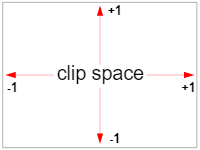
**WebGL初学笔记1 by Jiaqi Guo**

**2021-10-4** [**jacky\_guojj@outlook.com**](mailto:jacky_guojj@outlook.com)

Recommend Website from course：<https://webglfundamentals.org/webgl/lessons/zh_cn/>

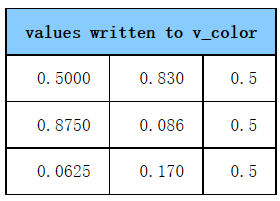
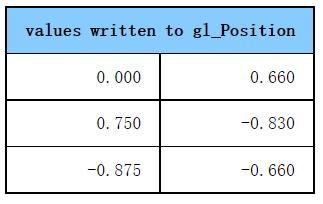
* **WebGL 基础概念：**

1. WebGL的定义一般只会涉及到两个参数，坐标空间和颜色，为此我们需要初始化两个shader来为WebGL提供以上信息。此外需要注意的是无论画布大小如何，我们的canvas空间坐标始终在-1 to 1之间，同理，颜色区间也被定义在0 to 1之间（本质是把像素值映射并归一化到特定空间，同理把RGB的值255映射到0-1的向量空间）



1. 为什么我们绘制出来的图形有时是彩色的，或者说表面是花的？

假设我们有三个点的vertex坐标



图表

描述已自动生成

由图可见，根据两个顶点的分别数据，v\_color将在这一对值中间进行内插

1. 缓冲区 buffer 和 attribute的实际作用

缓冲区是获取顶点和其他每个顶点数据到 GPU 的方式。 **gl.createBuffer**创建缓冲区。 **gl.bindBuffer**将该缓冲区设置为要处理的缓冲区。 **gl.bufferData**将数据复制到缓冲区中。这通常在初始化时完成。（具体例子： **程序1**的7，8步 ——— 分配vertex属性）

  var vertexBuffer = gl.createBuffer(); // get it's 'handle'

gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

然后我们再将数据拷贝到区域中：

  gl.bufferData(gl.ARRAY\_BUFFER, vertices, gl.STATIC\_DRAW);

之后，我们需要先询问WebGL，得到其分配给属性的位置，即**"a\_color"**和**"a\_position"**，分别对应着WebGL的颜色属性和位置坐标属性。（对应**程序1**的 9，10步）

  var a\_Position = gl.getAttribLocation(gl.program, 'a\_Position');

一旦我们知道属性的具体位置，我们就会在绘制之前发出剩下两个指令：

  gl.enableVertexAttribArray(a\_Position);

该命令告诉 WebGL 我们要从缓冲区提供数据

  gl.vertexAttribPointer(a\_Position, 4, gl.FLOAT, false, 0, 0);

//gl.vertexAttribPointer(

location,

numComponents,

typeOfData,

normalizeFlag,

strideToNextPieceOfData,

offsetIntoBuffer);

该命令告诉 WebGL 从当前绑定到 ARRAY\_BUFFER 绑定点的缓冲区中获取数据，每个顶点有多少个组件 (1 - 4)，数据类型是什么 ( BYTE, FLOAT, INT, UNSIGNED\_SHORT, 等等...)， stride 表示从一个数据到下一个数据要跳过多少字节，以及我们的数据在缓冲区中的偏移量

* **程序1：Draw Multiple Points (ch03: MultiPoint.js)**

1. WebGL 着色器初始化

// Vertex shader program..

var VSHADER\_SOURCE =

  'attribute vec4 a\_Position;\n' +

  'uniform mat4 u\_ModelMatrix;\n' +

  'void main() {\n' +

  '  gl\_Position = u\_ModelMatrix \* a\_Position;\n' +

  '}\n';

