLSL
- Animation and Interaction using WebGL

GL History

- Early History of APIs
  - Graphical Kernel System (GKS)
    - 2D but contained good workstation model
    - GKS adopted as IS0 and later ANSI standard (1980s)
    - GKS not easily extended to 3D (GKS-3D), Far behind hardware development
  - Core：Both 2D and 3D
  - Programmers Hierarchical Graphics System (PHIGS)
    - Arose from CAD community
    - Database model with retained graphics (structures)
  - X Window System
    - DEC/MIT effort
    - Client-server architecture with graphics
  - PEX combined the two PHIGS and X
    - Not easy to use (all the defects of each)

GL History (cont.)

➢Early History of APIs(cont.)

- Silicon Graphics (SGI)
  - SGI（Silicon Graphics ）
  - revolutionized the graphics workstation by implementing the pipeline in hardware (1982)

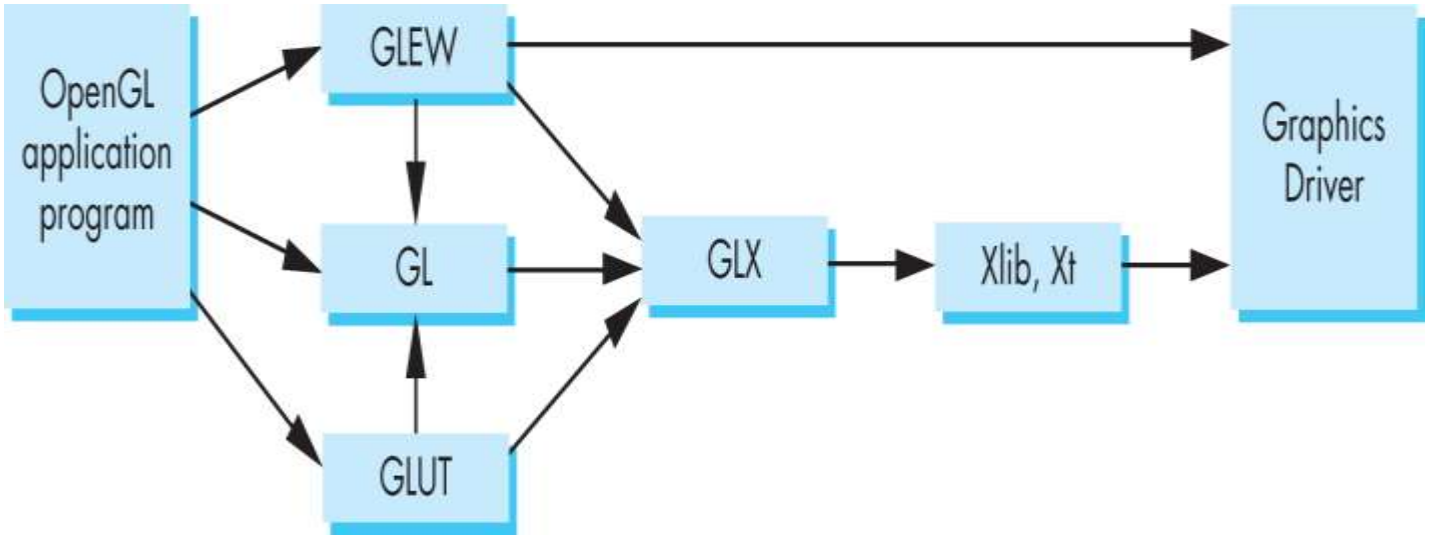
- GL
  - application programmers can access the system
  - With GL, it was relatively simple to program three dimensional interactive applications

GL History (cont.)

- Modern History of APIs
  - OpenGL
    - The success of GL lead to OpenGL (1992), "a platform-independent API ",
      - ***Easy to use***
      - ***Close enough to the hardware to get excellent performance***
      - ***Focus on rendering, Omitted windowing and input to avoid window system dependencies***
    - Originally controlled by an Architectural Review Board (ARB)(Members included SGI, Microsoft, Nvidia, HP, 3DLabs, IBM,…….), Now Kronos Group
      - **Was relatively stable稳定 (through version 2.5)**
        - **Backward compatible**
        - **Evolution reflected new hardware capabilities**
          - 3D texture mapping and texture objects
          - Vertex and fragment programs……
      - **Allows platform specific features through extensions**

GL History (cont.)

- Modern History of APIs
  - OpenGL(cont.)
    - Software Organization
      - ***GL:核心库, GLEW:扩展, GLUT:窗口管理***



GL History (cont.)

- Modern History of APIs(cont.)
  - WebGL
    - Derivation：OpenGL, OpenGL ES, WebGL
    - **WebGL是一种JavaScript API，用于在任何兼容的网页浏览器中不使用插件的情况下渲染2D图形和3D图形。**
    - 它基于OpenGL ES，是一个专为嵌入式系统设计的图形API。**兼容性**：WebGL 1.0基于OpenGL ES 2.0，WebGL 2.0基于OpenGL ES 3.0。**大多数现代浏览器都支持WebGL。**

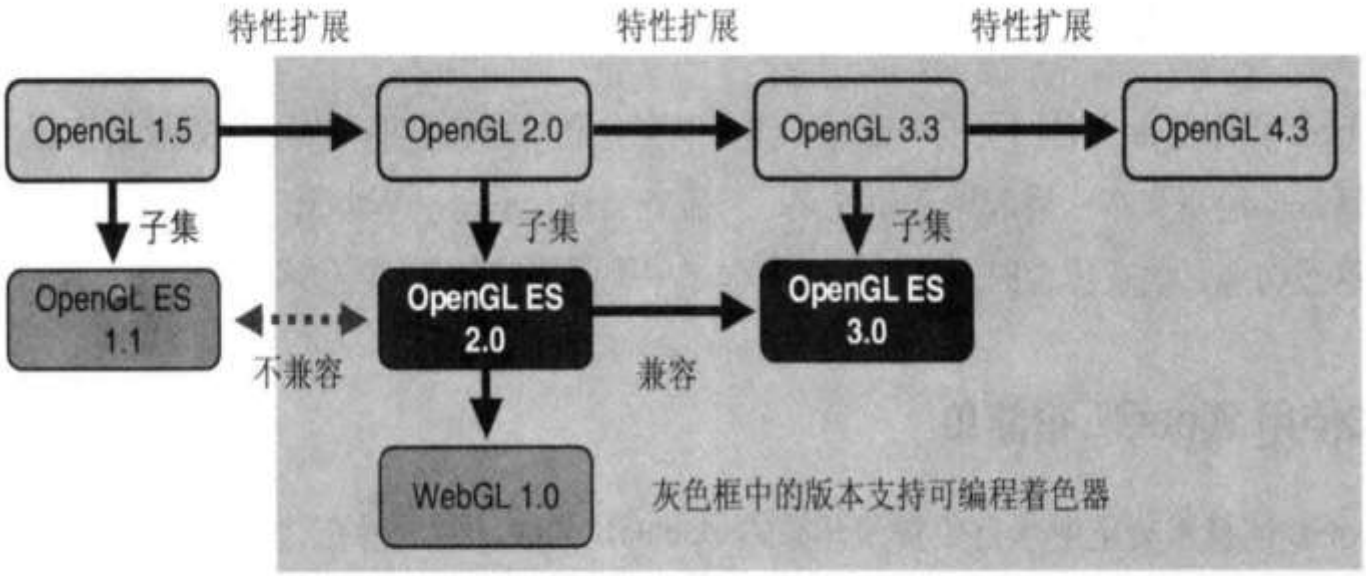


图 1.4 OpenGL、OpenGL ES 1.1//2.0/3.0 和 WebGL 之间的关系

GL History (cont.)

➢Modern Graphics Library Summary

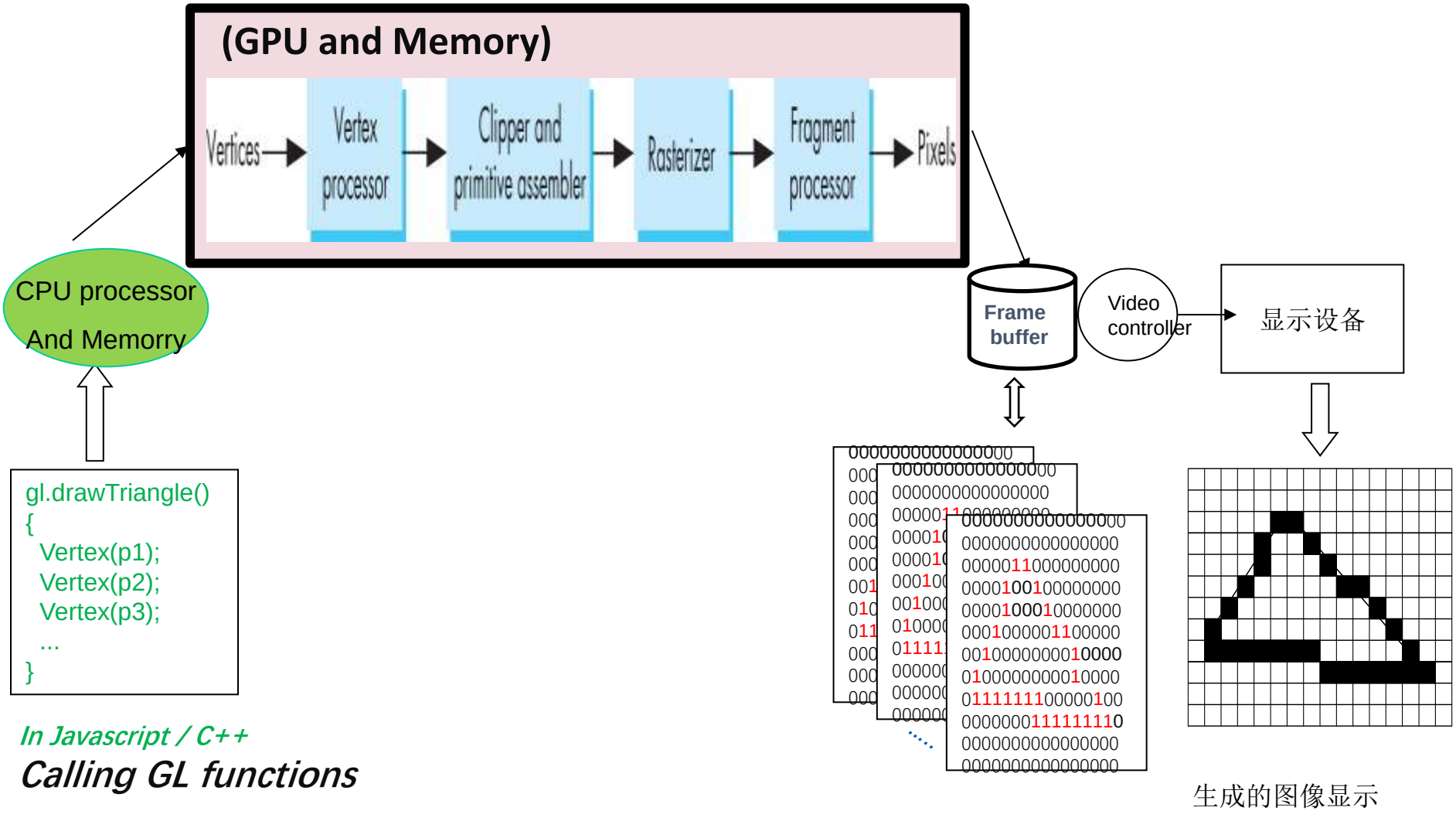
- WebGPU 是一个新兴的Web标准，旨在提供比WebGL更现代、更高效的图形和计算API。它受到了Vulkan、DirectX 12和Metal等现代图形API的影响。兼容性：目前，**WebGPU仍然是一个实验性的功能，只有部分浏览器的开发者版本支持它。**

Outlines

- GL History
- GL Architecture and Functions
- A Standard Program Structure using WebGL
- Shader Programing using GLSL
- Simple Interaction and Animation

GL Architecture

➢The Fix Functions Rendering Pipeline： Black Box View!

