Procesado de Triángulos de Bézier Cuadráticos por Rasterización Directa ajustada a la resolución de salida.

Trabajo Final del Máster en Inteligéncia Artificial,

Reconocimiento de Formas e Imagen Digital.

Departamento de Sistemas Informáticos y Computación.

Universidad Politécnica de Valencia.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Fernando Ruiz Velarde

Julio 2015

Agradecimientos

A mi pareja, M.ª Dolores, por animarme a continuar hasta el final.

A mi Tutor Ramón Mollá, por guiarme en la dirección y pasos a seguir.

A Vicente Julian Inglada y al Dpto. Sistemas Informáticos y Computación, por ayudarme a compaginar el Máster con mi actividad profesional.

A la Universidad Politécnica de Valencia, por las infraestructuras y recursos que pusieron a mi disposición.

Resumen

En este trabajo se parte de la idea de ampliar el pipeline gráfico actual, habilitando la programación de la fase de raster.
Partiendo de esta propuesta se muestra, como aplicación, la posibilidad de tratar superficies curvas de forma directa, y ajustada a la resolución de salida requerida.

Se realiza un estudio del estado del arte, tanto en la evolución de los sistemas de pipeline gráfico, como de las técnicas históricamente empleadas para la representación de superficies curvas.
Tras una clasificación de estas técnicas, se retoman ideas y trabajos menos evolucionados y se realiza una propuesta de método propia, observando su viabilidad actual, inconvenientes inherentes y una implementación inicial del mismo.

Tras ejecutar una serie de pruebas experimentales, se ha podido comprobar que el coste computacional (en tiempo de ejecución), aunque mayor que para el trazado de un triangulo simple, es sensiblemente inferior a aplicar métodos de subdivisión.
Destacar que este método se aproxima a la curva real de forma continua y más ajustada que por tramos lineales, y al ser su coste constante y solo dependiente del área a cubrir, cabe tenerla en consideración para futuros desarrollos.

Finalmente se describen líneas de continuación del trabajo, tanto en aspectos a mejorar sobre el método propuesto, como nuevas variaciones del mismo.

Índice general

Capítulo 1

Introducción

En base a la evolución del Pipeline gráfico, se observa que se han ido incorporando nuevas técnicas para mejorar los resultados obtenidos (texturas, shaders, LOD), pero que siguen basándose en métodos que quedaron establecidos en sus inicios (interpolación lineal y descomposición en superficies planas –triángulos-), por motivos de potencia y recursos de la época.

Considerando que actualmente hay potencia suficiente para incluir técnicas más avanzadas para los elementos base, se propone una revisión en la fase de Rasterización, de modo que se puedan emplear interpoladores no lineales. Esta propuesta proporcionaría (entre otras aplicaciones) la posibilidad de procesar superficies curvas (como triángulos de Bézier cuadráticos), obteniendo una representación ajustada a la resolución de salida requerida.

A continuación, en el Capítulo 2, se realiza una breve presentación de la evolución de las capacidades gráficas y librerías, tanto para un uso gráfico y usos generales; concluyendo con una crítica a esta situación.
Posteriormente, se muestra un estado del arte sobre el tratamiento de superficies curvas.

En el Capítulo 3 se da paso a la propuesta de incluir la Fase de Raster Programable. Tras una definición genérica de la forma en que esta fase podría integrarse en el pipeline actual, en el Apartado 3.2 se propone una aplicación práctica directa que daría uso a esta ampliación: El tratamiento de Superficies Curvas. A lo largo del Capítulo se hace una recopilación de las líneas de investigación que han existido hasta el momento para tratar este

tipo de geometría. Se clasifican en base a su forma de abordar el problema, haciendo finalmente una valoración de estas.

El Método Propuesto, principal objeto de este trabajo, es detallado en el Capítulo 4.
Tras compararlo con la técnica de trazado de triángulos, actualmente aplicada de forma fija en la fase de raster, se explica la operativa de esta propuesta con un pseudocódigo base para implementar el algoritmo.

Para sustentar la propuesta, se definen los elementos en los que esta se basa, y los estudios matemáticos que son de aplicación.
Se detallan los casos y situaciones que el problema presenta, las restricciones a aplicar para dirigirlo a un entorno manejable, y propuestas para la solución de este, siempre atendiendo a los objetivos de utilidad y aprovechamiento de los cálculos realizados.
Posteriormente se muestra el desarrollo de una primera implementación.

En el Capítulo 5 se muestran la serie de experimentos realizados, con el objeto de tomar medidas de tiempos de ejecución y hacer una valoración final de la viabilidad y perspectivas de utilidad del método.

Finalmente, en el Capítulo 6, se comentan una serie de trabajos posibles por los que continuar en esta línea. Quedan divididos en dos bloques: describiendo variaciones en las técnicas empleadas para la resolución de situación descrita; y apuntando a nuevos objetivos susceptibles de ser tratados con un método similar al propuesto.

Por ultimo, el Capítulo 7 presenta las conclusiones que se extraen de este trabajo, tanto en el ámbito de la utilidad de la propuesta de adoptar una fase de raster programable, como del interés histórico del tratamiento de curvas en 3D, así como a tenor de los resultados obtenidos con el método propuesto.

Capítulo 2

Estado del Arte

2.1. Evolución del Hardware gráfico

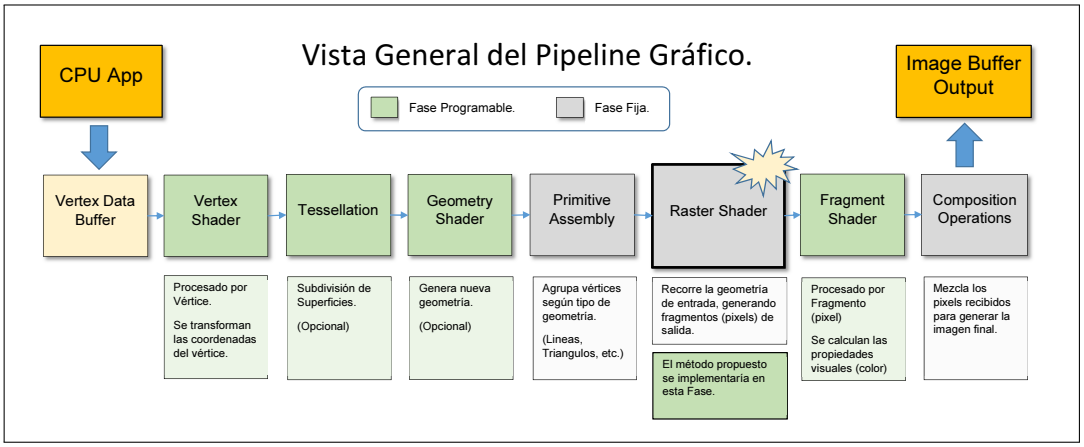
Desde la aparición de los primeros dispositivos gráficos, este aspecto de la informática ha ido evolucionando, tratando de presentar mejoras tanto en calidad como en rendimiento.

Las primeras tarjetas gráficas ofrecían un mecanismo simple de comunicación con el dispositivo de salida (monitor), pero fueron evolucionando para tratar de especializarse en las tareas propias de representación, dejando al procesador libre de estas cargas. Comenzaron a disponer de sus propios bancos de memoria, y hardware especializado para tareas tales como descompresión de vídeo (MPEG).

La aparición de los gráficos 3D, en tiempo real, forzaban a nuevas tareas de calculo para la generación de la imagen, y "previas" a la representación de esta.

En 2001 (con la tarjeta GeForce3) surgió el concepto de Unidad de Proceso Gráfico Programable. Permitía modificar el comportamiento de los cálculos que se realizaban en la tarjeta gráfica. Dotaba de esta forma un grado de libertad, al poder incluir los cálculos y nuevas técnicas en la tarjeta.

En aquel momento, solo dos fases eran programables: para el procesamiento de datos vectoriales (vertex shader), y en la fase final, para la determinación final del aspecto de los pixels de salida (fragment shader).



Vista General del Pipeline Gráfico.

Más recientemente, tanto los fabricantes de tarjetas gráficas, como los desarrolladores de APIs han centrado su avance en 2 aspectos:

Uso Gráfico

Tratando de evitar cuellos de botella, se busca reducir al máximo el flujo de información entre la memoria del sistema y la propia tarjeta gráfica.

- Datos Retenidos en la Tarjeta

Por una parte, mantener de forma permanente el mayor numero de datos en memoria gráfica.
En las ultimas especificaciones tanto de Direct3D como OpenGL, se fuerza (marcando como obsoletos los métodos anteriores) a enviar todos los datos de origen (vértices, índices, normales, texturas, etc) en bloque, quedando almacenados en la memoria de la tarjeta gráfica.

- Generación de Geometría al vuelo

Para el ahorro de memoria, en cuanto a la descripción de los objetos a mostrar, se ha añadido una nueva fase en el pipeline, que permite la subdivisión y generación de nuevas primitivas, a partir de las descritas en la memoria.

Esta fase también se ha beneficiado de la posibilidad de ser programable por el usuario. Concretamente se componen de varios estados: Geometry Shader, Tesellation Evaluator e Instanced Geometry; situados tras la fase de vertex-shader, y antes del rasterizado.

Uso General: (General Purpouse Graphic Processing Units)

El otro aspecto que mas está destacando es el uso de los procesadores gráficos para tareas de procesamiento genéricas, no directamente relacionadas con la generación de imágenes. Acuñándose el termino de GPGPU, Unidades de Proceso Gráfico para Propósitos Generales.

Critica al Estado actual

El presente trabajo parte la situación actual observada.

Queda vista la evolución de las fases del pipeline, la inclusión de nuevas fases y como el permitir su programación (programable shaders) ha contribuido a la eficiencia del sistema, y han permitido la aplicación de nuevas técnicas, aumentando así la calidad y posibilidades de estos sistemas gráficos. [Hillesland13b]

Si bien la tendencia actual es el uso de estas unidades gráficas para otras tareas (GPGPU), estas no tienen un impacto directo en la generación de gráficos 3D en tiempo real, ni en cuanto a eficiencia ni a la mejora de su calidad. Es de utilidad continuar revisando el pipeline existente, para que siga avanzando en favor de la tarea para la que fue concebido.

Por esto, se propone la posibilidad de habilitar la programación en fases actualmente fijas, más concretamente la fase de raster. En base a este modelo, se podrán tratar problemas actuales de forma distinta.
Concretamente, el presente trabajo, se centra en el estudio del tratamiento de superficies curvas, y propone un método para ello.

2.2. Tratamiento de Superficies Curvas

2.2.1. Precedentes

Existen múltiples estudios realizados a lo largo de los años, para la representación de superficies y líneas curvas [Goraud71] [Lane80] [Bezier86] [Rockwood89] [Hersch93], y otros que denotan interés en incorporar estos a los nuevos sistemas gráficos. [Levin76] [Blinn78a] [Nehab08] [Leben10] [Bruijns98]

Atendiendo a la forma de proceder, se pueden clasificar en 3 tipos:

- por subdivisión (tessellation)
- por trazado de rayos (raytrace)
- por rasterización (scanline)

Por Subdivisión (Tesselation)

Estos métodos se basan en la aproximación por medio de planos contiguos entre si. Se calculan una serie de puntos de la superficie, en base a una distribución. Estos puntos forman un entramado de superficies planas de menor tamaño, pudiendo ser cuadradas o triangulares. [Loop87]
Para cada cada uno de ellos se genera una nueva primitiva básica, que continúa su proceso por el pipeline habitual. [Krishnamurthy07]

En [Vlachos01] se utiliza una primitiva triangular para generar una superfice de Bézier, utilizando las normales de los vertices para curvar la superficie. Esta aproximación realizaba la subdivisión en la CPU, antes de enviar los polígonos a la GPU.

Actualmente, se encuentra el Shader Model 4.0 [Microsoft11], por medio del cual, en la fase de Teselación, se calculan en GPU una rejilla de nuevos vértices, que posteriormente serán modificados al pasar por el Vertex Shader.

La calidad/suavidad de la superficie obtenida dependerá del número de puntos o subdivisiones realizadas a la superficie definida originalmente.

Si la subdivisión no es suficiente, se podrá percibir la aproximación de manera evidente. A mayor cantidad de subdivisiones, mayor suavidad de la representación, pero con mayor coste computacional. Si la subdivisión es excesiva, se estarán generando un numero elevado de puntos y superficies que resultarán en información redundante. Un valor intermedio provocaría áreas de la curva supermuestreadas, mientras que otras carecerían de las divisiones suficientes.[Dyn90]

Debido a la propia naturaleza no lineal de una superficie Bézier, no es posible determinar un valor de subdivisión único que satisfaga todas las zonas de la superficie. El caso extremo se alcanza al realizar divisiones iterativas, con distribución no regular, en busca de obtener triángulos de 1 pixel de área. [Catmull74] [Clark79]

Por Trazado de rayos (Raytrace)

Estos métodos se basan en el calculo individual de cada píxel, respecto a la superficie. Inicialmente se procesa una geometría básica (triangulo, quad, tethaedro, etc), y se rasteriza su área.

Se encuentran dos variantes de estos métodos:

- Representar una figura completa

Trata de representar una figura completa, visible o contenida en el interior de la primitiva básica. [Schwarz06]
Para cada punto dentro de la primitiva básica, se calcula intersección de un rayo a la superficie definida. Este Rayo pasa por coordenadas (x,y) de pantalla, previamente convertidas a (u,v) respecto a la primitiva básica.
Esto puede dar lugar a puntos sin intersección, o múltiples intersecciones. [Blinn78b]
En estos casos, se busca la intersección mas próxima a la vista ($z = 0$).

En [Loop06], se dibuja un tetrahedro (pirámide de triangulos equilateros), calculando cada pixel como trazado de rayos en la fase de Fragment Shader. Se basa en una técnica analítica, limitada a superficies de grado4, y muestra errores de precisión. Una extensión [Toledo07], utiliza un cubo ocmo greometría base y ecuaciones de grado 6.

Por tanto, estos métodos realizan cálculos de raíces sobre la ecuación que define la superficie.

- Representación basada en contorno

Se centran en representar un área definida por los vértices de la primitiva básica, mas unos puntos de control asociados. [Pasko96]

El caso mas simple se centra en representar una curva de Bézier.
La primitiva básica utilizada es un triangulo, en el que uno de los vértices es considerado el punto de control de la curva.

Estos métodos se centran únicamente en el "relleno" del área. Es decir, en determinar si cada píxel concreto pertenece al interior o exterior del contorno. [Loop05] [Kokojima06] No tienen en cuenta la interpolación de valores sobre el área, tales como profundidad o coordenadas de textura, que siguen siendo interpolados de forma lineal.
Su principal aplicación está dirigida a mejorar métodos antialiasing de contornos, al poderse obtener un valor de distancia del pixel al contorno de la curva. [Krishnamurthy07] [Santina11]

Los primeros de estos métodos están orientados a la representación de superficies definidas matemáticamente y empleando la "fuerza-bruta" (Trazado de Rayos).
Los segundos, basándose en una definición de superficie mas concreta por medio de puntos de control, pueden ser fácilmente empleados por los diseñadores gráficos. Sin embargo, están mas orientados al calculo de contornos, para el relleno de áreas.