// Fragment shader program

var FSHADER\_SOURCE =

  'uniform int u\_colr;\n' +

  'void main() {\n' +

  '   if(u\_colr == 0) {' +

  '       gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +

// RED if u\_colr==0

  '   } else { '+

  '       gl\_FragColor = vec4(0.0, 1.0, 0.0, 1.0);\n' +

// GREEN otherwise

  '   }\n;' +

  '}\n';

1. 对于WebGL使用时JavaScript代码的初始化，以及资源调用是否成功的验证（作为模板可以套用）

// Retrieve <canvas> element

  var canvas = document.getElementById('webgl');

// Get the rendering context for WebGL

  var gl = getWebGLContext(canvas);

  if (!gl) {

    console.log('Failed to get the rendering context for WebGL');

    return;

  }

// Initialize shaders

  if (!initShaders(gl, VSHADER\_SOURCE, FSHADER\_SOURCE)) {

    console.log('Failed to intialize shaders.');

    return;

  }

  // Write buffer full of vertices to the GPU, and make it available to shaders

  var n = initVertexBuffers(gl);

  if (n < 0) {

    console.log('Failed to load vertices into the GPU');

    return;

  }

1. 规定Cavas画板的背景颜色

  // Specify the color for clearing <canvas>

  gl.clearColor(0, 0, 0, 1);

四个参数分别为 red, green, blue, alpha(背景图透明度)；

此时WebGL的背景为黑色：

图片包含 激光, 游戏机, 绿色, 灯光

描述已自动生成

1. 清空画板

// Clear <canvas>

  gl.clear(gl.COLOR\_BUFFER\_BIT);

1. 在Canvas上面绘制6个点

  // Draw 6 points. see http://www.khronos.org/opengles/sdk/docs/man/xhtml/glDrawArrays.xml

 gl.drawArrays(gl.LINE\_LOOP, 0, n); // gl.drawArrays(mode, first, count)

      //mode: sets drawing primitive to use. Other valid choices:

        // gl.LINES, gl.LINE\_STRIP, gl.LINE\_LOOP,

        // gl.TRIANGLES, gl.TRIANGLES\_STRIP, gl.TRIANGLE\_FAN

      // first: index of 1st element of array.

      // count; number of elements to read from the array.

三个参数分别对应： mode, first, count

<https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/drawArrays>

* **初始化顶点缓存区**

1. 定义vertices顶点，利用思维坐标系（右手坐标系）

图示

描述已自动生成  var vertices = new Float32Array([

     0.0,  0.5, 0.0, 1.0, // CAREFUL! I made these into 4D points/ vertices: x,y,z,w.

    -0.2,  0.0, 0.0, 1.0, // new point!  (? What happens if I make w=0 instead of 1.0?)

    -0.5, -0.5, 0.0, 1.0,

     0.0, -0.2, 0.0, 1.0,   // new point!

     0.5, -0.5, 0.0, 1.0, //

     0.2,  0.0, 0.0, 1.0,   // new point!  (Note we need a trailing comma here)

      ]);

定义图像的六个顶点。

1. 在定义完成图形的六个顶点之后，我们需要在显卡中创建一个 Vertex Buffer Object (VBO)

// Then in the Graphics hardware, create a vertex buffer object (VBO)

var vertexBuffer = gl.createBuffer(); // get it's 'handle'

如果有需要可以在之后验证是否创建VBO成功

  if (!vertexBuffer) {

    console.log('Failed to create the buffer object');

    return -1;

  }

1. 将刚才创建的顶点缓存器和数组缓存器进行绑定，随后再把刚才创建的vertex坐标组储存进缓存区内,以方便GPU对数据进行读取。

  // Bind the buffer object to target

  gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

  // COPY data from our 'vertices' array into the vertex buffer object in GPU:

  gl.bufferData(gl.ARRAY\_BUFFER, vertices, gl.STATIC\_DRAW);

1. 询问WebGL分配给vertex属性的地址，并验证

var a\_Position = gl.getAttribLocation(gl.program, 'a\_Position');

  if (a\_Position < 0) {

    console.log('Failed to get the storage location of a\_Position');

    return -1;