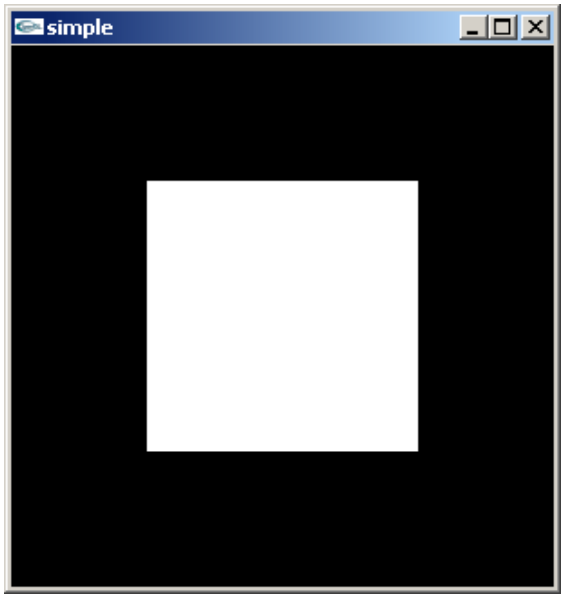


GL Architecture(cont.)

➢**The Fix Functions Rendering Pipeline： Black Box View（cont.)**

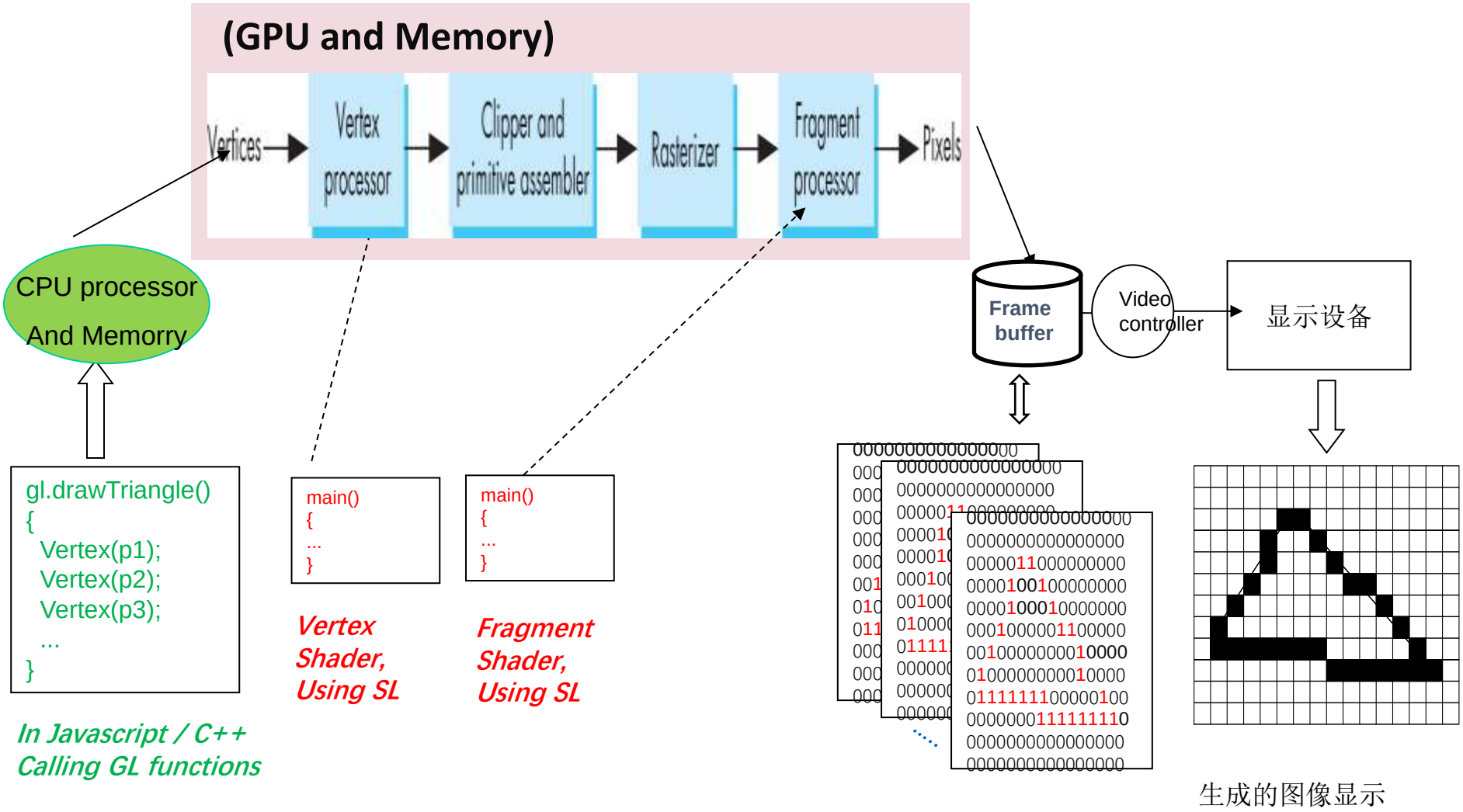
A Simple OpenGL Program: Generate a square on a solid background

```
#include <GL/glut.h>
void myrender(){
      glClear(GL_COLOR_BUFFER_BIT);
      glBegin(GL_QUAD;
            glVertex2f(-0.5, -0.5);
            glVertex2f(-0,5, 0,5);
            glVertex2f(0.5, 0.5);
            glVertex2f(0.5, -0.5);
      glEnd()
}
int main(int argc, char** argv){
      glutCreateWindow("simple");
      glutDisplayFunc(myrender);
      glutMainLoop();
}
```



GL Architecture(cont.)

➢ ***The Programmable Rendering Pipeline*: Extend view!**



GL Architecture(cont.)

- **The Programmable Rendering Pipeline: Extend view!(cont.)**

✓**Application's job is to send data to GPU**

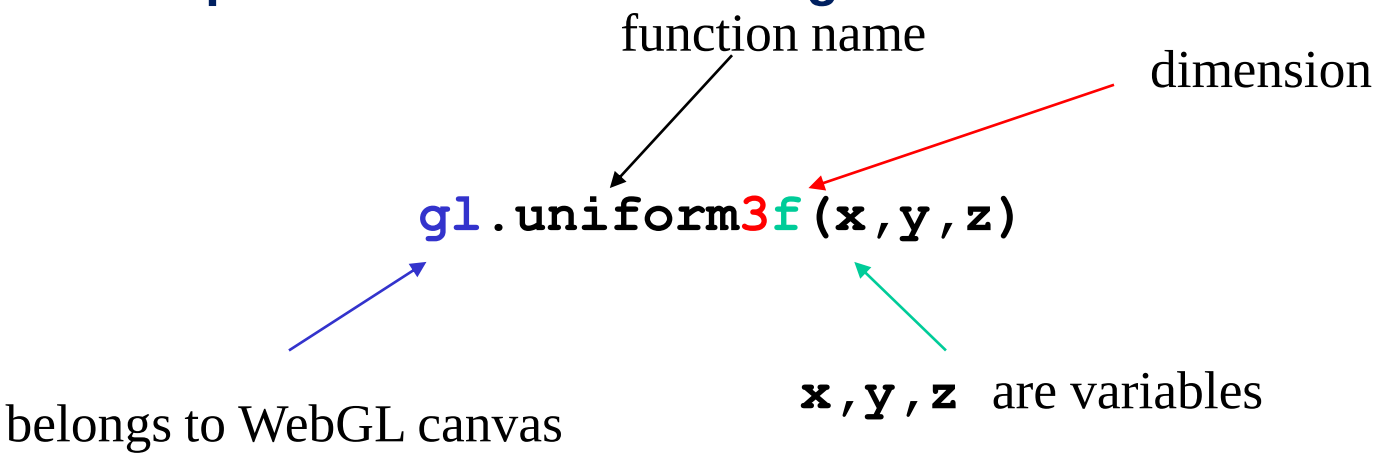
✓**GPU does all rendering**

- Performance is achieved by using GPU rather than CPU,
- Control GPU through programs called shaders（着色器）

◆**Performance is achieved by using GPU rather than CPU!**

GL Functions Formats

- **Formats**
  - **Lack of Object Orientation**
    - All versions of OpenGL are not object oriented , so that there are multiple functions for a given logical function
    - **Example: functions of sending uniform variable to shaders**




```
gl.uniform3fv(p)
```

p is an array

GL Functions Formats(cont.)

