Escuela
Politécnica
Superior

# Demo técnica para PC

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Luis González Aracil
Tutor/es:
Francisco José Gallego Durán

Universitat d'Alacant
Universidad de Alicante

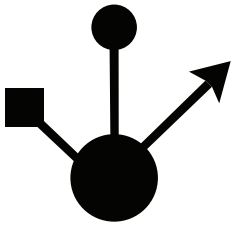
Marzo 2019

# Demo técnica para PC

**La demoscene**

**Autor**
Luis González Aracil

**Tutor/es**
Francisco José Gallego Durán
*Ciencia de la Computación e Inteligencia Artificial*



Grado en Ingeniería Multimedia

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Marzo 2019

# Resumen

Me dejo el resumen para el final

# Abstract

Some day I will write something in here, hopefuly...

# Motivación y objetivo general

Hablando con un compañero del trabajo sobre el lenguaje ensamblador, yo estaba intentando argumentarle su utilidad, a lo que él me contestó *"Hoy en día, saber ensamblador es como saber latín"*. Su afirmación zanjó el tema, pues a partir de ese momento no tuve ganas de seguir discutiendo, pero me hizo reflexionar. ¿Es inútil el ensamblador? ¿Sirve para algo el bajo nivel?

Para mí, la pregunta *"¿Para qué sirve saber ensamblador?"* es perfectamente equiparable a la pregunta "*¿Para qué le sirve a un arquitecto conocer las herramientas y materiales con los que va a construir una casa?"*.

Si una edificación cayera por una mala elección del material de los cimientos por parte del arquitecto, no habría duda en a quién culpar. Nadie abogaría que la culpa no es del arquitecto porque no es su responsabilidad conocer las bases. Sin embargo, hoy en día hay una enorme tendencia en el mundo del desarrollo software por menospreciar o infravalorar los cimientos de la programación, considerándolo algo arcaico y de carácter puramente didáctico, pero no práctico.

Yo me opongo radicalmente a esta visión, no sólo porque estoy convencido de la importancia de conocer el bajo nivel, si no que también encuentro cierta belleza en él. Cómo instrucciones en apariencia tan simples pueden construir sistemas tan complejos. A ello, se suma una gran curiosidad por saber cómo las cosas están hechas, desde el principio.

Una de las cosas que encuentro más apasionantes de la computación es la capacidad de los ordenadores, máquinas inertes y carentes de inteligencia real -por el momento- para reproducir nuestra realidad a partir de modelos matemáticos.

Los gráficos por computador son, por lo general, complejos. Sin embargo, hoy en día es posible crear con un ordenador imágenes que parecen fotografías y son capaces de engañar al ojo humano.

El objetivo principal de este trabajo es ir a las raíces, y revisar algunas de las técnicas que se usaban en los orígenes de los gráficos por computador para, a partir de operaciones con bajo coste computacional, generar escenas complejas.

*A mis padres, por estar ahí, siempre.*
*A mi hermana, por ser mi incordio y mi alegría.*
*A mi familia y amigos, por apoyarme y alegrarme los días.*

*A mi tutor,*
*por la visión que me ha dado sobre el mundo de la programación*
*y que tan valiosa es.*

*A Lola, por haber marcado la dirección cuando estábamos perdidos*

*If you give people*
*the choice of writing*
*good code or fast code,*
*there's something wrong.*
*Good code should be fast*

Bjarne Stroustrup

*When the whole world is silent,*
*even one voice becomes powerful.*

Malala Yousafzai

# Índice general

# Índice de figuras

# Índice de tablas

# Índice de Códigos

# 1 Introducción

Sin saber muy bien cómo, hemos llegado a un punto en el que conocer las bases del funcionamiento de un computador nos parece algo obsoleto, e incluso arcaico. Tecnologías de hace 20 años se tachan de reliquias en un mundo que aún no cuenta un siglo de antigüedad.

El irrefrenable avance de la tecnología y velocidad de evolución es innegable, pero a menudo, cuando se avanza muy rápido, también se pierde muy rápido.

