UNIVERSIDAD
DE GRANADA

Facultad de Ciencias

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Estructuras diferenciables sobre una superficie topológica y la visualización computacional de superficies.

Presentado por:
Norberto Fernández de la Higuera

Tutor:
Francisco José López Fernández
Departamento de Geometría y Topología

Carlos Ureña Almagro
Departamento de Lenguajes y Sistemas Informáticos

Curso académico 2020-2021

Estructuras diferenciables sobre una superficie topológica y la visualización computacional de superficies.

Norberto Fernández de la Higuera

Norberto Fernández de la Higuera *Estructuras diferenciables sobre una superficie topológica y la visualización computacional de superficies.*.
Trabajo de fin de Grado. Curso académico 2020-2021.

Responsable de tutorización

Francisco José López Fernández
Departamento de Geometría y Topología

Carlos Ureña Almagro
Departamento de Lenguajes y Sistemas Informáticos

Doble Grado en Ingeniería Informática y Matemáticas

Facultad de Ciencias

Universidad de Granada

Declaración de originalidad

D./Dña. Norberto Fernández de la Higuera

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2020-2021, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 6 de septiembre de 2021

Fdo: Norberto Fernández de la Higuera

Dedico este proyecto a mis familiares, que me han dado todo su apoyo sin esperar nada a cambio, y a mis compañeros, que también han puesto todo su esfuerzo y empeño en sus respectivos proyectos.

Índice general

Agradecimientos

Agradezco enormemente el apoyo que he recibido por parte de mi familia y mis dos tutores. Valoro todo el esfuerzo que han realizado por atender mis consultas durante un periodo educativo complicado, sobretodo durante las vacaciones de verano.

Resumen

El trabajo realizado consiste en el desarrollo de una demostración más sencilla de un teorema clásico de la topología diferencial. Está orientado a transmitir al lector la importancia de dicho resultado. Además, se traslada al ámbito informático para entender la dificultad que colleva representar correctamente una superficie.

En la parte matemática se trata la demostración de 2 resultados indicados en el artículo de Allen Hatcher [Hat13] cuyo corolario directo es el teorema central: toda variedad topológica 2-dimensional tiene una única estructura diferenciable salvo difeormorfismos. Su desarrollo requerirá de mucha información de la teoría de Morse, la cual es muy amplia y por tanto no se aportará un gran nivel de detalle, ya que el objetivo de este proyecto no es la investigación en el campo de dicha teoría.

El estudio de la demostración del teorema nos ha aportado varios recursos bastante interesantes, tales como la técnica del toro de Kirby o el uso del grafo asociado a una función de Morse para obtener difeomorfismos de partes de la superficie a elementos más sencillos, con sus estructuras diferenciables usuales (discos, anillos, "pantalones" o "pantalones cruzados").

Para la visualización de variedades diferenciales 2-dimensionales, se aproximarán con una malla indexada de triángulos, cuyos vértices siempre estarán en la superficie objetivo. La variedad se representará mediante cartas que el usuario puede definir como una expresión en texto. Dichas cartas podrán ser tan arbitrarias como el usuario desee, siempre que las pueda definir en el lenguaje, que serán traducidas a un lenguaje de programación y se compilará para ejecutarse en la GPU. Además, cabe destacar que se ha conseguido que absolutamente todos los cálculos relativos a la teselación se hagan en la GPU, en tiempo real, con tiempos por cada cuadro del orden de milisegundos.

El problema residirá en estudiar cuándo es necesario subdividir un triángulo para representar con mayor precisión esa zona.

Para visualizar correctamente una superficie cualquiera se han estudiado varias definiciones posibles, atendiendo a diferentes cualidades de las superficies. Puesto que lo que se desea es que la superficie se visualize correctamente, la definición escogida estará orientada a cómo la visión humana la percibe. Anque nos centremos en esta definición, trivialmente el límite de la malla indexada será la superficie a representar.

Finalmente, el programa implementado nos ha servido de herramienta de visualización de homotopías de manera eficiente, incluso para la función de Morse "altura" con dominio la superficie definida. En un futuro se podría incluir la posibilidad de que el usuario indique una función de Morse, pero habría que redefinir los programas "procesador" (procesa la parametrización en un lenguaje específico) y "visualizador" (visualiza la escena).

Las palabras clave que definen este proyecto son:

- Superficie.
- Profundizar.
- Entendimiento.
- Visualización.
- Eficiencia.

Summary

This project is about a very important result on differential topology. The main goal is to transmit how complex is the proof for a student, but been more comprehensible via Allen Hatcher's article [Hat13]. Also, we will translate this problem to the computer scope, showing how difficult is to represent a surface correctly.

A typical problem of differential topology is known if it is possible to find a unique smooth structure for every topological surface (except diffeomorphisms). J. R. Munkres proved that result in his thesis in 1955 but using some sophisticated analytical techniques. Allen Hatcher used "The Kirby Torus Trick" (a technique proposed by Kirby and Shiermann in 1960) to proof the theorem for 2-dimensional manifolds.

The initial main goals are to implement and design an efficient program to visualize homotopies, and proof the theorem as a corollary of the 2 results cited on Allen Hatcher's article [Hat13]. I think I have reached both goals and more because I found some interesting functionalities, like using partials of a defined function on the parametrization, and been able to visualize some characteristics of Morse functions (actually, only for "Height" function). However, it would have been interesting to have deepened more in Morse theory, but it would have required an exclusive project.

Mathematical part

I will introduce some concepts and results about differential topology, principally from the subjects Topology II and Smooth Manifolds, and obviously Morse theory. After that, I enunciate the important results of this part, the "facts" (six facts). Those results are supposed by the author (Allen Hatcher) for the proof of the "Handle Smoothing Theorem" and theorems A and B (the 2 main results whose corollary is the objective theorem). He proofs them at the end of his article.

Those facts need a large knowledge about Morse theory. To understand them, I used an article that concentrates some important results [CKoo]. Also, it gives summarized proofs for almost all of them.

Then, I enunciate and extend the proof of the "Handle Smoothing Theorem", whose objective is to smooth one embedding incrementally, via finding isotopies (first on the origin, then on the segment $[0, 1]$ and finally on the disk). It will help us to build a smooth structure starting from a base chart (topological chart, but being the only one it trivially defines a smooth structure) because we need a set of charts whose transition maps are smooth.

On the proof of that theorem, the author introduces the "Torus trick" (The Kirby Torus Trick), bringing up the structure of the surface to the Torus using an immersion to smooth

the induced structure and bringing it back to the original surface (plus other steps). This together with Morse theory (specially, the "facts") are the key to build the entire proof of the theorems A and B because the rest needs only a correct connection.

Thanks to the previous theorem and the facts, I will enunciate and extend the proofs of theorems A and B (remember that in our case a manifold is a 2-manifold):

- **Theorem A**: Every manifold has a smooth structure.
- **Theorem B**: Every homeomorphism between smooth manifolds can be isotoped to a diffeomorphism.

For theorem A, using a triangulation of the open set of the transition map's preimage (supposing that is not smooth), we will be able to use the "Handle Smoothing Theorem" first on the vertex, then on the edges and finally on the interiors of the triangles. Therefore, we will have an isotopy of the main chart (whose have been affected by non-smooth transition map). Equivalently, we will have a chart with a smooth transition map with the previous charts. Then, by induction, we will define a smooth structure of the manifold, starting with a topological one.

Theorem B's proof is very similar to the proof of theorem A. We only need to translate chart smoothing for smoothing the homeomorphism, using the fact that exists a smooth triangulation on every smooth manifold.

Eventually, the central theorem of the project is a trivial corollary of theorems A and B.

Computational part

Before explaining anything about this part, like the first part I will enunciate some useful concepts about 3D visualization, especially OpenGL (Khronos Wiki [Khr] was the main source in this topic).

After that, I begin with the development of tessellation study, in other words, how the program will subdivide the triangles to approximate better the surface. The first thing to do is select the correct definition of "good surface approach", under some threshold $\epsilon > 0$.

Thanks to computer vision we know that the human vision perceives a surface through its geometry, edges, and textures. Because I use plain textures in this case and the charts defines a unique surface (only 1 geometry element), we can reduce vision perception elements to edges and texture (how the light reacts to the surface). Abbreviating, the program will tessellate depending on the curvature (Gauss curvature, K, is the first characteristic of a surface that directly influences the light) and it will tessellate more on edges and areas with more light (dark areas will not be tessellated).

Also, I have implemented some typical techniques to improve the efficiency of the program, like adding tessellation distance (max distance to tessellate a triangle) or avoiding tessellation out of view frustum (out of the user's view). In some special cases we can also use an additional improvement, that prevent tessellate on hidden triangles. E. g. if we have a sphere,

some triangles are visible on the front and others are not, on the back, then the program will not tessellate them.

Previously, I have studied more definitions, but the results were bad, because it was not able to detect critical areas like peaks (volume-based definition) or tessellates when it was not necessary (area-based definition).

In practice, before starting with tessellation study, it was needed to implement a program that pre-processes the charts (that defines the surface), giving as output a translation to GLSL language (used on the shaders) with additional functions, like the normal function of the charts and Gaussian curvature. For that, the program must recognize the expression trees of the functions and be able to derivate them (partial derivative respect to x_i, where x_i is a function's variable). That GLSL code will be included in the shaders that will need it (tessellation shaders).

For the implementation of the "processor" I used a base code, provided in the subject Language Processors. It gave me an initial implementation that, after adapting it, can detect lexical, syntactical, and semantical errors.

In case of the visualizer, I used some example code that implements a basic shader for a simple 3D scene (one object with one light, using Phong lighting). For its interface I used a free software library called Dear ImGui [Cor], with some addons for showing the light vector [Mor] and a file selection dialog [Cui], to select a parametrization.

The measures used to know if the program with tessellation works better than not using tessellation, have been the frames per second (how many frames have been calculated on one second) and the number of generated primitives. I want a program that visualizes the similar surface using less triangles than uniform approximation (with another initial mesh size), and generating more fps. This would mean that the program manages all the resources efficiently.