- **Formats (cont.)**
  - Functions（函数）: gl作为前缀
  - Constants（常量）:大写，一般也有gl前缀
    - Most constants are defined in the canvas object
    - In desktop OpenGL, constands were in #include files such as `gl.h`

      – **Examples in `OpenGL`:**

        • `glEnable(GL_DEPTH_TEST);`

      – **Examples in WebGL:**

```
gl.enable(gl.DEPTH_TEST)
gl.clear(gl.COLOR_BUFFER_BIT)
```

Outlines

- GL History
- GL Architecture and Functions Formats
- A Standard Program Structure using WebGL
- Shader Programing using GLSL
- Animation and Interaction using WebGL

A Standard Program Structure

- 网页结构
  - 传统动态网页：Web(HTML5，JavaScript)
  - 3D图形处理的动态网页：Web(HTML5，JavaScript)+ WebGL(GL,GLSL)
    - GLSL: 着色器语言，类C语法，用于编写shader程序

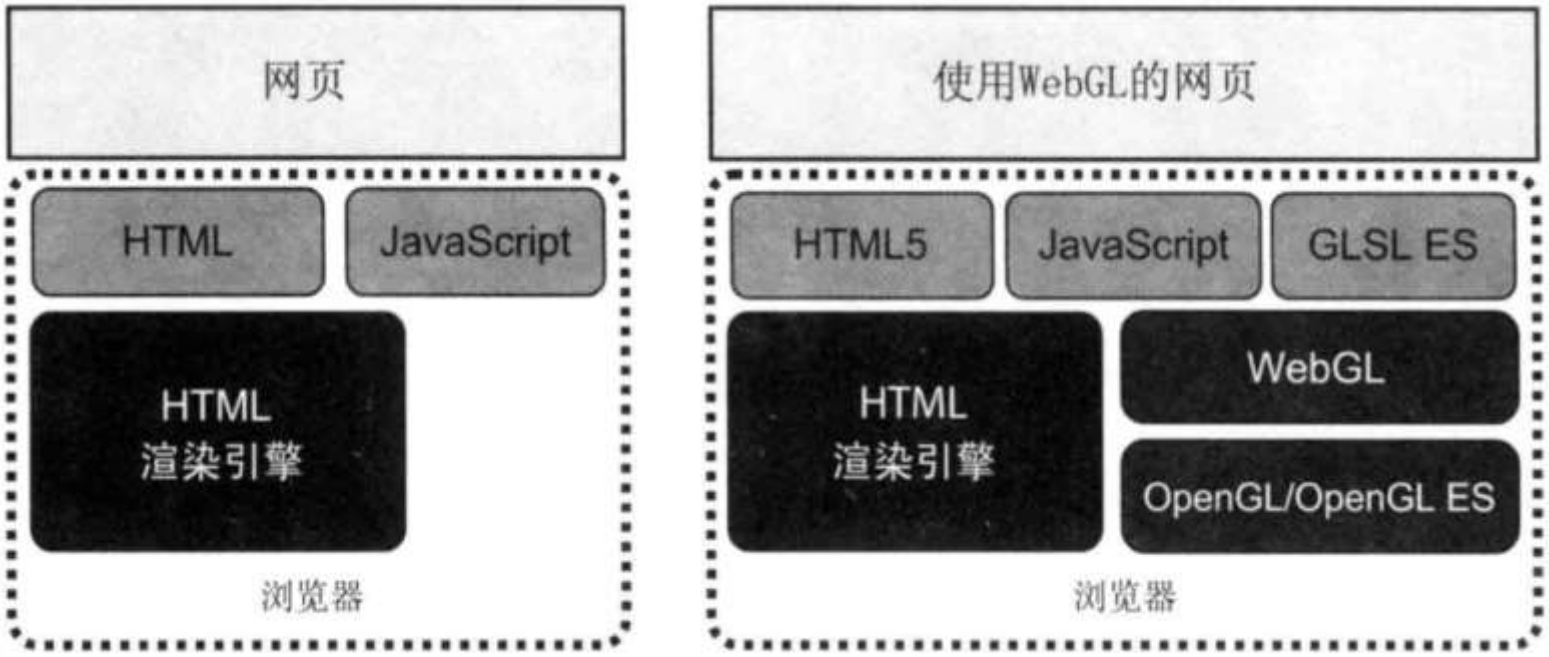
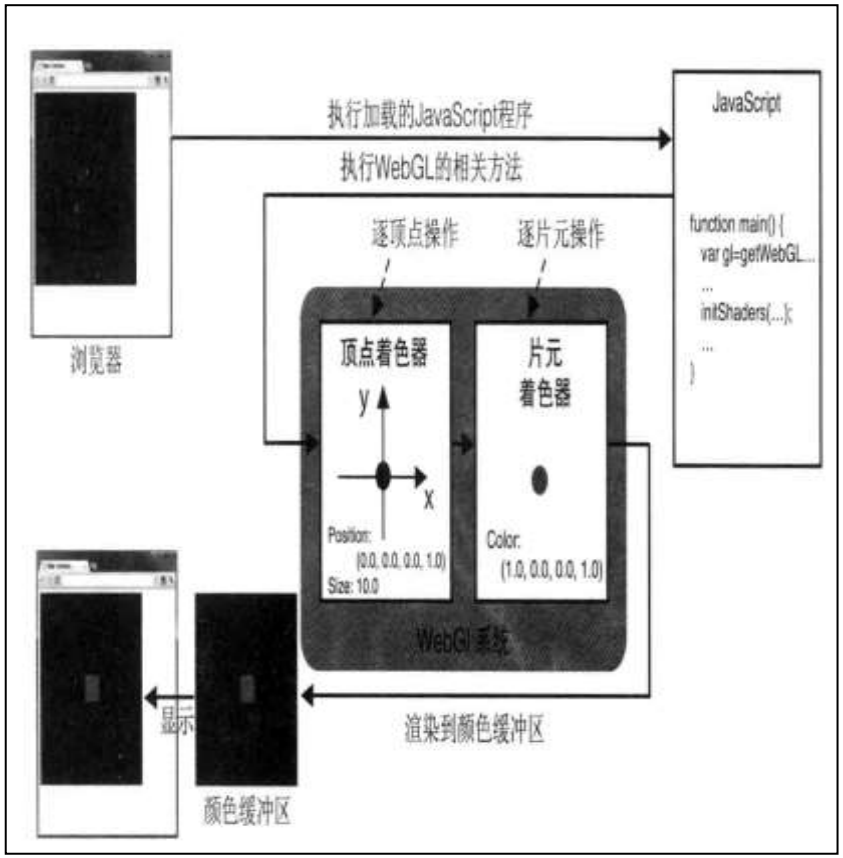
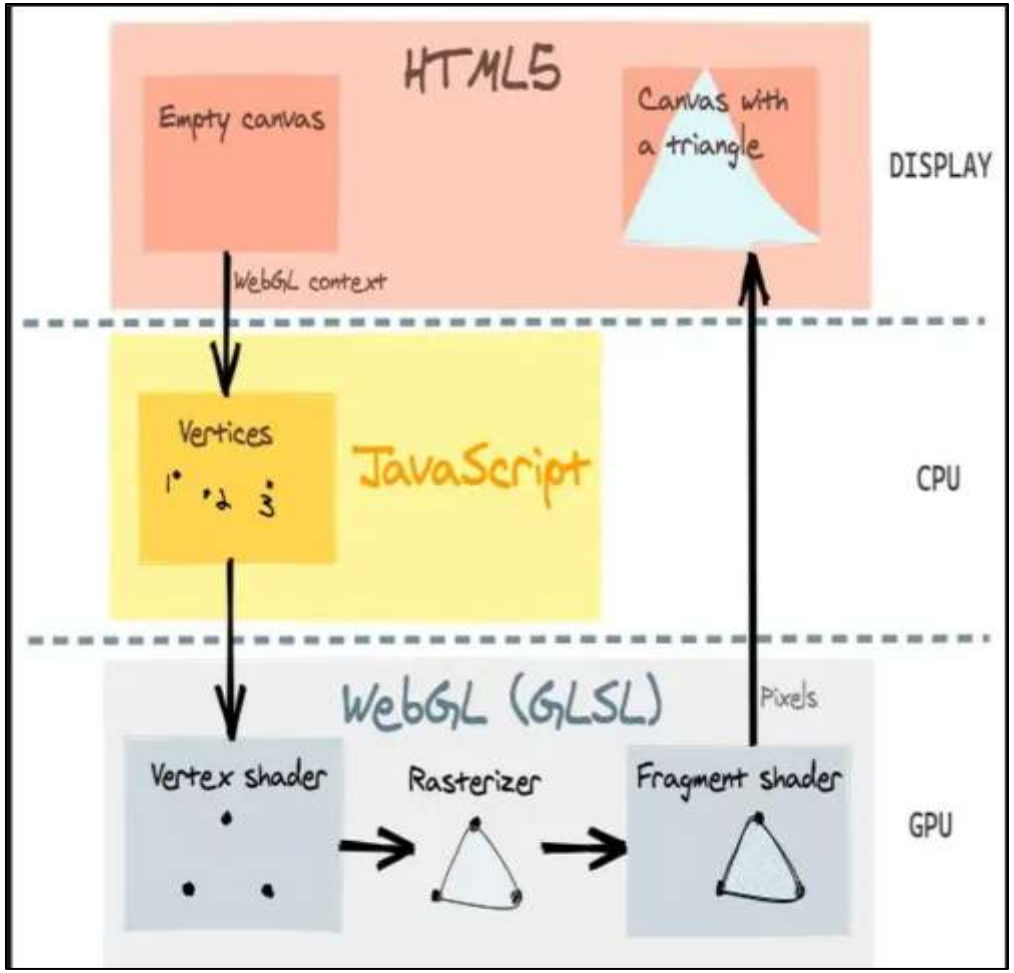


图 1.5 传统的动态网页（左侧）和 WebGL 网页（右侧）的软件结构

A Standard Program Structure（cont.)

- 执行过程/程序架构

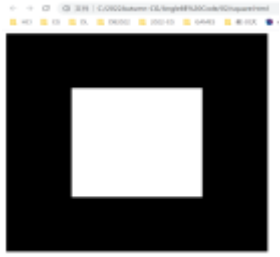




A Standard Program Structure（cont.)

- Program Five steps(编程五步骤）
  1. Describe page (HTML)
  2. Compute or Specify Geometry Data (JS)
  3. Send data to GPU (JS)
  4. Call Render (JS)
  5. Define Shaders (in any file with GLSL)
     1. could be done with a separate file (browser dependent)

- Example：\Angle8E Code\02\square



A Standard Program Structure（cont.)

- Program Five steps(编程五步骤）

  Step1: Describe page (HTML) //参见squre.html

  -定义界面内容：主要是定义“画布canvas”，以及界面的各种交互控件

  -加载程序中需要的各个js文件

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

A Standard Program Structure（cont.)