Por Rasterización (Scanline)

Estos métodos, aunque similares a los mencionados en el bloque anterior, se diferencian en que no realizan cálculos para posiciones de pixel que quedan fuera de la superficie a representar.
Los anteriores, al recorrer los pixeles generados por una primitiva básica, realizaban cálculos sobre todos ellos, aunque no pertenecieran a la superficie final y quedaran descartados. En este aspecto, se podría considerar que se están realizando cálculos innecesarios, que aumentan el proceso sin obtener con ellos beneficio alguno.

En este apartado se incluyen técnicas que identifican directamente el contorno de la superficie a representar, y se centran únicamente en recorrer su interior.

La técnica básica, lineal, es la ya incluida en la fase fija de rasterización del pipeline actual. Dado un polígono de 3 lados rectos (triangulo), se recorren estos por coordenada Y creciente. Al tratarse de lados rectos, este recorrido es realizado por interpolación lineal, y se pueden beneficiar de aplicar técnicas como el algoritmo de Bresenham. [Bresenham65]

Esto va generando puntos del contorno, que delimitan lineas horizontales (scanline) dentro del triangulo. (técnica de Flag Fill Algorithm [Ackland81])
De nuevo, se recorren estas lineas, interpolando linealmente los valores asociados. (u,v, normal, etc.)

Trabajos al respecto han tratado de ampliar este mismo concepto haciendo uso de triángulos de Bézier. Sin embargo, se remontan a momentos en que no se disponían de dispositivos de aceleración gráfica (GPU), y no se podían aprovechar de esta potencia de calculo. [Turner78]
Igualmente, se limitaban a determinar el área o contorno de la figura, sin interpolar (o hacerlo linealmente) los valores de las posiciones internas.

En la mayoría de estos estudios se interpolan los valores internos, como coordenadas de textura, profundidad-z, etc. de la forma habitual, linealmente. Su empleo se centra en la representación 2D de áreas curvas, y al trazado de fuentes.[Hersch93]

2.2.2. Valoraciones

Tras el estudio de estas líneas de investigación sobre la incorporación de superficies curvas a los pipelines gráficos, se pueden extraer las siguientes valoraciones:

Los métodos de subdivisión requieren determinar, de antemano o durante la ejecución, el nivel de subdivisión aplicar.
Si se trata de llegar a una representación lo mas fiel a la curvatura, es necesario llegar a niveles de patches de tamaño 1 pixel, lo cual implica que los resultados de cada refinamiento convergen a una misma coordenada final. Toda la carga queda en las primeras fases del pipeline (entradas al vertex shader), pudiendo ocasionar aquí un cuello de botella.
Posteriormente, al trazar un triangulo de una área tan ínfima, se desaprovecha la capacidad de la fase de rasterizado (aun con la implementación fija actual), de distribuir en paralelo el trabajo de la apariencia final del pixel.
La intención del método propuesto es de no requerir grandes esfuezos en fases previas, y tratar de aprovechar (no descartar) los cálculos que se realicen.

Los métodos de trazado de rayos consiguen una alta fidelidad a la curvatura requerida, sin embargo, entre sus mayores inconvenientes se encuentra el realizar cálculos de alto coste computacional, y que pueden resultar completamente descartados al comprobar que corresponden a posiciones fuera de la superficie.

Por ultimo, entre los métodos basados en rasterización, se observa que estos han quedado algo relegados en pro de métodos de subdivisión, probablemente debido a la incorporación en el pipeline de fases que promueven su uso.

Se centran en el trazado del contorno, sin proporcionar una solución al recorrido interno de la superficie, interpolando adecuadamente los parámetros sobre ella.

Otra intención del método propuesto es la de reaprovechar en la medida de lo posible los calculos que se realizan a cada iteración. Por esto, se hace interesante observar los métodos de recorrido de curvas por diferencias avanzadas.[Bartley97]

El método de diferencias adaptativas [Lien87] permite avanzar sobre la curva con una distancia fija (deseable de 1 pixel), y se han realizado experiencias con él. [Shantz88] [Chang89] A cada paso se calcula la distancia respecto a la división anterior, modificando los incrementos para obtener pasos del tamaño requerido.
Pero a cada cambio de tamaño de paso, se descartan y repiten los cálculos desde el paso anterior.

En el método propuesto se intenta no llegar a descartar ningún calculo, realizando únicamente aquellos que son útiles en la representación final.

Capítulo 3

Propuestas

3.1. Propuesta de Modelo

Partiendo del estado actual del pipeline, se propone la posibilidad de convertir la actual fase de Rasterización en una fase programable. (raster-shader) [Patney13] Esta propuesta abriría la puerta a la aplicación de técnicas personalizables, que resulten en nuevas capacidades y formas de obtener efectos finales. [Bruijns98] [Toledo04]

3.1.1. Pipeline con Fase de Raster Programable (Raster Shader)

Definición:

Otorgar al pipeline gráfico la posibilidad de incorporar un comportamiento dado por el usuario, en sustitución de la actual función fija de raster. Este programa sería ejecutado tras la ultima fase de Vertex/Geometry Shader y antes de ejecutar los Fragment Shaders.

Datos de Entrada:

Se recibirían todos los datos de salida del VertexShader, junto con información general de la geometría.

- Tipo de Primitiva a tratar. (POINT, LINE, TRIANGLE, QUAD, EXTENDED, etc...)
- Información de Vértices. (ya transformados y proyectados)

- Información adicional asociada al vértice (U,V,W, Normal, Color, etc)
- Información adicional de Geometry Shader (Convex Hull, Control Points...)

Datos de Salida:

Se calcularían los datos que recibirá como entrada la fase de Fragment Shader.

- Valores interpolados. (x,y,z, u,v,w, normal, color, alpha, etc. . .)

Imitando al actual "Geometry Shader" (generando nuevos vértices "al vuelo"), esta fase podrá generar múltiples instancias de ejecución del Fragment Shader. Posteriormente el pipe line continuaría de la forma habitual, con la ejecución de los Fragment Shaders y resto del Pipeline.

3.1.2. Aplicaciones para el Modelo Propuesto

- Tratamiento directo de superficies curvas.
- Representación de materiales multipass sin pasar por las fases previas.
  (en un mismo paso del raster pueden generarse multiples fragments a calcular, para una misma posición)
- Efectos de postproceso en linea. (generar varios fragment de salida con alphas variables para simular desenfoque por profundidad o movimiento)
- Control adaptativo de lineas y antialias.
  Se podría personalizar las condiciones en las que generar múltiples fragments para ajustar así el antialias por zonas críticas (contornos).
  En el dibujado de lineas, podría ajustarse el grosor en base a parámetros (p.ej. profundidad) o para añadir características visuales (patrones o grosor variable)

3.2. Propuesta de Aplicación

Concretamente, el presente trabajo se centra en proponer una programación alternativa para la fase de rasterizado, que permita hacer un tratamiento directo de superficies curvas. (Quadratic Bézier TriPatches)[Farin86]

Las ventajas de retrasar su tratamiento hasta esta fase se concentrarian en dos aspectos:

- Por una parte, se reduciría la carga de trabajo en fases iniciales, siendo no necesaria (o en menor medida) la subdivisión o generación de geometría de teselación.
- Por otra parte, la calidad final obtenida se vería incrementada, al obtenerse una representación mas próxima a la geometría descrita (superficie curva), siempre ajustada al tamaño de salida final.

De hecho, ciertas técnicas actuales como el almacenamiento o generación de múltiples geometrías por LoD no serian necesarias, así como tampoco los cálculos para estimar su impacto en la imagen a generar, para decidir el LoD a aplicar o nivel de teselación que generar.
Se reduciría este coste computacional, quedando disponible para otros usos.

Otras ventajas del tratamiento directo de este tipo de primitivas serían:

- Reducir la cantidad de memoria requerida para definir superficies de modelos orgánicos.
- Reducir la carga de trabajo en Vertex Shader debido a Teselación (incluida la realizada en GPU).
- Mejorar la calidad de la respresentación, con un coste computacional lineal al área afectada, y no dependiente de la calidad requerida.
- Poder evitar trabajos previos, como preparación de múltiples modelos para LOD, el calculo de estos en tiempo de ejecución, o incluso los cálculos para determinar el LOD a aplicar.

Aplicar esta técnica es completamente compatible y en ningún momento trata de sustituir las ya existentes.
Anterior a esta fase se podrá seguir utilizando subdivisión de geometría, o generación de esta 'al vuelo'.
Igualmente, en fases posteriores se podrán seguir aplicando técnicas como el normal mapping, parallax, etc. para aumentar el detalle del aspecto final de la geometría.

Por tanto, el objetivo de este trabajo es el de establecer dicho programa para tratar de forma directa la representación de superficies curvas.

Concretamente, y para simplificar una primera aplicación de este tipo de shader propuesto, se ciñe al uso de triángulos de Bézier cuadráticos.

Se espera una entrada que defina la superficie Bézier en los siguientes términos:

- 3 vértices, formando las esquinas del triangulo.
- 3 vértices de control, afectando la curvatura de las 3 aristas, y de la propia superficie.

  Asociado a cada vértice (tanto de superficie como de control), se obtendría información de posición (x, y, z), normal (nx, ny, nz), color (r, g, b, a), coordenadas de textura (u, v), etc.

El resultado consistirá en múltiples instancias de datos de salida, que continuarán por el pipeline gráfico habitual.
Para cada una de estas instancias se entregarán valores interpolados de la información de entrada. (posición, normal, color, coordenadas de textura, etc). Esta información habrá sido interpolada siguiendo la superficie Bézier definida por los datos de entrada.

Capítulo 4

Desarrollo

4.1. Método Propuesto

El objetivo, al iniciar los trabajos para este proyecto, era el de encontrar y proponer un Algoritmo de Representación de superficies curvas (triángulos cuadráticos de Bézier) con un planteamiento equivalente o similar al método de relleno de triángulos por lineas de raster.

Según se ha descrito en el capítulo "Tratamiento de Superficies Curvas", existen métodos que también tratan el problema de dibujar/rellenar la superficie, en base a lineas horizontales o raster ("Por Rasterización").
En estos métodos, se utilizan cálculos genéricos:

- Realizan una intersección de superficie con plano horizontal (que coincide con la linea de raster en el plano de proyección) [Manocha94] [Manocha95],
- o bien recorren la arista curva por medio del calculo directo de su inversa (con el consiguiente calculo de raíz cuadrada) [Blinn78b],
- o por aproximaciones iterativas (método de newton).
- o por aproximaciones precalculadas en una tabla al inicio del proceso (interpolando para obtener resultados en valores intermedios a los presentes en la tabla)

En cualquier caso, estos cálculos solo pueden ser aproximados, por la propia naturaleza de la función raíz cuadrada, por su imposibilidad de ser representado con exactitud (en el campo de la informática), y por su alto coste (en los métodos iterativos) para refinar la solución.

Por tanto, no se pretende en este trabajo, obtener una solución matemáticamente precisa, sino únicamente aproximar el resultado a las necesidades del problema. Es decir, obtener una representación visualmente aceptable, en relación a la resolución y área que sea preciso cubrir.

El método propuesto seguirá esta segunda filosofía de recorrer progresivamente la arista curva, pero aplicando ciertas consideraciones para reducir la complejidad del algoritmo. Se busca obtener una calidad en el contorno ajustada "al píxel", sin depender de niveles de detalle; y al mismo tiempo tratando de realizar una cantidad de cálculos lo mas reducidos y aprovechables posible.

Así mismo, se trata de aplicar el método de diferencias avanzadas, para poder avanzar sobre la arista con un coste fijo. El coste quedará claramente definido desde el comienzo del tratamiento del problema. Los cálculos concretos relativos a la apariencia, se aplicarán únicamente para cada píxel/fragment generado, siendo estos validos y no descartables desde su generación.

Por tanto, el método se aleja de toda aproximación por tramos lineales (Teselación), dependientes de una subdivisión ajustada a priori; o de la realización de cálculos que puedan quedar finalmente desechados, como en los casos de Traza de Rayos, o en la aplicación del método de diferencias avanzadas adaptativas.

4.1.1. Espacio de trabajo

El Método está orientado a la representación de superficies triangulares de Bézier (Bézier Tri-Patch) [Farin86] en base a su contorno.
Esta quedará definida en base a sus 3 vértices (extremos de las aristas), y 3 puntos de control, cada uno asociado a una arista, definiendo su curvatura. Por tanto, las aristas son, en base, curvas Bézier cuadráticas (grado 2).

Indicar que, todo el trabajo se realiza sobre un espacio de coordenadas ya proyectado, lo que comúnmente en el pipeline gráfico es denominado "clipping coordinates", que han sido tratadas previamente en la fase de "Vertex Shader".

4.1.2. Método para recorrido de superficies triangulares cuadráticas.

El Método sigue una analogía al método de trazado de triángulos por recorrido de aristas. Realizaré una descripción paso a paso, partiendo del método de relleno de triángulos clásico, mostrando las adaptaciones al uso de aristas curvas.
El método esta sujeto a restricciones, que serán descritas en próximos capítulos.

Analogías

Triángulos con aristas rectas. (metodo lineal)

1. Se recorre cada arista por el metodo de bresenham [Bresenham65] (linealmente desde su vertice inicial-superior al final-inferior)
2. Tomando de aquí: un punto inicial y otro final, se conforma una recta horizontal o linea de raster.
3. Se interpola dicha linea de raster linealmente
4. Se repite desde 2, tomando los próximos puntos de cada arista
5. Cuando la 1ª arista se completa, se toma la 3ª en su lugar y se continua.

Triángulos con aristas curvas. (método propuesto)

1. Se recorre cada arista curva por el método de recorrido propuesto a continuación.
   Adicionalmente se calcula y recorre una linea central sobre la superficie. (que será denominada "cuerda central de la superficie").
2. Tomando de aquí: un punto inicial, otro final, y un punto central (sobre la cuerda), se conforma la curva horizontal o curva de raster.

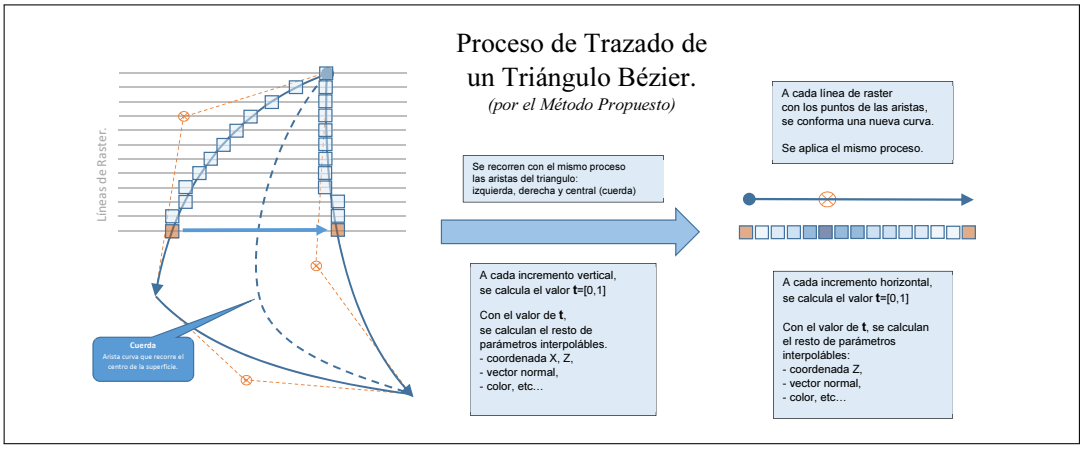
   Durante el recorrido se toman siempre puntos con igual coordenada Y, por lo que la curva formada queda siempre horizontal en la vista.
   Realmente sigue siendo una curva, al poder variar la coordenada Z
   (u otros componentes) no linealmente respecto a su inicio y final.

