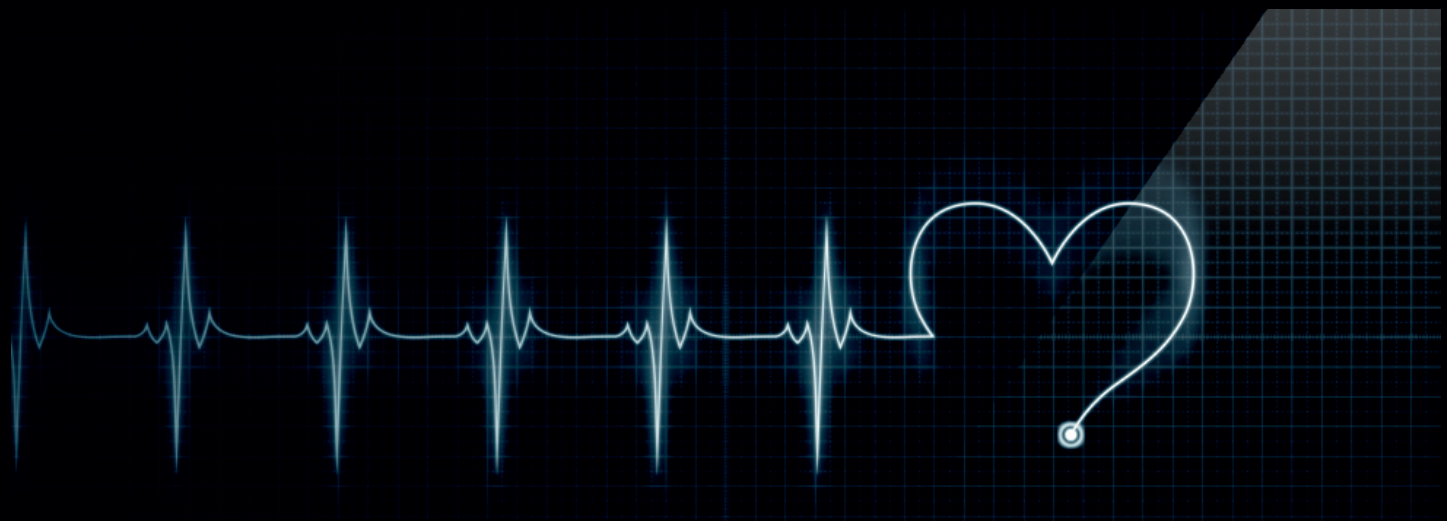
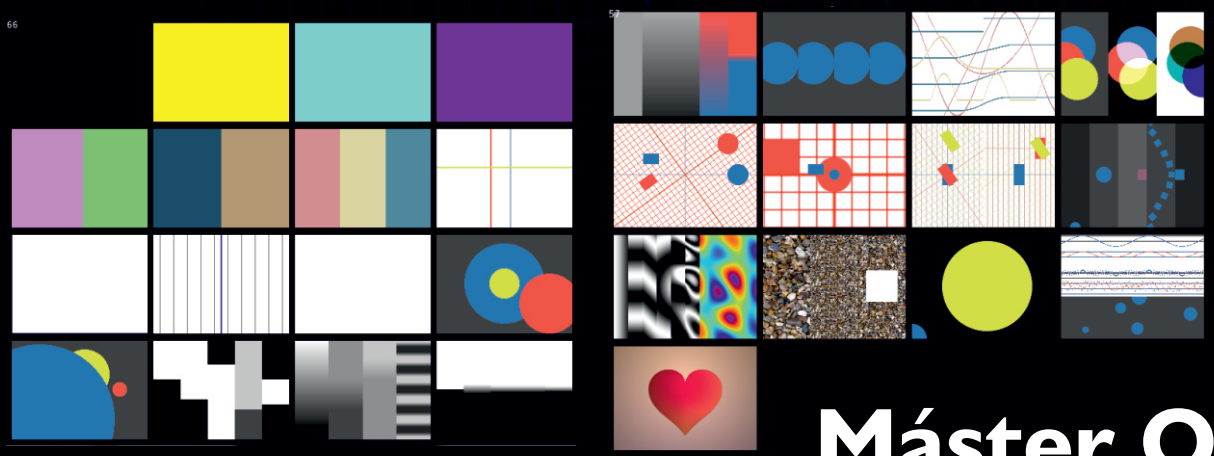
Procesadores Gráficos, Juegos y Realidad Virtual