- Program Five steps(编程五步骤）

  Step1.Describe page (HTML) //参见squre.html

  -定义界面内容:主要是定义“画布canvas”，以及界面的各种交互控件

  -加载程序中需要的各个js文件

  - “`../Common/initShaders.js`”: **contains JS and WebGL code for reading, compiling and linking the shaders（公用JS代码）**
  - “`../Common/MV.js`”: **matrix-vector package（作者自编向量矩阵函数）**
  - “`square.js`”: **the application file（用于绘制squere的主程序代码）**

```
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="square.js"></script>
```

A Standard Program Structure（cont.）

- Program Five steps(编程五步骤)

  2.Compute or specify geometry data (JS)

  ① 加载配置webGL绘图环境

  ② 创建几何对象

➢ **Init(): determines where to start execution when all code is loaded入口函数**

➢ Canvas: 获取HTML中定义的可视画布对象“gl-canvas”

➢ gl: 建立webGL上下文环境，使画布canvas能够进行3D绘图功能

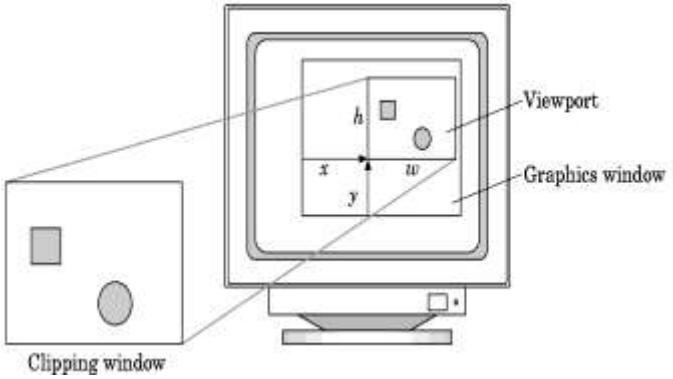
➢ gl.viewport(): GL函数，在画布canvas上指定区域作为视口区域(对应裁剪窗口-NDC空间)

➢ gl.clearColor(): GL函数，设置清屏的颜色(常量:即将帧缓存像素颜色的初值(对应画布区间)

```
var canvas;
var gl;
window.onload = function init() {
    canvas = document.getElementById( "gl-canvas" );
    gl = canvas.getContext('webgl2');
    if ( !gl ) { alert( "WebGL 2.0 isn't available" );

    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 0.0, 0.0, 0.0, 1.0 );
    …
```

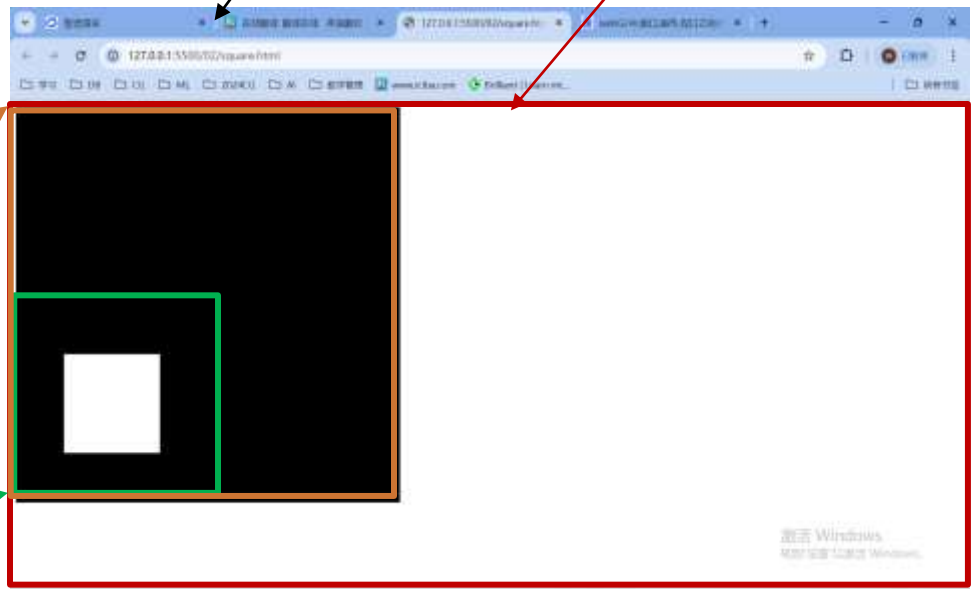
A Standard Program Structure（cont.)

- Window, canvas, viewport



Window

innerWindow



```
<canvas id="gl-canvas"
width="512" height="512">
```
-坐标原点(0,0)在左上角

若gl.Viewport(0, 0, w/2,h/2)
-在canvas划出的区域,
-坐标原点(0, 0)在左下角！
注: viewport才是真正绘图区域,
裁剪空间NDC中的图形绘制

A Standard Program Structure（cont.)

- Program Five steps(编程五步骤）

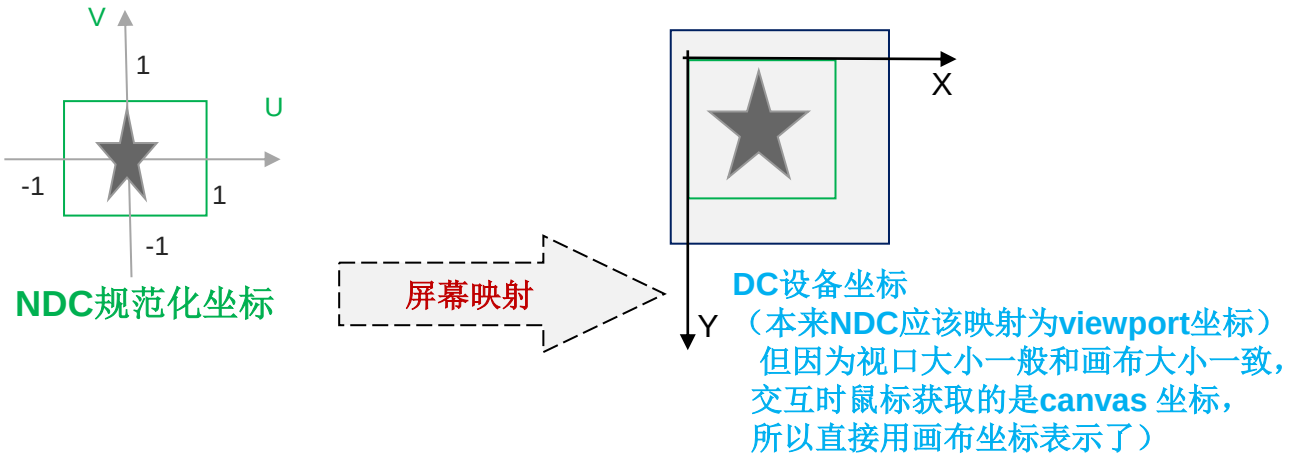
  step2.Compute or specify geometry data (JS)

  ① 加载配置webGL绘图环境

  ② 创建几何对象:

  - 顶点坐标目前设置在 **NDC规范化坐标范围内, 取值在 (-1,-1)(1,1)范围！**

```
// Four Vertices of Square
    var vertices = [
        vec2( -0.5, -0.5 ),
        vec2(  -0.5,  0.5 ),
        vec2(  0.5, 0.5 ),
        vec2( 0.5, -0.5)    ];
```



A Standard Program Structure（cont.)

- Program Five steps(编程五步骤）

  step3.Send data to GPU (JS)

  - 创建program对象，加载shader程序
  - 创建缓冲缓冲区对象,将顶点属性数据放入并关联顶点着色器,发送数据给GPU

```
  //Load shaders
 program = initShaders( gl, "vertex-shader", "fragment-shader" );
 gl.useProgram( program );

  // Load the data into the GPU
 var bufferId = gl.createBuffer();
 gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
 gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );

// Associate out shader variable with our data buffer
 var vPosition = gl.getAttribLocation( program, "vPosition" );
 gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
 gl.enableVertexAttribArray(vPosition););
```

*注:**flatten()** 是定义在 MV.js 中的函数，to convert JS array to an array of C-like array of float32's*

A Standard Program Structure（cont.)

- Program Five steps(编程五步骤）

  step4. Call Render (JS)

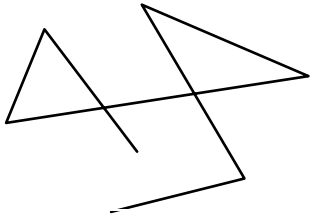
  - 首先清屏操作gl.clear();
  - 然后告诉GPU，用发送的数据绘制什么“图元”！

```
window.onload = function init() {
      ……
      render();
};
function render() {
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 );//三角扇
}
```

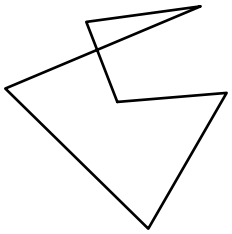
A Standard Program Structure（cont.)

- WebGL Primitives (图元) 主要下面7种：

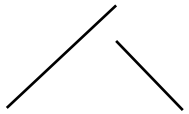
gl.POINTS



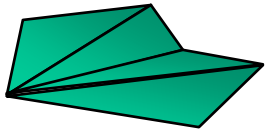
gl.LINE_STRIP



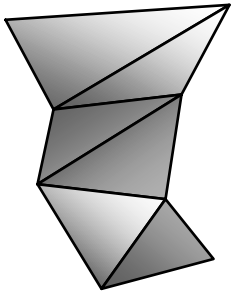
gl.LINE_LOOP



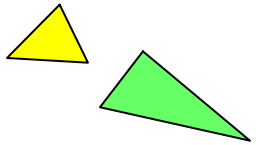
gl.LINES



gl.TRIANGLE_FAN



gl.TRIANGLE_STRIP

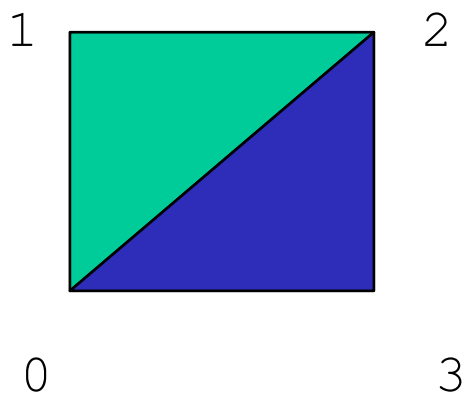


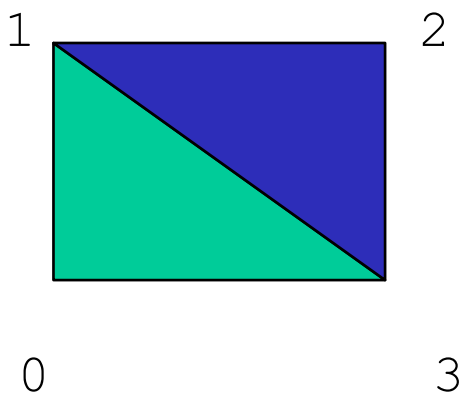
gl.TRIANGLES

A Standard Program Structure（cont.)