Because there are a lot of shading code that is executed many times (once per primitive or per vertex), it is required the best possible optimization. On Khronos Wiki [Khr] there is some information about that. The more important thing is to avoid repeated calculations, like the inverse of a constant matrix (constant during rendering one frame). Apart of that, we can group some scalar calculations in a vector operation, reducing the number of operations (the number of ticks to resolve a vector operation is similar than a scalar operation, due to the GPU characteristics).

Conclusions

After doing all the project, I can conclude that Allen Hatcher's article [Hat13] provides a lot of results, which are so important in differential topology, supported by a large and interesting theory (Morse theory). It would be interesting going deeper into that theory, but it is very complex and will require a lot of time (it has content as for an own subject).

Respecting to the program, I realized how difficult is to correctly define a good approxi-

mation of a surface. Also, I have noticed that this program would be able to have more functionalities, like adding the possibility of using partials in the parametrization (implemented) and visualize Morse functions, and some characteristics (like critical points and, possibly, them indices).

There is a topic that I have not been able to study, because of the time. It is "Transform feedback", a functionality of OpenGL that allows render again the output mesh. It would add a lot of possibilities, including recursive tessellation (the originally thought algorithm). Logically, it will have a high negative impact on efficiency, but it should be studied carefully.

Also, we could modify the program to give as output a mesh that represents a surface with a certain number of primitives as goal. It would improve the performance of the program if we are treating with static objects (only surfaces, not homotopies), avoiding all the calculations for each vertex (calculate image through parameterization and the normal through the normal function of the parameterization).

It would be a good idea to implement the program with WebGL, to make it more portable and flexible.

These are the key words of the project:

- Surface.
- Deepen.
- Comprehension.
- Visualization.
- Efficiency.

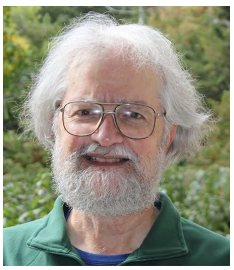
Introducción

En el ámbito matemático, cuando se define una estructura ligada a un elemento, la pregunta natural es si siempre es posible encontrar una para un elemento concreto (existencia) y en caso de que exista, si es la única bajo ciertas condiciones (unicidad).

Un problema clásico de la topología diferencial es si es posible definir para toda superficie topológica una única estructura diferenciable salvo difeomorfismos. J. R. Munkres lo demostró en su tesis doctoral (1955) utilizando que toda superficie topológica admite una triangulación diferenciable, junto con ciertas técnicas analíticas sofisticadas.

J. R. Munkres.

Dicha cuestión ha estado ligada a la clasificación de estructuras lineales a trozos, para variedades topológicas n-dimensionales. En 1960, Kirby y Shiermann usaron una técnica novedosa en ese campo, el "Truco del Toro" de Kirby, para demostrarlo en dimensiones superiores. Más tarde, esta técnica fue adaptada por A. J. S. Hamilton en 1976, demostrando de nuevo los resultados sobre existencia y unicidad de estructuras lineales a trozos en variedades tridimensionales aportados por Moise. Parte de esta información se puede encontrar en la introducción que hace J. R. Munkres en su artículo "Obstructions to Imposing Differentiable Structures" [Mun64] y la recopilación histórica de Ciprian Manolescu [Man14].



Allen Hatcher.

Lo que consigue Allen Hatcher en su artículo [Hat13] es trasladar la técnica del toro de Kirby a variedades topológicas 2-dimensionales, enunciando y demostrando 2 resultados cuyo corolario trivial es el teorema de existencia y unicidad para el caso de superficies topológicas.

Cabe destacar que el teorema deja de ser cierto para estructuras de mayor complejidad, como la conforme y la isométrica, he incluso para variedades de dimensión $n \geq 3$.

Para comprender la importancia del problema de suavizar una "estructura topológica", se ha propuesto el estudio de la realización de una buena aproximación a una superficie, gestionando los recursos de manera eficiente para su visualización en tiempo real. Además, permitirá la visualización de homotopías a modo de animaciones fluidas.

Actualmente estamos observando el auge del hardware para renderizado mediante trazados de rayos, muy útil para la correcta representación de una superficie (bajo un cierto margen

de error), pero está lejos de las capacidades de un computador medio si se quiere ejecutar en tiempo real. Es por ello que se utiliza una malla indexada de triángulos para aproximar la variedad diferenciable 2-dimensional, la cual se definirá mediante cartas que el usuario indicará en un lenguaje específico.

Dichas cartas serán procesadas por medio de un programa llamado "Procesador" que dará como salida un código GLSL, el cual se incluirá en los shaders donde sea necesario. Este programa no funcionará sólo como traductor, sino que tiene el objetivo principal de obtener los árboles de expresión de las funciones definidas y ser capaz de calcular los árboles de las derivadas parciales, para obtener las funciones típicas de una superficie (normal, curvatura, etc).

El estudio en sí consistirá en detectar en qué zonas será necesaria una mayor cantidad de triángulos para aproximarla mejor. Inicialmente se propuso el uso del Geometry shader para implementar un algoritmo de teselado propio, ya que el Geometry shader requiere una versión no muy reciente de OpenGL y aportaría mayor flexibilidad. Sin embargo, por algunas causas que se comentarán en el apartado de **Implementación y pruebas**, decidí utilizar el Tessellation shader, que aunque sea más reciente, está diseñado específicamente para el teselado, incrementando considerablemente el rendimiento de la aplicación.

El problema a tratar es la explicación de la prueba de existencia y unicidad, salvo difeomorfismos, de una estructura diferenciable para toda variedad topológica 2-dimensional sin bordes, de manera que un estudiante del grado de Matemáticas lo entienda con claridad, siempre que suponga ciertos los resultados utilizados de la teoría de Morse. También se diseñará he implementará un programa capaz de visualizar una superficie por cartas, mediante rasterizado, gestionando correctamente los recursos. Además, se ha conseguido incluir la posibilidad de utilizar derivadas parciales de funciones definidas por el usuario y la visualización de funciones de Morse por niveles (isobaras) junto con sus puntos críticos (actualmente sólo para la función de Morse "altura").

Contenido de la memoria

El trabajo, sin contar las secciones comunes, se ha dividido en 2 partes, una para la demostración del teorema central (parte matemática) y otra para el estudio de visualización de superficies (parte informática). Cada parte tiene un capítulo dedicado a los conceptos básicos utilizados, para orientar al lector.

- Para la parte matemática, el capítulo que trata los conceptos básicos se complementará con otro en el que se indiquen los resultados más importantes utilizados de la teoría de recubridores y la de Morse.
- En cuanto a la parte informática, además de la descripción del trabajo base, se han añadido los capítulos referentes al desarrollo de la aplicación (capítulos 7, 8 y 9), aunque cabe destacar que al haber tenido bastante peso el estudio de la teselación, algunos diagramas no se han tenido en cuenta. Es decir, el diseño de la aplicación en sí ha sido el más sencillo posible, para dedicar el mayor tiempo posible a la teselación, eficiencia, robusted y portabilidad de la aplicación.

Se han añadido como apéndices una **Guía de instalación**, donde además se indicarán los requisitos previos, y una **Guía de uso**.

Fuentes principales

Como principales fuentes de información para la realización del proyecto, se han consultado las siguientes referencias:

- En la demostración del teorema, los artículos de Allen Hatcher [Hat13] y [CK00].
- Para la implementación del programa, la documentación de OpenGL en la Wiki de Khronos [Khr] y la de Dear ImGui para la interfaz [Cor].

Objetivos

A continuación se mostrarán los objetivos inicialmente propuestos, los alcanzados y aquellos campos de estudio más utilizados.

Objetivos y métodos inicialmente propuestos:

- Ámbito de las Matemáticas:
  - Se tratarán algunos aspectos básicos de topología diferencial en dimensión baja.
  - Se abordará una prueba más sencilla de un teorema clásico de Munkres sobre la unicidad de las estructuras diferenciables soportadas por una superficie topológica.
- Ámbito de la Informática:
  - Desarrollar software de visualización y animación 3D de superficies, que permitirá ilustrar visualmente algunos conceptos matemáticos del ámbito de las superficies topológicas diferenciables.
  - Se analizarán los algoritmos relacionados descritos en la literatura y se implementarán los más adecuados a los conceptos a ilustrar.
  - El software debe ser eficiente, robusto y portable.

Objetivos alcanzados y métodos usados:

- Ámbito de las Matemáticas:
  - Se han tratado algunos aspectos básicos de topología diferencial en dimensión baja. Para su correcto tratamiento han sido necesarias algunas herramientas provenientes de la teoría de Morse, que hemos utilizado sin incluir sus demostraciones. Han sido necesarios algunos conocimientos de análisis complejo (esfera de Riemann) y álgebra (para trabajar con los grupos fundamentales).
  - Se ha abordado una prueba más sencilla de un teorema clásico de Munkres sobre la unicidad de las estructuras diferenciables soportadas por una superficie topológica. Sólo se ha alcanzado para el caso de superficies sin borde.
- Ámbito de la Informática:
  - Se ha desarrollado un software de visualización y animación 3D de superficies definidas por cartas, que permitirá ilustrar visualmente conceptos matemáticos como el cálculo de normales, la curvatura de Gauss, funciones de Morse (sólo función altura) y sus puntos críticos.
  - El usuario puede escribir cualquier función, que pueda definir con el lenguaje, que seguidamente será traducida a un lenguaje de programación (GLSL) el cual se compilará para ejecutarse en la GPU.

  - Se han analizado los algoritmos de generación de árboles de expresión y árboles derivados, junto con algoritmos de teselado y procedimientos de muestreo. Se han implementado los más adecuados para la definición escogida de "buena aproximación" a una superficie, con el objetivo de mejorar la calidad visual con un buen rendimiento.
  - El software realiza absolutamente todos los cálculos relativos a la teselación en la GPU, consiguiendo por tanto visualizar animaciones fluidas en tiempo real, con tiempos por cuadro del orden de milisegundos. Su portabilidad está sujeta a las librerías utilizadas, por lo que si es posible instalarlas con versiones iguales o superiores, será viable su instalación y ejecución.