Práctica 1: Shader de San Valentín





Máster Oficial en Informática Gráfica, Juegos y Realidad Virtual

Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

http://dac.etsii.urjc.es/rvmaster/

"Lo que oyes lo olvidas, lo que ves lo recuerdas,
lo que haces lo aprendes."
-- Confuncio

"By three methods we may approach wisdom:
First, by reflection, which is the noblest;
second, by imitation, which is the easiest;
and third, by experience, which is the bitterest."
-- Confuncio

Práctica 1: Introducción a la programación gráfica en GPUs

Objetivo de la práctica

El objetivo de esta práctica consiste en profundizar en los conceptos de arquitectura más básicos que se han introducido en las asignaturas de Gráficos 3D y en la primera parte de Procesadores Gráficos y Aplicaciones en Tiempo Real, aplicando estos conocimientos y reforzándolos sobre la base práctica de la programación de procesadores gráficos de última generación para PC.

En este primer acercamiento, vamos a familiarizarnos con una de las partes programables del cauce gráfico: el procesador de fragmentos; y asentar las bases (previa lectura de los capítulos recomendados) de la sencillas reglas de programación del lenguaje GLSL de medio-alto nivel[1], que nos permitirán apreciar los detalles de arquitectura durante la realización de estos ejercicios y poder realizar sofisticados efectos gráficos en posteriores asignaturas del Máster de Informática Gráfica, Juegos y Realidad Virtual.

No se va a hacer hincapié en el análisis y diseño en esta primera toma de contacto, dado lo pequeños y sencillos que van a ser estos programas iniciales a modo de ejemplo, pero son dos aspectos muy a tener en cuenta en cualquier desarrollo software, y en especial en *shaders* en los que la arquitectura puede condicionar fuertemente la forma de enfrentarse al problema a solucionar[2].

Planteamiento

La presente práctica consta de dos partes. La primera consiste en una serie de pequeños tutoriales que el alumno debe estudiar, para posteriormente realizar un shader que tenga una temática inspirada en San Valentín.

IMPORTANTE:

[1] Se ha elegido este lenguaje por su sencilla sintaxis (muy cercana al lenguaje C) que nos abstrae de problemas que nos alejarían de la función didáctica que buscamos en la asignatura (como la correspondencia entre registros y parámetros en los programas, por citar uno de muchos) , y sin embargo es lo suficientemente cercano al hardware como para poder apreciar las particularidades de la arquitectura de las GPUs actuales que tenemos en las tarjetas gráficas para ordenadores personales.

[2] Por supuesto, los alumnos son libres de hacerlo y serán bien valoradas este tipo de consideraciones en las memorias de las prácticas, pero en este momento es totalmente opcional dado el carácter inicial de esta práctica.

- En esta práctica habrá que entregar una memoria escrita en la que se respondan claramente a las preguntas y cuestiones propuestas en cada apartado.
- La entrega se realizará a través del Campus Virtual
- Es necesario incluir el código fuente o, al menos, indicar las modificaciones realizadas sobre el código existente.
- Es necesario incluir también una captura de pantalla (y un enlace a un video en YouTube en el que se visualice la animación en el caso de que las imágenes generadas cambien con el tiempo).
- Será absolutamente **obligatorio** comentar los resultados obtenidos.

Lecturas y tutoriales previos

Con el fin de adecuar la asignatura al nuevo espacio de europeo de enseñanza superior, y dado el reducido tiempo presencial que tenemos en la asignatura, parte de la explicación relativa al lenguaje de programación que vamos a utilizar se hará fuera del horario de clase por parte del alumno, mediante la realización de un conjunto de tutoriales, adaptados del trabajo de Uğur Güney e Iñigo Quilez.