- 面图元:三角形，三角扇，三角带
  - 注意:各自需要的顶点数，顶点顺序不同！

- gl.drawArrays(gl.TRIANGLES, 0, 6); // 0, 1, 2, 0, 2, 3
- gl.drawArrays(gl.TRIANGLE_FAN, 0, 4); // 0, 1 , 2, 3
- gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4); // 0, 1, 3, 2





A Standard Program Structure（cont.）

- Program Five steps(编程五步骤)

  Step5. Define Shaders

  - 可以html里定义，也可以单独的文件.
  - 至少定义一组着色器(含1个顶点着色器，1个片元着色器)

```
<html>
<script id="vertex-shader" type="x-shader/x-vertex">
#version 300 es
in vec4 vPosition;

void main()
{
    gl_Position = vPosition;
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
#version 300 es
precision mediump float;
out vec4 fColor;

void main()
{
    fColor = vec4( 1.0, 1.0, 1.0, 1.0 );
}
</script>
```

Outlines

- GL History
- GL Architecture and Functions Formats
- A Standard Program Structure using WebGL
- Shader Programing using GLSL
- Animation and Interaction using WebGL

Shader Programing using GLSL

- First programmable shaders were programmed in an assembly-like manner.
- OpenGL extensions added functions for vertex and fragment shaders
  - Cg (C for graphics) C-like language for programming shaders
    - Works with both OpenGL and DirectX
    - Interface to OpenGL complex
  - OpenGL Shading Language (GLSL)
    - Part of OpenGL 2.0 and up
    - High level C-like language
    - New data types:Matrices, Vectors, Samplers
  - As of OpenGL 3.1, application must provide shaders

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
- Because 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

Operators and Functions

- Standard C functions
  - Trigonometric
  - Arithmetic
  - Normalize(归一化), reflect（反射）, 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
```

Selection and Swizzling

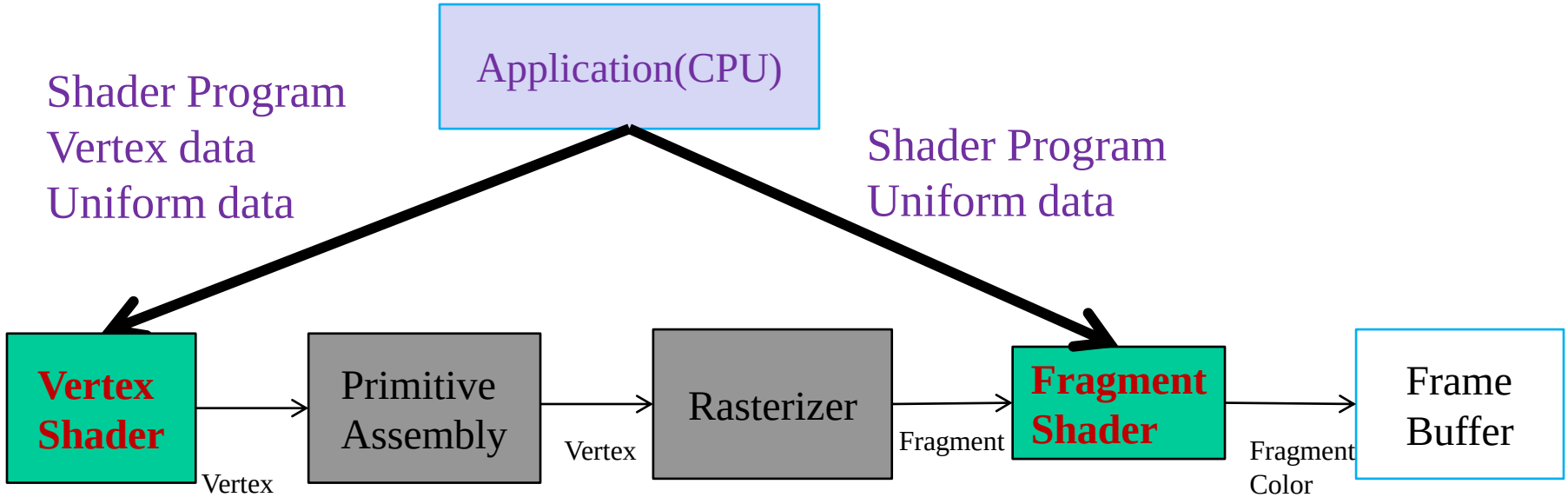
- Can refer to array elements by element using [] or selection (.) operator with
  - x, y, z, w
  - r, g, b, a:

  `Ex:a[2], a.b, a.z, a.p` are the same
- **Swizzling** operator lets us manipulate components

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

Shader Programing using GLSL（cont.)

- 执行流程示意图
  - 顶点着色器：所有图形的顶点，并发执行的相同代码
  - 片元着色器：所有图形生成的片元，并发执行的相同代码
  - ➢Vertex attributes are interpolated by the rasterizer into fragment attributes



Shader Programing using GLSL（cont.)

- Vertex Shader
  - webGL2.0中，用in替换了attribute修饰符
  - webGL2.0中，必须标注版本

Simple Vertex Shader (WebGL 2.0)

```
#version 300 es
in vec4 vPosition;
void main(void)
{
    gl_Position = vPosition;
}
```

compiler directive版本2.0标注

input from application

must link to variable in application

built in variable

Simple Vertex Shader (WebGL 1.0)

```
attribute vec4 vPosition;
void main(void)
{
    gl_Position = vPosition;
}
```

input from application

must link to variable in application

built in variable

Shader Programing using GLSL（cont.)

- Fragment Shader
  - webGL2.0中，必须标注版本，并且标注精度
  - webGL2.0中，输出片元颜色不再使用内置变量

Simple Fragment Program (WebGL 2.0)

```
#version 300 es
precision mediump float;
out fragColor;
void main(void)
{
   fragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

compiler directive

required

Simple Fragment Program (WebGL 1.0)

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

built in variable

Shader Programing using GLSL（cont.)

- Variable Qualifiers变量修饰符
  - Attribute
    - Attribute-qualified variables can change at most once per vertex
    - **webGL2.0中用in替换, 一般只修饰顶点着色器中输入的顶点属性**
  - Uniform
    - Variables that are constant for an entire primitive(对一个图元来说，改变量是一个常量), Used to pass information to shader such as the time or a bounding box of a primitive or transformation matrices
    - Can be changed in application and sent to shaders
    - Cannot be changed in shader
  - Varying
    - Variables that are passed from vertex shader to fragment shader
    - **webGL2.0中用in和out替换, 仍要求变量同名**

```
out vec4 color; //vertex shader
in vec4 color;  // fragment shader
```

Shader Programing using GLSL（cont.)

- Variable Qualifiers变量修饰符（cont.)
  - Example1: Set Colors from Application

```
var cBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(colors),  gl.STATIC_DRAW );

var aColor = gl.getAttribLocation( program, ”aColor" );
gl.vertexAttribPointer( aColor, 3, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( aColor );
```

```
in vec4 aColor, aPosition;
out vec4 vColor;
void main()
{
  gl_Position = aPosition;
  vColor = aColor;
}
```

```
precision mediump float;
in vec4 vColor;
out vec4 fColor
void main()
{
   fColor = vColor;
}
```

Shader Programing using GLSL（cont.)

- Variable Qualifiers变量修饰符（cont.)
  - Example2：Sending a Uniform Variable to set color

```
// in application
vec4 uColor = vec4(1.0, 0.0, 0.0, 1.0);
gl.uniform4f(gl.getUniformLocation( program, ”color" ), uColor);
```

```
// in fragment shader (similar in vertex shader)
uniform vec4 uColor;
out vec4 fColor;
void main()
{
    gl_FragColor = uColor;
}
```

Summary

- GL History
- GL Architecture and Functions Formats
- A Standard Program Structure using WebGL
- Shader Programing using GLSL
- Animation and Interaction using WebGL

Animation Type and FPS

- **Animation:** *Reference: GAMES101_Lecture21*

"Bring things to life"

- Communication tool
- Aesthetic issues often dominate technical issues

An extension of modeling

- Represent scene models as a function of time

Output: sequence of images that when viewed sequentially provide a sense of motion

(Phenakistoscope, 1831)

Animation Type and FPS(cont.)

Animation Types

- 逐帧动画（frame-by-frame animation）
  - Offline generation and real-time playback.
  - Such as: movies, TV
- 实时动画（Online real-time animation）：
  - Online generated and play immediately.
  - Such as: games, interactive applications.

Animation Type and FPS(cont.)

Frame frequency(帧频)：**frames per second (FPS)**

- 帧率能够达到 50 ～ 60 FPS 的动画将会相当流畅，让人倍感舒适；
- 帧率在 30 ～ 50 FPS 之间的动画，因各人敏感程度不同，舒适度因人而异；
- 帧率在 30 FPS 以下的动画，让人感觉到明显的卡顿和不适感；
- 帧率波动很大的动画，亦会使人感觉到卡顿。

- Display Refresh frequency:显示器刷新频率
  - （每秒刷新60次以上，**屏幕显示感觉不闪烁**）

Output: sequence of images that when viewed sequentially provide a sense of motion

- Film: 24 frames per second
- Video (in general): 30 fps
- Virtual reality: 90 fps

Animation Type and FPS(cont.)

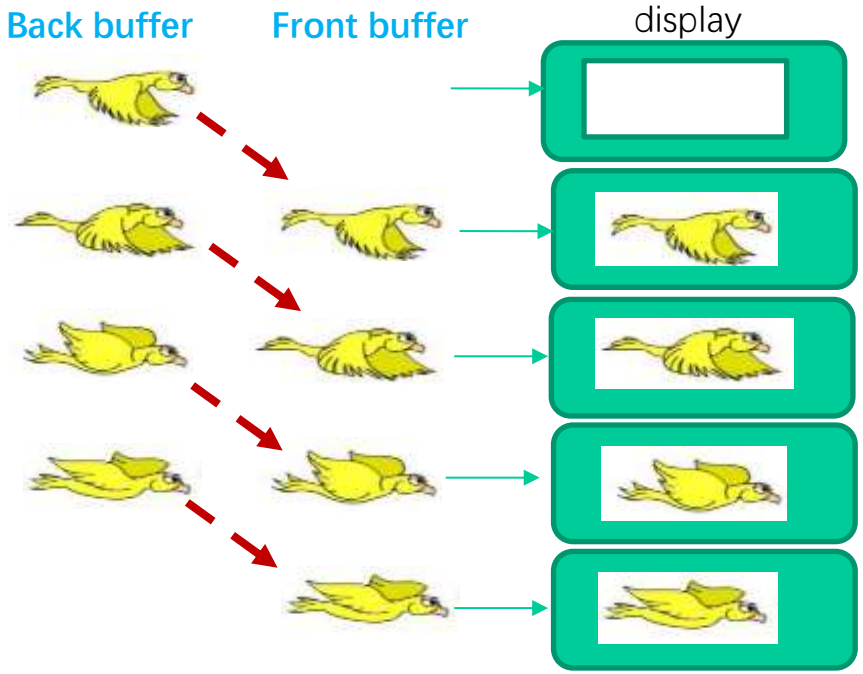
- FPS>150：帧拖延效果（晕炫视觉，拖影）
  - 原因~ 若创建一帧时间很短，远远小于屏幕的刷新周期，则导致拖影。
  - ✓**solution：Time Delay时延**

- FPS<24：帧断裂 or 帧飘移 (破帧)
  - 原因~ 单帧缓存(只有1个颜色帧缓存)：若创建一帧时间大于 屏幕一个刷新周期，则会partial rendering:部分绘制，导致帧飘移或帧破裂。
  - ✓**Solution：Double Buffering双帧缓存**

Double Buffering for Smooth Animations

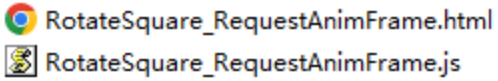
- Double Buffering
  - when we are rendering a frame, it always be render into a buffer that is not displayed to prevents display of a “partial rendering”.
  - front buffer前帧: always display
  - back buffer后帧: rendering into

openGL use a buffer swap



Animation in Brower

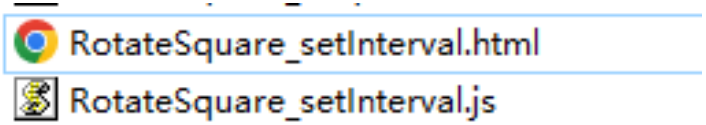
- requestAnimationFrame()



  - The function is now part of JS ,
  - Browser refresh the display at 60 Hz, this function will allow only one execution per display refresh

```
function render {
    gl.clear(gl.COLOR_BUFFER_BIT);
    uTheta += 0.1;
    gl.uniform1f(thetaLoc, uTtheta);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    requestAnimationFrame(render);
}
```

Animation in Brower(cont.)