   A partir de la curva definida por los 3 puntos, se pasa a una formulación basada en pto.inicial pto.final y pto.control equivalente.
   De este modo, se podrá procesar esta curva de igual forma que las aristas.
3. Se recorre esta nueva curva por el mismo método propuesto a continuación.

4. Se repite desde 2, tomando los próximos puntos de cada arista.

5. Cuando la 1ª arista se completa, se toma la 3ª en su lugar y se continua.

Tomar en consideración que la arista central (cuerda), deberá estar definida para todo el rango a procesar.



Proceso de Trazado de un Triángulo de Bézier.

4.1.3. Método para recorrido de curvas cuadráticas.

Al igual que en el apartado anterior, realizaré una comparativa paso a paso, desde los métodos utilizados actualmente para recorrer lineas rectas, al método propuesto para el recorrido de curvas cuadráticas.
En ambos casos, se presenta como el recorrido de una función de dimensión 1.
Es directa la extensión del método a lineas en entornos de mayor dimensión.

Analogías

Lineas Rectas (lineal)

- Se presenta la recta X como formula paramétrica en función de t.

$$X = f(t) \quad : \quad t = [0, 1]$$

- Se divide t en un numero N de pasos de tamaño Δt.
  (se obtienen incrementos de t constantes)

$$\Delta t = \frac{1}{N}$$

- Se toman valores t_n avanzando desde $t = 0$ con pasos Δt
  (diferencias avanzadas con solo 1 nivel de incremento constante Δt)
- Al evaluar repetidamente X en cada t_n se obtienen valores X_n

$$X_n = X(t_n)$$

  Al ser una recta, la distribución es lineal, es decir:

$$\Delta X = X(t_n) - X(t_{n-1}) = cte.$$

  Se puede aplicar directamente diferencias avanzadas sobre X, y obtener el valor ΔX a aplicar para recorrer la recta.

- Para avanzar un $\Delta X = 1$, y así avanzar píxel a píxel, el Δt requerido es:

$$\Delta t = \frac{1}{N} \quad : \quad N = X_f in - X_i ni$$

  En rectas de dimensión 2 o superior se calculará el Δt que interese (coincidiendo con $\Delta X = 1$, o $\Delta Y = 1$), según la dirección del eje sobre el que se quiera avanzar

  En general, se calculan valores $X(t_n)$, $Y(t_n)$ etc. usando y avanzando sobre un mismo t_n.
  Se puede aplicar directamente diferencias avanzadas sobre X e Y, para obtener los ΔX y ΔY a aplicar.

Lineas Curvas (cuadráticas)

- Se presenta la curva X como formula paramétrica en función de t.

$$X = f(t) \quad : \quad t = [0, 1]$$

- Si (análogamente) se divide t en un numero N de pasos de tamaño Δt, se obtienen incrementos de t constantes

$$\Delta t = \frac{1}{N}$$

- Si se toman valores t_n avanzando desde $t = 0$ con pasos Δt

- Al evaluar repetidamente X en cada t_n se obtienen valores X_n
  pero con incrementos no constantes.
  Al NO ser una recta, la distribución NO es lineal, es decir:

$$\Delta X = X(t_n) - X(t_{n-1}) \neq cte.$$

Esta es la forma habitual para recorrer/evaluar una curva en función de un parámetro t.[Bartels87]

- Al utilizar un valor N demasiado pequeño, los valores obtenidos de X pueden estar demasiado separados. ($\Delta X > 1$)
  Esto generaría "huecos"que pueden ser unidos con lineas rectas, pero se obtiene como resultado un contorno no completamente curvo.

- Al utilizar un valor N demasiado grande, los valores obtenidos de X pueden estar demasiado juntos ($\Delta X < 1$)
  Esto generaría posiciones (discretas) repetidas, y se estarían realizando cálculos innecesarios o descartables.
  Esta aproximación sobre-utiliza los recursos.

Ambas aproximaciones están sujetas a cálculos iniciales para ajustar el nivel de detalle.

En cualquier caso, un mismo valor de N presentará ambos problemas, ya que ningún ajuste de nivel de detalle inicial es correcto para toda la curva.
También es posible ir ajustando el Δt progresivamente [Lien87], e ir recalculando sus incrementos (matriz de incrementos en el método de diferencias avanzadas), sin embargo:

- Para controlar este ajuste, se realiza el calculo del siguiente punto, y sobre el resultado obtenido se decide el ajuste necesario.

- A cada momento de ajuste, el calculo realizado es desechado (por estar demasiado cerca, o demasiado alejado)
  De nuevo esta aproximación sobre utiliza recursos, realizando cálculos no aprovechables finalmente.

4.1.4. Descripción del Método

Tratando de obtener los índices t_n para obtener puntos X a una distancia cte. [Peters94], y de este modo ni descartar cálculos (por repetición), ni "dejar huecos"(que deban ser unidos linealmente), se propone el siguiente método:

- Buscar una funcion g (que será denominada "función interpoladora cuadrática inversa")
  tal que:
  $$t = g(u) \quad : \quad u = [0, 1]$$
  se define la curva parametrizada como:
  $$X = f(g(u))$$

- Al dividir u en un numero N de pasos de tamaño Δu se obtienen incrementos de u constantes.
  $$\Delta u = \frac{1}{N}$$

- Al evaluar t en cada u_n se obtendrán valores t_n (a incrementos no constantes)

- Al evaluar X en cada uno de esos t_n se obtendrán valores X_n (a incrementos constantes)

Se está mapeando el espacio $t = [0, 1]$ sobre un espacio $u = [0, 1]$ de forma que sobre una interpolación lineal de u (con Δu constante) se puedan obtener valores de t suficientes y necesarios para calcular $X(t)$ espaciados adecuadamente.

En curvas de dimensión 2 o superior:

- Se ajusta el Δu para obtener incrementos constantes sobre un eje ($\Delta X = 1$, o $\Delta Y = 1$, según el eje a seguir que interese)

- Se van obteniendo los valores de t (con incrementos no constantes).

- Se calculan valores de $X(t_n)$, usando y avanzando sobre un mismo t_n

En el eje de referencia (X por ejemplo), se obtendrán los incrementos constantes deseados (Δu para obtener $\Delta X = 1$)
En el resto de ejes, se obtendrán los valores correspondientes a la distribución cuadrática correspondiente, al evaluar la curva sobre el t obtenido.

4.1.5. Pseudocódigo

Dibujado de un Triangulo de Bézier Cuadrático por Rasterización.

Entrada: 3 vertices del triangulo y 3 vertices de control (arista curva).
Salida: Se dibujan los puntos del interior del triangulo

{Definir aristas}
$A \leftarrow$ arista desde punto mas alto a punto de altura medio.
$B \leftarrow$ arista desde punto mas alto a punto mas bajo.
$C \leftarrow$ arista desde punto de altura media a punto mas bajo.
$Cuerda \leftarrow$ arista desde pto. mas alto al mas bajo, por el centro de la superficie.

{Definir interpoladores verticales}
$intA \leftarrow$ interpolador cuadrático que recorre la arista A (a incrementos $\Delta Y = 1$)
$intB \leftarrow$ interpolador cuadrático que recorre la arista B (a incrementos $\Delta Y = 1$)
$intCuerda \leftarrow$ interp. cuadrático que recorre la arista $Cuerda$ (a incr. $\Delta Y = 1$)

{Recorrer contorno en vertical, generando lineas de raster horizontales}
repetir
  {Calcular puntos para la iteración actual}
  $rasterIni \leftarrow$ punto calculado por $intA$.
  $rasterFin \leftarrow$ punto calculado por $intB$.
  $rasterMedio \leftarrow$ punto calculado por $intCuerda$.

  {Definir línea actual de raster (horizontal)}
  $aristaRaster \leftarrow$ arista desde pto. $rasterIni$ a $rasterFin$, pasando por $rasterMedio$.
  $intR \leftarrow$ interp. cuadrático que recorre $aristaRaster$ (a incrementos $\Delta X = 1$)

  {Recorrer linea de raster horizontal}
  repetir
    {Calcular punto interior}
    $F \leftarrow$ punto calculado por $intR$
    Dibujar(F)
    {Avanzar por linea de raster}
    $intR \rightarrow$ avanzar 1 paso.
  hasta que $intR$ llegue al final

{Avanzar por contorno de lineas verticales}
si $intA$ no ha llegado al final **entonces**
{Avanzar por arista A}
$intA \rightarrow$ avanzar 1 paso.
si no
{Sustituir arista A por arista C}
$intA \leftarrow$ interp. cuadrático que recorre la arista C (a incrementos $\Delta Y = 1$)
fin si
si $intB$ no ha llegado al final **entonces**
{Avanzar por arista B}
$intB \rightarrow$ avanzar 1 paso.
si no
{Sustituir arista B por arista C}
$intB \leftarrow$ interp. cuadrático que recorre la arista C (a incrementos $\Delta Y = 1$)
fin si
{Avanzar por arista Cuerda}
$intCuerda \rightarrow$ avanzar 1 paso.
hasta que $intCuerda$ llegue al final
Fin

4.2. Bases Matemáticas

4.2.1. Definiciones y Herramientas a utilizar.

A continuación se muestra una serie de herramientas matemáticas que han sido empleadas en el desarrollo del Método propuesto.
Entre ellas destacar la definición de 2 términos propios: "Cuerda Bézier", e "Interpolador inverso". Estos términos (no consensuados) identifican los elementos clave en que se basa el método propuesto, y se les presta una especial atención en su definición.

Curva de Bézier Cuadrática.

Se toma la definición paramétrica $f(t)$ de Bézier [Bezier67] y [Casteljau59], haciendo referencia a sus punto de inicio, punto de control y punto final.
Si bien la definición como polinomio de Bézier es mas adecuada para realizar cálculos eficientes, repetidamente se hará uso de la expresión dada por DeCasteljau, de la cual se pueden extraer mas directamente algunas conclusiones útiles.

Conversión de Curva sobre 3 puntos a Bézier.

En ciertas ocasiones se presenta la definición de la curva a utilizar en base a 3 puntos de la misma. Es importante aplicar una conversión a este caso, para obtener una definición equivalente de la curva, basada en los términos definidos como curva de Bézier.

Se toma la definición paramétrica de curva cuadrática que pasa por 3 puntos conocidos. Se toma el método de [Cordero02] para la conversión de esta curva a una en la forma de Polinomio de Bézier Cuadrático, obteniendo las referencias a su punto de inicio, punto de control y punto final.

La aplicación de este método se resume en:

Dada la expresión de una curva como polinomio de Bézier (basada en la expresión de DeCasteljau)

$$y = t^2 \cdot x_1 + t \cdot (1-t) \cdot x_2 + (1-t)^2 \cdot x_3$$

Conocidos los puntos $x_1, x_2, x_3, ...$ para cada valor de t se resuelve la ecuación.

Se Toma ahora un punto conocido por el que pasa la curva (valor de y en dicho punto). Se Tiene por tanto conocidos $x_1, y, x_3, ...$ por lo que dado t se resuelve x_2. Fijando este valor de t, se obtiene el valor x_2 como punto de control de la curva Bézier.

Si se trata la curva sobre 3 puntos expresada en función de un parámetro t, como la curva que "pasa"por el punto central dado cuando $t = 0{,}5$, y se fija este valor en la expresión anterior, queda despejado el valor para x_2 como punto de control para una curva Bézier equivalente.

Destacar que es necesario fijar un valor de t correspondiente al momento de paso por el punto central. En nuestra aplicación, se emplea $t = 0{,}5$, considerando siempre que el punto está situado a mitad de recorrido de la curva de entrada.

Superficie de Bézier Cuadrática.

Se toma la definición paramétrica de Bézier [Farin88], haciendo referencia a sus 3 vértices y 3 puntos de control.

Los vértices corresponderán con los vértices de sus aristas.
Los puntos de control corresponderán con los puntos de control de sus aristas.

Cuerda sobre superficie de Bézier. (arista central)

Cada linea de raster será a su vez una curva cuadrática.
Para definirla es necesario conocer los puntos iniciales y finales (obtenidos al recorrer las aristas), pero también un punto de control, que será el encargado de determinar la curvatura sobre esta linea de raster.

Se emplea una nueva linea sobre la superficie, cuya extensión abarque todo el rango vertical (eje Y) del triangulo Bézier a representar.
Esta "cuerda" permitirá obtener puntos sobre la superficie a cada coordenada Y evaluada (linea de raster).
Inicialmente se propone utilizar como vértices de esta curva: el vértice superior del triangulo, el vértice inferior del triangulo, y el punto sobre la superficie correspondiente a los parámetros $s = t = 0{,}5$ (centro de la superficie).

Así mismo, para obtener una sucesión de puntos eficiente, se transformaría la expresión de esta curva a polinomio de Bézier, para (de nuevo) aplicar el método de recorrido de aristas curvas descrito en este trabajo.
Para cada nueva linea de raster, el punto de la cuerda obtenido será un punto sobre la superficie (y por tanto punto de la curva).
Transformando de nuevo esta expresión de curva a la forma de Bézier, se obtienen las referencias adecuadas para realizar el recorrido de la linea de raster por el método propuesto.

Función Interpoladora Cuadrática inversa.

Sea la funcion g tal que:

$$t = g(u) \quad : \quad u = [0, 1]$$

Obteniendo la curva parametrizada como:

$$X = f(g(u))$$

A partir de la función cuadrática f (de 2^o grado), su inversa g es una función raíz con múltiples soluciones. Para evitar esta situación, se opta por restringir la curvatura máxima de las aristas de entrada a tratar.

Esta definición se complementa y detalla en las siguientes secciones.

Método de diferencias avanzadas.