El tiempo de clase lo aprovecharemos para concentrarnos en resolver las posibles dudas que hayan surgido a partir de la visualización del video y el "cacharreo" al realizar realizando los pasos del tutorial.

Estos pequeños tutoriales pueden ser objeto de preguntas en el examen, además es muy conveniente haberlos leído de cara a poder hacer con soltura las prácticas, ya que en ellos se explica de manera práctica un conjunto de conceptos relativos a la programación práctica de la GPU en lenguaje GLSL y la generación procedural de contenido.

Normativa del laboratorio

- El criterio más importante es la funcionalidad: un programa que funciona siempre tiene más posibilidades de llevarse una buena puntuación; no se valorarán aquellos programas que no funcionen. Una práctica o proyecto modesto será evaluada mucho más favorablemente que un "proyecto" ambicioso que sólo da *core-dumps*. Los siguientes criterios que se tendrán en cuenta (y que hay que cuidar al realizar las prácticas) son:
    - La manera de resolver el problema con el programa
    - Estructuras de datos y diseño de los algoritmos
    - Claridad y documentación en el código
    - Eficiencia y elegancia en la implementación.
- ¡Por favor, no hagáis trampas! Se procura alentar el diálogo y el trabajo en equipo, pero por favor trabajad de forma independiente (a menos que el trabajo sea en grupos). Trabajos muy similares serán considerados como copias, a menos que la naturaleza lo pedido sea tan restrictiva que justifique las similitudes. Y una copia implica el suspenso automático. Simplemente piénsalo de esta manera: hacer trampas dificulta el aprendizaje y la diversión de conseguir hacerlo. Es vuestra responsabilidad proteger vuestro trabajo y aseguraros que no se convierte en el de otro.

- Si se utiliza (o mejora) código fuente u otro material obtenido a través de internet, la biblioteca... debe darse el crédito a los autores y pedir permiso de ser necesario (si tiene una licencia restrictiva). Tomar código de un libro o de internet sin tener en cuenta estas consideraciones será considerado copia.

- Está terminantemente prohibido la práctica de técnicas de *overclocking* en las tarjetas gráficas del laboratorio, así como desbloquear los procesadores de de los chips gráficos. Este tipo de acciones pueden dañar físicamente el equipo del laboratorio y los alumnos responsables serán amonestados severamente.

- Las prácticas (código y memoria explicativa) deberán entregarse en los plazos indicados con las herramientas del portal de la asignatura en el Campus Virtual. La recepción de los trabajos a través del Campus Virtual restringe el periodo en el que se pueden enviar. Por favor, organizaros bien para evitar retrasos.

Recomendaciones

En principio la práctica se puede hacer en parejas, si alguien no encuentra pareja o por problemas de horario tiene complicado el poder equilibrar el trabajo también es posible hacerla de forma individual, esta práctica es sencilla y no debería suponer una dificultad añadida el no tener compañero/a para realizarla.

Índice del enunciado de la práctica

Objetivo de la práctica 1
Planteamiento 1
Lecturas y tutoriales previos 2
Normativa del laboratorio 2
Recomendaciones 3
Índice del enunciado de la práctica 3
1. Lenguajes de programación de shaders (de medio-alto nivel) 4
1.1 HLSL (High Level Shading Language) 4
1.2 Cg (C for Graphics) 5
1.3 RenderMonkey 5
1.4 GLSL (OpenGL Shading Language) 5
1.5 Otros lenguajes de programación de shaders 6
2. Escena básica 6
3. Entorno básico de programación (etapa de sombreado) 7
4. Shader de San Valentín 8
5. Ejemplo de práctica 9
Forma y fecha de entrega 9
Nota aclaratoria 9

1. Lenguajes de programación de shaders (de medio-alto nivel)

Hasta la llegada de los lenguajes de medio-alto nivel para *shaders*[3], la programación de los procesadores de la GPU debía hacerse directamente en ensamblador. El programador tenía que utilizar directamente los atributos de los vértices (y fragmentos) y los mnemotécnicos de sus operaciones. Éste era un proceso sujeto a errores, aunque el juego de instrucciones no era muy amplio en aquella época, lo que facilitaba su memorización; además los programas no podían tener muchas instrucciones debido a las limitaciones del hardware.