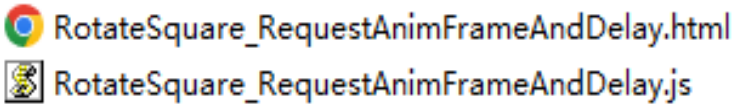


- Interval Timer()
  - Executes a function after a specified number of milliseconds
  - setInterval(render, interval);

- Note an interval of 0 generates buffer swaps as fast as possible,
- Defects:
  - Stack overflow
  - 每种浏览器中可能有区别，难以得到平滑动画显示！不兼容

Animation in Brower(cont.)

- Combination of “requestAnimationFrame” and “Interval timer”

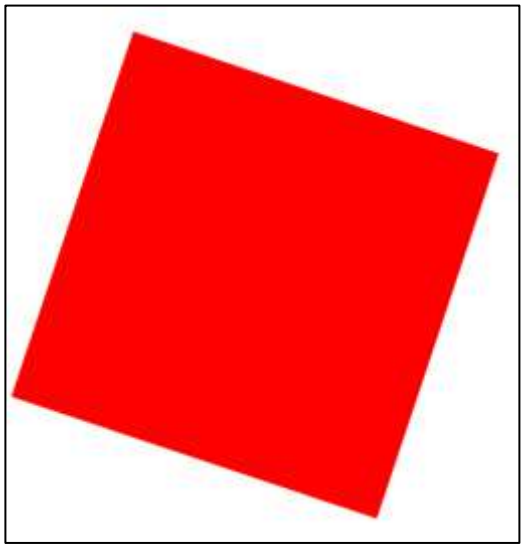
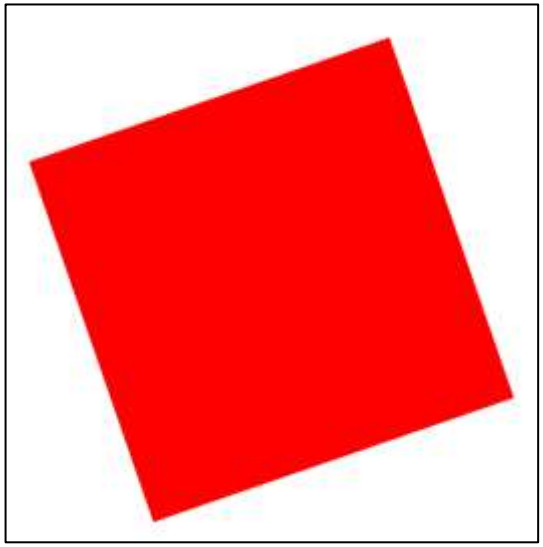
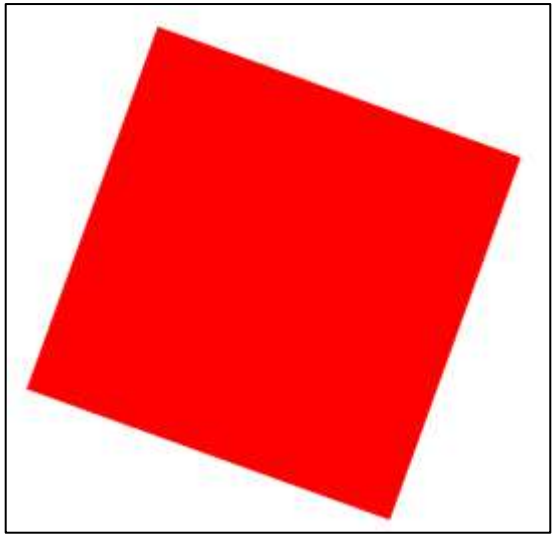


➢setTimeout()

```
function render()
{
    gl.clear(gl.COLOR_BUFFER_BIT);

    theta += (direction ? 0.1 : -0.1);
    gl.uniform1f(thetaLoc, theta);

    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

    setTimeout(
        function (){requestAnimationFrame(render);}, delay
    );
}
```

Animation Example:

- Rotated Square
  - Animate display by rerendering with different values of θ

Animation Example(cont.)

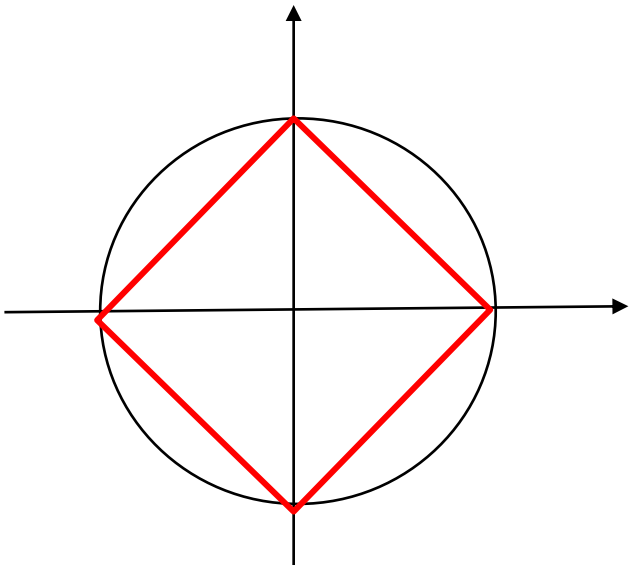
- 绘制方法1:在主程序中计算下一帧图形顶点后再绘制
  - Recompute the vertices locations by new theta value
  - Resend new data array from CPU to GPU
  - ✓Simple but slow! //see RotateSquare_Interaction

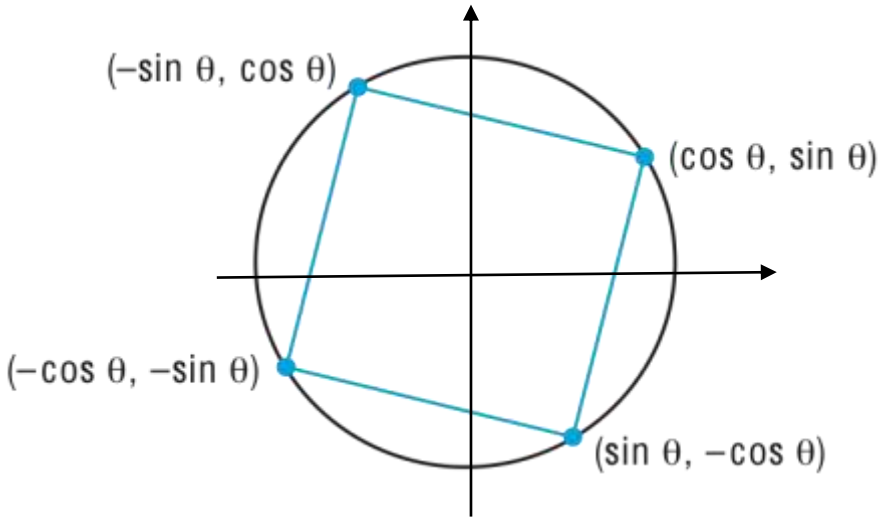
```
for(var theta = 0.0; theta <thetaMax; theta += dtheta; ){
      vertices[0] = vec2(Math.sin(theta), Math.cos.(theta));
      vertices[1] = vec2(Math.sin(theta), -Math.cos.(theta));
      vertices[2] = vec2(-Math.sin(theta), -Math.cos.(theta));
      vertices[3] = vec2(-Math.sin(theta), Math.cos.(theta));

      gl.bufferData(…)

       render();
}
```

Animation Example(cont.)

✓上页中新帧中正方形顶点计算方法如下:

✓注意:四个顶点的计算方法各自不同





Consider the four point, circle radius=1

$$x_{i+1} = R\cos\theta_{i+1}$$

$$y_{i+1} = R\sin\theta_{i+1}$$

Animation Example(cont.)

绘制方法2:在“顶点着色器里”重新计算下帧顶点位置并绘制,

- Send original vertices to vertex shader(only once)
- Send new θ to shader as a uniform variable, and **Compute new vertices in vertex shader**
- Better Way, *//see “rotatingSquare1*

```
var vertices = [
    vec2(0, 1),
    vec2(-1, 0),
    vec2(1, 0),
    vec2(0, -1)
];

// Load the data into the GPU
var bufferId = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, bufferId);
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);

// Associate out shader variables with our data bufferData
var positionLoc = gl.getAttribLocation(program, "aPosition");
gl.vertexAttribPointer(positionLoc, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(positionLoc);

thetaLoc = gl.getUniformLocation(program, "uTheta");

render();
};
```

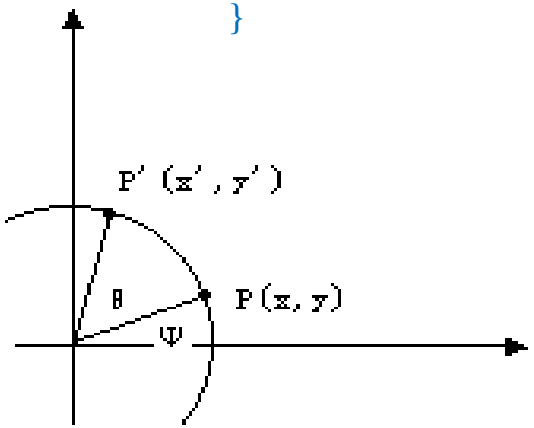
```
function render()
{    gl.clear(gl.COLOR_BUFFER_BIT);
    uTheta += 0.1;
    gl.uniform1f(thetaLoc, uTheta);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    render();
}
```

```
in vec4 aPosition;
uniform float uTheta;
void main()
{
  gl_Position.x = -sin(uTheta) * aPosition.x + cos(uTheta) * aPosition.y;
  gl_Position.y = sin(uTheta) * aPosition.y + cos(uTheta) * aPosition.x;
  gl_Position.z = 0.0;
  gl_Position.w = 1.0;
}
```

Animation Example(cont.)

