# PEC1: Mi propia demo, DemoScene

Pedro Jesús Sánchez Illescas

April 3, 2024

# Contents

# 1 Librerías utilizadas

- SDL 2.30.0
- SDL-image 2.8.2
- SDL-mixer 2.8.0
- SDL-ttf 2.22.0

# 2 Proceso de creación

Todo el proyecto se ha realizado usando como base el código proporcionado para la práctica. Para empezar a usar todos los efectos en un mismo ejecutable, la primera fase del proyecto ha sido refactorizar la estructura de todos los efectos, viendo que todos los efectos tiene una estructura de inicialización, actualización y renderización. De esta manera, existe la clase abstracta *EffectTemplate*, con los métodos *init()*, *update(deltaTime)* y *render()*, que serán implementados en cada efecto. Estos efectos serán ejecutados por un algoritmo principal que empezará lanzando las inicializaciones, y los pares actualización −− renderización en cada frame, llevando también el cálculo del tiempo usando la clase *Clock*, basada en los ticks de SDL. Esta clase proporciona los valores de tiempo entre frames (*deltaTime*) y el tiempo total de ejecución para poder ser usados en los distintos efectos de una manera limpia.

La entrega consta de un efecto principal que contiene el tema musical "Road to Nowhere" (`https://www.youtube.com/watch?v=7k5H-sDAxwc`), pieza sin copyright de Roman Dudchyk, junto con una serie de efectos que estarán sincronizados con cada una de las secciones del tema. Debido a la duración de la música, algo más de 3 minutos, y para no repetir demasiado los efectos y quede una demo demasiado aburrida Entre estos efectos hay algunos que son exactamente los que vienen dados de ejemplo o tienen modificaciones muy ligeras como para que puedan ser consideradas como efectos personalizados (quitando el proceso de refactorización anteriormente citado).

Seguidamente se muestra una lista con los efectos añadidos a la entrega. Los efectos sin modificaciones importantes vienen marcados como "default".

- *BlackScreenEffect*: efecto simple que pinta la pantalla de negro usado como efecto dramático en los compases de cambio de sección.
- *BarsEffect*
- *FlockingEffect*
- *PlasmaPec1Effect*: efecto "default" con una función personalizada de generación de patrones.
- *FractalPec1Effect*: efecto "default" sin modificaciones del efecto de Mandelbrot.

- *DistortionPec1Effect*: efecto " default" de distorsión con una imagen personalizada.
- *SpyralEffect*
- *WhirlpoolEffect*
- *Pec1FlashEffect*
- *TexturizationEffect*
- *Pec1AudioEffect*

## 2.1 BarsEffect

Este efecto se empezó a desarrollar como una simulación de los barrotes de un campo de fuerza a partir de aplicar un filtro de detección de ejes al resultado del efecto de fuego, pero el resultado acabó pareciéndose más al efecto de una fotograma en pausa de las antiguas cintas VHS. El filtro total viene dada en la expresión (1).

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & -7 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{1}$$

Al resultado de la convolución se le ha añadido un pequeño umbral variante en el tiempo mediante la ecuación $u = 150 + 150 \sin(st)$, siendo $s = \frac{1}{BPM}$ y $t$ el tiempo de simulación para que los pulsos vayan al ritmo de la canción.

## 2.2 FlockingEffect

Este algoritmo simula un movimiento coordinado e independiente entre partículas que están a una distancia cercana, basado en la clusterización de partículas según su posición e intentando mantener uniforme la velocidad media del cluster y las separaciones entre los elementos del mismo. Este algoritmo viene explicado en la url `https://medium.com/@pramodayajayalath/flocking-algorithm-simulating-collective-behavior-in-nature-inspired-systems-dc6d7fb884cc`.

## 2.3 SpyralEffect

Este efecto representa el movimiento de los planetas formando galaxias. Es muy vistoso y fácil de implementar. Consiste en disponer de un número de partículas dispuestos sobre un plano, y se van desplazando en trayectorias circulares desde el centro de la pantalla siempre a la misma velocidad lineal. El tener todas las partículas la misma velocidad lineal hace que los puntos más cercanos al centro de la pantalla volteen el centro más rápidamente, creando los brazos de las galaxias y otras estructuras interesantes.

## 2.4 WhirlpoolEffect

Este efecto tiene una implementación más interesante, y consta de los siguientes pasos:

1. Construir un degradado radial a partir del degradado lineal dado en una textura.

2. Añadir un tiling pasado por parámetro para generar varios discos en la pantalla.

3. Implementar un offset de generación (píxel por donde se empieza a renderizar el degradado de forma cíclica).

4. Aumentar el offset en función del tiempo de simulación a una velocidad $\frac{screenWidth}{2BPM}$ para que en cada golpe llegue un círculo al centro.