Con la versión del Shader Model 3.0, estas restricciones han ido desapareciendo y la necesidad de programar efectos gráficos más sofisticados ha hecho que pronto sean comunes los *shaders* de una longitud considerable.

Programar un código de cientos de líneas en ensamblador es una tarea ardua y tediosa. En las arquitecturas PC aún tenemos el consuelo de ganar algo de rendimiento respecto a la versión compilada en lenguajes de alto nivel. Pero en el caso de las GPUs ni siquiera tenemos ese aliciente, porque el código que programa un *shader* no se ejecuta realmente en las tarjetas de cauce clásico, sino que describe una configuración del hardware a que premite llevar a cabo el renderizado.

Por ello surgió la necesidad de facilitar la labor de programación, y los fabricantes de GPUs son cada vez más conscientes de la importancia crítica que tiene esto para atraer a los desarrolladores en un mercado -el de los videojuegos- donde prima la eficacia en la utilización de los recursos de la máquina y la velocidad de desarrollo por partes iguales.

En el mundo del PC los avances arquitecturales han llegado a un punto en el que cada vez es más difícil conseguir una mejora de rendimiento con los métodos tradicionales de mejora de prestaciones (sin que vayan acompañadas de un aumento considerable en el consumo), por lo que se invierte un esfuerzo más que notable por mejorar los compiladores. Sin embargo en el de las GPUs eso ha sido dejado de lado hasta hace bien poco[4], ya que las diferencias en la arquitectura no permitían aprovechar las mejoras en este sentido que se han venido haciendo durante años en arquitecturas orientadas a instrucciones, por ello incluso los llamados lenguajes de medio-alto nivel son muy dependientes de las arquitectura del chip gráfico para el que estemos realizando la aplicación. Un aspecto a nuestro favor en esta asignatura, ya que con ellos podemos aprovechar para estudiarlas al mismo tiempo que entendemos los entresijos de la creación de efectos gráficos muy avanzados.

Los 3 lenguajes de programación de *shaders* más populares son: HLSL, Cg y GLSL.

1.1 HLSL (High Level Shading Language)

Apareció en el año 2002 de la mano de Microsoft y algunos fabricantes de chips gráficos, como iniciativa para programar *shaders* apoyándose en el *Real-Time Shading Language* (RTSL) desarrollado por la Universidad de Stanford el año anterior.

[3]Buena parte de este apartado ha sido adaptado del texto del capítulo 10 (apartado 10.4 ayudas para la programación de la GPU: lenguajes de alto nivel) del libro *Procesadores Gráficos para PC* de Manuel Ujaldón. Para una explicación detallada, consúltese la referencia.

[4]Una excepción a esta regla es la aparición a principios de Noviembre (2006) de la tecnología *"Close to the Metal"* de ATI-AMD y *CUDA* de nVIDIA que permiten aprovechar la capacidad de cálculo de las tarjetas sin tener que lidiar con las APIs gráficas y el *driver* de la tarjeta, y abstrayendo totalmente al programador de las particularidades de este tipo de arquitecturas *streaming*.

Con este lenguaje los *shaders* ganaron mucho en elegancia y legibilidad, lo que se tradujo en una menor cantidad de errores, mejor depuración y se hicieron más fáciles de mantener y ampliar. Una de las mayores aportaciones de este lenguaje fue sacar a los *shaders* de la oscuridad críptica del ensamblador y con ello empezaron a ganar una cierta popularidad.