  }

1. 将缓冲区object赋给上述指定变量，指定坐标维度，数据类型等，最后激活该变量

  // Assign the buffer object to a\_Position variable

  gl.vertexAttribPointer(a\_Position, 4, gl.FLOAT, false, 0, 0);

  // vertexAttributePointer(index, x,y,z,w size=4, type=FLOAT,

  // NOT normalized, NO stride)

// Enable the assignment to a\_Position variable

  gl.enableVertexAttribArray(a\_Position);

* **程序2: 旋转三角形 (ch04 RotatingTriangle\_withButtons.js)**

形状

描述已自动生成

1. 初始化旋转角度，建立一个4×4的JS矩阵来向GPU传递数据

  // Current rotation angle

  var currentAngle = 0.0;

  // Create, JS matrix whose values we will send to GPU to set the 'uniform'

  // (a 4x4 matrix in a single var) named u\_ModelMatrix

  var modelMatrix = new Matrix4();

* **画一个旋转的三角形**

  // Start drawing

  var tick = function() {

    currentAngle = animate(currentAngle);  // Update the rotation angle

    draw(gl, n, currentAngle, modelMatrix, u\_ModelMatrixLoc, u\_colrLoc);

    // Draw the triangle

    requestAnimationFrame(tick, canvas);   // Request that the browser ?calls tick

  };

  tick();

}

上述代码中有两个自定义的函数 分别是 animate和draw，其中：

**animate**：对三角形的旋转角进行实时更新

function animate(angle) {

//==========================================================================

// Calculate the elapsed time

  var now = Date.now();

  var elapsed = now - g\_last;

  g\_last = now;

  // Update the current rotation angle (adjusted by the elapsed time)

  var newAngle = angle + (ANGLE\_STEP \* elapsed) / 1000.0;

  return newAngle %= 360;

}

**draw: 绘制图中旋转的三角形，一共有红绿两个图形，这里以绘制红色三角为例**

function draw(gl, n, currentAngle, modelMatrix, u\_ModelMatrixLoc, u\_colrLoc) {

  // Clear <canvas> AND clear the Z-buffer for proper occlusion/depth test.

  gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

//--------------------RED triangle at Z== -0.5

  // Set the rotation matrix

  modelMatrix.setRotate(currentAngle, 0, 0, 1);

  modelMatrix.translate(0.35, 0, -0.5); // -z is away from eye

  // Pass the rotation matrix to the vertex shader

  gl.uniformMatrix4fv(u\_ModelMatrixLoc, false, modelMatrix.elements);

  // Pass the 'colr' value  to the fragment shader:

  gl.uniform1i(u\_colrLoc, 0); // set value for RED.

  // Draw the triangle

  gl.drawArrays(gl.TRIANGLES, 0, n);

* **程序2: 绘制旋转的3D图形(ch05 ColoredMultiObject.js)**

1. **准备一个4×4的模型矩阵，并将该模型矩阵的值转换成GPU可以解析的变量类型**

  // Create a local version of our model matrix in JavaScript

  var modelMatrix = new Matrix4();

  // Constructor for 4x4 matrix, defined in the 'cuon-matrix-quat03.js' library

  // supplied by your textbook.  (Chapter 3)

  // Initialize the matrix:

  modelMatrix.setIdentity(); // (not req'd: constructor makes identity matrix)

  // Transfer modelMatrix values to the u\_ModelMatrix variable in the GPU

   gl.uniformMatrix4fv(u\_ModelLoc, false, modelMatrix.elements);

1. **如何让绘制的图形转起来**

**将函数储存进一个变量中，然后反复执行：**

**其中**requestAnimationFrame**主要用于让浏览器不停更新图像**

//---------------Interactive Animation: draw repeatedly

  // Create an endlessly repeated 'tick()' function by this clever method:

  // a)  Declare the 'tick' variable whose value is this function:

  var tick = function() {

    currentAngle = animate(currentAngle);  // Update the rotation angle

    draw(currentAngle, modelMatrix, u\_ModelLoc);   // Draw shapes

//    console.log('currentAngle=',currentAngle);

    requestAnimationFrame(tick, myCanvas);

                      // Request that the browser re-draw the webpage

  };

  // AFTER that, call the function.

  tick();             // start (and continue) animation:

                      // HOW?  Execution j

umps to the 'tick()' function; it

                      // completes each statement inside the curly-braces {}

                      // and then goes on to the next statement.  That next

                      // statement calls 'tick()'--thus an infinite loop!

