CLASE 1 IG: Intro WebGL

El material lo estoy cogiendo de las siguientes fuentes:

- Libro Real-Time 3D Graphics with WebGL 2 en Safari Online
- Tutorial en WebGL2 Fundamentals
- Tutorial en Tutorials Point.

Práctica 1

Empezamos por las diapos clase 1, y damos el script de ejemplo clase1-1 para empezar.

Las primeras diapos de la clase están siguiendo el ejemplo principal del tutorial de Tutorials Point.

Probando las primitivas

Podemos ir probando los diferentes tipos de primitivas: gl.POINTS, gl.LINES, gl.LINE_STRIP, gl.LINE_LOOP, gl.TRIANGLES en la llamada a gl.drawArrays()

Pasando uniforms

Comenzamos por pasar un color desde el main usando un uniform en el main.

Esto va en el fragment shader:

```
uniform vec4 uColor;
fragColor = uColor;
```

Y esto va en el main, al final del bloque step 3:

```
// look up uniform locations
var colorLocation = gl.getUniformLocation(shaderProgram,
"uColor");
// Set a random color.
gl.uniform4f(colorLocation, 0, 1, 0, 1);
```

Pintando rectángulos

Meter el cuerpo js en una función init

Para poder meter más funciones el javascript, vamos a meter todo lo que tenemos en una función llamada init, y luego añadimos la llamada al final de todo el script para que llame a nuestra función en la carga de la página

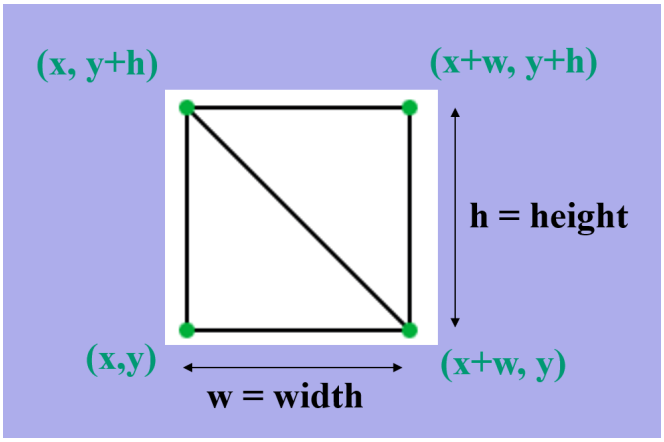
```
// function init() {
```

```
      …
}

// Call init once the document has loaded
       window.onload = init;
```

Creamos función para dibujar un rectángulo

Para pintar un rectángulo, lo haremos pintando dos triángulos conectados, como se ve en la figura:



De esta manera, voy a crear una función setRectangle que cree un array javascript de 6 vértices (para 2 triángulos) y lo meta en el buffer:

```
function drawRectangle(x, y, width, height) {
  var x1 = x;
  var x2 = x + width;
  var y1 = y;
  var y2 = y + height;

  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([
     x1, y1,
     x2, y1,
     x1, y2,
     x1, y2,
     x2, y1,
     x2, y2]), gl.STATIC_DRAW);

  gl.drawArrays(gl.TRIANGLES, 0, 6);
}
```

Y entonces comento la parte de pintar los tres puntos, y la sustituyo por una llamada a draw Rectangle.

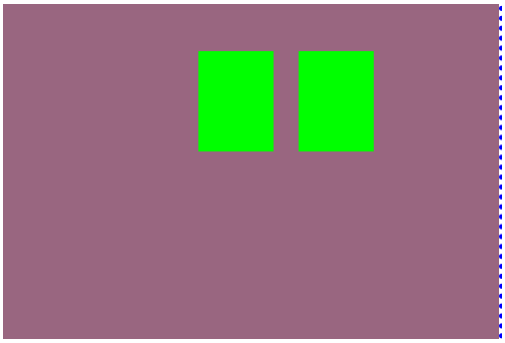
```
// Creates a rectangle and insert data in the buffer
drawRectangle(-0.2, 0.4, 0.3, 0.4);
```

Ojo que la variable gl debemos pasarla como parámetro al método, o bien declararla global, que nos será más sencillo. Para ello eliminamos la palabra var antes del primer uso de la variable.

Pintando varios rectángulos

Si queremos pintar un segundo rectángulo, basta con añadir una segunda llamada a drawRectangle:

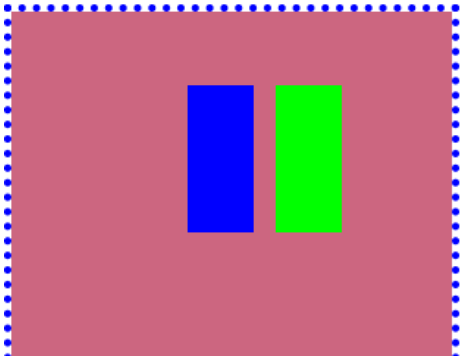
```
// Creates a rectangle and insert
data in the buffer
drawRectangle(0.2, 0.4, 0.3, 0.4);
```



Añadiendo color a la función setRectangle

Vamos a añadir el parámetro color al método setRectangle, para poder dibujar varios rectángulos a la vez en pantalla con colores diferentes. Simplemente lo añadimos como argumento, y movemos la llamada a gl.uniform dentro de la función

```
drawRectangle(-0.2, 0.4, 0.3, 0.4,
[0,0,1,1]);
drawRectangle(0.2, 0.4, 0.3, 0.4,
[0,1,0,1]);
```



Además habrá que pasar a global también la variable colorLocation (quitando el comando var previo)

Verlo en móviles

Solo tenemos que copiar el enlace al Live Site que nos da Glitch, y pegar la url en nuestro navegador móvil. Además, si cambiamos el código en glitch, y refrescamos la página en el móvil, **veremos los cambios actualizados**!

Animando la escena

Creando método render para animación

La parte exclusiva de renderización (bloque step 4) la metemos en un método aparte. Pero para eso habrá que pasar el canvas y el vertex_buffer a variable global, y que puedan ser accedidos desde el render. Y finalmente, al final del init, llamar al render();

```
function render() {

  //=== STEP 4: Create the geometry and draw =====

   // Clear the canvas
  gl.clearColor(0.6, 0.4, 0.5, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);

   // Set the view port
  gl.viewport(0,0,canvas.width,canvas.height);

  // Bind vertex buffer object
  gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);

  // pintar
  ……
    gl.drawArrays(gl.TRIANGLES, 0, 6);
  }

  // Unbind the buffer
  gl.bindBuffer(gl.ARRAY_BUFFER, null);
}


Function init(){
   …

   // renderizamos el frame
   render();
```

Si añadimos un mensaje por consola veríamos que el método render solo se está ejecutando una vez.

```
console.log ('mi primer programa WebGL!!');
```

Para que se ejecute varias veces por segundo, lo que se suele hacer es llamar al método window.requestAnimationFrame al final del método render, y esto hará que se llame a nuestra función de render en cada fotograma, para que siempre se esté refrescando

```
// empezamos bucle animación
window.requestAnimationFrame(render);
```

Haciendo que uno de los rectángulos caiga

Para animar uno de los rectángulos hay que tener en una variable la coordenada y de uno de los rectángulos, y hacer que en cada ejecución del render se vaya decrementando una cierta cantidad. La variable y1 la haremos global, para que coja su valor al principio, y luego dentro del método render la vamos decrementando

```
var y1=0.4;
```

```
y1 -= 0.01;
drawRectangle(-0.2, y1, 0.3, 0.4, [0,0,1,1]);
```

De esta manera el rectángulo irá cayendo poco a poco