Al igual que en otros lenguajes de medio-alto nivel, el programador debe conocer la arquitectura gráfica y sus muchas limitaciones, ya que el compilador no genera un programa especialmente portable, y en sus inicios ni siquiera se encargaba de reescribir las operaciones que no eran directamente soportadas por el hardware.

1.2 Cg (C for Graphics)

Muy poco después, a finales del 2002, apareció el lenguaje Cg junto con un entorno de programación desarrollado por nVIDIA en estrecha colaboración con Microsoft e igualmente orientado a la rápida creación de efectos gráficos en múltiples plataformas software y hardware, que abarca toda clase de sistemas operativos (Windows, Linux, MacOS...) en las diferentes APIs gráficas (OpenGL y DirectX) y plataformas (PC, XBoX...). No está pensado sólo para los chips gráficos de nVIDIA, sino que también da soporte a las GPUs de otros fabricantes (ATI, Matrox...), de manera que un mismo *shader* puede optimizarse de forma sencilla para un gran número de sistemas que tengan los potenciales usuarios.

Esta gran portabilidad ha sido una de las principales razones para adoptarlo en la asignatura como lenguaje con el que reforzar los conceptos de arquitecturas gráficas para PC, ya que nos permite verlas desde dentro y tener la posibilidad de "cacharrear" con ellas.

Además nVIDIA, de forma gratuita, pone a disposición de los programadores de aplicaciones gráficas un conjunto muy atractivo de herramientas y SDKs de desarrollo para sacar el mejor partido de las tarjetas.

Este lenguaje puede ser utilizado de forma directa en los programas para realizar efectos gráficos y también embebido en ficheros con metainformación en el formato CgFX.

1.3 RenderMonkey

RenderMonkey es un entorno integrado, abierto y extensible de herramientas de desarrollo para la programación de *shaders* creado por ATI, para facilitar la cooperación entre programadores y artistas durante la creación de efectos gráficos en tiempo real. Hasta el momento de su aparición, las herramientas para el desarrollo gráfico tenían un aspecto rudimentario. Actualmente, tanto las herramientas de desarrollo de nVIDIA como las de ATI, permiten visualizar de forma interactiva los efectos gráficos a medida que se van realizando.

RenderMokey tiene soporte para HLSL y GLSL, dado que es ofrecida por ATI, parece lógico pensar que hará un mayor esfuerzo en incorporar aquellas prestaciones y efectos que favorezcan el lucimiento de la infraestructura hardware de que disponen sus tarjetas gráficas.

1.4 GLSL (OpenGL Shading Language)

El estándar OpenGL no ha sido el impulsor de ninguna de las novedades de programación referentes a los procesadores de la GPU, se ha visto arrastrado y afectado por el éxito de algunas de ellas. En realidad, más que por cubrir este nicho de mercado, OpenGL ha tenido que dar soporte a *shaders* por puro instinto de supervivencia, como respuesta a una comunidad de usuarios muy activa que comenzaba a plantearse otras posibilidades en vista de esta falta de funcionalidad.

Y a su vez, se podría decir, que los *shaders* han salvado a OpenGL, ya que desde su aparición en 1992 había ido perdiendo cada vez más portabilidad (perdiéndose su principal objetivo) debido a las exóticas extensiones que hacían unos y otros fabricantes de chips gráficos cuando todavía el cauce gráfico en hardware no tenía etapas programables y que ahora se han "unificado" gracias a estas nuevas funcionalidades.

Así pues, a finales del 2003 apareció GLSL, tras una evolución en paralelo con Cg, y que también toma a C y Renderman como base para su especificación.