}

**draw函数解析：**

1. **清空画布**

  // Clear <canvas>  colors AND the depth buffer

  gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

1. **设置图形出现的位置**

  modelMatrix.setTranslate(-0.4,-0.4, 0.0);

// 'set' means DISCARD old matrix,

              // (drawing axes centered in CVV), and then make new

              // drawing axes moved to the lower-left corner of CVV.

形状

描述已自动生成坐标是（-0.4, -0.4, 0.0）所以正四面体会出现在左下角，即第三象限，如果将这组数据修改为（0.4，0.4，0.0）那么正四面体将与正立方体重合，如下图所示

形状

描述已自动生成

1. 从右手坐标系转换为左手坐标系，再进行坐标的缩放

  modelMatrix.scale(1,1,-1);

// convert to left-handed coord sys

// to match WebGL display canvas.

  modelMatrix.scale(0.5, 0.5, 0.5);

// if you DON'T scale, tetra goes outside the CVV; clipped!

1. 让绘制的图形旋转起来，利用模型矩阵的rotate属性

  modelMatrix.rotate(currentAngle, 0, 1, 0);  // Make new drawing axes that

  // modelMatrix.rotate(90.0, 0,1,0);

              // that spin around y axis (0,1,0) of the previous

              // drawing axes, using the same origin.

currentAngle 是一个不断在变化并且更新的变量，决定了旋转的角度

后面三个参数（0，1，0）决定了旋转的对称轴。

1. 将定义的顶点坐标数据传入

  gl.uniformMatrix4fv(u\_ModelLoc, false, modelMatrix.elements);

      // Draw just the first set of vertices: start at vertex 0...

在本程序中具体定义为

      // Face 0: (left side)

     0.0,  0.0, sq2, 1.0,     1.0,  1.0,  1.0,  // Node 0

     c30, -0.5, 0.0, 1.0,     0.0,  0.0,  1.0,  // Node 1

     0.0,  1.0, 0.0, 1.0,     1.0,  0.0,  0.0,  // Node 2

      // Face 1: (right side)

     0.0,  0.0, sq2, 1.0,     1.0,  1.0,  1.0,  // Node 0

     0.0,  1.0, 0.0, 1.0,     1.0,  0.0,  0.0,  // Node 2

    -c30, -0.5, 0.0, 1.0,     0.0,  1.0,  0.0,  // Node 3

      // Face 2: (lower side)

     0.0,  0.0, sq2, 1.0,     1.0,  1.0,  1.0,  // Node 0

    -c30, -0.5, 0.0, 1.0,     0.0,  1.0,  0.0,  // Node 3

     c30, -0.5, 0.0, 1.0,     0.0,  0.0,  1.0,  // Node 1

      // Face 3: (base side)

    -c30, -0.5, 0.0, 1.0,     0.0,  1.0,  0.0,  // Node 3

     0.0,  1.0, 0.0, 1.0,   1.0,  0.0,  0.0,  // Node 2

     c30, -0.5, 0.0, 1.0,     0.0,  0.0,  1.0,  // Node 1

每一个面由三个顶点（4D 左边）确定，右边的3×3的矩阵为图形颜色的定义矩阵（RGB）

1. 最后一步开始绘图

  gl.drawArrays(gl.LINE\_LOOP, 0, 12);

fig 2 Draw with TRIANGLES

形状

中度可信度描述已自动生成形状

描述已自动生成

fig 1 Draw with Line\_Loop