Se toma el método de Diferencias Avanzadas [Lien87], llegándose a aplicar en este caso sobre ecuaciones de grado 4. [Bartley97]

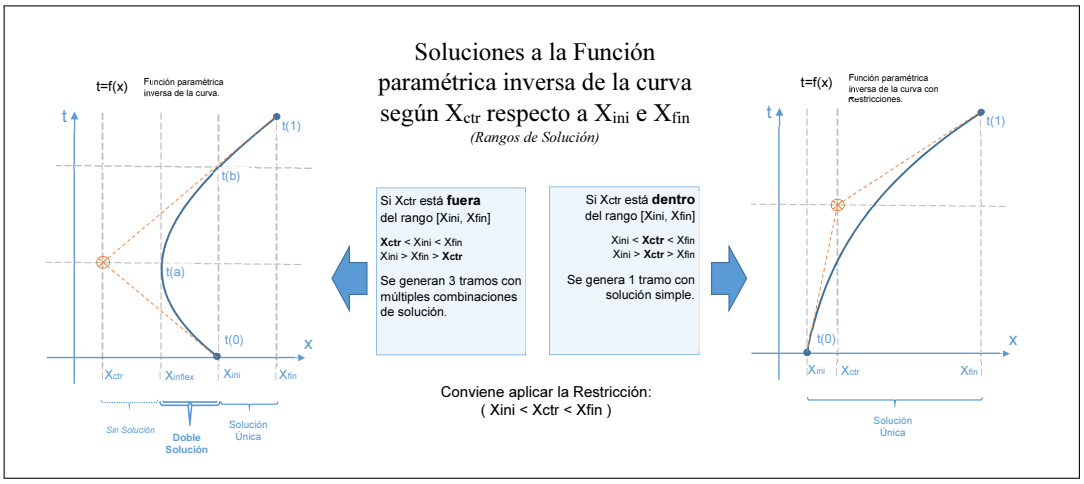
El principal interés en la utilización de este método, es el de poder avanzar en el recorrido de las arista, aprovechando cálculos anteriores, que no suponga un calculo completo desde cero.
Este método reduce drásticamente el coste computacional al avanzar progresivamente sobre la curva.

4.2.2. Análisis de la Función Interpoladora Cuadrática Inversa.

Basándose en la definición de Bézier de las curvas cuadráticas, y aplicando el teorema de inclusión, se pueden determinar en qué casos se producen situaciones no deseadas (múltiples soluciones, sin solución, etc.) [Blinn05a] [Blinn05b], y como acotar los rangos que sean de interés.

Entre las propiedades de este tipo de curva se encuentra que:

- La tangente en el punto inicial sigue la dirección del punto inicial al punto de control.
- La tangente en el punto final sigue la dirección del punto de control al punto final.
- La evolución de la tangente es de forma lineal (al reducirse el grado de su ecuación a 1).
- Este tipo de curva tienen, como máximo, un único punto de inflexión (cambio de sentido).
- Todos los puntos están incluidos en el triangulo formado por los puntos inicial, final y control.



Análisis de la curva.

Análisis de la curva. Situaciones de la solución.

Situación con doble solución.

Visualmente se entiende que, al recorrer la curva (avanzar en el parámetro t), los valores inicialmente se dirigen hacia el punto de control (aunque sin alcanzarlo explícitamente), y se desvían posteriormente en dirección al punto final.

Por tanto, para que un mismo valor sucediera 2 veces (mismo resultado para 2 posiciones de t, o lo que es lo mismo 2 posibles soluciones de t en la inversa de la función), el recorrido debería volver sobre valores anteriores, esto es, cambiar su sentido de avance.

Esto ocurre únicamente cuando la tangente inicial (dirección de partida) tiene distinto signo a la tangente final (dirección de llegada).
Por tanto, se puede determinar y restringir la situación a curvas monótonas.
Es fácilmente comprobable por medio del signo de ambas tangentes, calculadas como la diferencia entre el punto de control y el de inicio/final respectivamente.

Restricción 1.

$$(tan_{inicial} \cdot tan_{final}) >= 0 \quad : \quad (P_{ctrl} - P_{ini}) \cdot (P_{fin} - P_{ctrl}) >= 0$$

Situación sin solución.

En una curva sin restricciones, el rango de valores queda acotado entre el max. y min. de los puntos inicial, final y control.

El rango natural sobre el que realizar los cálculos podría ser este, sin embargo, puesto que la curva no pasa explícitamente por el punto de control, se obtendrían situaciones (posibles valores de X) sobre los que no existe una solución para t.
Tratar de calcular la inversa en estos puntos carecería de sentido y utilidad.
Incluso el tener que detectar cada situación tendría un coste, sin obtener con ello un resultado aprovechable.

Se debe acotar mejor el rango a recorrer, o bien, restringir de nuevo la curva de entrada de forma que se eviten estas situaciones.

Situación con Infinitas soluciones.

Este tipo de situación se da cuando la curva queda colapsada completamente en una coordenada única.

Por ejemplo, si la curva resulta ser una recta (aun expresada como curva cuadrática), y se encuentra en la misma vertical correspondiente al punto X de búsqueda. En estos casos, restringir la curva carecería de sentido, puesto que no se atenderían situaciones que deberían (en principio) ser un problema mas simple (rectas).

Se Puede optar por dos tratamientos:

- Modificar el comportamiento del algoritmo para poder tratar con aristas tanto curvas como rectas en un mismo problema, aplicando dos métodos distintos.
- Modificar el problema, para poder seguir tratándolo como aristas curvas en todo momento.
  Una modificación simple seria desplazar levemente (un delta mínimo) aquel parámetro que colapse la forma propia de curva, siempre atendiendo a que esta modificación no sea perceptible en el resultado final.

Análisis de la curva, en base a sus parámetros.

En base a la posición del punto de control respecto a los puntos de inicio y final, se observan 3 casos de estudio:

Punto de control situado fuera del rango [ini,fin].

$$P_{ctrl} < P_{ini} \quad o \quad P_{ctrl} > P_{fin}$$

Para este análisis se toman $P_{ctrl} < P_{ini}$, siendo análogo/simétrico el caso con $P_{ctrl} > P_{fin}$

En este caso, la tangente inicial y final tienen signo contrario, por lo que se produce un cambio de dirección de esta.
Dado que la progresión de la tangente es lineal desde $t_0 \rightarrow tan_{inicial}$ hasta $t_1 \rightarrow tan_{final}$, el cambio de dirección ($tan = 0$) se producirá en $t = 0{,}5$

Por tanto, el punto de inflexión de la curva será $X_{infl} = f(t = 0{,}5)$

Se destacan 4 fragmentos de la curva:

a) Desde X.ini hasta X.infl : correspondiente a valores de t en el rango $[0, 0{,}5]$

b,c) Desde X_{infl} hasta X_{fin} : correspondiente a valores de t en el rango $[0{,}5, 1]$ pudiendo dividirse en:

- **b)** rango $[X_{infl}, X_{ini}]$, con valores repetidos con a).
- **c)** rango $[X_{ini}, X_{fin}]$, con nuevos valores no repetidos.

d) desde X_{ctrl} hasta X_{infl} : correspondiente a valores de X fuera de la curva (y por tanto sin solución en la función inversa)

Resumiendo: de extremo izquiero al derecho

rango d) $[X_{ctrl}, X_{infl}[$ = sin solución.
(no existen puntos en este rango que pertenezcan a la curva)

rango a,b) $[X_{infl}, X_{ini}]$ = solución doble.
(puntos que pertenece doblemente a la curva, dos valores de t responden a estos valores de X)

rango c) $]X_{ini}, X_{fin}]$ = solución simple.
(puntos que pertenecen a la curva, con un único valor de t correspondiente a cada valor de X)

Punto de control situado dentro del rango $[ini, fin]$.

$$P_{ini} <= P_{ctrl} <= P_{fin}$$

Las tangentes inicial y final tienen mismo signo, por lo que no existe cambio de dirección, ni punto de inflexión correspondiente.
El valor de X es monótono (creciente/decreciente), y por tanto la inversa solo tendrá una única solución.
Así mismo, ambos extremos, por coincidir con los puntos inicial y final, también tendrán solución definida.

En este caso, se puede tomar el rango $[X_{ini}, X_{fin}]$ como tramo a recorrer.

Aun en el caso en que el punto de control esté situado sobre uno de los extremos ($P_{ctrl} = P_{ini} \quad o \quad P_{ctrl} = P_{fin}$), se tendría una tangente inicial o final nula, pero que avanzaría hacia una tangente no nula de forma lineal.
Igualmente, X sería monótona, y la inversa tendría una solución única, y quedaría definida en todo el rango.

Esta situación es la mas adecuada para nuestro problema, y se puede garantizar aplicando la:

Restricción 2.

$$P_{ini} <= P_{ctrl} <= P_{fin}$$

Todos los puntos colapsados sobre un mismo valor.

$$X = P_{ini} = P_{ctrl} = P_{fin}$$

En esta situación se no se podría determinar un valor de t concreto, ya que todos serian validos, presentándose infinitas soluciones.
La forma de acotar este caso, seria refinando la Restricción 2 como:

Restricción 3.

$$P_{ini} <= P_{ctrl} <= P_{fin} \quad : \quad P_{ini} \neq P_{fin}$$

Se observar que, la Restricción 3 satisface al mismo tiempo la Restricción 1, por lo que es la única necesaria a observar.
En realidad, puesto que con la Función interpoladora cuadrática inversa solo se pretenden obtener valores t para posiciones concretas de Y, solo es necesario comprobar que la función es Ymonótona, restringiendo solo en este eje (ignorando la forma de la arista en el eje X)

Durante el recorrido de la arista de raster (horizontal), se aplica la misma restricción, pero sobre el eje X, ignorando la forma en Y (que particularmente será constante).

Para la generalización del método, será necesaria una primera fase de preproceso, capaz de subdividir la superficie de entrada, en otras que cumplan dicha restricción. En los test realizados, todos los triángulos hacen uso exclusivo de aristas que cumplen esta restricción.

4.2.3. Calculo de la Función Interpoladora Cuadrática Inversa.

Calculo directo.

Aun tras acotar los rangos del problema, para asegurar soluciones existentes y únicas sobre el punto a calcular, esta función sigue una distribución de función raíz, y hallar el valor exacto es difícilmente calculable.

Existen métodos para ello [Dellajustina14] [Banerjee15] [Kwon08], etc. pero su calculo (aun siendo aproximado) requiere un gran esfuerzo computacional.
También hacer constar que estos métodos están orientados al calculo de un valor concreto, y ante sucesivos cálculos (para recorrer valores contiguos de la función) no se aprovecha el esfuerzo anterior realizado.

Es por esto que se descartó el uso de un calculo directo, en pro de variantes aproximadas, pero que puedan aportar soluciones mas eficientes, y con reaprovechamiento de esfuerzo anterior.

Calculo por aproximación polinomial.

Se presenta como alternativa la de realizar un calculo por una función de aproximación, sobre la cual se pueda hacer un recorrido aplicando técnicas iterativas/incrementales.

Inicialmente se definió g (función interpoladora cuadrática inversa) para un rango $[0, 1]$, con la intención de ser recorrida a intervalos constantes.
Puesto que esto es análogo al recorrido de una curva cuadrática (función $f(t)$) en un mismo rango $[0, 1]$ y a intervalos también constantes, se puede aplicar a nuestro problema el método de diferencias avanzadas, siempre y cuando nuestra función g aproximadora tenga una forma similar, es decir un polinomio.

En general, el método de diferencias avanzadas puede ser aplicado a polinomios de cualquier grado, por lo que nuestra función aproximadora también puede serlo.

Claramente, no es posible obtener una buena aproximación de una función raíz, por medio de un polinomio cuadrático. Por este motivo, se realizaron varios tests, aplicando diferentes configuraciones en los parámetros y grados para tratar de ajustarse lo más posible a la función real.

En todo caso, este polinomio vendrá siempre determinado por los propios parámetros de la curva ($Pto_{inicial}$, Pto_{ctrl} y Pto_{final}), siendo los que variarán en cada aplicación.

Búsqueda del polinomio a emplear.

Tomando la expresión paramétrica de la curva, en base a sus extremos y punto de control, hay que centrarse en 3 instantes destacables:

$f(0) = X_1$ $\leftarrow$ siendo el punto inicial.

$f(0{,}5) = X_2$ $\leftarrow$ siendo un punto por el que pasa la función.
(define la intensidad de su curvatura)

$f(1) = X_3$ $\leftarrow$ siendo el punto final.

Se define la inversa como:

$g(X_1) = 0$ $\leftarrow$ parámetro 0 en el punto inicial.

$g(X_2) = 0{,}5$ $\leftarrow$ parámetro 0.5 en el punto central.

$g(X_3) = 1$ $\leftarrow$ parámetro 1 en el punto final.

Se estan sustituyendo los valores por los parámetros $t \leftrightarrow X$, esto es:

$$X = f(t) \quad \text{pasa a ser} \quad t = g(X)$$

Observando las restricciones descritas anteriormente, el rango queda definido en $[0, 1]$ y se obtiene una solución única para cada valor de $t = [0, 1]$.

Por tanto, estos 3 puntos están definiendo una curva de igual grado, que podría considerarse como aproximación a la inversa de la función/curva original.
Sin embargo, esta función sigue una distribución de grado 2, y no función raíz como correspondería a su inversa; por lo que solo se obtienen valores "correctos" en los puntos descritos.

En la práctica, esta curva ha sido definida por medio de un polinomio de Lagrange, siendo aplicable un recorrido por el método de diferencias avanzadas.

Entre los experimentos realizados en busca de una curva (polinómica) que obtenga una mejor aproximación a la función inversa, se optó por probar:

- Por una parte, incrementar el grado de la función, calculándose nuevos valores sobre la función original, para utilizarlos como puntos de control en la aproximación.

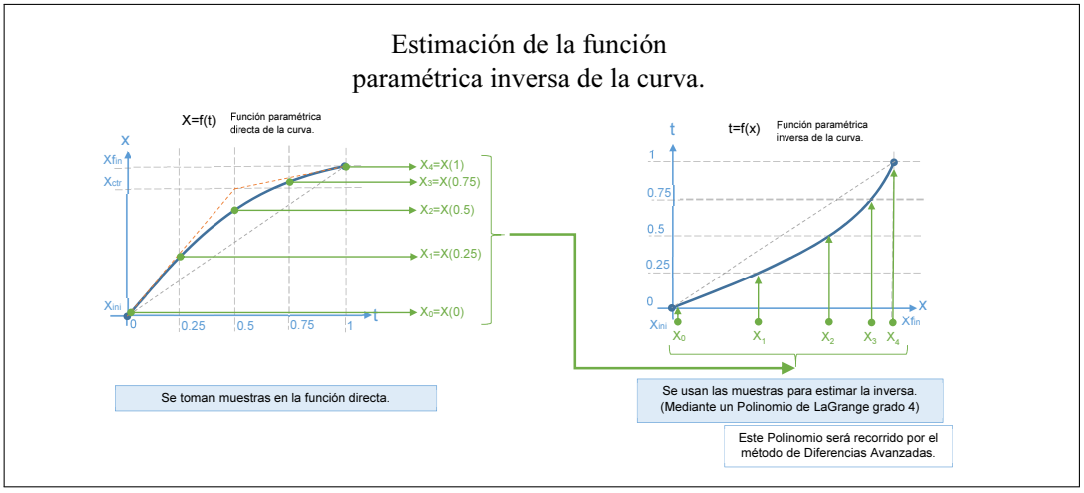
  Ejemplo. *para* $t = 0, \quad t = 0{,}25, \quad t = 0{,}5, \quad t = 0{,}75, \quad t = 1$

- Se varió la distribución de estos valores, de forma que trataran de forzar mas la aproximación en los rangos con mayor curvatura.