Campos de estudio más utilizados:

- Ámbito de las Matemáticas:
  - Topología diferencial en dimensión baja, análisis complejo y álgebra.
- Ámbito de la Informática:
  - Para el procesador: Procesadores de Lenguajes y Programación.
  - Para el programa de visualización: Informática gráfica, Diseño Orientado a Objetos, Sistemas Operativos y algunos conceptos de Visión por Computador.

Parte I.

Teorema clásico de Munkres

En esta parte se desarrollará una demostración más sencilla de un teorema clásico sobre variedades 2-dimensionales, haciendo uso de teoría de Morse.

No profundizaremos demasiado en dicha teoría, puesto que es un campo bastante amplio y no es el objetivo principal del proyecto.

1. Conceptos previos

Este capítulo tiene el objetivo de introducir los conceptos básicos utilizados en esta primera parte del proyecto. Dichos conceptos se han obtenido de las referencias [CKoo], [Chh17] y [VCo7], junto con el conocimiento adquirido de las asignaturas de Topología II y Variedades Diferenciables.

Definición 1.1. Una **variedad topológica** 2-dimensional es un espacio de Hausdorff localmente Euclídeo que verifica el segundo axioma de numerabilidad, es decir, su topología tiene una base numerable.

Definición 1.2. Un **embebimiento o encaje** es una aplicación continua e inyectiva de un espacio topológico en otro que es un homeomorfismo sobre la imagen dotada por la topología inducida.

Definición 1.3. Un **sistema coordenado** (o carta) sobre S es un embebimiento $h : \mathbb{R}^2 \to S$.

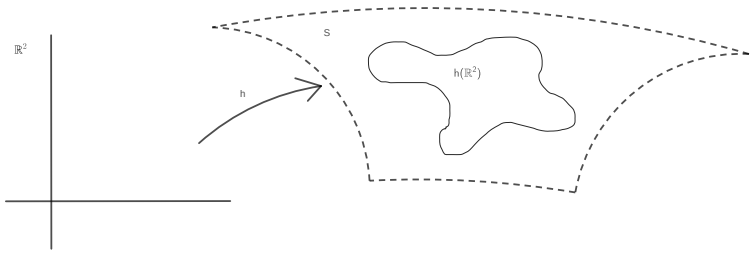


Figura 1.1.: Ejemplo de carta.

Definición 1.4. Sea S un espacio topológico Hausdorff, un **atlas** diferenciable 2-dimensional sobre S es una familia de cartas $E = \{h_i\}_{i\in\Lambda}$ verificando:

1. $\{h_i(\mathbb{R}^2)\}_{i\in\Lambda}$ es un recubrimiento abierto de S.
2. Si $h_i(\mathbb{R}^2) \cap h_j(\mathbb{R}^2) \neq \varnothing$ entonces $h_j^{-1} \circ h_i : h_i^{-1}(h_i(\mathbb{R}^2) \cap h_j(\mathbb{R}^2)) \to h_j^{-1}(h_i(\mathbb{R}^2) \cap h_j(\mathbb{R}^2))$ es un difeomorfismo.

Definición 1.5. Sea S un espacio topológico Hausdorff, una **estructura diferenciable** 2-dimensional sobre S es un atlas diferenciable maximal.

Definición 1.6. Una **variedad diferenciable** 2-dimensional es una variedad topológica 2-dimensional S junto con una estructura diferenciable E, es decir, el par (S, E).

Definición 1.7. Una aplicación $f : N \to M$ entre las variedades diferenciables N y M se dice que es **diferenciable** si la composición con las cartas de ambas estructuras diferenciables es diferenciable en el sentido real, es decir, $\forall \phi \in E_N$, $\forall \psi \in E_M$ se tiene que $\psi^{-1} \circ f \circ \phi$ es diferenciable.

Definición 1.8. Una **inmersión** es una aplicación diferenciable entre variedades diferenciables cuya diferencial es inyectiva en todo punto.

Definición 1.9. Una n-celda $n \in \mathbb{N}$, en un espacio topológico X, es una aplicación continua $f : \overline{B}^n \to X$ tal que, si llamamos $C = f(B^n)$, la aplicación $f|_{B^n} : B^n \to C$ es un homeomorfismo (donde B^n es la bola unidad abierta en $\mathbb{R}^n$ y $\overline{B}^n$ la cerrada). Si X es una variedad diferenciable y f es diferenciable (es decir, extiende diferenciablemente más allá de $\overline{B}^n$), la celda se dirá diferenciable.

Al conjunto $f(\partial B^n) = f(S^{n-1})$ se le llama caras borde de la celda (lados si $n = 2$, vértices si $n = 1$), mientras que al conjunto C se le refiere como el interior de la celda. Es común identificar la celda con el conjunto C si ello no conlleva a ambigüedad.

Definición 1.10. Una **triangulación diferenciable** de una variedad diferenciable X es un conjunto de celdas diferenciables con interiores disjuntos dos a dos y cuya unión es X.

Definición 1.11. Sean f y g homeomorfismos entre los espacios topológicos X e Y. Una **isotopía** es una homotopía que deforma f en g, $H : X \times [0,1] \to Y$, y que satisface:

1. $H(\cdot, t) : X \to Y$ es un homeomorfismo $\forall t \in [0,1]$.
2. $H(p,0) = f(p)$, $H(p,1) = g(p)$ $\forall p \in X$.

Definición 1.12. Dos embebimientos $f, g : N \to M$ de una variedad N en otra M se dicen **isotópicos** si existe $H : N \times [0,1] \to M$ aplicación continua tal que:

1. $H(\cdot, t) : N \to M$ es un embebimiento $\forall t \in [0,1]$.
2. $H(p,0) = f(p)$, $H(p,1) = g(p)$ $\forall p \in N$.

Dados dos embebimientos $f, g : N \to M$, en la mayoría de casos las isotopías $H : N \times [0,1] \to M$ que consideraremos entre ambos serán de la siguiente forma:

$$H = H_0 \circ (f \times Id_{[0,1]})$$

Donde $H_0 : M \times [0,1] \to M$ es una isotopía en M que deforma Id_M en un homeomorfismo $g_0 : M \to M$, en particular, $g = g_0 \circ f = H_0(f, 1)$.

Definición 1.13. Sea M una variedad diferenciable y $f : M \to \mathbb{R}$ una función diferenciable en M. Un punto $p \in M$ se dice que es un **punto crítico** de f si $Df(p) = 0$ en T_pM. A su imagen por f, $f(p)$, se le dice **valor crítico** de f.

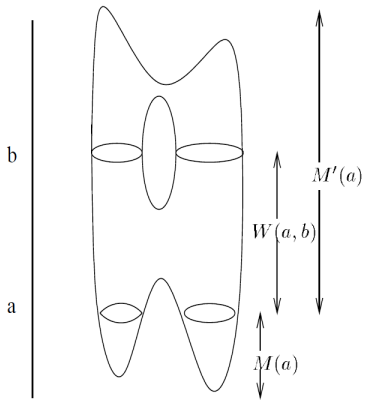
Definición 1.14. Sea M una variedad diferenciable y $f : M \to \mathbb{R}$ una función diferenciable en M. Un punto crítico $p \in M$ se dice que es **no degenerado** si $H_\phi(f)(p)$ es regular para cualquier parametrización ϕ centrada en p, donde $H_\phi(f) = H(f \circ \phi)$ es la matriz Hessiana de $f \circ \phi$.

El **índice** de dicho punto es la dimensión del mayor subespacio de T_pM donde $H_\phi(f)$ es definida negativa. No depende de ϕ por la regla de Sylvester, ya que H para otra parametrización ψ cumple $H_\psi(f) = J^t(\theta) H_\phi(f) J(\theta)$, con θ el cambio de coordenadas y J la matriz Jacobiana.

Definición 1.15. Sea $f : M \to \mathbb{R}$ función diferenciable, con M una variedad, se dice que es una **función de Morse** si todos sus puntos críticos son no degenerados.

Definición 1.16. Sea f una función de Morse en una superficie diferenciable S, dados $a < b$ valores regulares de f en $\mathbb{R}$ definimos:

- $V(a) = f^{-1}(a)$, se suele referir como una curva de nivel f.
- $M(a) = f^{-1}((-\infty, a])$, el conjunto que hay "debajo" de la curva de nivel para el valor regular a.
- $M'(a) = f^{-1}([a, \infty))$, el conjunto que hay "encima" de la curva de nivel para el valor regular a.
- $W(a, b) = f^{-1}([a, b])$, el conjunto contenido entre 2 curvas de nivel.



Definición 1.17. Una aplicación continua f se dice **propia** si para todo compacto M, su imagen por f^{-1} es compacta.

2. Resultados previos

2.1. Teoría de recubridores

A continuación se enuncia el resultado utilizado de la Teoría de Recubridores.

Teorema 2.1 (de Levantamiento de aplicaciones)**.** *Sea $(\tilde{X}, \pi)$ recubrimiento de X, Y espacio topológico y $f : Y \to X$ aplicación continua. Sea $y_0 \in Y$, $x_0 = f(y_0) \in X$ y $\tilde{x_0} \in \pi^{-1}(x_0)$ entonces son equivalentes:*

1. $\exists! \tilde{f} : Y \to \tilde{X}$ *continua tal que* $\tilde{f}(y_0) = \tilde{x_0}$ *y* $\pi \circ \tilde{f} = f$.
2. $f_*(\Pi_1(Y, y_0)) \subset \pi_*(\Pi_1(\tilde{X}, \tilde{x_0}))$.

2.2. Teoría de Morse

A continuación se enuncian resultados de la Teoría de Morse, principalmente utilizados para la demostración de los Hechos.

Teorema 2.2. *Sea M subvariedad de un espacio euclídeo E. Entonces:*

- *Para casi todo $v \in E^*$ (dual de E) y $p \in E$, las funciones $h_v, r_p : M \to \mathbb{R}$ definidas así:*

$$h_v(x) = v(x)$$

$$r_p(x) = \frac{1}{2}||x - p||^2$$

  son funciones de Morse.