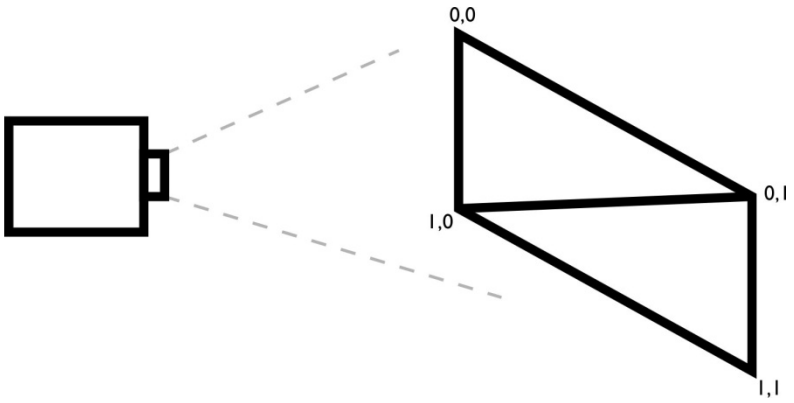
El lenguaje GLSL se incorporó a OpenGL a partir de su versión 2.0, y como consecuencia de ello, el cauce segmentado de OpenGL se modificó para considerar que los *shaders* "oscurezcan" algunas de las funciones tradicionales. El largamente esperado OpenGL 4.0 no ha traído consigo la reestructuración y orientación a shaders que se esperaba en esta nueva versión, aún así abre en la API las puertas a muchas novedades arquitecturales importantes que se han realizado en las tarjetas graficas más modernas.

1.5 Otros lenguajes de programación de shaders

Aunque los lenguajes comentados son los más utilizados, existe una gran variedad de lenguajes de programación para chips gráficos, tanto en el ambiente universitario (más orientados a experimentación e investigación) como en el mundo profesional de la programación de videojuegos. De entre ellos podemos destacar:

- Sh[5]: lenguaje de meta programación de *shaders*, desarrollado por la universidad de Waterloo (Canadá). Está orientado tanto a GPUs como procesadores no puramente *streaming* como los basados en la arquitectura Cell. Este lenguaje pretende dar un mayor grado de abstracción respecto a la arquitectura, pero su implementación actual (como librería para lenguajes orientados a objetos) no es capaz de obtener el mismo rendimiento que otras opciones de programación de *shaders* más cercanas al hardware.
- BrookGPU[6]: es la alternativa desarrollada en la Universidad de Stanford (USA), está más orientada a realizar aplicaciones de tipo general que aprovechen la potencia de cálculo de la GPU. Se podría considerar una variante de C, y necesita de una API gráfica (OpenGL o DirectX) para cargar sus programas en los procesadores del chip gráfico. El desarrollo se estancó en Octubre de 2004, fue retomado con cierta intensidad en Noviembre del 2007, pero parece haber sido abandondo frente a alternativas más totalmente orientadas a GPGPU coom OpenCL.

2. Escena básica



En esta primera práctica vamos a utilizar una herramienta especialmente diseñada para la codificación de shaders de fragmentos en la etapa de sombreado. Este tipo de shaders es muy utilizado en dos nichos que tienen mucho en común sus orígenes: la *demoscene* y la computación genérica

[5] Podeis encontrar más información de este lenguaje en http://libsh.org/
[6] La página web de este proyecto es: http://graphics.stanford.edu/projects/brookgpu/

en procesadores gráficos utilizando el cauce de OpenGL.

La escena de partida es muy sencilla, una cámara que apunta a un par de triángulos colocados a modo de *quad*, que cubren perfectamente el campo de visión. Las coordenadas de textura varían de cero a uno entre el vértice superior izquierdo y las coordenadas correspondientes de cada uno de los tres vértices en los extremos del campo de visión, de modo que tenemos una correspondencia perfecta entre los píxeles del framebuffer y los fragmentos que cubren la superficie del *quad*.
Podemos considerar que los colores calculados para cada uno de esos los fragmentos son los téxeles de la textura que lo recubre en el caso más simple de proyección, ya que la superficie es ortogonal al eje óptico de la cámara.

Este tipo de configuración se aprovecha en efectos gráficos en tiempo real de lo más variopinto (procesado de capas en postprocesado, filtrados, raymarching...). En el contexto de esta práctica es elegido por dos motivos: Desde el punto de vista didáctico los shaders de fragmentos son los más sencillos de abordar, nos ayudan a entender la interfaz de memoria en modo gráfico y tienen la gran ventaja de tener una salida casi directa, lo que permite poder visualizar el resultado del Shader con facilidad. Además, su parecido con los shaders de cómputo nos permite experimentar con técnicas de Computational Graphics y con la generación de texturas procedurales.