- 上页中，顶点着色器中的每顶点的计算公式相同！
- P是初始顶点位置，P’是旋转theta后的顶点位置，可以推出它们之间的计算关系

```
in vec4 aPosition;
uniform float uTheta;
void main()
{
   gl_Position.x = -sin(uTheta) * aPosition.x + cos(uTheta) * aPosition.y;
   gl_Position.y = sin(uTheta) * aPosition.y + cos(uTheta) * aPosition.x;
   gl_Position.z = 0.0;
   gl_Position.w = 1.0;
}
```



$x' = r\cos(\theta+\Psi)$

$= r\cos\theta\cos\Psi - r\sin\theta\sin\Psi$

$y' = r\sin(\theta+\Psi)$

$= r\sin\theta\cos\Psi + r\cos\theta\sin\Psi$

for : $x = r\cos\Psi$, $y = r\sin\Psi$

So : $x' = x\cos\theta - y\sin\theta$

$y' = x\sin\theta + y\cos\theta$

Interaction
Working with callback

- **Callback Function/Event Listeners**
  - Programming interface(API) for event-driven input uses *"callback functions"* (回调函数）or *"event listeners"*(事件监听器)
    - ✓**Define 定义回调函数/事件监听器**
      - – ***define a callback function for each event*** **the graphics system recognizes**
    - ✓**Register 注册回调函数/事件监听器**
      - ✓***Browsers enters an event loop and responds to those events for which it has callbacks registered***
    - ✓Execute 执行回调函数**/事件监听器**
      - – ***The function is executed when the event occurs***
- **Ex.** use the onload window event to initiate execution of the init（）function.

  定义，注册，执行window的onload事件的回调函数init（）

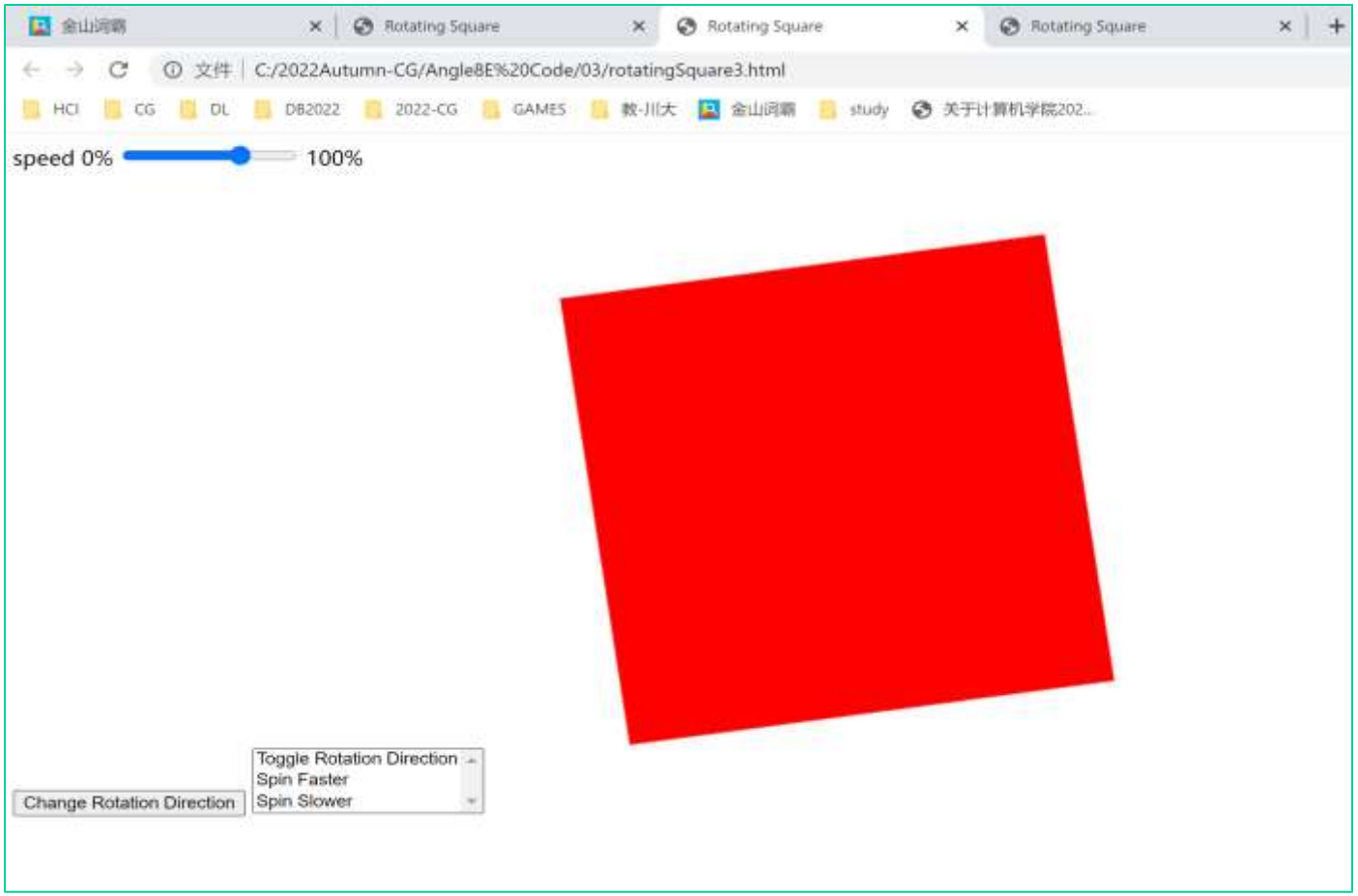
  window.onload = init(){…}；//init()相当于C中的main()函数

Interaction
Working with callback（cont.)

- 一般常用的Target（目标），Event types（事件类型）
  - Button: click,
  - Menu: click
  - Slider: change
  - Mouse: mousedown,mouseup,mousemove
  - Window: onload, keydown, resize

Interaction
Working with callback（cont.）

- Example: angelcode8/03/rotatingSquare3



Interaction
Working with callback（cont.)

1.Adding a Button

- In the HTML file,Uses HTML button tag
  - id gives an identifier we can use in JS file
  - Text “Change Rotation Direction” displayed in button
  - Clicking on button generates a click event


```
<button id="DirectionButton">
            Change Rotation Direction
</button>
```

Interaction
Working with callback（cont.)

1.Adding a Button(cont.)

- Declare variable “direction”
  - In the render function we can use a var direction which is true or false to add or subtract a constant to the angle

```
var direction = true; // global initialization
```

```
Render(){
…
   if(direction)
        theta += 0.1;
   else
        theta -= 0.1;
   …}
```

Interaction
Working with callback（cont.）

1.Adding a Button(cont.)

- Register Button Event Listener
- **We still need to define the listener:** **no listener and the event occurs but is ignored**
- **Two forms for event listener in JS file:** **choose one**

```
var myButton = document.getElementById("DirectionButton");

myButton.addEventListener("click", function() {
    direction = !direction;
});
```

```
document.getElementById("DirectionButton").onclick =
function() {
    direction = !direction;
};
```

Interaction
Working with callback（cont.)

1.Adding a Button(cont.)

- More **onclick variants**：

```
myButton.addEventListener(“click", function(event) {
  if (event.button == 0) { direction = !direction; }
});
//Event.button=0表示是鼠标左键
```

```
myButton.addEventListener("click", function(event) {
  if (event.shiftKey == 0) { direction = !direction; }
});
//Enent.shiftkey 表示按下shift键
```

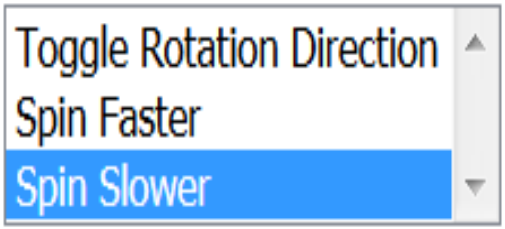
```
<button onclick="direction = !direction"></button>
```

Interaction
Working with callback（cont.)

2.Menus

- Use the HTML select element
- Each entry in the menu is an option element with an integer value returned by click event

```
<select id="mymenu" size="3">
        <option value="0">Toggle Rotation Direction</option>
        <option value="1">Spin Faster</option>
        <option value="2">Spin Slower</option>
</select>
```



Interaction
Working with callback（cont.)

- **2.Menus(cont.)**
- Add and Register Menu Listener

```
var m = document.getElementById("mymenu");
m.addEventListener("click", function() {
   switch (m.selectedIndex) {
      case 0:
         direction = !direction;
         break;
      case 1:
         delay /= 2.0;
         break;
      case 2:
         delay *= 2.0;
         break;
       }
});
```

Interaction
Working with callback（cont.)

- **3.keyboard**
  - **Using window on “keydown” Event**

```
window.addEventListener("keydown", function() {
  switch (event.keyCode) {//按键:数字键1,2, 3代替菜单选择
    case 49: // ’1’ key
      direction = !direction;
      break;
    case 50: // ’2’ key
      delay /= 2.0;
      break;
    case 51: // ’3’ key
      delay *= 2.0;
      break;
  }
});
```

Interaction
Working with callback（cont.）

- **3.keyboard(cont.)**
  - Don’t Know Unicode

```
window.onkeydown = function(event) {
  var key = String.fromCharCode(event.keyCode);
  switch (key){
    case ’1’:
      direction = !direction;
      break;
    case ’2’:
      delay /= 2.0;
      break;
    case ’3’:
      delay *= 2.0;
      break;
  }};
```

Interaction
Working with callback（cont.)

4. Slider Element

- In HTML file: Puts slider on page
  - Give it **an identifier**
  - Give it **minimum and maximum values**
  - Give it **a step size** needed to generate an event
  - Give it **an initial value**

```
<div>
speed 0 %<input id=“slider" type="range"
   min="0" max="100" step="10" value="50" />100%
</div>
```

Interaction
Working with callback（cont.)