  Ejemplo. *para* $t = 0, \quad t = 0{,}1, \quad t = 0{,}25, \quad t = 0{,}5, \quad t = 1$

- Incluso se tomaron valores fuera del rango $[0, 1]$, basándose en la simetría sobre el eje $X = 0$ de una función raíz.

  Ejemplo. *para* $t = -0{,}5 \quad$ *con valor* $\quad -f(0{,}5)$



Estimación de la función inversa de la curva.

Tal como es conocido para los polinomios de Lagrange [Cordero02], a mayor grado la función se vuelve mas inestable en los extremos. Es por esto que, por el momento, se ha optado por un compromiso en cuanto a su nivel de aproximación, y estabilidad.

Finalmente, para una primera implementación, se utiliza como función inversa la aproximación polinomial de Lagrange de grado 4, basada en muestras para valores fijos del parámetro t en:

$$t = 0, \quad t = 0{,}25, \quad t = 0{,}5, \quad t = 0{,}75, \quad t = 1$$

A futuro se barajan implementar otras aproximaciones para obtener mejores resultados. Posiblemente, para una curvatura positiva máxima seria mas conveniente las muestras en $t = (0, 0{,}1, 0{,}3, 0{,}6, 1)$, mientras que en curvaturas negativas la tendencia sería al contrario con $t = (0, 0{,}4, 0{,}7, 0{,}9, 1)$; por lo que las muestras a tomar deberían variar en función de la intensidad y signo de curvatura del caso a tratar.

4.3. Implementación

Para poner en practica lo descrito hasta el momento, se ha realizado un desarrollo básico del método. En primer lugar, describiré las funciones y estructuras de datos implementadas, continuando después con la forma de operar con la aplicación de laboratorio.

4.3.1. Funciones implementadas.

Estructuras de datos

En general se utilizan unas estructuras de datos (y no modelo de objetos), para emplear un modelo de memoria estática, conociéndose de antemano su tamaño.
Esto debería hacer mas fácilmente la incorporación al modelo de shaders ya existente.

Vértice.

Los vértices, en estos tests, solo incluyen las componentes x,y. Siendo del tipo son float 32 de bits.
En un caso mas general podrán incluir mas parámetros (z, normal, uv, color, etc), y análogamente al modelo de shaders ya existente, podrían ser recibidos como un array de datos, estableciéndose el orden en el cual se encuentra cada uno.

Arista curva.

Estructura con 3 vértices: inicio, medio/control y final.
Adicionalmente (para ayudar en la eficiencia de calculo), se tendrán los coeficientes del polinomio de Bézier, de forma que puedan utilizarse para calcular puntos concretos sobre la curva.

Superficie cuadrática.

Almacena la información de las 3 aristas curvas que definen su contorno.
Adicionalmente, se incluye la arista curva central (cuerda), la cual debe ser valida en todo el rango Y de la superficie.

Interpolador Cuadrático Inverso. (en Y)

Estructura con los datos necesarios para iterar sucesivamente por una función cuadrática.
Se tendrá el último valor útil, y los coeficientes de primer a cuarto orden, para realizar un recorrido por el método de diferencias avanzadas.

El valor t transcurrirá desde 0 hasta 1, con incrementos no constantes, pero que corresponderán a posiciones de Y a incrementos constantes. A partir de este valor t se pueden estimar el resto de componentes (X, Z y otros)
El valor Y transcurrirá desde su valor inicial, con incrementos constantes (de tamaño unitario), hasta alcanzar su valor final.
Ajustando el incremento (haciendolo menor a la unidad) se pueden obtener muestras más próximas para la aplicación de técnicas de "antialiasing".

Los valores para X transcurrirán desde su valores inicial, con incrementos no necesariamente constantes, hasta alcanzar su valor final. Estos son estimados a partir del valor t, y los datos generales de la arista recorrida, aplicando la formula de Polinomio de Bézier.
Un valor N indicará la cantidad de pasos o iteraciones pendientes de realizar.
Con este valor se podrá determinar (sin errores de precisión) cuando el interpolador ha concluido el recorrido por la arista.

Interpolador Cuadrático Inverso. (en X)

Se utiliza la misma estructura que el "Interpolador Cuadrático Inverso (en Y)".
Únicamente se diferencia en la función que inicializa la estructura, haciendo que los incrementos constantes se produzcan sobre la componente X, para un recorrido en horizontal, quedando el valor para Y estimado a partir del parámetro t, como con el resto de componentes.

Funciones.

Las funciones se basarán en la manipulación de las estructuras de datos antes mencionadas.

Funciones de Interpolación Cuadrática Inversa.

`interp_init_y`

Entrada: Una arista curva.

Salida: Un interpolador cuadrático inverso (en Y) inicializado.

En base a los valores de entrada, se estimarán los coeficientes de la función de aproximación inversa. (para la curva en Y)
En base a la longitud (en Y), se determinará el numero de pasos a realizar y a partir de estos, los valores iniciales e incrementos para un recorrido por el método de diferencias avanzadas.

`interp_init_x`

Entrada: Una arista curva.

Salida: Un interpolador cuadrático inverso (en X) inicializado.

En base a los valores de entrada, se estimarán los coeficientes de la función de aproximación inversa. (para la curva en X)
En base a la longitud (en X), se determinará el numero de pasos a realizar y a partir de estos, los valores iniciales e incrementos para un recorrido por el método de diferencias avanzadas.

`interp_step`

Entrada: Interpolador cuadrático inverso (será modificado)

Realizará un paso en el recorrido por el método de diferencias avanzadas.
Con esto quedará actualizado el valor de t.
Con esto quedará decrementado el valor N de pasos restantes.

`interp_calc`

Entrada: Interpolador cuadrático inverso.

Entrada: Arista curva.

Salida: Vértice.

Se calcularán todos los componentes de un vértice, en base a la arista curva y el valor t actual del interpolador cuadrático inverso.
Se utilizará (para cada componente) la función auxiliar `curve_calc`. Por tanto, todos los componentes son estimados por calculo directo.
En futuras implementaciones, se propone ampliar la estructura del interpolador cuadrático inverso, de forma que también aproxime por una función inversa iterable por diferencias avanzadas, cada uno de los componentes.
Esto requerirá algo más de espacio en memoria, cálculos adicionales en la inicialización del interpolador, y lo haría dependiente a la cantidad de componentes de entrada. Sin embargo, reduciría notablemente los cálculos de las componentes a cada iteración.
Al presentarse en el marco de una fase de raster programable, la decisión final queda en manos del usuario.

`interp_end`

Entrada: Interpolador cuadrático inverso

Salida: Boleano indicando si la interpolación ha llegado a su fin.

Básicamente comprueba si el valor N del interpolador es 0.
En cualquier caso, la salida es falsa, indicando que aun no se ha alcanzado el final, y se puede seguir utilizando el interpolador en nuevas iteraciones.

Funciones de Superficie

`surface_init`

Entrada: Vertices A, B y C

Entrada: Vertices control AB, BC, CA

Salida: Superficie cuadrática

Dados los vértices de esquina y control de entrada, quedarán definidas las 3 arista de contorno.
Se definirán de forma que las arista A y B sean las que comiencen mas arriba (Y_min), siendo la tercera arista C, la que sustituya a la mas corta de las anteriores durante el rasterizado, hasta alcanzar el punto mas bajo (Y_max).
Por medio de la función auxiliar `surface_central_point` se calculará el punto centro sobre la superficie.
Empleando el vértice mas alto (Y_min), el vértice mas bajo (Y_max) y este punto central, y por medio de la función auxiliar `curve_trought_points` se calculará la arista curva central.
Con esta información se tiene preparada la superficie para ser rasterizada.

Funciones auxiliares

`curve_calc_point`

Entrada: Coeficientes polinomio Bézier: A,B,C

Entrada: Valor t

Salida: Valor interpolado sobre la curva, en t

En base al valor t se estimará por calculo directo, el valor interpolado sobre la curva cuadrática con los coeficientes de entrada: A,B y C.

`surface_central_point`

Entrada: Superficie cuadrática (ignorando cuerda central)

Salida: Vertice

Tomando como definición de superficie los vértices y puntos de control, se calcula un punto sobre la superficie centrado ($s = t = 0{,}5$).
A futuro se calcularían todos los componentes del vértice (por el momento, solo X,Y,Z).

`curve_trought_points`

Entrada: Arista curva (con punto de control 'falso', sobre la curva)

Salida: Arista curva (con punto de control 'verdadero', definiendo curva Bézier)

Se reciben los vértices de inicio y fin de la curva, y un punto sobre la misma (central $t = 0{,}5$).
Este punto sobre se recibe como vertice de control, aunque realmente no lo es.
En base a estos 3 puntos sobre la curva, se calcula el punto de control equivalente.
Se obtiene, por tanto, el punto de control real que, junto con los puntos inicio y fin, determinan una curva cuadrática que pasa por los 3 puntos iniciales de entrada.
Esta función será empleada para calcular la curva de rastreo (horizontal), que será recorrida.
También será empleada para calcular una curva central sobre la superficie, a partir de un punto sobre la superficie.

Otras funciones análogas.

A fin de poder realizar múltiples pruebas de forma automatizada y comparar los resultados contra métodos lineales mas habituales, se han generado las siguientes funciones.

`interp_lineal_init_y`

Entrada: Una arista curva. (se ignora el punto de control)

Salida: Un interpolador lineal inverso (en Y) inicializado.

En base a los valores de entrada, se estimarán coeficientes lineales, valor inicial e incremento constante para un recorrido por el método de diferencias avanzadas.
En base a la longitud (en Y), se determinará el numero de pasos a realizar.
En este caso (al ser lineal), se tendrá un único coeficiente de incrementos.

`interp_lineal_init_x`

Análogo a la función `interp_lineal_init_y`, pero en base a la longitud en X.

`render_lineal`

Entrada: 3 vertices y 3 vertices de control (los vertices de control son ignorados)

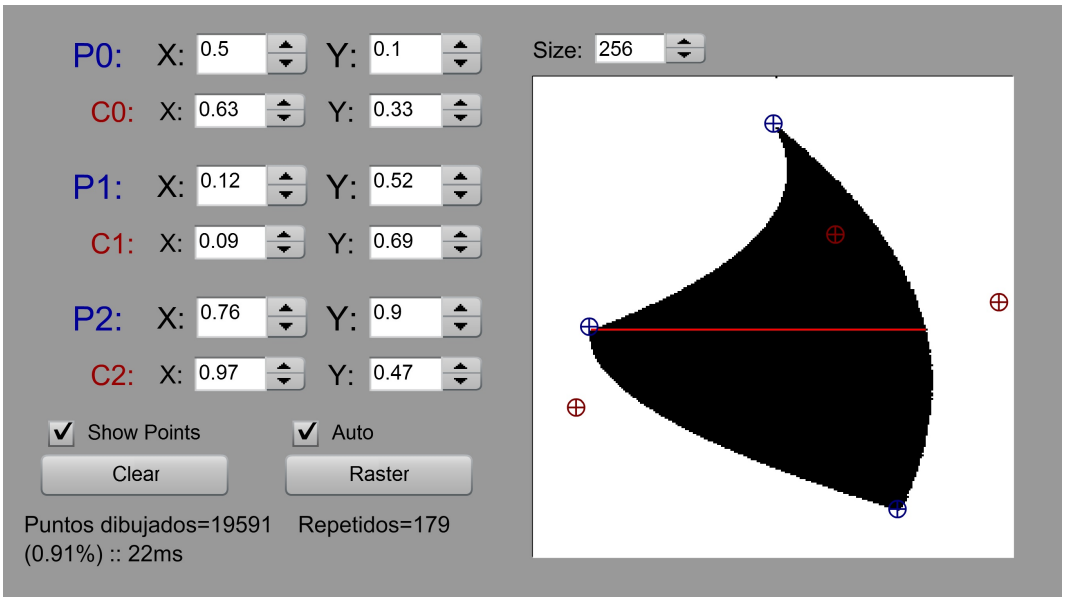
Análoga a la función `render`, pero en su lugar empleará interpoladores inversos lineales, de forma que se obtendrá una representación plana de la superficie definida por los 3 vértices.
Al admitir los mismo parámetros de entrada, se podrá comparar un mismo caso con su versión cuadrática.

4.3.2. Aplicación de Laboratorio.

Este conjunto de funciones a sido integrado dentro de una aplicación (Evaluador) que las controla y hace uso de ellas.

Pueden ver una versión de Pseudocódigo en capitulo "Propuesta de Método".

Una primera implementación, para facilitar la validación del método, ha sido realizada por medio de la tecnología de Adobe Flash y su lenguaje AS3.
Esta vía generó una aplicación interactiva, con la que se puede fácilmente alterar los valores de entrada al problema, así como ciertos factores de entorno, como el tamaño del canvas de trabajo.
Si bien la velocidad de ejecución no es su principal característica, permite comprobar si la programación introducida sigue correctamente las pautas del método descrito.
Se obtiene así mismo una salida visual de su ejecución, pudiendo percibir la forma en que afectan al resultado los errores por aproximación de la inversa, e incluso errores por precisión de los datos manejados.



Aplicación gráfica interactiva. (Flash/AS3)

Para la implementación final, el lenguaje utilizado ha sido C++, tratando en todo momento de no hacer uso de librerías externas, de forma que el núcleo del método quede lo mas limpio e independiente posible.

Esta implementación (por defecto) no genera ninguna imagen de salida, y es utilizada únicamente para el propósito de medir el consumo de recursos, principalmente tiempo.
Se controla mediante paso de parámetros por linea de comandos, por medio de los cuales se pueden establecer diferentes configuraciones con las que operar.
Se controlan las dimensiones del problema, parámetros de entrada y precisión requerida en las mediciones de tiempo.

En caso de querer obtener una imagen de salida (especificando parámetro -o), esta es guardada en formato `.pgm` en escala de gris.

Opciones parametrizables

Concretamente se dispone de las siguiente opciones:

Espacio de trabajo (-c canvas):

Establece el área o extensión sobre el que se representa el triangulo a evaluar. Hace referencia al lado de la imagen cuadrada sobre la que dibujar la figura. A mayor tamaño, mayor nivel de detalle se podrá observar.
Las coordenadas de entrada (expresadas en forma canónica) se escalarán a este espacio antes de comenzar la evaluación del caso.

Repeticiones (-r repeat):

Puesto que el proceso se estima se complete en un margen de tiempo muy pequeño, las mediciones de su duración pueden resultar imprecisas.
Para ampliar esta precisión, se repite el mismo caso `repeat` veces, midiendo el tiempo total, y posteriormente obteniendo la duración unitaria del caso.

La aplicación del tamaño del canvas (escalado desde coordeandas canónicas) se realiza antes de comenzar la medición, para que no sea contabilizado como tiempo de calculo. La preparación de los interpoladores si se contempla como parte del coste del proceso.