- *Si M no contiene al origen, entonces la función $q_A : M \to \mathbb{R}$ definida por:*

$$q_A(x) = \frac{1}{2} < Ax, x >$$

  es una función de Morse para casi todo A endomorfismo simétrico positivo de E.

Lema 2.1. *Sea S una superficie diferenciable compacta y $f : S \to \mathbb{R}$ una función de Morse en S, entonces f puede verse como la función altura respecto a un adecuado embebimiento de S en un espacio euclídeo.*

Corolario 2.1. *Para cualquier S superficie diferenciable se puede encontrar una función de Morse $f : S \to \mathbb{R}$ que tenga todos sus puntos críticos en niveles distintos.*

Demostración. La primera parte es una consecuencia trivial del teorema 2.2, obteniendo una f que por el lema anterior se pueda ver como la función altura (respecto a un embebimiento adecuado). La parte restante se puede conseguir deformando dicha f para conseguir que los puntos críticos estén a distintos niveles, gracias a que son puntos aislados. □

Teorema 2.3. *Sea S una superficie y $f : S \to \mathbb{R}$ una función de Morse. Tomamos $a < b$ valores regulares tales que $W(a, b)$ no contenga ningún punto crítico de f. Entonces:*

- *$M(b)$ es difeomorfo a $M(a)$.*
- *$V(b)$ es difeomorfo a $V(a)$.*
- *$W(a, b)$ es difeomorfo a $V(a) \times [a, b]$, o de forma equivalente, cada componente conexa de $W(a, b)$ es difeomorfa a un anillo de $\mathbb{R}^2$.*

Demostración. Se puede consultar en el artículo "Clasification of Surface via Morse Theory" [CKoo], que corresponde con el teorema 8. □

Teorema 2.4. *Sea S una superficie diferenciable y $f : S \to \mathbb{R}$ una función de Morse. Sea p un punto crítico y $a < b$ valores regulares tales que $p \in W(a, b)$ y no contenga ningún punto crítico de f aparte de p. Entonces:*

- *Si el índice de p es 0 o 2, $M(b)$ es difeomorfo a la unión disjunta de $M(a)$ con un disco D, es decir, $W(a, b)$ es difeomorfo a un disco D.*

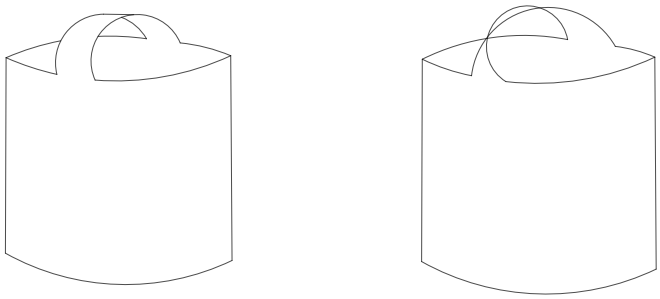


Figura 2.1.: Pantalones normales y cruzados (esquematizado)

- *Si el índice de p es 1, $M(b)$ es difeomorfo a $M(a)$ junto con un rectángulo pegado en dos segmentos disjnutos de $V(a)$, que se puede ver como unos "pantalones" si el pegado se realiza de acuerdo a la orientación, o unos "pantalones cruzados" en caso contrario.*

Demostración. El primer punto se demuestra en el artículo "Clasification of Surface via Morse Theory" [CKoo], el teorema 13. El punto siguiente se demuestra en el mismo artículo, el teorema 16. □

2.3. Hechos utilizados para los teoremas

A continuación se enuncian los "hechos" utilizados para las demostraciones de los teoremas. Su demostración detallada en varios casos requerirá asumir como válidos algunos resultados de teoría de Morse.

Hecho 2.1. *Todo $W \subset \mathbb{R}^2$ abierto tiene una triangulación clásica tal que el diámetro euclídeo de los triángulos se aproxima a 0 en la frontera topológica de W.*

Demostración. Tomamos la cuadrícula generada de forma natural por $\mathbb{Z}^2$ sobre $\mathbb{R}^2$. Vamos a definir de forma incremental el conjunto de cuadrados que cubren todo W.

Tomamos primero todos los cuadrados (cerrados) que estén contenidos estrictamente en W. Vamos a llamar U a la parte cubierta por el conjunto de cuadrados actual.

Dado un $p \in W$ y $p \notin U$ entonces existe un cuadrado que lo contiene pero que no está contenido estrictamente en W. Por ser W abierto sabemos que para p existe una bola abierta $B \subset W$ que lo contiene. De forma equivalente podemos subdividir el cuadrado inicial en 4 cuadrados iguales, 8 ... y así sucesivamente hasta encontrar una subdivisión en la que algún cuadrado contenga a p y sea lo suficientemente pequeño como para que esté contenido en la bola $B \subset W$. Añadimos al conjunto todos los cuadrados anteriores que estén contenidos en W. Realizamos esta operación de manera indefinida.

Una vez definido el conjunto de cuadrados, podemos definir la triangulación como los triángulos resultantes de dividir por la diagonal dichos cuadrados. De esta forma tenemos una triangulación T tal que la unión de sus triángulos, U, está contenida en W pero todo punto $p \in W$ está en algún triángulo, por lo que $U = W$. Además, cuando tomamos p tendiendo a ∂W, la bola $B \subset W$ que lo contiene tiene radio ϵ tendiendo a 0, es decir, el cuadrado necesario para cubrirlo correctamente tiende a 0, y como consecuencia los dos triángulos de los que se compone también. □

Hecho 2.2. *Toda variedad diferenciable S tiene una triangulación diferenciable.*

Demostración. Vamos a contruir una malla de polígonos diferenciables, que nos dará paso de forma trivial a una malla de triángulos diferenciables.

Por teoría de Morse, se sabe que existe $f : S \to \mathbb{R}$ función de Morse propia, de forma que los inversos de compactos sean compactos y todos sus puntos críticos estén en distintos niveles. Cortamos S por los niveles de puntos no críticos, para separar los puntos críticos entre sí, obteniendo así una descomposición de S en piezas difeomorfas a:

- Discos, tiene un punto crítico de orden 0 o 2.
- Anillos, no tiene puntos críticos.
- "Pantalones", o de forma equivalente, medio toro al que se le ha quitado un disco en el interior. Tiene un punto crítico de orden 1.
- "Pantalones cruzados", o también se pueden ver como medio toro suma conexa con un espacio proyectivo $\mathbb{RP}^2$. Por ello, se puede dividir en unos "pantalones" normales y una cinta de Möbius. Tiene un punto crítico de orden 1.

Por la teoría de Morse tenemos una división de S en conjuntos difeomorfos a alguno de los anteriores, todos ellos pegados por circunferencias (los bordes de los conjuntos descritos). Podemos obtener una malla añadiendo un vértice a cada circunferencia y seguidamente si es:

- Un disco, se toma el centro y se divide el disco en 3 partes, teniendo 3 sectores difeomorfos a un triángulo.
- Un anillo, se unen los 2 vértices (uno de cada circunferencia del borde) mediante un arco, obteniendo así un cuadrilátero.
- Unos "pantalones", se unen los 3 vértices mediante 2 arcos (un vértice común a los 2 arcos), dando lugar a un heptágono.
- Una cinta de Möbius, se une el vértice con él mismo mediante un arco, el cual recorre la mitad de la cara de la cinta, obteniendo así un triángulo.

Finalmente tenemos la malla de polígonos diferenciables, la cual podemos convertir en una auténtica triangulación en el sentido clásico de Radó, que además sea diferenciable, subdividiendo de forma adecuada los polígonos. □

Hecho 2.3. *Para toda estructura diferenciable E del toro punteado T'_E existe un subconjunto compacto suyo cuyo complemento es difeomorfo a $S^1 \times \mathbb{R}$ con la estructura diferenciable usual.*

Demostración. Dividimos T'_E tal y como lo hicimos en el **Hecho 2.2**, obteniendo así que está formado por piezas P_j separadas por circunferencias C_j. Las piezas pueden ser discos, anillos o pantalones (cintas de Möbius no puesto que T'_E es orientable y la orientabilidad es una propiedad topológica).

La forma en la que se pegan esas piezas P_j, se asocia a un grafo G donde los vértices representan a cada P_j y las aristas indican adyacencia (cada arista está asociada a una circunferencia C_j engarzando dos piezas contiguas, que representan los vértices en el grafo asociado a esa arista). Existe una aplicación cociente $q : T'_S \to G$ que lleva cada punto del entorno de una circunferencia C_j a la correspondiente proyección según la función de Morse propia usada, sobre el arco de G que representa a C_j.

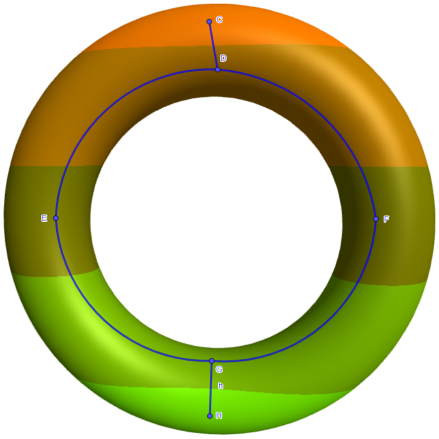


Figura 2.2.: Ejemplo de grafo asociado para la función de Morse "altura" que parte del Toro.

La aplicación $q_* : \Pi_1(T'_E) \to \Pi_1(G)$ es un homomorfismo sobreyectivo con inversa a la izquierda. Por tanto, $\Pi_1(G)$ es un cociente de $\Pi_1(T'_E)$, por lo que está finitamente generado. Esto implica que existe un subgrafo $G_0 \subset G$ tal que la clausura de $G - G_0$ consiste en un número finito de árboles (G se puede retraer a G_0). Sólo uno de esos árboles puede no ser compacto puesto que a T'_E le falta un único punto. Además, el no ser compacto implica que ese árbol está compuesto por un subárbol homeomorfo a $[0, \infty)$ junto con subárboles finitos pegados a él.