3. Entorno básico de programación (etapa de sombreado)

En esta primera práctica vamos a utilizar un entorno de desarrollo intuitivo para explorar y realizar las modificaciones a una serie de programas en el procesador de fragmentos que nos ayuden a reforzar los conceptos vistos en clase. Esta IDE es muy limitada, pero nos sirve para abstraernos de las complejidades en la comunicación entre la CPU y la GPU a través de las APIs gráficas, en las siguientes prácticas analizaremos el código de un *framework* que realice estas funciones, mucho más cercano a lo que se utiliza en aplicaciones reales y que nos permitirá ahondar en otras partes del cauce gráfico que no son programables pero sí configurables.

Cuando se inicia el entorno de trabajo (ShadorZ), se muestra una matriz de 4x4 pequeñas imágenes que se corresponden con los 16 primeros pasos de un tutorial de GLSL realizado por Uğur Güney, que hemos adaptado para que aprendais de forma interactiva unas primeras nociones de este lenguaje de programación de shaders. El tutorial se compone de un conjunto de 28 pequeñas lecciones[7] que cubren algunos aspectos sobre el sistema de coordenadas en la etapa de sombreado, la generación de formas analíticas en 2D, las funciones básicas de GLSL que suelen utilizarse en esta etapa, y cómo aprovechar las posibilidades que nos brinda el entorno de trabajo para hacer nuestras propias creaciones en forma de textura procederá la animada.

Al pasar el cursor sobre cada uno de los recuadros, se muestra el nombre del Shader correspondiente en el nombre de la ventana. Aunque los shaders se ejecutan de forma automática en esta ventana de bienvenida y previsualización, es conveniente seleccionarlos con un clic de ratón para trabajar con cada uno de ellos. Es conveniente que trabajes con ellos de forma ordenada, al inicio de cada uno de los nombres aparece un número que indica la secuencia de realización:

1. Blank screen.
2. Solid color
3. GLSL vectors
4. RGB color model and components of vectors
5. The coordinate system

[7] Para acceder a los pasos del tutorial 17-18, hay que presionar la tecla de la flecha derecha. Los cursores nos permite movernos entre los shaders que hayamos programado desde el entorno (y que se almacenan en la carpeta ./shaders).

6. The frame size
7. Coordinate transformation
8. Horizontal and vertical lines
9. Visualising the coordinate system
10. Moving the coordinate center to the center of the frame
11. Making the aspect ratio of the coordinate system 1.0
12. Disk
13. Functions
14. Built-in functions: step
15. Built-in functions: clamp
16. Built-in functions: smoothstep
17. Built-in functions: mix
18. Anti-aliasing with smoothstep
19. Function plotting
20. Color addition and substraction
21. Coordinate transformations: rotation
22. Coordinate transformations: scaling
23. Successive coordinate transformations
24. Time, motion and animation
25. Plasma effect
26. Texture
27. Mouse input
28. Randomness

Pulsando la tecla F1 obtener una lista con los atajos de teclado del entorno de trabajo. El más relevante es el que os permite abrir en un editor el código del Shader que estáis observando en un determinado momento (**mayúsculas+click derecho** del ratón).
Debéis estudiar el código de cada uno de los shaders, poniendo especial atención en los comentarios que se han escrito al inicio y en el interior del código. Podréis observar el efecto producido de cualquier cualquier modificación del código en el entorno de trabajo, simplemente editando y guardando el fichero desde el editor.