Método a emplear (-m método):

Por medio de este parámetro se podrá emplear una u otra implementación, para comparar resultados.
Los valores aceptados serán: lineal y quad.

lineal Método por interpolador lineal clásico, con subdivisión jerárquica.
Se observará el parámetro (-l `nivel`), dibujando el triangulo final de forma lineal.

quad Método por interpoladores cuadráticos inversos.
(método objetivo del presente trabajo)

Parámetro nivel (-l nivel):

Indica el nivel de subdivisión jerárquica que se aplicará al triangulo inicial.
A cada nivel se generarán 4 triángulos, calculando puntos sobre las curvas y nuevos puntos de control para continuar con siguientes niveles de subdivisión.
Por tanto, la cantidad de triángulos finales a dibujar será de 4^{nivel}.

Caso a evaluar (input):

Se indican las coordenadas (X,Y) de cada vértice y punto de control.
Se esperan que el orden de estos puntos sea el adecuado, concretamente comenzando por el vértice superior, vértice final de la arista izquierda, vértice final de la arista derecha; el punto de control a la arista izquierda, punto de control de la arista inferior, y punto de control de la arista derecha.

Resultado (output):

Se generará una línea de salida, resumiendo los parámetros e indicando la medición de tiempo obtenida:

output: `lineal, 0, 256, 100, 13778, 73.7152, 0.737152`

Correspondiendose con la información de:
método, nivel de subdivisión, canvas, num.repeticiones, puntos calculados,
tiempo total y tiempo individual (en milisegundos).

Aplicación de proceso en lote.

Como apoyo, se ha generado una aplicación para lanzar casos y configuraciones de forma automatizada. Es útil para probar diferentes configuraciones, y al mismo tiempo poder repetir el mismo experimento con otros parámetros.

Esta aplicación toma una serie de "casos" desde un archivo de texto con coordenadas previamente generadas. (ver a continuación) Permite comparar la variación de comportamiento respecto a parámetros como el tamaño de canvas o el método empleado.

Procesa todos los casos sobre una misma configuración, obteniendo los tiempos individuales de estos, y calculando finalmente el tiempo medio, máximo y mínimo. Genera una línea de salida con los resultados, junto con la combinación de parámetros empleados. (tamaño de canvas, método, repeticiones, tiempos y numero de casos evaluados.

comando:

```
batch -c 100 -m lineal -r 100 casos.txt
```

Utilidad de generación de casos.

Esta utilidad se emplea para preparar un conjunto de datos de entrada (coordenadas) aleatorio, para ser evaluadas posteriormente.

parámetros: `numero de muestras a generar.`

Cada linea representa un caso, generándose las coordenadas de 3 vértices y 3 puntos de control. Las coordenadas están expresadas en forma canónica [0..1], para ser independientes del canvas utilizado en su evaluación.

En todo momento, se generan puntos de control que observen las restricciones del problema: Vértices de control dentro de la caja de contorno de los extremos de las aristas, para limitar la curvatura máxima.

Capítulo 5

Pruebas y Resultados

5.1. Pruebas Realizadas

Con las herramientas descritas en el capitulo anterior, se ha procedido a realizar una serie de experimentos, variando parámetros tales como el tamaño de canvas, método a emplear y coordenadas de entrada.

Los diferentes métodos llegan a calcular los valores de coordenada X e Y, tal como quedarían dispuestas para enviarlas a la siguiente fase de render (Fragment Shader), pero sin llegar a realizar ninguna operación de dibujado propiamente.

No se ha realizado ninguna medición sobre el requerimiento de memoria, ya que, todos los métodos emplean estructuras de datos estáticas, no aumentando en ningún momento, siendo independientes a la situación o tamaño del canvas.

La ejecución es lineal, sin aplicar ningún tipo de optimización ni paralelización de tareas. Esto sitúa a ambos métodos en un marco similar de recursos disponibles.

Se han preparado dos bloques de muestras a analizar, compuestas por 100 conjuntos de coordenadas. Estas fueron obtenidas a través de la utilidad de generación de casos expuesta en el capitulo anterior, la cual provee un conjunto de coordenadas aleatorias, pero atendiendo a las restricciones iniciales descritas.

Así se presentan todo tipo de casos, desde polígonos con una mayor superficie, otros mas "estilizados" y combinando situaciones con aristas de mayor o menor nivel de curvatura.

Para cada conjunto de cada bloque, se lanzó el algoritmo variando el tamaño de canvas en los valores: 256, 512, 1024, 2048 y 4096.
Por tanto, los resultados son la media de 1000 casos por algoritmo, con un total de 12000 experimentos.
Esto pretende exponer los casos a situaciones de estrés, al requerir la generación de un gran número de puntos de salida.

En las mismas condiciones se han ejecutado las muestras para los algoritmos:

interpolación cuadrática: método propuesto.

lineal: empleando únicamente los vértices del patch.

subdivisión recursiva: generando nuevos puntos sobre la curva, y lanzando 4 triángulos independientes (que a su vez podían volver a ser subdivididos recursivamente).
Se han empleado desde 1 a 10 subdivisiones.
(Llegándose a generar más de 1 millos de triángulos)

Para obtener una mejor estimación del tiempo empleado, cada ejecución constó de 100 repeticiones del mismo caso.
El equipo empleado contaba con un procesador Intel Core i7-4960HQ 2.6GHz corriendo sobre entorno Windows 8.1 Pro.

En las mediciones solo se tiene en cuenta el tiempo empleado para el método en sí, sin incluir la lectura y escalado de las coordenadas de entrada al tamaño de canvas indicado en el experimento.

Al comparar el método de subdivisión, se tiene en cuenta la generación de las nuevas coordenadas, puesto que serian unos cálculos iniciales no necesarios para el método propuesto. Comúnmente esta subdivisión se realiza en fases previas del pipeline, o de forma paralela.

En general, se emplea como marco de referencia los resultados del algoritmo de subdivisión aplicando un nivel 10.
Esta situación genera una cantidad enorme de triángulos a dibujar, llegando a representar cada uno un único píxel, tomandose como un método que genera la imagen de mayor precisión, siendo por contra la menos eficiente.
En la situación extrema, en un canvas de 256x256 solo existen un máximo de 65.536 píxeles a cubrir, por lo que el generar más de un millón de triángulos sobrepasa claramente la resolución de salida.

5.2. Resultados Obtenidos

5.2.1. Tiempos

De las mediciones obtenidas se muestran a continuación los tiempos medio de las ejecuciones dentro de una misma configuración.

Los tiempos hacen referencia a milisegundos, con una precisión de al menos milésimas, para una única ejecución del método.
(precisión de 2533207 ticks/sec = 0.000394757ms evaluando sobre 100 casos)
Están señalados los casos en que la subdivisión recursiva tiene un coste mayor que empleando el método propuesto.

método	256	512	1024	2048	4096
lineal	0.2982	1.1310	4.5696	17.9516	73.0835
1 sdiv	0.3249	1.2455	4.8760	18.4243	73.1023
2 sdiv	0.4207	1.3725	5.1482	19.3865	75.1276
3 sdiv	0.5869	1.6716	5.7567	20.4531	76.5365
4 sdiv	1.0789	2.4621	7.1241	23.1178	81.8947
5 sdiv	2.2735	4.4088	10.4634	29.4171	93.0096
6 sdiv	5.6611	8.9390	16.9754	40.1351	116.3750
7 sdiv	17.1670	22.1599	35.3551	69.2826	159.6840
8 sdiv	49.9626	58.1822	76.6251	121.8010	267.4650
9 sdiv	136.7130	136.4400	158.5980	211.2440	457.3470
10 sdiv	541.7340	546.7540	548.9780	637.6560	848.4090
quad	0.4451	1.6584	6.3097	24.6395	96.9871

Tiempos de Ejecución. (en ms.)

Podemos observar que, a pesar de realizar cálculos mas complejos, el método propuesto aún puede aventajar al de subdivisión a ciertos niveles.
En general los tiempos de ejecución han sido menores que a nivel de subdivisión 3 y 6, para resoluciones bajas y altas respectivamente.

Tomando como referencia subdivisión a nivel 10 (con mas de 1 millón de triángulos), el método propuesto en altas resoluciones representa poco mas del 10 %, y en bajas resoluciones a penas un 0.1 %, debido en gran parte (y como se analizará mas tarde) al ajuste de los cálculos en base a la resolución de salida, frente a la gran redundancia de cálculos del método de referencia.

5.2.2. Similitud de Imagen.

Así mismo, se realizó una comparación entre imágenes generadas por el método propuesto y cada caso de nivel de subdivisión, con la intención de poder establecer un nivel de similitud entre ellas.

En general, ante escenarios de baja resolución (canvas 256 o 512), bajos niveles de subdivisión ya consiguen un resultado visual aceptable y altamente similar al generado por el método propuesto.
Sin embargo, en escenarios de mayor resolución (canvas 2048 o 4096), es preciso aumentar el nivel de subdivisión a emplear.

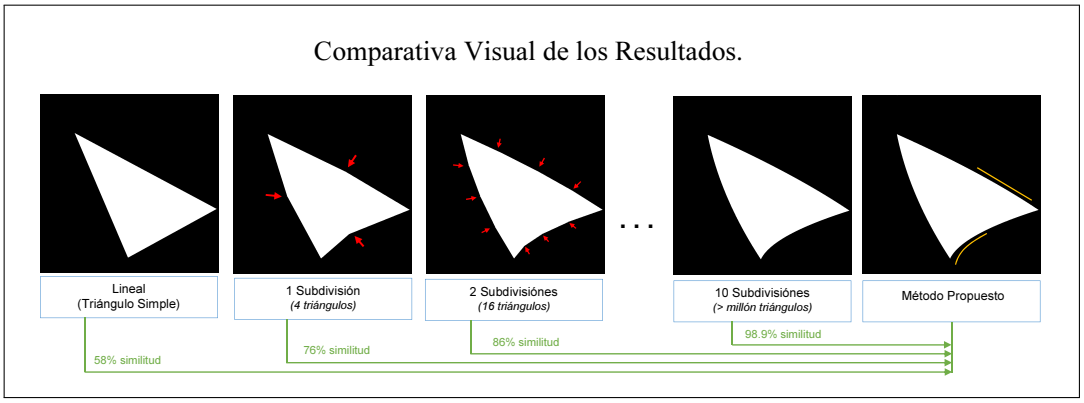


Imagen obtenida aumentando el nivel de subdivisión.

En la siguiente tabla se muestra el porcentaje de similitud entre las imágenes generadas por los diferentes métodos (propuesto y subdivisión a varios niveles), respecto a la imagen obtenida por subdivisión a nivel 10, que se toma como referencia de mejor representación de la figura.

método	256	512	1024	2048	4096
lineal	57.9410 %	57.5333 %	57.3262 %	56.1581 %	54.6012 %
1 sdiv	57.9410 %	57.5333 %	57.3262 %	56.1581 %	54.6012 %
2 sdiv	77.7760 %	79.9828 %	81.6062 %	79.8486 %	77.9490 %
3 sdiv	78.7628 %	80.7614 %	82.4000 %	81.0587 %	79.3740 %
4 sdiv	81.3701 %	81.6516 %	82.4827 %	81.1648 %	79.5540 %
5 sdiv	87.1318 %	83.7704 %	83.4063 %	81.1573 %	79.5511 %
6 sdiv	91.7213 %	89.3844 %	84.9050 %	82.2365 %	79.5197 %
7 sdiv	94.7098 %	93.1466 %	90.5781 %	83.3722 %	80.6288 %
8 sdiv	98.4003 %	96.3603 %	94.1591 %	89.0097 %	81.6454 %
9 sdiv	99.5177 %	99.4089 %	97.2609 %	93.6945 %	87.3687 %
quad	81.0416 %	85.0662 %	87.3257 %	84.1072 %	83.0878 %

Cuadro 5.1: % Similitud de imagenes respecto a subdivisión 10.

En general, la similitud es mayor conforme se aumenta el nivel de subdivisión. Cuanto mayor es la resolución de la imagen, los errores representan un porcentaje menor de diferencia, pudiéndose medir con mas precisión.

Hacer constar que debido al uso de una ecuación inversa por aproximación (ver apartado 4.2.3), la curva obtenida no es completamente correcta, por lo que también presenta diferencias de forma respecto a las imágenes de referencia.

Se observa que el método de triangulo simple presenta un grado de similitud bajo (entre el 54 % y el 58 %), dejando claro que no es una buena aproximación a una superficie curva.

Para alcanzar representaciones curvas mas ajustadas es necesario emplear niveles de subdivisión entre 5 y 6 en resoluciones bajas (con similitudes en torno al 87 % y 89 %), llegando a ser necesarios niveles por encima de 7 en resoluciones altas para alcanzar similitudes que superan el 90 %.

5.2.3. Redundancia en los cálculos.

Uno de los principales objetivos al realizar la propuesta del método, era el de poder ajustar los cálculos realizados a la resolución de salida.
De esta forma se evitan fases previas para ajustar el proceso, y se asegura que en cualquier situación, la suavidad de la curva no se viera interrumpida, y que no se realizase proceso por encima de las necesidades.

A este respecto, se han obtenido medidas de la cantidad de puntos finales calculados (y que posteriormente se convertirán en instancias de proceso para la fase de Fragment Shader).

Se muestra a continuación los valores medios de generación de puntos de los experimentos realizados:

método	256	512	1024	2048	4096
lineal	8350	33268	132707	529989	2118999
1 sdiv	8536	33734	134808	538030	2151202
2 sdiv	8672	33916	135301	540235	2159083
3 sdiv	8820	34076	135732	541187	2162640
4 sdiv	8943	34178	136355	542625	2164614
5 sdiv	9616	35729	136647	545030	2170143
6 sdiv	13542	38428	142907	546285	2179636
7 sdiv	26008	54139	153711	571301	2184439
8 sdiv	65536	104032	216544	614723	2285807
9 sdiv	262144	262144	416128	866169	2458829
10 sdiv	1048576	1048576	1048576	1664511	3464605
quad	8514	33879	135332	540588	2161759

Cuadro 5.2: Volumen medio de puntos calculados.

El método de subdivisión consigue una mayor definición del contorno en base a dividir el problema hasta que este sea suficientemente pequeño como para no percibir discontinuidades.

Sin embargo, esto lleva a la generación de una cantidad enorme de subprocesos, que en muchos casos aportan muy poco a la solución final, puesto que a menudo coinciden entre sí.

En un caso extremo, en un entorno cuadrado de 256 por 256 píxeles, una figura con un nivel de 10 subdivisiones, llega a generar mas de 1 millón de triángulos.

Prácticamente la totalidad de estos afectarán únicamente a 1 píxel de la imagen final, pero teniendo en cuenta que (aún cuando la figura ocupara todo el área visible) no se disponen de más de 65536 píxeles, prácticamente cada triángulo generado estaría sobrescribiendo el mismo punto al menos 16 veces. (si no se cubre todo el área, la repetición sería aún mayor)

Aquí es donde reside la fuerza del método propuesto, al tratar de evitar (en la medida de lo posible, aún sin haber sido optimizado) la duplicidad en el calculo de puntos de salida.

método	256	512	1024	2048	4096
lineal	101.97 %	101.84 %	101.98 %	102.00 %	102.02 %
1 sdiv	99.75 %	100.43 %	100.39 %	100.48 %	100.49 %
2 sdiv	98.18 %	99.89 %	100.02 %	100.07 %	100.12 %
3 sdiv	96.54 %	99.42 %	99.71 %	99.89 %	99.96 %
4 sdiv	95.20 %	99.13 %	99.25 %	99.62 %	99.87 %
5 sdiv	88.54 %	94.82 %	99.04 %	99.18 %	99.61 %
6 sdiv	62.87 %	88.16 %	94.70 %	98.96 %	99.18 %
7 sdiv	32.74 %	62.58 %	88.04 %	94.62 %	98.96 %
8 sdiv	12.99 %	32.57 %	62.50 %	87.94 %	94.57 %
9 sdiv	3.25 %	12.92 %	32.52 %	62.41 %	87.92 %
10 sdiv	0.81 %	3.23 %	12.91 %	32.48 %	62.40 %

Cuadro 5.3: % de puntos cálculados por el método propuesto.