Podemos eliminar estos árboles finitos quitando las circunferencias que corresponden a dichos segmentos, cuyos vértices están asociados a discos P_j. Estas simplificaciones en G se pueden ver como simplificaciones en la función de Morse entendiendo los C_j como curvas de nivel, cancelando "sillines" con extremos locales (si es un disco que se adhiere a unos "pantalones") y anillos (si un C_j separa un anillo de un disco).

Finalmente tenemos que G tiene un subárbol no compacto G_f, homeomorfo a $[0, \infty)$ y la única posibilidad es que los segmentos correspondan a anillos, debido a que a T'_S sólo le falta un punto, que se puede entender como el límite de dicha sucesión de P_j y C_j. Tenemos que existe un compacto (la unión de los P_j y C_j correspondientes a $G - G_f$) cuyo complementario (los correspondientes a G_f) es una sucesión de anillos "pegados" de forma diferenciable en el sentido usual y por tanto es difeomorfo al cilindro con la estructura usual. □

Hecho 2.4. *Toda estructura diferenciable E del toro $(S^1 \times S^1)_E$ es difeomorfa a la estructura usual del toro $S^1 \times S^1$.*

Demostración. Partiendo de la demostración del hecho anterior, el grafo asociado G ahora es finito con $\Pi_1(G)$ el cociente del grupo abeliano $\Pi_1(T_E)$, es decir, necesariamente $\Pi_1(G) = \mathbb{Z}$. Podemos reducir G a una circunferencia, retrayendo los árboles finitos tal y como se hizo en la demostración anterior y haciendo los cambios necesarios en la función de *Morse*. La única posibilidad es que T_E sea una sucesión de anillos pegados diferenciablemente, así que T_E es difeomorfo a T con la estructura usual o a una botella de Klein, pero no puede ser éste último por ser $\Pi_1(T_E)$ abeliano (ya que el grupo fundamental de la botella de Klein no es abeliano y no es posible siquiera un homeomorfismo entre ellos). □

Hecho 2.5. *Sea E una estructura diferenciable en $D^1 \times \mathbb{R}$ tal que es la usual en un entorno del borde. Entonces existe un difeomorfismo $g : (D^1 \times \mathbb{R})_E \to (D^1 \times \mathbb{R})_U$, con U la estructura usual, que además es la identidad entorno a $\partial D^1 \times \mathbb{R}$.*

Demostración. Tomamos la proyección $\pi : (D^1 \times \mathbb{R})_E \to \mathbb{R}$ que ya es diferenciable en un entorno del borde. Si vemos $(D^1 \times \mathbb{R})_E$ como una variedad sabemos que existe una función de Morse $f : (D^1 \times \mathbb{R})_E \to \mathbb{R}$, a la que podemos obligar que coincida con π en un entorno del borde más pequeño que en el que es diferenciable π, donde no tendrá puntos críticos.

La función f es una función de Morse propia, con todos sus puntos críticos en distintos niveles. Los niveles de puntos no críticos están formados por un único arco y una o varias circunferencias. Al cortar por dichos niveles obtenemos discos, anillos, "pantalones", rectángulos y rectángulos con un "agujero" (un rectángulo menos un disco abierto). El grafo asociado G es un árbol debido a que $\Pi_1(D^1 \times \mathbb{R}) = 0$, y procedemos como en la demostración del hecho 2.3, alterando f de forma que el G asociado sea homeomorfo a $\mathbb{R}$.

La nueva función f no tiene puntos críticos, por lo que puede ser la segunda componente de un difeomorfismo $g : (D^1 \times \mathbb{R})_E \to D^1 \times \mathbb{R}$, que coincidirá con la identidad en un entorno del borde. La primera componente se puede obtener a partir del campo de vectores gradientes de f, de forma similar a como se prosigue en la demostración del teorema 2.2 de la teoría de Morse. □

Hecho 2.6. *Sea E una estructura diferenciable en D^2 tal que es la usual en un entorno del borde. Entonces existe un difeomorfismo $g : D^2_E \to D^2_U$, con U la estructura usual, que además es la identidad entorno a ∂D^2.*

Demostración. Tomamos la función que devuelve el radio (distancia al origen) en un entorno de ∂D^2_E y como es diferenciable, la extendemos a una función de Morse propia $f : D^2_E \to (0, 1]$, cuyos puntos críticos están en el interior y $f^{-1}(1) = \partial D^2_E$.

El grafo asociado G es un árbol por ser $\Pi_1(D^2) = 0$ y deformando f como se hizo anteriormente, podemos simplificar G a un único punto. Entonces, f sólo tiene un punto crítico p, que necesariamente debe de ser de índice 0 porque coincide con la función "radio" en el borde (f decrece a medida que nos acercamos p, es decir, en p la matriz Hessiana es definida positiva y por tanto el índice es 0).

Construimos $g : D^2_E \to D^2$ difeomorfismo a partir de f de manera similar a como se hizo en el apartado anterior, reproduciendo parte de la demostración del teorema 2.2 (siguiendo el flujo del campo de vectores gradientes de f). En particular, tenemos que g es la identidad en un entorno de ∂D^2_E. □

2.4. Teorema de Alisamiento de Asas

Teorema 2.5. *(de "alisamiento de asas") Sea S una superficie diferenciable, entonces:*

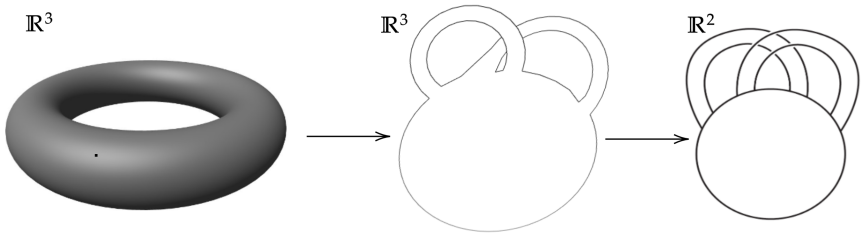
1. *Un embebimiento $h : \mathbb{R}^2 \to S$ puede isotoparse a un embebimiento diferenciable en un entorno compacto del origen. Además, la isotopía actúa como la identidad fuera de un entorno compacto que contiene al anterior.*
2. *Un embebimiento $h : D^1 \times \mathbb{R} \to S$ que es diferenciable entorno a $\partial D^1 \times \mathbb{R}$ puede isotoparse a un embebimiento diferenciable entorno a $D^1 \times 0$. Dicha isotopía actúa como la identidad sobre un entorno de $\partial D^1 \times \mathbb{R}$ y fuera de un entorno compacto de $D^1 \times 0$.*
3. *Un embebimiento $h : D^2 \to S$ que es diferenciable entorno a ∂D^2 puede isotoparse a un embebimiento diferenciable en todo D^2. La isotopía actúa como la identidad en un entorno de ∂D^2.*

Demostración. Voy a proceder a la demostración de cada uno de los apartados:

1. La idea de la demostración es arrastrar la estructura diferenciable de S (E_S) a una estructura diferenciable sobre el Toro (T_S) , y aprovechar que en tales condiciones existe un difeomorfismo de T_S al Toro con la estructura diferenciable estándar (por el **Hecho 4**), que nos permitirá construir la isotopía deseada.

   Vamos a utilizar la técnica del toro de Kirby, para ello veremos el toro T como el espacio de órbitas $\mathbb{R}^2/\mathbb{Z}^2$ con su estructura topológica y diferenciable estándar, tomando el 0 como imagen del $0 \in \mathbb{R}^2$. Eliminamos un punto del toro distinto del 0, y a esta nueva variedad la llamamos T'.

   Consideremos una inmersión $q : T' \to \mathbb{R}^2$ diferenciable que fija el 0. Dicha inmersión se puede construir partiendo del embebimiento del toro punteado T' en un disco con dos "1-asas" en $\mathbb{R}^3$ y seguidamente "aplanando" la figura, es decir, llevar diferenciablemente el disco con asas a $\mathbb{R}^2$. Las asas se embeben por separado ya que como se observa, se solapan en $\mathbb{R}^2$.



   Sea $h : \mathbb{R}^2 \to S$ el embebimiento del enunciado, por el cual, haciendo "pull-back", la estructura diferenciable de S induce una estructura diferenciable en $\mathbb{R}^2$, que denotaremos E_1. Por el mismo razonamiento pero para la inmersión q, $\mathbb{R}^2$ con la estructura E_1 induce una estructura diferenciable en T' que llamaremos E_2.

   Sabemos por el **Hecho 3** que existe un conjunto compacto en T'_{E_2} cuyo complemento

es difeomorfo a $S^1 \times \mathbb{R}$ con su estructura diferenciable estándar, equivalentemente es difeomorfo a $D^2 - (0,0)$ como abierto de $\mathbb{R}^2$ con su estructura diferenciable estándar. Si vemos el disco punteado como un subconjunto del plano complejo $\mathbb{C}$, el 0 se puede añadir de forma natural puesto que la estructura diferenciable usada hasta el momento es la usual en el cilindro (que induce la usual en D^2, en el plano complejo y en la esfera de Riemann). Esto nos permite extender la estructura diferenciable E_2 de T' a T, la cual seguiremos llamando E_2.