## 2.5 Pec1FlashEffect

Este efecto de flash es una parte de la refactorización del efecto de sincronización entregado en los recursos de la práctica, pero en lugar de cambiar el alpha de un color hasta ser totalmente visible, el color se va haciendo invisible hasta dejar ver una imagen, que va cambiando de forma cíclica entre un grupo de imágenes dentro de la carpeta *assets/flash*. Al igual que con los efectos anteriores, la velocidad de cambio es proporcional a la duración de un pulso de la canción para sincronizar los cambios de imágenes con la música.

## 2.6 TexturizationEffect

Este efecto trata de texturizar un casquete esférico centrado en el origen, donde se encuentra la cámara, en lugar del interior del toro de la demo inicial. Para calcular los puntos de la esfera se utiliza la técnica de "distancia2", es decir, que en lugar de usar la ecuación tradicional $\sqrt{x^2 + y^2} = R$, se usa la expresión $x^2 + y^2 = R^2$, ahorrando las raíces cuadradas en todos los cálculos de todos los frames, con el consiguiente ahorro en rendimiento.

## 2.7 Pec1AudioEffect

Este efecto es la base que, además de ejecutar la canción, se encarga de administrar e ir ejecutando los efectos visuales de la demo, haciendo las veces de una modificación de la función main para administrar la ejecución de los efectos anteriores.

## 2.8 Transiciones

Las transiciones han sido diseñadas como efectos de segundo orden, que reciben dos efectos, fuente y destino, ya inicializados y se ocupan de ir actualizando ambos efectos y renderizando los píxeles que correspondan de cada uno de los efectos según un método abstracto *useSourceBuffer(i, j)*, que devuelve true si hay que renderizar

el píxel $(i, j)$ del efecto fuente en pantalla. En caso contrario, se usa el píxel del efecto destino. Se usa la terminología fuente – destino porque la renderización final se hace en la superficie del efecto destino. Esta funcionalidad base se implementa en la clase *TransitionEffect*, y actualmente hay tres implementaciones de transiciones:

- *CircleTransitionEffect*: un círculo se va abriendo o cerrando desde / hacia el centro de la pantalla, dependiendo de la dirección del efecto, añadiendo cada vez más píxeles del destino.
- *RowTransitionEffect*: en cada iteración se van añadiendo más filas del efecto destino a la imagen, ya sea de arriba a abajo o viceversa, según el sentido de la transición.
- *ColumnTransitionEffect*: similar al anterior, pero en columnas de izquierda a derecha o viceversa.

En el menú main se usan directamente estos efectos para transicionar entre los efectos dados en un vector, pero para usarlos dentro del efecto de audio se ha optado por usar una clase extra *EffectWithTransition* que tiene su efecto inicial (fuente) y final (destino). Esta clase elige una transición de forma aleatoria al iniciarse, y va revisando en cada frame si la transición ha terminado. En cuanto esto ocurre, solamente el efecto destino se ejecuta. De esta manera se puede usar transiciones de una manera razonablemente eficiente sin que resulte en un código muy denso, como ocurre con la función main, que primeramente ejecuta la transición con el efecto anterior y al terminar sigue ejecutando el efecto siguiente solo.

# 3 Compilando el proyecto

Como se indicó al principio de este documento, el proyecto solo tiene como dependencias la librería básica SDL más las extensiones image, mixer y ttf. En el archivo main.cpp hay cuatro flags de depuración para poder mostrar por pantalla los fps actuales, el título del efecto actual (en el main, con lo que solamente saldrá el efecto *PEC1AudioEffect*), el tiempo restante de simulación y la sección del tema musical que está activo actualmente. Por defecto está todo a $false$ para no tener distracciones en la pantalla, pero se pueden activar poniendo sus flags a true. Un ejemplo de estos flags en funcionamiento puede verse en el video `https://youtu.be/CDG5kHigrSQ` donde se ejecuta una serie con los efectos proporcionados para la PEC, tanto a resolución 640x480 como a 1440x1080.

# 4 Conclusiones

En esta PEC se ha entregado una demo con varios efectos basados en los dados en los materiales para la PEC. Aunque se ha intentado mantener una temática sobre el cielo (imágenes de nubes y planetas, movimiento espiral de galaxia), no ha sido posible mantenerlo en todos los efectos. Es imposible aplicar esa temática en el efecto de flocking y no sería muy vistoso una imagen con tantos tonos oscuros en el de texturización, por poner un par de ejemplos. La experiencia en general

aprendiendo sobre los efectos y diseñando los propios ha sido bastante buena, aun con el handicap de las técnicas de programación tan "oscuras" utilizadas en los códigos iniciales.

Así, para suplir mis carencias artísticas, he optado por implementar un pseudo motor de ejecución de efectos en el que se inicializa un vector con la lista de efectos a ejecutar hasta un tiempo determinado o una condición predefinida, pasando de un efecto a otro mediante una transición aleatoria. Estas transiciones son facilmente ampliables, puesto que se trata de una función que define qué píxeles de la imagen pertenecen a cada uno de los efectos implicados en la misma.