Además , es importante visualizar el siguiente tutorial llamado "formulanimations", en el que Iñigo Quilez muestra los principios para pintar figuras 2D a partir de sencillas fórmulas matemáticas:

https://www.youtube.com/watch?v=0ifChJ0nJfM

El entorno de trabajo que se muestra en el video es muy parecido al que nosotros utilizamos. La ventaja de ShadorZ es la posibilidad de trabajar sin conexión a internet. Si deseáis realizar la práctica en ShaderToy (el entorno de trabajo para el desarrollo de este tipo de Shaders en WebGL creado por Iñigo Quilez y Pol Jeremias), debéis aseguraros que también funciona en el entorno propuesto para la práctica (las limitaciones de los dos entornos son ligeramente diferentes).

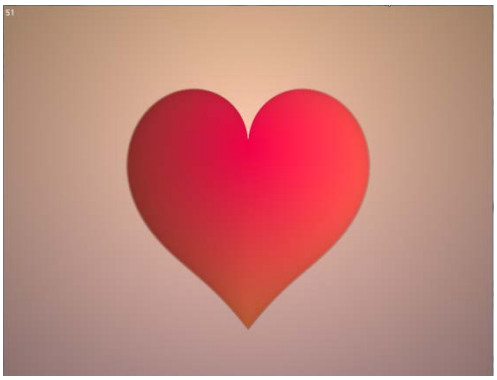
4. Shader de San Valentín

El objetivo de la práctica es realizar un Shader que muestre en pantalla (con ayuda del entorno de trabajo) un efecto gráfico en dos o tres dimensiones inspirado en la temática de San Valentín. Se valorará tanto el carácter técnico como el artístico de vuestra creación; se premiará especialmente la originalidad y la creatividad del shader.

En la carpeta tex se puede colocar una imagen arbitraria que sirva de base para el efecto gráfico que vosotros queréis conseguir, dicha textura debe ser únicamente un apoyo. El efecto gráfico generado con vuestro shader debe tener entidad de por sí, no un añadido a la imagen alojada en esta carpeta.

5. Ejemplo de práctica

El último de los recuadros en la segunda página del entorno de trabajo muestra un ejemplo de shader inspirado en San Valentín, en el que se muestra una animación de un corazón que late.



Es el shader número 29.

La imagen se ha sintetizado / modelado con un conjunto de fórmulas sencillas que modifican el color y de un pequeño conjunto de formas base (también generadas con fórmulas sencillas). La combinación de estas piezas da lugar a una textura procedural que a primera vista parece haberse generado en las distintas etapas del cauce tradicional a partir de una lista de vértices.

Forma y fecha de entrega

Se entregará la memoria en el formato de cualquier procesador de textos (ó PDF) debidamente identificada junto con el shader en GLSL en formato texto (y la textura que opcionalmente podéis añadir), tal y como los utilizaríais en el entorno de trabajo que se ha presentado en esta práctica.

La memoria debe estar acompañada por una captura (que puede volcarse desde el entorno de trabajo con la tecla F12) y un enlace a un pequeño video del shader en YouTube, si habéis realizado una animación.

Puede enviarse directamente mediante la aplicación correspondiente en el Campus Virtual dentro del plazo, e indicando "[PG Practica01]". No olvidéis indicar vuestros nombres y apellidos en el cuerpo del mensaje.

Las entregas realizadas antes **de las 15h del día 14 de Febrero** tendrán un punto extra.
La entrega de la práctica finalizará a las **15h del día 16 de Febrero**.
Un jurado independiente valorará el carácter artístico de vuestras creaciones de San Valentín, otorgando un premio a los tres shaders que más les gusten:
- 1er premio: 2 puntos extra
- 2o premio: 1'5 puntos extra
- 3r premio: 1 punto extra

Planificaros bien.

Nota aclaratoria

Todas las marcas y productos mencionados en este enunciado de prácticas están registradas por sus respectivas compañías, y su uso es de carácter descriptivo con fines docentes.

Tal como se aclara en el enunciado, la primera sección de esta práctica, con contenido teórico, está fuertemente inspirada en el apartado 10.4 del libro *Procesadores Gráficos para PC* de Manuel Ujaldón, y la práctica se basa parcialmente en las ideas y el entorno de programación de StorMoid y Uğur Güney.