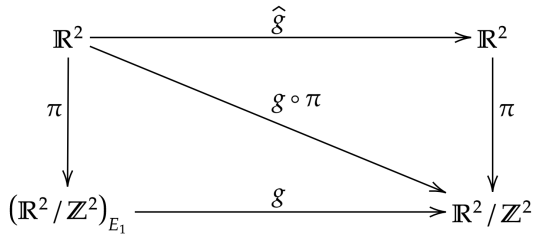
Por el **Hecho 4** sabemos que toda estructura diferenciable del toro ($T \equiv S^1 \times S^1$) es difeomorfa a la estándar. Por tanto, existe un difeomorfismo $g : T_{E_2} \to T$. Para poder utilizar el Teorema de Levantamiento de aplicaciones de la teoría de recubridores, necesitamos normalizar dicha función g:

- Aplicando rotaciones en el toro T (lo vemos como $S^1 \times S^1$) podemos hacer que g lleve el 0 en el 0.
- Necesitamos que el homomorfismo g_* sea la identidad a nivel de grupos fundamentales para que el difeomorfismo g pueda ser levantado a un difeomorfismo $\widehat{g} : \mathbb{R}^2_{E_1} \to \mathbb{R}^2$ fijando el origen. Para ello basta con componer g con el automorfismo lineal $L \in GL_2(\mathbb{Z})$ apropiado, es decir, aquel tal que al componerlo con g_* queden fijos los dos generadores del grupo fundamental del toro topológico. La nueva g sigue llevando el 0 en el 0 y g_* es la identidad.

$$\left(\mathbb{R}^2/\mathbb{Z}^2\right)_{E_1} \xrightarrow{\ g\ } \mathbb{R}^2/\mathbb{Z}^2 \xrightarrow{\ L\ } \mathbb{R}^2/\mathbb{Z}^2$$

$$\pi_1(0,\ T_{E_1}) \xrightarrow{\ g_*\ } \pi_1(0,\ T) \xrightarrow{\ L\ } \pi_1(0,\ T)$$

De esta forma construimos un difeomorfismo $\widehat{g} : \mathbb{R}^2_{E_1} \to \mathbb{R}^2$ como el levantamiento de g fijando el origen, que de forma natural, al compararlo con la identidad ($\widehat{g}(x) - x$, $x \in \mathbb{R}^2$) es doblemente periódico.



Identifiquemos $\mathbb{R}^2$ con el disco unidad D^2 de $\mathbb{R}^2$ vía un difeomorfismo radial (respecto de la estructura diferenciable estándar), que sea la identidad en un entorno del origen. A continuación trasladamos vía ese difeomorfismo la estructura diferenciable E_1 de $\mathbb{R}^2$ al disco D^2, generando una estructura E en el disco D^2, y análogamente la estructura diferenciable estándar de $\mathbb{R}^2$ que induce la estructura diferenciable estándar en

D^2. Entonces, salvo estas identificaciones en el dominio y codominio de $\widehat{g}$, obtenemos $G : D^2_E \to D^2$ automorfismo diferenciable, que sigue siendo $\widehat{g}$ entorno al 0 y tiende a ser la identidad en el borde (por la periodicidad $\|\widehat{g}(x) - x\|$ está acotado para todo x, y por consiguiente al tender x a infinito las variaciones tienden a 0 con la reparametrización, es decir, $G(x)$ tiende a x). Podemos extender G a un homeomorfismo $G : \mathbb{R}^2_E \to \mathbb{R}^2$, siendo la identidad fuera del interior del disco.

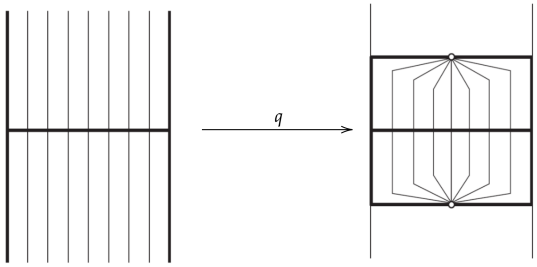
Por el truco de Alexander, G es isotópica a la identidad. Se puede obtener la isotopía G_t variando el radio del disco de origen y destino ($G_t(x) = tG(\frac{x}{t})$ para $x \in D((0,0),t)$ y la identidad fuera del disco $D((0,0),t)$), por lo que G_0 es la identidad y $G_1 = G$.

Definimos la isotopía que resuelva el problema como $h_t = h \circ G_t^{-1}$, teniendo que $h_0 = h$ por ser G_0 la identidad. Además, h_t se queda fija fuera del disco unidad ya que G_t es la identidad en dicho conjunto. También tenemos que $h_t(0) = h(0) = 0$ ya que para todo t, $G_t^{-1} = \widehat{g}^{-1}$ entorno al 0 y $\widehat{g}(0) = 0$. Finalmente, h_1 es diferenciable entorno al 0 porque $G_1^{-1} = G^{-1} = \widehat{g}^{-1}$ entorno al 0, que localmente es un difeomorfismo de la estructura usual de $\mathbb{R}^2$ en la inducida por S mediante h, que habíamos denotado por E_1.

2. La idea es, al igual que en el punto anterior, encontrar un automorfismo diferenciable del dominio de h con diferentes estructuras diferenciables y restringirlo para que esté fijo donde lo solicite el enunciado.

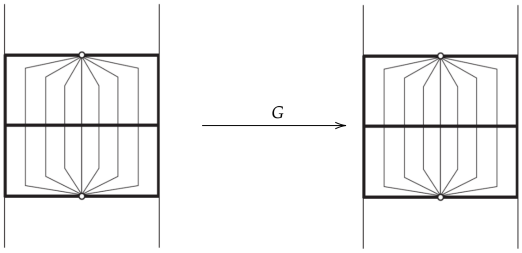
   Tenemos $h : D^1 \times \mathbb{R} \to S$ embebimiento diferenciable en un entorno del borde del dominio. Dicho embebimiento induce una estructura diferenciable E_1 en $D^1 \times \mathbb{R}$ que coincidirá con la estructura diferenciable estándar de $D^1 \times \mathbb{R}$ entorno al borde ya que h es diferenciable en el sentido usual ahí.

   Por el **Hecho 5** tenemos que existe un difeomorfismo g entre la estructura inducida E_1 y la estructura estándar de $D^1 \times \mathbb{R}$ que es la identidad entorno al borde del conjunto. Tomamos el homeomorfismo $q : D^1 \times \mathbb{R} \to (D^1 \times D^1) - (0 \times \partial D^1)$ cuya acción se sugiere en la siguiente figura; en particular q es la identidad en un entorno de $D^1 \times 0$.



   Como en el caso anterior trasladamos las correspondientes estructuras diferenciables vía q y definimos $G : ((D^1 \times D^1) - (0 \times \partial D^1))_{E_1} \to (D^1 \times D^1) - (0 \times \partial D^1)$ por $G = q \circ g \circ q^{-1}$, que como es la identidad entorno al borde del dominio, se puede extender a $G : \mathbb{R}^2_{E_1} \to \mathbb{R}^2$ como la identidad fuera de $D^1 \times D^1 - (0 \times \partial D^1)$. No hay problema en los dos puntos de $0 \times \partial D^1$ ya que por la definición de q el difeomorfismo

G extiende dejándolos fijos.. Su comportamiento entorno a $D^1 \times 0$ es igual que el de g (q es la identidad) y es la identidad fuera de $D^1 \times D^1$ y entorno a $\partial D^1 \times \mathbb{R}$.



Podemos adaptar el truco de Alexander definiendo una isotopía G_t de homemomorfismos en $\mathbb{R}^2$ rescalando el cuadrado $D^1 \times D^1$ al igual que en el apartado anterior hicimos con el disco, de forma que G sea isotópica a la identidad en $\mathbb{R}^2$.

Finalmente basta con definir $h_t = h \circ G_t^{-1}$. Cumple claramente que $h_0 = h$, h_1 es diferenciable en un entorno de $D^1 \times 0$ y h_t es la identidad en un entorno de $\partial D^1 \times \mathbb{R}$.

3. Tenemos $h : D^2 \to S$ embebimiento que es diferenciable entorno al borde del dominio. Este embebimiento induce mediante "pull-back" la estructura diferenciable de S a D^2 que coincide con la estructura diferenciable estándar de D^2 en un entorno del borde, ya que h es diferenciable en dicha zona en el sentido usual.

   Por el **Hecho 6** existe un difeomorfismo g entre D^2 con la estructura estándar y la inducida, que además es la identidad entorno al borde. Una adaptación del truco de Alexander al disco nos aporta la isotopía g_t de homeomorfismos de D^2, donde t va variando el tamaño de los discos de dominio e imagen de g y extendiendo por la identidad, por lo que g_0 sería la identidad y $g_1 = g$. Tomando la isotopía $h_t = h \circ g_t$ tendríamos lo solicitado, ya que $h_0 = h$ y h_1 es igual que h en un entorno del borde y es diferenciable en todo el disco.

□

Corolario 2.2. *El primer apartado del teorema anterior sigue siendo cierto para un abierto W de $\mathbb{R}^2$ en vez de para todo $\mathbb{R}^2$, con el objetivo de suavizarlo en un punto $p \in W$.*

Demostración. Dado un punto $p \in W$, por ser un conjunto abierto existe una bola abierta $p \in B_p \subset W$. Tomamos $f : \mathbb{R}^2 \to B_p$ homeomorfismo que lleva el origen a p, cuya restricción a la bola abierta de radio 2 centrada en el origen ($B((0,0),2) = B_2$) es diferenciable. Tenemos que $f|_{B_2} : B_2 \to f(B_2) \subset B_p$ es un difeomorfismo.

Definimos $g = h \circ f : \mathbb{R}^2 \to S$ que es trivialmente continua e inyectiva. Como f es un homeomorfismo y h un embebimiento, la restricción de g a su imagen es un homeomorfismo, y por tanto g es un embebimiento.

Aplicamos el primer apartado del teorema anterior a g, obteniendo la isotopía g_t, donde g_1 es diferenciable en un entorno del origen y queda fija fuera de un compacto que contiene al anterior. Además, por la demostración del teorema sabemos que dicho compacto está

contenido en la bola abierta B_2, por lo que podemos construir la isotopía h_t de la siguiente forma:

- $h_t(p) = (g_t \circ f|^{-1}_{f(B_2)})(p)$ para todo $p \in f(B_2)$.
- $h_t(p) = h(p)$ para todo $p \in W - f(B_2)$.