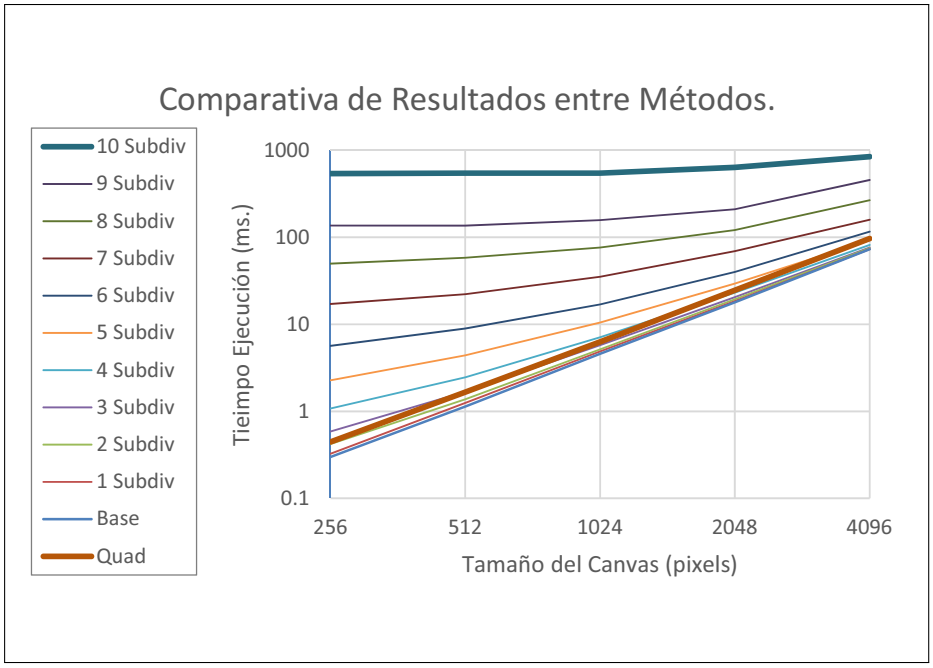
En esta tabla se muestra en porcentaje la proporción de puntos generados por el método propuesto, en contraposición a la subdivisión a varios niveles.
Podemos observar como la cantidad de puntos generados es muy similar respecto a los primeros niveles de subdivisión, y como a partir de un punto la diferencia es mas notable.

Esto se debe al alcanzar el momento en que las subdivisiones comienzan a generar puntos duplicados, y a realizar proceso no efectivo, mientras que el método propuesto se contiene en generar solo los puntos necesarios.

A resoluciones mayores este momento de redundancia se retrasa hasta niveles de subdivisión mas altos, pero aún así, el método propuesto genera al menos un 40 % menos de puntos (incluso mas de un 65 % menos en resoluciones medias).

5.3. Valoración

Como se puede observar en la siguiente gráfica, el método lineal siempre es mas rápido que el resto. Así mismo, este método es el que peor se aproxima a una línea o superficie curva. Para mejorar la apariencia de la figura a representar se debe emplear alguna otra técnica.



Comparativa en Tiempos de Ejecución.

Comparando los tiempos para niveles de subdivisión mayor, claramente se observa como estos se incrementan, pero son admitidos en pro de una mejor aproximación. El nivel de subdivisión debe ser aplicado en función del tamaño y curvatura a mostrar, pudiendo ser necesario generar niveles de subdivisión altos. Si bien esta aproximación es prácticamente indistinguible de una curvatura real, se genera un gran volumen de triángulos a representar, con tendencia a cubrir individualmente un área ínfima.

Con el método propuesto, esta aproximación se hace invariable al tamaño, generando en cualquier caso una aproximación continua a la curva.
Se observa que el tiempo de ejecución es similar (o sensiblemente inferior) a métodos con niveles de subdivisión medios.

De media, el método propuesto aventaja al uso de subdivisión con niveles 3 en adelante. En escenarios de alta resolución, al cubrir un área mayor, el tiempo aumenta, pero también cabe destacar en estos casos la necesidad de emplear mayores niveles de subdivisión para obtener representaciones equivalentes con las técnicas habituales.

El método propuesto es ventajoso comparado con subdivisión a niveles 5 y 6 (y superiores). Por tanto, cabe tener en consideración el método propuesto, sobre todo ante superficies que cubran grandes áreas, o cuando se requieran imágenes de alta resolución.

Las ventajas e inconvenientes de cada método pueden definirse en base a cuatro escenarios principales:

	áreas pequeñas	áreas extensas
baja subdiv.	**lineal** baja preparación preproceso poco recorrido por interpolación **quad** preparación constante (mayor) poco recorrido por interpolación	**lineal (baja precisión)** baja preparación preproceso mucho recorrido interp. lineal **quad** preparación constante (mayor) mucho recorrido (a mayor coste)
alta subdiv.	**lineal** mucho preproceso (mayor) poco recorrido por interpolación **quad** preparación constante (menor) poco recorrido por interpolación	**lineal** mucho preproceso (mayor) mucho recorrido interp. lineal **quad** preparación constante (menor) mucho recorrido (a mayor coste)

Cuadro 5.4: Comparación de Métodos según Escenario

En general, el método de subdivisión se ve afectado por el coste de una preparación previa (la subdivisión) que aumenta conforme se requiere mayor precisión.

En este aspecto, el método propuesto mantiene un coste constante en esta fase, que si bien es mayor comparado con niveles de subdivisión bajos, ofrece ventaja cuando se compara con niveles de subdivisión altos.
Respecto a esto, el método propuesto aventaja al clásico en las situaciones de altos niveles de subdivisión o requisito de suavidad en la curva a obtener.

Por otra parte, una vez realizadas las tareas de preparación, ambos métodos se basan en avanzar recorriendo la superficie por medio de interpolaciones con diferencias avanzadas.

El método lineal tiene un coste mucho menor (grado 1) frente al método propuesto (que emplea ecuaciones de grado 4), por lo que en el momento de recorrer el interior de las figuras este ultimo tienen un coste mayor. Esto es especialmente apreciable en casos de áreas extensas.

Se puede concluir con la observación de que el método propuesto tiene ciertas ventajas ante niveles de alta subdivisión, pero su alto coste de recorrido por interpolación (frente al lineal), puede penalizarlo cuando tiene que cubrir grandes áreas.

En resumen:

Los tiempos de ejecución obtenidos comparados al uso de triángulos simples es lógicamente mayor, pero al enfrentarse con niveles de subdivisión mas altos (en busca de una mejor representación de la superficie curva) podemos observar que se obtiene un cierto beneficio.

La precisión (y similitud respecto a subdivisión nivel 10) de las imágenes resultantes no alcanzar a ser suficiente, no tanto por la suavidad del contorno mostrado, si no por la desviación de estas respecto de una curva perfectamente calculada.
Por esto se presenta necesario estudiar nuevas formas de realizar los cálculos de forma mas adecuada.

En cuanto al aprovechamiento de los cálculos realizados, y dado que el objetivo del método propuesto era ajustarse a la resolución de salida, se observa una gran ahorro. La reducción de puntos redundantes ya supone por sí mismo una menor carga de trabajo para las siguientes fases del pipeline (Fragment Shader, etc), y por tanto un ahorro de recursos y energía.

Ante una fase de Raster Shader programable por el usuario soportada por el hardware gráfico, sería susceptible de aplicarse también otras optimizaciones y paralelización al método propuesto, haciendo uso de los recursos que en este entorno se presentan.

Indicar que, aunque en los tests realizados no ha empleado ningún tipo de optimización, la subdivisión aprovecha (hoy por hoy) la posibilidad de tratar cada triángulo de forma paralela.
El método propuesto también podrá aprovechar las de unidades de proceso que se asignen a tareas de Raster.

Cabe esperar que, tras perfeccionar y optimizar el método, se obtengan resultados mejores, con figuras mas precisas y reducción en el tiempo de proceso.

Capítulo 6

Línea de Trabajo

6.1. Continuación de los Trabajos

El desarrollo expuesto es una experiencia inicial sobre el problema presentado. Sobre este trabajo caben futuras ampliaciones y mejoras, entre las que planteo las siguientes:

6.1.1. Comprobación de aristas rectas.

En la situación en que alguna de las aristas no tuviera curvatura, esto es, fuesen lineas rectas, seria interesante detectar este hecho y emplear para su recorrido un algoritmo lineal.
Esto evitaría la necesidad del calculo y recorrido de la función interpoladora inversa.

6.1.2. Comprobación de la monotonía de las aristas.

(y subdivisión de aristas y/o superficie en caso necesario)

En los test expuestos, los casos siempre entran dentro de las restricciones descritas. Sin embargo, para permitir aplicar esta técnica a situaciones mas generales será preciso detectar si se encuentra ante un caso que no cumpla las restricciones. En caso negativo, concretamente en el caso en que la arista tenga una curvatura excesiva, se encontraría con que la función no es monótona en el eje a recorrer, obteniendo grandes errores en la aplicación del método.

Una primera solución propuesta es la de dividir dicha arista en su punto de inflexión. Esto generará dos nuevas aristas, en cuyo rango sí se cumple la monotonía en el eje. Para realizar esta operación planteo el uso de los estudios de DeCasteljau en cuanto a la subdivisión de una curva en un punto aleatorio. [Alois12]

Posteriormente, y puesto que la arista dividida forma parte del contorno de una superficie, cabria estudiar la forma en la que esta superficie queda igualmente dividida. En general, se hace necesario comprobar que esta división genere siempre aristas que mantengan la monotonía.

También es valorable la posibilidad de realizar una subdivisión estática de la superficie en una fase previa, asegurando así siempre la monotonía de las aristas, y evitando la convivencia de diferentes topologías finales.

6.1.3. Adaptación de la dirección de raster.

En general, las lineas de raster se tratan horizontalmente.
En nuestro caso, a cada nueva linea de raster, se habrá realizado un avance en ambas aristas laterales (contorno), y en la cuerda de superficie. A partir de estos puntos se genera la curva de raster horizontal, sobre la cual se calcula su función interpoladora inversa, para que su posterior recorrido de la linea de raster tiene un coste menor.

Este calculo inicial por cada linea de raster es notable, y conviene reducirlo al máximo. Una forma de mejorar este comportamiento es plantear el uso de lineas de raster verticales.

Ante triángulos con poca variación vertical (frente a su amplitud horizontal) se obtendrán mejores resultados, al trazarse un número menor de lineas de raster. Sin embargo, en función de la amplitud horizontal y vertical del triangulo a tratar se puede escoger una u otra dirección de raster.[Hersch88]

De esta forma, siempre se generarán el menor número de lineas de raster posible, minimizando los costes de preparación de su interpolador.

6.1.4. Nuevas funciones de aproximación.

La función de aproximación empleada en los tests adolece de problemas de precisión, básicamente por la imposibilidad o alto coste de obtener resultados exactos. Sin embargo, se pueden tratar de buscar nuevas funciones de aproximación, o variar las condiciones actuales que se utilizan para la aproximación presentada.

Aproximar con la combinación lineal de múltiples funciones polinomiales.

Se podrían buscar 2 o mas funciones polinomiales que, combinadas linealmente (en base al valor t a calcular en cada paso), y dando mayor peso a la función que mejor se aproxime en ese punto, resulten en una mejor función de la inversa.

Aproximar con polinomio de lagrange, con referencias variables.

Esto es, no tomar como referencia unos puntos fijos ($t = 0/0{,}1/0{,}3/0{,}6/1{,}0$), ya que estos puntos de referencia pueden ser adecuados ante una curvatura dada, pero desastrosos ante otra opuesta.
Será necesario estudiar la distribución adecuada a tomar para estas de referencias, en base a la curvatura original del caso; que básicamente estaría dada en función al punto de control.

Un punto de control centrado (con valor al 50 % entre el valor inicial y final) expresaría una distribución lineal, pudiéndose tomar puntos de referencia centrados y equiespaciados. Un punto de control mas próximo a al inicio requeriría mayor concentración de puntos de control en ese rango; y a la inversa ante un punto de control mas próximo al valor final, dado que representa una curvatura en sentido contrario (cóncava vs convexa).

6.2. Ampliaciones y Avances a Explorar

El método presentado se centra en un caso concreto y restringido de menor complejidad, donde ha sido posible explorar los problemas propios de la técnica propuesta. Este método puede ser modificado y mejorado para obtener otras soluciones y adaptarse a nuevas condiciones.

En general, las vías de ampliación sobre las que seria interesante emplear esta técnica podrían ser:

6.2.1. Aplicación sobre otras geometrías base.

Trabajo con superficies cuadradas. (Quads)

Se ha partido de la base de superficies triangulares (tri-patch), por tener ofrecer una mayor similitud para emplear los algoritmos clásicos de procesado de triángulos. La evolución directa sería aplicar la técnica sobre superficies con cuatro lados (quads), pudiéndose investigar en dos líneas:

- Descomponiendo cada polígono en triángulos, generando una nueva arista curva äl vuelo.
- Tratando directamente un contorno con 4 lados, sin descomposición previa.

Superficies curvas de grado superior. (Cúbicas)

Igualmente, en cuanto a la forma matemática de las curvas, se ha empleado grado 2. Son conocidas las limitaciones de este grado, y en general, se suele optar por curvas de grado 3. Su uso requiere un estudio mas pormenorizado para tratar de trasladar el método propuesto a estos niveles.

Por una parte, se necesitarían nuevos métodos o funciones aproximadoras inversas; pero anterior a esto cabría un análisis [Blinn06a] [Blinn06b] de las restricciones necesarias a aplicar, para garantizar regiones en que la inversa tenga una solución única.

6.2.2. Mejoras en la eficiencia de los cálculos a realizar.

Empleo de múltiples interpoladores.

En lugar de emplear el interpolador para obtener valores del parámetro t a cada paso de X, y con él calcular el resto de información (Z, u,v, normal, etc); dado que el problema está acotado obteniendo unicidad en la función inversa, se pueden emplear múltiples interpoladores inversos independientes (o sus aproximaciones) simultáneos para cada uno de los parámetros.

Tener en consideración que para el calculo directo de una función cuadrática ya se dispone de una implementación hardware, dedicada actualmente al muestreo de texturas, pudiendo ser otra propuesta la generalización de su uso.

En cualquier caso, al proponer un shader de raster abierto, esta opción queda abierta a ser empleada por el usuario según la aplicación a desarrollar.