La abstracción en el mundo de la computación ha sido un factor clave, de hecho es el factor que ha permitido que un set reducido de instrucciones como el que tienen los ordenadores sea capaz de imitar la realidad. Abstraer el software y llevarlo hacia modelos más cercanos al ser humano ha permitido pensar más en términos de nuestro día a día y menos en términos de mover memoria y realizar sumas y restas. Y esto es bueno, si para realizar cualquier mínima tarea tuviéramos que preocuparnos de hasta el más mínimo detalle de implementación, la curva de aprendizaje sería demasiado inclinada, y la eficiencia de la producción del software caería en picado.

Sin embargo, a más nos alejamos del hardware y más capas de abstracción añadimos, las instrucciones que escribimos se alejan más y más del reducido set de instrucciones que nuestro ordenador puede ejecutar. Como dijo una vez David J. Wheeler, *"Todo problema en computación puede resolverse con otra capa de indirección, excepto el problema de tener demasiada indirección"*[1].

Esta frase, además de tener un punto cómico, plantea un problema más serio del que muchas veces nos damos cuenta. Hoy en día hay aplicaciones construidas dentro de webs. Para ejecutar código en el cliente de una web se usa *JavaScript*, un lenguaje interpretado. Esto significa que para ejecutar una instrucción de código máquina proveniente de *JavaScript* es necesario, primero, interpretar la línea de código, compilarla y traducirla al lenguaje de la máquina virtual de *JavaScript* que integra el navegador y que esta máquina virtual interactúe con el sistema operativo para ejecutar la orden necesaria. Esto obviando una gran cantidad de pasos intermedios e ignorando las propias capas de abstracción de la memoria, el funcionamiento del procesador, los posibles fallos de la caché del procesador... Y si a todo este proceso, ya de por sí complejo y con muchas capas de por medio, añadimos una aplicación web compleja que añade nuevas capas de abstracción sobre el propio código que se ejecuta en *JavaScript*... ¿No parece demasiado?

Y sin embargo hoy en día prácticas como esta son perfectamente aceptadas, e incluso a veces, son punteras en la industria. Se justifica el hecho de tener una gran capacidad de

[1] `http://www.stroustrup.com/bs_faq.html`

cómputo para poder añadir más y más carga computacional. Se habla de las ventajas en la velocidad de producción, o en la sencillez de manejo del alto nivel en comparación del bajo nivel. Y es cierto que en muchos casos se gana eficiencia o productividad, ¿pero y lo que estamos perdiendo a cambio?

Podemos estar reduciendo la velocidad de nuestro programa cientos de veces, y aún así muchas veces no importa, porque la diferencia entre que algo tarde en ejecutarse 0,001s a que tarde 0,1s se nota, pero tampoco importa tanto. Sin embargo, pensamientos como este son peligrosos, y son los que han llevado a que programas aparentemente sencillos y ligeros incluyan tiempos de espera al iniciarlos, tal y como argumenta Mike Acton [2].

Y lo que es más, cabe preguntarse, ¿de verdad tanta abstracción simplifica el problema?

La realidad es que hay software donde la gran cantidad de capas que lo forman no solo reduce su tiempo de ejecución, si no que aumenta su complejidad de forma innecesaria, hecho que al que se llama popularmente *overengineering*.

De hecho, hoy en día existen hasta aplicaciones gráficas y juegos dentro de la web. Capas de abstracción dentro de capas de abstracción. Pero en aplicaciones tan computacionalmente costosas como aquellas que manejan gráficos y/o modelos matemáticos, toda esta abstracción tiene un coste que pasa factura. Quizá es precisamente por este motivo, que empiezan a surgir iniciativas interesantes, como Wasm[3].

Los gráficos siempre han requerido grandes capacidades de cómputo, aumentar la resolución de pantalla supone un coste cuadrático. Es por ello, que en el terreno de los gráficos, la simplicidad, la sencillez y la eficiencia priman. En una web, la diferencia entre 10 o 100 fotogramas por segundo puede no ser relevante, pero en una aplicación gráfica, en una reproducción de vídeo o en un videojuego, es un factor clave.