Sabemos que pega bien debido a que g_t deja fija a g fuera de un compacto que contiene al origen y está contenido en B_2. De esta forma tendríamos una isotopía que deforma h en una función que es diferenciable en un entorno del punto p y queda fija fuera de un compacto que contiene al anterior, concluyendo así la prueba. □

3. Resultados principales

3.1. Enunciados

Teorema A. *Toda variedad topológica 2-dimensional tiene una estructura diferenciable.*

Teorema B. *Todo homeomorfismo entre variedades diferenciables 2-dimensionales es isotópico a un difeomorfismo.*

Suponiendo ciertos los teoremas anteriores, es directa la obtención del siguiente resultado, ya que por A tenemos que toda variedad topológica 2-dimensional tiene una estructura diferenciable y por B sabemos que es única salvo difeomorfismos:

Corolario 3.1. *(Teorema clásico de Munkres) Toda variedad topológica 2-dimensional tiene una única estructura diferenciable salvo difeomorfismos.*

3.2. Demostración del Teorema A

Teorema A. *Toda variedad topológica tiene una estructura diferenciable.*

Demostración. Sea S una variedad topológica, podemos coger un atlas $\{h_i | 1 \leq i < N\}$ con $N \in \mathbb{N}$ si es finito o $N = \infty$ si no lo es. Vamos a construir por inducción una estuctura diferenciable en el conjunto $U_n = \cup_{i \leq n} h_i(\mathbb{R}^2)$ para cada $n < N$, que por tratarse de sistemas coordenados de un atlas su límite debe de ser S, probando así el resultado. Cabe destacar que cada U_n contiene a todos los anteriores.

La inducción empieza tomando una carta cualquiera del sistema, $U_1 = h_1(\mathbb{R}^2)$ por ejemplo. Si se considera la variedad U_1 con el atlas $\{h_1\}$ entonces h_1 es diferenciable para ésta de forma trivial (se compone con la inversa y queda la identidad en $\mathbb{R}^2$).

Una vez arrancada la inducción, suponiendo cierto para el paso $n-1$ vamos a extender la diferenciabilidad de U_{n-1} a U_n. Sea la carta h_n, tomamos entonces $W = h_n^{-1}(U_{n-1}) = h_n^{-1}(U_{n-1} \cap h_n(\mathbb{R}^2))$, que es un abierto de $\mathbb{R}^2$ por ser h_n continua.

Tenemos $W \subset \mathbb{R}^2$ abierto, por el **Hecho 1** sabemos que existe una triangulación geométrica suya y al ir acercándose a la frontera topológica los triángulos convergen a puntos. Queremos aplicar el "Teorema de alisamiento de asas" en los vértices de los triángulos, seguidamente en los lados y finalmente en el interior de cada triángulo (aplicar los 3 apartados del teorema de forma consecutiva), pero para ello es necesario partir de un embebimiento de $\mathbb{R}^2$:

1. Para todos y cada uno de los vértices de la triangulación elegimos una bola $B(p, \epsilon_p) \subset W$ cuyos cierres topológicos en $\mathbb{R}^2$ no se corten mutuamente. $B(p, \varepsilon_p)$ es abierto y queremos obtener $\widehat{h}$ diferenciable entorno a p.

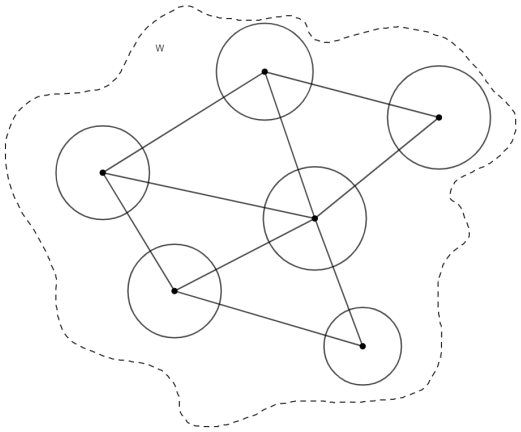


Figura 3.1.: $B(p, \epsilon_p)$ para cada vértice.

Podemos aplicar el Corolario del apartado 1 del Teorema de Alisamiento de Asas ya que cumplimos todas las hipótesis necesarias. Así obtenemos una $\widehat{h}$ isotópica a h, que es diferenciable en O_p entorno abierto de p y además queda fija fuera de otro entorno un poco mayor $O'_p \supset O_p$, con $O'_p \subset B(p, \epsilon_p)$.

De manera acumulativa, este procedimiento se puede realizar simultáneamente en todos los vértices p en la triangulación de W. Esto prueba que $h_n : W \to U_{n-1}$ es isotópica a un homeomorfismo $\widehat{h}_n : W \to U_{n-1}$ que es diferenciable, como aplicación sobre la superficie diferenciable U_{n-1}, en un entorno $O_p \subset W$ alrededor de cada vértice p de la triangulación de W. Además la isotopía coincide con h_n fuera de entornos $O'_p \subset W$ mayores que O_p para cada p, disjuntos 2 a 2.

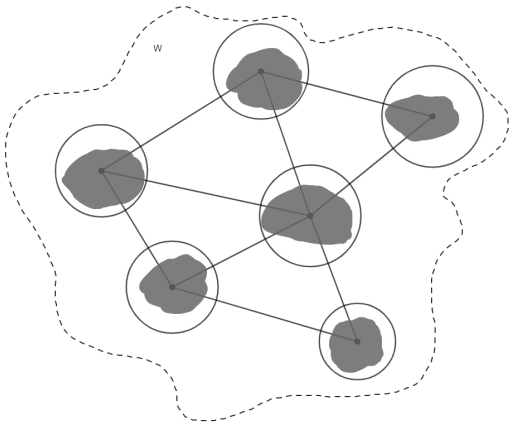


Figura 3.2.: O_p para cada vértice.

2. Tenemos por el paso anterior un $h_n : \mathbb{R}^2 \to h_n(\mathbb{R}^2)$ isotópico al original en las condiciones explicadas, y que es diferenciable como aplicación $W \to U_{n-1}$ entorno a los

vértices de la triangulación de W. Queremos utilizar el apartado 2 del Teorema de Alisamiento de Asas para generar otra isotopía que nos lleve h_n a otro homeomorfismo (al que le daremos el mismo nombre) cuya restricción a $W \to U_{n-1}$ sea diferenciable además entorno a los lados de la triangulación anterior, coincidiendo con el h_n original fuera de un entorno del 1-esqueleto de esa triangulación.

Para ello, consideramos para cada lado l de la triangulación de W un subconjunto R_l (rectángulo) dentro de W que sea difeomorfo a $D^1 \times \mathbb{R}$, cumpliendo:

a) R_l corta a l en un segmento compacto y es disjunto con cualquier otro lado de la triangulación de W. En particular, R_l no contiene ningún vértice de la triangulación de W.

b) Si p_1 y p_2 son los vértices extremos de l, una componente del borde de R_l está contenida en O_{p_1} y la otra en O_{p_2}, es decir, h_n es diferenciable en 2 componentes de ∂R_l.

c) Los cierres de los rectángulos R_l en $\mathbb{R}^2$ son disjuntos 2 a 2 y están contenidos en W.

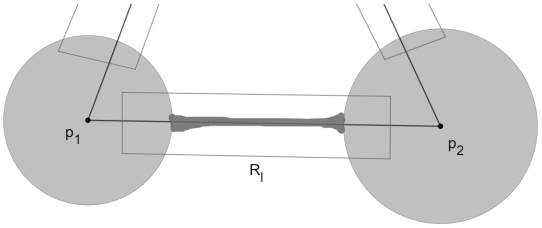


Figura 3.3.: Entorno de l donde h_n es diferenciable.

Ahora es evidente como en el apartado 2 del Teorema de Alisamiento de Asas nos produce la isotopía deseada realizando el trabajo simultáneamente en todos los rectángulos R_l, generando el nuevo $h_n : \mathbb{R}^2 \to h_n(\mathbb{R}^2)$ deseado.

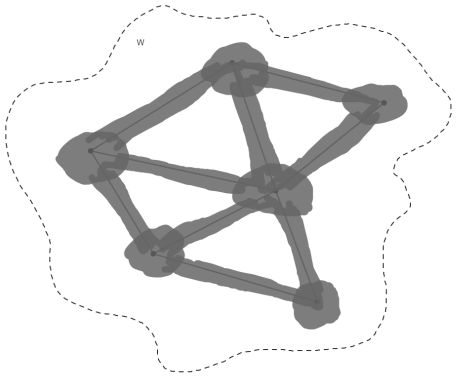


Figura 3.4.: Entorno del 1-esqueleto donde h_n es diferenciable.

3. El tercer paso es similar a los anteriores, pero ahora usando el apartado 3 del Teorema

de Alisamiento de Asas. Lo que hacemos en considerar para cada triángulo T de la triangulación de W un dominio de Jordan D_T satisfaciendo:

a) $D_T \subset \mathring{T}$.

b) $h_n : W \to U_{n-1}$ es diferenciable sobre $T - \mathring{D}_T$.

c) Los D_T son disjuntos dos a dos.

Aplicando el Teorema de Carathéodory obtenemos que es difeomorfo a la bola unidad y por tanto podemos proceder de manera similar a los apartados anteriores.

A continuación producimos otra isotopía que nos lleve el h_n generado en el apartado 2 a otro homeomorfismo (al que daremos el mismo nombre) cuya restricción $W \to U_{n-1}$ sea diferenciable sobre D_T, coincidiendo en cada instante con la h_n anterior en un entorno de ∂D_T y de hecho fuera de D_T, para cada triángulo T. Esto concluiría la prueba.

Para probar la existencia de D_T vamos a definir la curva de Jordan cuyo interior es de forma trivial un dominio de Jordan, que será dicho D_T. La curva debe ser diferenciable, cerrada y simple, que es la caracterización de una curva de Jordan. Reducimos el problema a buscar dicha curva para el entorno tubular de un triángulo equilátero, ya que es difeomorfo al de un triángulo cualquiera. Podemos simplificarlo más aportando únicamente una curva no cerrada cuyos extremos se puedan pegar consecutivamente, siendo infinitamente derivable en los puntos donde se unen.

Haciendo uso de una función meseta f que vale 0 en $\mathbb{R}^-$ y 1 a partir de $\epsilon > 0$, si tomamos $g(x) = tg(\frac{\pi}{3})xf(x)$ en el intervalo $[-1, \epsilon]$, tenemos que $g(-1) = 0$ y $g(\epsilon) = tg(\frac{\pi}{3})\epsilon$ al igual que sus derivadas, por lo que si vamos alternando $g(x)$ y $g(-x)$ mediante rotaciones y traslaciones, tendremos una curva α diferenciable (suavización del triángulo equilátero).