4. Slider Element(cont.)

- Add onchange Event Listener

  - Two usages:

```
document.getElementById("slider").onchange =
function(event)
{
        delay= 100-event.target.value;
};
```

```
document.getElementById("slider").onchange =
function()
{
                delay =100- event.srcElement.value;
};
```

Interaction
CAD-like Examples

// CAD-like Examples： angleCode/03/*.*

square.html: puts a colored square at location of each mouse click

triangle.html: first three mouse clicks define first triangle of triangle strip. Each succeeding mouse clicks adds a new triangle at end of strip

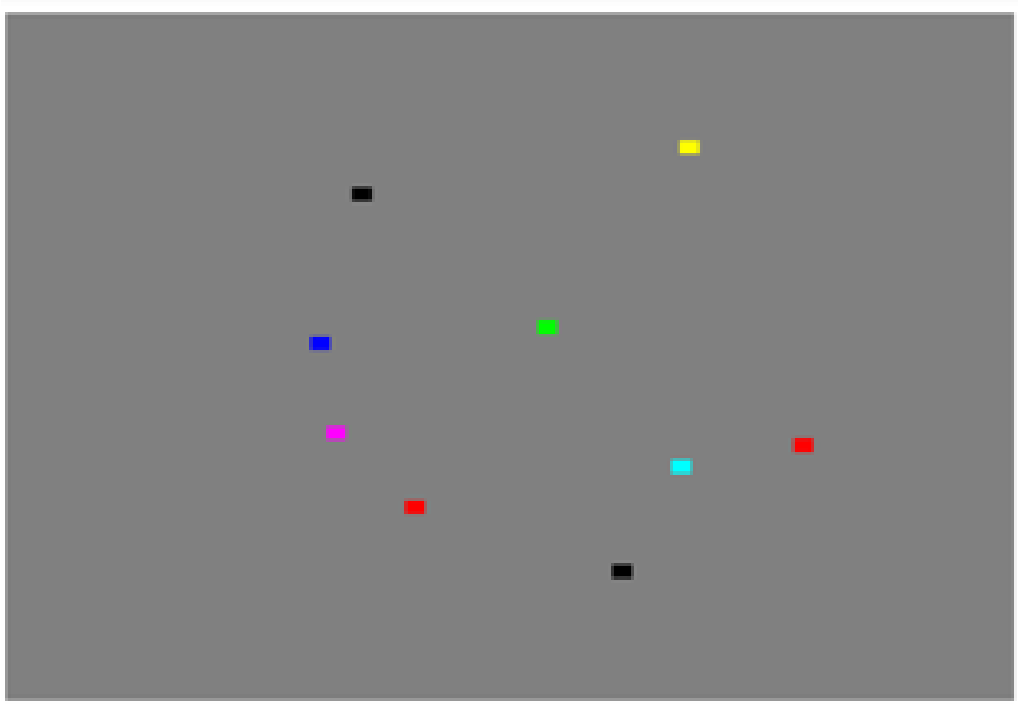
cad1.html: draw a rectangle for each two successive mouse clicks

cad2.html: draws arbitrary polygons

Interaction
CAD-like Examples(cont.）

- Position Input: use the mouse to give locations
  - Must convert from position on canvas to position in application(需要将屏幕桌标转换为NDC坐标）

“square.html” and “square.js”



Interaction
CAD-like Examples(cont.）

如何把鼠标击“点”的屏幕画布坐标，转换为绘制用的NDC坐标？

- Canvas specified in HTML file of size: canvas.width , canvas.height.(获取屏幕的宽度W，和高度H)
- System Returned Canvas coordinates are event.clientX and event.client（获取点的屏幕坐标Xs,Ys)

➢计算NDC下的坐标 t(x,y):

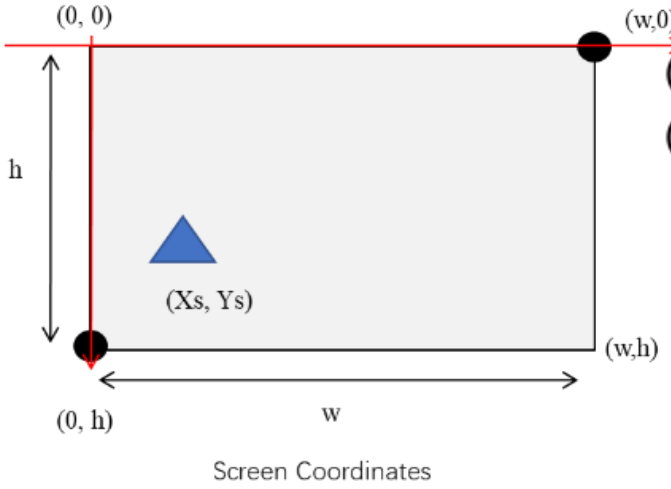
$$\frac{Xw-(-1)}{1-(-1)}=\frac{Xs-0}{w-0} \quad \text{得到 } Xw=-1+\frac{2Xs}{w}$$

$$\frac{Yw-(-1)}{1-(-1)}=\frac{Ys-h}{0-h} \quad \text{得到 } Yw=-1+\frac{2(Ys-h)}{-h}=-1+\frac{2(h-Ys)}{h}$$

```
canvas.addEventListener("mousedown", function(event){
    gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
    var t = vec2(2*event.clientX/canvas.width-1,
       2*(canvas.height-event.clientY)/canvas.height-1);
    gl.bufferSubData(gl.ARRAY_BUFFER, 8*index, flatten(t));
```

Interaction
CAD-like Examples(cont.)

Screen Coordinates（Xs,Ys）
/Canvas Coordinates

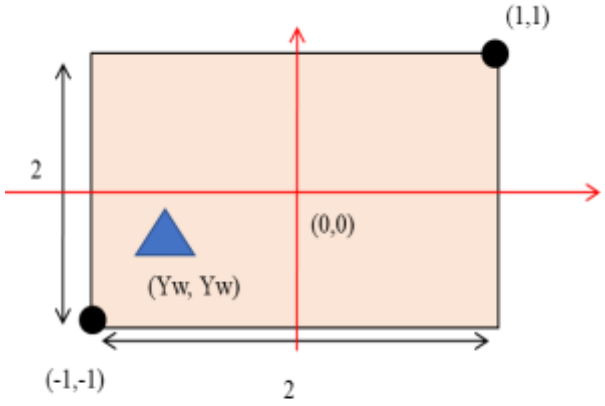


Screen Coordinates

$(0,h) \rightarrow (-1,-1)$

$(w,0) \rightarrow (1,1)$

NDC Coordinates(Xw,Yw)



View/Clip Coordinates

1.直接根据对应成比例进行推导：

$$\frac{Xw-Xwleft}{Xwright-Xwleft}=\frac{Xs-Xsleft}{Xsritht-Xsleft}$$

$$\frac{Yw-Ywbottom}{Ywtop-Ywbottom}=\frac{Ys-Ysbottom}{Ystop-Ysbottom}$$

屏幕视区左上角 (xsleft, ysbottom) = (0, 0) 代入数据：

$$\frac{Xw-(-1)}{1-(-1)}=\frac{Xs-0}{w-0}$$ 得到 $Xw=-1+\frac{2Xs}{w}$

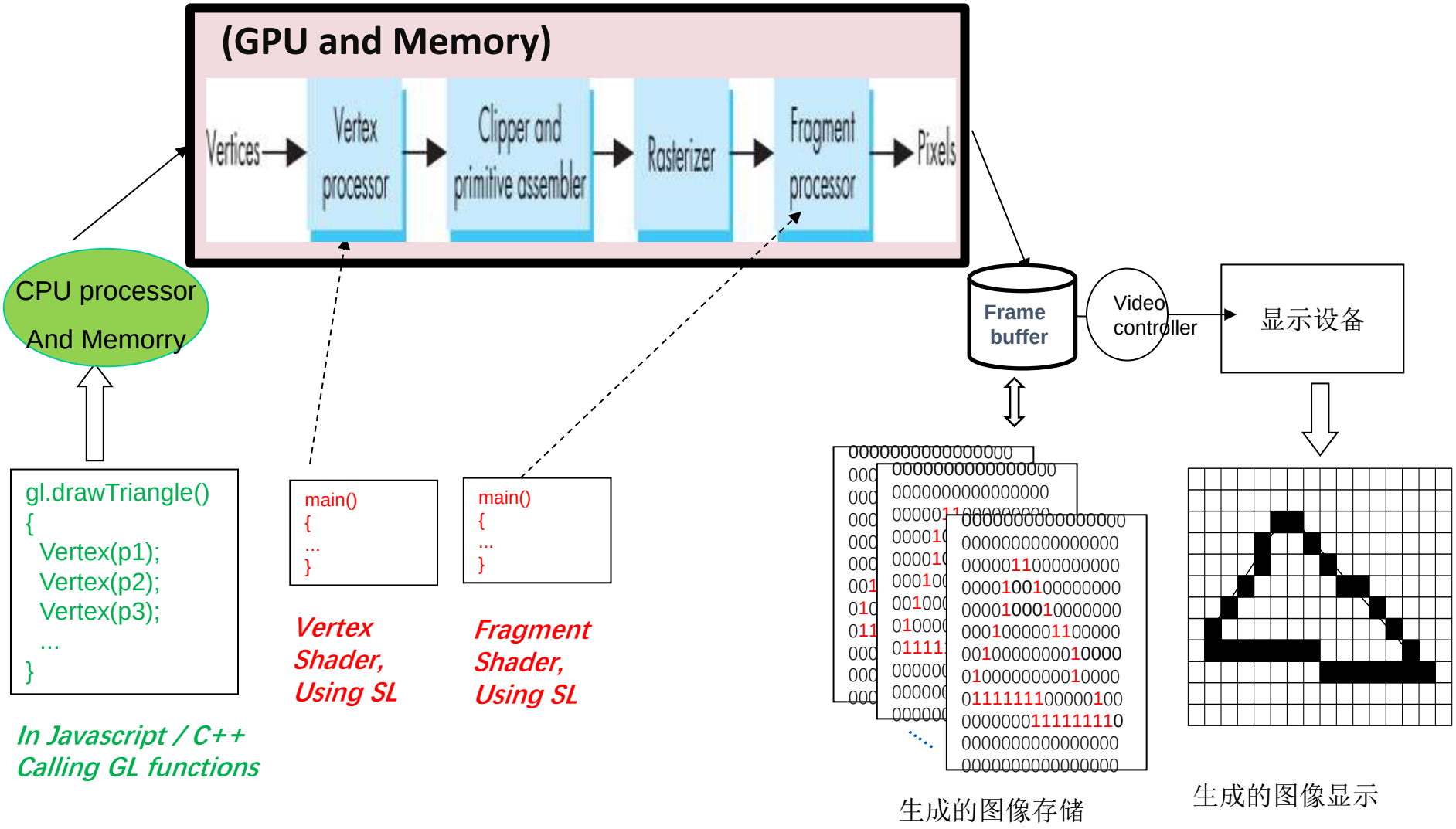
$$\frac{Yw-(-1)}{1-(-1)}=\frac{Ys-h}{0-h}$$ 得到 $Yw=-1+\frac{2(Ys-h)}{-h}=-1+\frac{2(h-Ys)}{h}$

Outlines

- GL History
- GL Architecture and Functions Formats
- A Standard Program Structure using WebGL*
- Shader Programing using GLSL *
- Animation and Interaction using WebGL*

Summary

➢The Programmable Rendering Pipeline



Summary(cont.)

- **OpenGL references:**

  www.opengl.org

  https://learnopengl-cn.readthedocs.io/zh/latest/

- **WebGL references:**

  -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://get.webgl.org/
  - https://webglfundamentals.org/webgl/lessons/zh_cn

- **GL函数查询：**

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

  使用方法：不需要注册登陆，右上角搜索框内输入函数名