Y sin embargo, a pesar de su altísimo coste computacional, los gráficos por computador nos acompañan desde principios de los años 50, (dónde los ordenadores eran miles de veces menos potentes) en forma de hacks rudimentarios que permitían generar gráficos a partir de ficheros de texto[4]. No obstante, el boom de los gráficos por computador se produciría en los años 80, con la aparición y popularización de los primeros ordenadores personales, así como del videojuego. Es en este marco en el que se originaría la ***demoscene***.

El propósito de este estudio es, pues, visitar el arte del *demoscening* e investigar y exponer algunas de las técnicas gráficas que más comúnmente se usaban en los orígenes de la *demoscene*. Pretende ser una vuelta a los orígenes, donde se exploren los gráficos desde una perspectiva actual pero cercana al bajo nivel.

[2] `https://www.youtube.com/watch?v=rX0ItVEVjHc&t=4620s`
[3] `https://webassembly.org`
[4] `http://www.catb.org/jargon/html/D/display-hack.html`

# 2 Estado del arte

## 2.1 La demoscene

Qué es la demoscene y to eso, origen, historia, motivación, perfil del demoscener

## 2.2 Grupos de demoscening

Grupos de demoscene, principales demos

# 3 Objetivos

Aprender el bajo nivel y rendimiento, crear varios efectos básicos, crear un efecto más complejo

# 4 Metodología

## 4.1 Tests de rendimiento

Herramientas que voy a usar y cómo los voy a medir

## 4.2 Las demos

### 4.2.1 Planteamiento inicial

Comprensión del problema (de la demo) y búsqueda de información

### 4.2.2 Planteamiento matemático inicial

Intentar reproducir sin código, de forma matemática, el problema a resolver

### 4.2.3 Implementación

Implementación de la ideal inicial a grosso modo

### 4.2.4 Refinamiento

Refactor code, improve code

## 4.3 Software

Crear un sistema que me permita ejecutar las demos desde 0, de forma sencilla, sin dependencias, dándome un array de datos y que sea **multiplataforma**

# 5 Tests de rendimiento

## 5.1 Implementación

## 5.2 Resultados

# 6 Demos clásicas

## 6.1 Fuego

### 6.1.1 Investigación inicial

### 6.1.2 Planteamiento formal

### 6.1.3 Implementación

### 6.1.4 Refinamiento

### 6.1.5 Resultado

## 6.2 Geometría

### 6.2.1 Investigación inicial

### 6.2.2 Planteamiento formal

### 6.2.3 Implementación

### 6.2.4 Refinamiento

### 6.2.5 Resultado

## 6.3 Planos infinitos

### 6.3.1 Investigación inicial

### 6.3.2 Planteamiento formal

### 6.3.3 Implementación

### 6.3.4 Refinamiento

### 6.3.5 Resultado

## 6.4 Plasma

### 6.4.1 Investigación inicial

### 6.4.2 Planteamiento formal

### 6.4.3 Implementación

### 6.4.4 Refinamiento

### 6.4.5 Resultado

## 6.5 RotoZoom

### 6.5.1 Investigación inicial

### 6.5.2 Planteamiento formal

### 6.5.3 Implementación

### 6.5.4 Refinamiento

### 6.5.5 Resultado

## 6.6 Deformaciones de imagen

# 7 Demo final

### 7.0.1 Investigación inicial

### 7.0.2 Planteamiento formal

### 7.0.3 Implementación

### 7.0.4 Refinamiento

### 7.0.5 Resultado

# 8 Conclusiones

El bajo nivel es importante, la demoscene no debe perderse, las matemáticas son fundamentales, avanzar hacia el futuro teniendo muy presente el pasado, a veces para avanzar hay que mirar atrás, entender lo que hicieron los que iban por detrás de nosotros y saber aplicarlo

A test for bibliography with XeLaTeX.[1]

# Bibliografía

M. Alfonso, B. Bernardo, C. Carlos, and D. Domingo. El problema de los gatos y los perros. *Mascotas*, 50:112–115, 2010.

M. Alfonso, M. Marta, and N. Nuria. Mi viaje a EEUU. *Revista de viajes*, 14:50–56, 2010.