Práctica 2 Imagen Sintética: Visualización de música con WebGL

Adrià Garriga Alonso 158636

13 de junio de 2015

En este documento describiré las técnicas utilizadas para las cinco versiones del visualizador de música. Las cinco versiones se encuentran en los directorios `v1` a `v5`. Necesitan ser servidos por un servidor HTTP, se pueden encontrar online en http://rhaps0dy.bananabo.xyz/. Deberían funcionar con tan sólo abrirlos en un navegador que soporte la WebAudio y WebGL, como pueden ser Mozilla Firefox, Google Chrome y Chromium.

Algunas versiones de Mozilla Firefox y Chromium no soportan mp3, así que puede haber problemas para reproducir la versión 1.

1. Versión 1

Implementación en Elm. `index.html` es la versión compilada, la fuente se encuentra en `v1/src`.

En esta demo, la CPU se encarga de reproducir la música, y analizar las frecuencias actuales con la API WebAudio. Se hacen 32 grupos de bins contiguas de FFT, se toma el logaritmo base 4000 de la media aritmética de cada grupo, y se asigna ese valor a cada una de las 32 fuentes. Código relevante en `processFrequencyData` en `Main.elm`.

Estos valores se utilizan, junto con el tiempo transcurrido en el último frame, para actualizar la representación en memoria de las fuentes. Esta representación sólo contiene el tiempo actual, el número de particulas actuales y la posición, velocidad y tiempo inicial de cada una. En particular, el valor obtenido de la FFT se utiliza para controlar la velocidad media de las partículas nuevas y el intervalo de tiempo entre creación de partículas. Código relevante en `update` en `Fountain.elm`.

La posición de las partículas se calcula en el vertex shader en GPU. Este y el fragment shader se pueden encontrar al final de `Fountain.elm`.

2. Versión 2

Implementación en Javascript con `THREE.js` y `MIDI.js`, así que los archivos que se distribuyen son la misma fuente.

Creamos la estrella en un canvas para usarla como textura (`function generateSprite` en `script.js`) y creamos las 28 fuentes (justo debajo de `function createEmitter`). En `function onload` cargamos el fichero midi, y decimos que cuando se reproduzca una nota se active la fuente correspondiente, y 0,25 s después se desactive.

Por cada partícula que se emite, sumamos una pequeña cantidad al hue del color de la siguiente partícula, para que las fuentes cambien de color.

3. Versión 3

Aquí volvemos a utilizar la Web Audio API. Creamos un analizador de FFT y forma de onda con 128 bins (`function onload` en `script.js`) y sus buffers correspondientes, que mapearemos a texturas para los shaders (líneas 178 a 201).

Los shaders relevantes se encuentran en las líneas 63 a 104 de `index.html`. Calculamos las coordenadas polares de cada vértice. Mapeamos entonces el valor de la textura con FFT en la posición determinada por la longitud a la posición en Z del vértice. Lo mismo con los colores en el fragment shader, multiplicamos la altura por el color blanco.

A esta posición determinada por la longitud le sumamos un "shift" que depende del valor de la forma de onda actual en la posición determinada por el ángulo de las coordenadas polares.

En la demo, en la parte x negativa (izquierda desde la cámara), se puede observar un artefacto. Esto es debido a la interpolación lineal de las coordenadas en el fragment shader, y a que las texturas a las que está mapeada la altura y el "shift" son buffers con principio y final. Este artefacto también está presente en la versión 5. Queda pendiente de arreglar de alguna forma.

Esta visualización también tiene, en el centro, un emisor de partículas parecidas a fuego. Este efecto se genera con un sprite de partícula en forma de círculo con los bordes muy difuminados, sacando partículas con hue entre 0.0 y 0.15, es decir, rojas, naranjas y amarillas. Además, el blending de color entre las partículas es aditivo.

El emisor de partículas se activa cuando hay un "kick" en la canción. Estos kicks los detectamos guardando el sexto bin de la FFT, y comparando su valor actual con el de hace 20 frames. Si pasa de un threshold, hay un kick. Funciona bastante bien. El código relevante está en las líneas 397 a 408.

A destacar también que para el texto generado se utiliza shading con normales interpoladas por píxel para las curvas exteriores, y shading con normales sin interpolar para las caras planas frontales y traseras.

4. Versión 4

En esta demo, en vez de mapear la FFT a la altura de una especie de onda circular, la utilizamos como mapa de alturas para generar terreno.

A medida que el tiempo avanza, se van guardando las alturas anteriores para formar la segunda dimensión del mapa de alturas. Líneas claves para esto son de la 132 a la 137.

Estos valores históricos de la FFT se utilizan para generar la altura del terreno en el vertex shader, en `index.html` en las líneas de la 31 a la 48. Las normales por vértice también se calculan aquí.

En el fragment shader, justo debajo, se utilizan las normales calculadas para calcular la iluminación, sólamente con el componente difuso. Dependiendo también de la altura, se colorea el fragmento de un color diferente. Si se colorea como agua, la normal se pone vertical, para dar la ilusión que el terreno es plano en esa zona.

5. Versión 5

Esta demo es una versión bien coloreada y iluminada de la 3. Hemos añadido iluminación básica, sólamente la componente difusa, de una luz central. Cuando hay un "kick", esta luz se activa con un color con hue aleatorio, y se va apagando suavemente después. También hemos coloreado las ondas con un degradado de negro a blanco repetido, radialmente.