Paralelización de las tareas.

Se podría aprovechar el entorno y la naturaleza del problema, realizando cálculos de forma paralela.
Concretamente, según se avanza por el contorno (y cuerda central), la linea de raster queda ya definida, y su recorrido interno puede ser lanzado como tarea independiente, paralelizando el calculo de estas.

Evaluación de operaciones generales

Dentro del actual contexto reducido, sería preciso realizar ajustes en ciertas operaciones generales, como:

Estimación de la cuerda central

Ya que, actualmente, se emplea simplemente un punto de referencia centrado en la superficie, terminando en el vértice mas inferior.
Esto puede no ser del todo correcto, observándose que si la geometría finaliza con la ultima arista en una orientación casi horizontal, esta cuerda se desviaría significativamente hacia un lateral.

Se propone observar la conveniencia de, finalizar la cuerda sobre un punto de la arista inferior, o generar una nueva cuerda una vez alcanzado el centro de la superficie.

Estimación de puntos de control

Actualmente, al convertir una curva que pasa por 3 puntos a una curva Bézier, se está estimando que el corresponde al valor $t = 0{,}5$. Sin embargo, esta apreciación puede no ser correcta, debiéndose aplicar la conversión con un valor distinto de t, tal vez dependiente de la proximidad del punto a los extremos.

6.2.3. Otras Ampliaciones.

Submuestreo y Multimuestreo

Valorar la viabilidad del método aplicándose a problemas de aliasing. Básicamente, se podrían realizar recorridos a incrementos constantes, menores de 1 pixel. Notar que este submuestreo puede ser de diferente nivel en cada eje, y por tanto, tener niveles de antialising distinto en el contorno que en el interior de la superficie.

Combinación con otros estudios.

Retomando alguno de los trabajos consultados referentes al tema, se podrían aplicar sobre este método como, por ejemplo, hacer uso de información de distancia al borde como factor para determinar el impacto del error producido por las aproximaciones.

Capítulo 7

Conclusiones

Al comienzo de este trabajo se planteó la posibilidad de ampliar las capacidades gráficas con la inclusión de una fase de raster programable por el usuario.
Un aplicación para esta idea, de forma que fuese atractivo su estudio, sería el poder tratar de forma directa superficies curvas, ajustándose perfectamente a la resolución de salida.

Tras las debidas investigaciones en cuanto a la evolución y estado del arte de los pipelines gráficos, y de los métodos de representación de superficies curvas, se ha constatado un interés real y actual sobre este tipo de problemas.

Si bien el método propuesto no puede llegar a obtener una precisión perfecta, debido a la propia naturaleza de las funciones que definen la curva y la imposibilidad del calculo de su inversa de forma precisa, no por ello debe ser desechada.

Al igual que otros métodos (subdivisión), una aproximación adecuada es suficiente para obtener unos resultados visualmente agradables.

El aspecto que en un principio suscitaba mayores dudas era si resultaría competente. Era obvio que el coste computacional sería mayor que los empleados para la interpolación lineal, así mismo, estos han tenido ya mucho tiempo para evolucionar y ser cada vez mas óptimos.
Su principal baza consiste en poder obtener un mismo nivel de calidad con un coste constante, únicamente dependiente del área a cubrir.

Al observar los resultados obtenidos tras los experimentos, su coste se sitúa cerca (e incluso levemente inferior) al correspondiente de aplicar uno o dos niveles de subdivisión. Concluir con esto la utilidad y viabilidad del método.

Si bien, para área reducidas, la aproximación por un único polígono, o una única iteración de subdivisión puede ser suficiente, para áreas mayores se hace necesario aumentar el número de iteraciones, lo que a su vez incrementa el coste de computación.
Es en esta situación donde el método propuesto representa una alternativa, no solo viable, sino ventajosa, obteniendo una representación mas ajustada a la superficie curva, y a su vez siendo menos costosa que al aumentar el nivel de subdivisión, ya que su coste es constante.

Destacar también el volumen de puntos calculados, que reducen drásticamente los obtenidos por altos grados de subdivisión, evitando redundancia y reduciendo carga en las siguientes fases del pipeline. (con el mejor aprovechamiento y ahorro de energía que esto supone).

Para poder confirmar estos datos, sería deseable implementar el método con la ayuda de los sistemas gráficos actuales, aprovechando la especialización del hardware, y la paralelización de los cálculos. Sin embargo, mientras la fase de raster no sea programable, estas pruebas se realizarán en entornos de computación genérica, no dentro de un pipeline gráfico real.

En cualquier caso, aun quedan pendientes múltiples tareas para mejorar el método propuesto, haciéndolo lo más eficiente posible y con menores margenes de error, e incluso ampliar el ámbito de aplicación, tratando geometrías mas útiles e intuitivas para los diseñadores gráficos.

Finalmente concluir que queda, al menos mínimamente, justificada la propuesta del "Raster Shader" programable, abriendo también en esta línea nuevos estudios de aplicaciones a desarrollar aprovechando esta funcionalidad.

Bibliografía

[Ackland81] B D Ackland, N H Weste. *The Edge Flag Algorithm - A Fill Method for Raster Scan Displays.* 1981. URL: http://citeseerx.ist.psu.edu/showciting?cid=264958.

[Alois12] Alois Zingl. *A Rasterizing Algorithm for Drawing Curves.* 2012. URL: http://members.chello.at/easyfilter/bresenham.pdf.

[Banerjee15] A Banerjee, A Ghosh, M Das. *High Performance Novel Square Root Architecture Using Ancient Indian Mathematics for High Speed Signal Processing.* 2015. URL: doi:10.4236/apm.2015.58042.

[Bartels87] R H Bartels, J C Beatty, B A Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* 1987. URL: http://citeseerx.ist.psu.edu/showciting?cid=14047.

[Bartley97] Curtis Bartley. *Forward Difference Calculation of Bezier Curves.* 1997. URL: http://dl.acm.org/citation.cfm?id=282762.

[Bezier67] P Bézier, J Thilliez. *La commande numérique des machines.* 1967. URL: http://cds.cern.ch/record/275531?ln=es.

[Bezier86] Pierre Bezier. *The Mathematical Basis of the UNISURF CAD System.* 1986. URL: http://dl.acm.org/citation.cfm?id=536514.

[Blinn05a] J F Blinn. *How to Solve a Quadratic Equation.* 2005. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1528437.

[Blinn05b] J F Blinn. *How to Solve a Quadratic Equation, Part 2.* 2005. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1607926.

[Blinn06a] J F Blinn. *How to solve a cubic equation, part 1: The shape of the discriminant.* 2006. URL: http://dl.acm.org/citation.cfm?id=1137279.

[Blinn06b] J F Blinn. *How to solve a cubic equation, part 2: The 11 case.* 2006. URL: http://dl.acm.org/citation.cfm?id=1158859.

[Blinn78a] J F Blinn. *Computer Display of Curved Surfaces.* 1978. URL: http://dl.acm.org/citation.cfm?id=908845&coll=DL&dl=GUIDE&CFID=297527239&CFTOKEN=55068479.

[Blinn78b] J F Blinn. *A scan line algorithm for displaying parametrically defined surfaces.* 1978. URL: http://dl.acm.org/citation.cfm?doid=988437.988439.

[Bresenham65] J E Bresenham. *Algorithm for computer control of a digital plotter.* 1965. URL: http://dx.doi.org/10.1147/sj.41.0025.

[Bruijns98] J. Bruijns. *Quadratic Bezier Triangles As Drawing Primitives.* 1998. URL: http://dl.acm.org/citation.cfm?id=285305.285307&coll=DL&dl=GUIDE&CFID=415646505&CFTOKEN=24344302.

[Casteljau59] P de Casteljau. *Outillage méthodes calcul.* 1959. URL: http://citeseerx.ist.psu.edu/showciting?cid=741672.

[Catmull74] E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* 1974. URL: http://dl.acm.org/citation.cfm?id=907242.

[Chang89] S L Chang, M Schantz, R Rocchetti. *Rendering Cubic Curves and Surfaces with Integer Adaptive Forward Differencing.* 1989. URL: http://dl.acm.org/citation.cfm?doid=74334.74349.

[Clark79] J H Clark. *A fast scan-line algorithm for rendering parametric surfaces.* 1979. URL: http://old.siggraph.org/publications/rarities/clark-parametric-1979supp.pdf.

[Cordero02] J M Cordero Valle, J Cortés Parejo. *Curvas y superficies para modelado geométrico.* 2002. URL: http://dialnet.unirioja.es/servlet/libro?codigo=142787.

[Dellajustina14] F Dellajustina, L Martins. *The Hidden Geometry of the Babylonian Square Root Method.* 2014. URL: doi:10.4236/am.2014.519284.

[Dyn90] Dyn, Levine. *A butterfly subdivision scheme for surface interpolation with tension control.* 1990. URL: http://dl.acm.org/citation.cfm?id=78958.

[Farin86] G Farin. *Triangular Bernstein-Bezier Patches.* 1996. URL: http://citeseerx.ist.psu.edu/showciting?cid=552780.

[Farin88] G Farin. *Curves and Surfaces for Computer Aided Geometric Design.* 1988. URL: http://dl.acm.org/citation.cfm?id=61954&coll=DL&dl=GUIDE&CFID=420367584&CFTOKEN=94153377.

[Goraud71] H. Gouraud. *Computer Display of Curved Surfaces.* 1971. URL: http://content.lib.utah.edu/cdm/ref/collection/uspace/id/4477.

[Hersch88] R D Hersch. *Vertical scan-conversion for filling purposes.* 1988. URL: http://link.springer.com/chapter/10.1007/978-3-642-83492-9_28.

[Hersch93] Roger D. Hersch. *Font Rasterization: State of the Art.* 1993. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.94.5523.

[Hillesland13b] Karl Hillesland. *Open GL and Direct X.* 2013. URL: http://dx.doi.org/10.1145/2542266.2542276.

[Kokojima06] Yoshiyuki Kokojima, Kaoru Sugita, Takahiro Saito, Takashi Takemoto. *Resolution Independent Rendering of Deformable vector objects using graphics hardware.* 2006. URL: http://dl.acm.org/citation.cfm?id=1179997.

[Krishnamurthy07] Adarsh Krishnamurthy, Rahul Khardekar, Sara McMains. *Direct evaluation of nurbs curves and surfaces on the gpu.* 2007. URL: http://dl.acm.org/citation.cfm?id=1236246.1236293&coll=DL&dl=GUIDE&CFID=372487142&CFTOKEN=11052246.

[Kwon08] Taek-Jun Kwon. *Floating-point division and square root implementation using a Taylor-series expansion algorithm.* 2008. URL: http://dx.doi.org/10.1109/ICECS.2008.4674950.

[Lane80] J M Lane, L C Carpenter, T Whitted, J F Blinn. *Scan line methods for displaying parametrically defined surfaces.* 1980. URL: http://dl.acm.org/citation.cfm?id=358815.

[Leben10] Ivan Leben. *Random Access Rendering of Animated Vector Graphics Using GPU.* 2010. URL: http://ivanleben.blogspot.com.es/.

[Levin76] Joshua Levin. *A Parametric Algorithm for Drawing Pictures of Solid Objects Composed of Quadric Surfaces.* 1976. URL: http://dl.acm.org/citation.cfm?id=360355.

[Lien87] Sheue-Ling Lien, Michael Shantz, Vaughan Pratt. *Adaptive forward differencing for rendering curves and surfaces.* 1987. URL: http://dx.doi.org/10.1145/37401.37416.

[Loop05] Charles Loop, Jim Blinn. *Resolution Independent Curve Rendering Using Programmable Graphics Hardware.* 2005. URL: http://research.microsoft.com/en-us/um/people/cloop/loopblinn05.pdf.

[Loop06] Charles Loop. *Realtime GPU rendering of piecewise algebraic surfaces.* 2006. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.187.9980.

[Loop87] Charles Loop. *Smooth subdivision surfaces based on triangles.* 1987. URL: http://citeseerx.ist.psu.edu/showciting?cid=86692.

[Manocha94] Dinesh Manocha, James Demmel. *Algorithms for Intersecting Parametric and Algebraic Curves I: Simple Intersections.* 1994. URL: http://dl.acm.org/citation.cfm?id=174617.

[Manocha95] Dinesh Manocha, James Demmel. *Algorithms for Intersecting Parametric and Algebraic Curves II: Multiple Intersections.* 1995. URL: http://www.sciencedirect.com/science/article/pii/S1077316985710106.

[Microsoft11] Microsoft. *Direct 3D 11 Graphics Pipeline.* 2011. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ff476882(v=vs.85).aspx.

[Nehab08] Diego Nehab, Hugues Hope. *Random Access Rendering of General Vector Graphics.* 2008. URL: http://dl.acm.org/citation.cfm?id=1457515.1409088&coll=DL&dl=GUIDE&CFID=415467094&CFTOKEN=69296340.

[Pasko96] Alexander A Pasko, Andrei V Savchenko, Vladimir V Savhenko. *Implicit Curved Polygons.* 1996. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.991.

[Patney13] Anjul Patney. *Programmable Graphics Pipelines.* 2013. URL: http://idav.ucdavis.edu/~anjul/docs/dissertation_anjul_patney.pdf.

[Peters94] Jörg Peters. *Evaluation and approximate evaluation of the multivariate Bernstein-Bezier form on a regularly partitioned samples.* 1994. URL: http://dl.acm.org/citation.cfm?id=198434&CFID=415646505&CFTOKEN=24344302.

[Rockwood89] A Rockwood, K Heaton, T Davis. *Real-time Rendering of Trimmed Surfaces.* 1089. URL: http://dl.acm.org/citation.cfm?doid=74334.74344.

[Santina11] Rami Santina. *Resolution Independent NURBS Curves Rendering using Programmable Graphics Pipeline.* 2011. URL: http://www.researchgate.net/publication/258568429_Resolution_Independent_NURBS_Curves_Rendering_using_Programmable_Graphics_Pipeline.

[Schwarz06] Michael Schwarz, Marc Stamminger. *Pixel-Shader-Based Curved Triangles.* 2006. URL: http://dl.acm.org/citation.cfm?id=1179622.1179680&coll=DL&dl=GUIDE.

[Shantz88] M Shantz, S Shang. *Rendering trimmed NURBS with adaptive forward differencing.* 1988. URL: http://dl.acm.org/citation.cfm?id=378510.

[Toledo04] Rodrigo De Toledo, Bruno Levy, Jean-claude Paul. *Extending the graphic pipeline with new gpu-accelerated primitives.* 2004. URL: http://scholar.google.es/scholar?cluster=14027368352311278 29&hl=es&as_sdt=0,5&as_ylo=2004&as_yhi=2004.

[Toledo07] Rodrigo de Toledo, Bruno Levy, Jean-claude Paul. *Iterative Methods for Visualization of Implicit Surfaces on GPU.* 2007. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.102.5396.

[Turner78] Turner Whitted. *A scanline algorithm for computer display of curved surfaces.* 1978. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.108.7055.

[Vlachos01] Alex Vlachos, Jörg Peters, Chas Boyd, Jason L.Mitchell. *Curved PN Triangles.* 2001. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.104.4338.