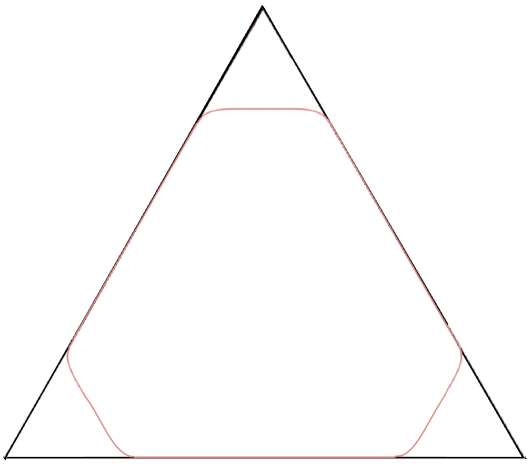


Figura 3.5.: Curva de Jordan cercana al triángulo

Se puede observar que es válido $\forall \epsilon > 0$ y que al hacer tender ϵ a 0, la curva será el

propio triángulo equilátero. Es por ello que podemos tomar el ϵ lo suficientemente pequeño como para que la curva α quepa en el entorno tubular y siga siendo una curva de Jordan. Como exigimos que $D_T \subset \mathring{T}$ podemos aplicar a la curva un factor de escala para así no contener ningún punto del borde del triángulo T. Además, de forma evidente obtenemos que los dominos D_T son disjuntos 2 a 2.

Todas las isotopías de los pasos anteriores coinciden por extensión continua con el homeomorfismo $h_n : \mathbb{R}^2 \to h_n(\mathbb{R}^2)$ original en la frontera de W en $\mathbb{R}^2$ por construcción, ya que el diámetro de los triángulos en W tiende a 0 al acercarnos a la frontera. Por tanto pueden ser extendidas como isotopías de $\mathbb{R}^2 \to h(\mathbb{R}^2)$ coincidentes con h_n en $\mathbb{R}^2 - W$.

Como conclusión, el homeomorfismo $h_n : \mathbb{R}^2 \to h_n(\mathbb{R}^2)$ resultante es compatible con la estructura diferenciable en U_{n-1} y junto con $h_1, \ldots, h_{n-1}$, nos define una estructura diferenciable sobre U_n. Esto cierra la inducción y prueba el teorema.

□

3.3. Demostración del Teorema B

Teorema B. *Todo homeomorfismo entre variedades diferenciables 2-dimensionales es isotópico a un difeomorfismo.*

Demostración. Sea $f : S \to S'$ un homeomorfismo entre superficies diferenciables. Por el Hecho 2.2 (ver su demostración) podemos encontrar una triangulación clásica en el sentido de Radó de S que también sea diferenciable. Eso significa que las 2-celdas son embebimientos diferenciables de triángulos geométricos en $\mathbb{R}^2$ en S, las 1-celdas embebimientos diferenciables de segmentos de $\mathbb{R}^2$ en S, y las 0-celdas (vértices de la triangulación) puntos en S. La prueba sigue las mismas ideas del Teorema A:

1. Para cada uno de los vértices p de la triangulación se considera una carta diferenciable $\varphi_p : \overline{D} \to \overline{B}_p \subset S$, $\varphi_p(D) = B_p$, $\varphi_p((0,0)) = p$, donde como siempre D es el disco unidad abierto y $\overline{D}$ el cerrado en $\mathbb{R}^2$. Haremos la elección de cartas de forma que los discos topológicos $\{\overline{B}_p \colon p \text{ vértice de la triangulación}\}$ sean disjuntos dos a dos. Para cada vértice p llamaremos $g_p = f \circ \varphi_p : D \to f(B_p) \subset S'$, y procediendo como en el Apartado 1 de la demostración del Teorema A, encontraremos un homeomorfismo $\hat{g}_p : D \to f(B_p)$ isotópico a g_p, diferenciable en un entorno O_p de $(0,0)$, y que coincida con g_p fuera de un entorno compacto O'_p de $(0,0)$ en D conteniendo a O_p. En particular el homeomorfismo $\hat{\varphi}_p = \hat{g}_p \circ \varphi_p^{-1} : B_p \to f(B_p)$ es isotópico a $f|_{B_p}$, diferenciable en un entorno de p, y que coincide con f fuera de un entorno compacto de p en B_p. Todas estas últimas isotopías se pegan de forma natural, extendidas como f fuera de las bolas B_p, para construir una isotopía entre f y un homeomorfismo $\hat{f} \colon S \to S'$ que es diferenciable en el abierto $O = \cup_p \varphi_p(O_p)$, y coincide con f fuera de $O' = \cup_p \varphi_p(O'_p)$.

2. Al homomorfismo $\hat{f}$ generado en el apartado anterior, por comodidad en la notación, le seguiremos llamando f. Para cada uno de los lados l de la triangulación se considera una carta diferenciable $\varphi_l \colon [-1,1]^2 \to \overline{R}_l \subset S$, $\varphi_l([-1,1]\times] - 1, 1[) = R_l$, $\varphi_l([-1,1] \times 0) = l_0$, donde l_0 es un subsegmento de l no conteniendo sus extremos. Haremos la elección de cartas de forma que los discos topológicos $\{\overline{R}_l \colon l \text{ lado de la triangulación}\}$ sean disjuntos dos a dos y además $\varphi_l(\partial D^1 \times [-1,1]) \subset O$. Para cada lado l llamaremos

$g_l = f \circ \varphi_l : [-1,1]\times]-1,1[\to f(R_l) \subset S'$, y procediendo como en el Apartado 2 de la demostración del Teorema A, encontraremos un homeomorfismo $\hat{g}_l : [-1,1]\times]-1,1[\to f(R_l)$ isotópico a g_l, diferenciable en un entorno U_l de $[-1,1]\times 0$, y que coincida con g_l en U_l', donde U_l' ahora es la unión de un entorno de $\partial D^1 \times]-1,1[$ en $[-1,1]\times]-1,1[$ y el complemento de un entorno compacto de $[-1,1]\times 0$ en $[-1,1]\times]-1,1[$ conteniendo a U_l. En particular el homeomorfismo $\hat{\varphi}_l = \hat{g}_l \circ \varphi_l^{-1} : R_l \to f(R_l)$ es isotópico a $f|_{R_l}$, diferenciable en un entorno de l_0, y coincide con f en un entorno de ∂R_l y fuera de un entorno compacto de l_0 en R_l. Todas estas últimas isotopías se pegan de forma natural, extendidas como f fuera de los rectángulos topológicos R_l, para construir una isotopía entre f y un homeomorfismo $\hat{f}\colon S \to S'$ que es diferenciable en un entorno U del 1-esqueleto de la triangulación y coincide con f fuera de un entorno cerrado $\overline{U}'$ de ese 1-esqueleto conteniendo a $\overline{U}$ en su interior U'.

3. Al homomorfismo $\hat{f}$ generado en el apartado anterior, por comodidad en la notación, le seguiremos llamando f. En éste último paso para cada uno de los triángulos T de la triangulación tomaremos una carta diferenciable $\varphi_T\colon \Delta \to T \subset S$, donde Δ representa la envolvente convexa en $\mathbb{R}^2$ de los puntos $(-1,0),(1,0),(0,1)$. Consideraremos también un disco abierto $D_T \subset \Delta$ con borde diferenciable de forma que $\varphi_T(\Delta \setminus D_T) \subset U$, y llamaremos $g_T = f \circ \varphi_T : \Delta \to f(T) \subset S'$. Procediendo como en el Apartado 3 de la demostración del Teorema A, encontraremos un difeomorfismo $\hat{g}_T : \Delta \to f(T)$ isotópico a g_T y que coincida con g_T en un entorno de $\partial\Delta$. En particular el difeomorfismo $\hat{\varphi}_T = \hat{g}_T \circ \varphi_T^{-1} : T \to f(R_l)$ es isotópico a $f|_T$ y coincide con f en un entorno de ∂T. Todas estas últimas isotopías se pegan de forma natural para construir una isotopía entre f y un difeomorfismo global $\hat{f}\colon S \to S'$, completando la prueba.

□

Parte II.

Visualización de superficies

Procederemos al estudio de la representación de una superficie dada las cartas que la definen. El estudio estará orientado al ajuste del nivel de subdivisión de la malla inicial, para alcanzar una cierta precisión en la representación.

Para ello será necesario elegir una definición de "buena aproximación" y tener en cuenta la percepción de la visión humana.

Además, se necesitará implementar un programa que extraiga los elementos relevantes de las cartas y lo ponga en práctica.

4. Conceptos básicos

Este capítulo tiene el objetivo de introducir los conceptos básicos sobre Informática Gráfica y OpenGL, tales como aquellos relacionados con la representación computacional de objetos "3D" y algunos resultados matemáticos usados, del campo de la topología diferencial. Dichos conceptos se han obtenido de las referencias [Khr] y [dV], junto con el conocimiento adquirido de la asignatura Informática Gráfica y las del departamento de Geometría y Topología.

Se sugiere la lectura de las definiciones de "sistema coordenado" (o carta), "variedad topológica", "variedad diferenciable" y "función de Morse" del apartado de **Conceptos Previos** de la parte I, con el objetivo de orientar al lector en el sentido matemático que subyace en esta parte informática (aunque no es necesario si ya se tiene una ligera idea).

Definición 4.1. Una **GPU** es una unidad de procesamiento gráfico, especialmente diseñada para la visualización de elementos gráficos (ya sean 2D o 3D mediante una proyección), en particular para el cálculo vectorial. Realizar este tipo de cálculos en esta unidad reducirá el uso de CPU, se completarán en menor tiempo e incluso con mayor paralelismo (contiene una gran cantidad de unidades de cálculo en coma flotante).

Además, normalmente suele albergar una unidad de almacenamiento temporal (VRAM), para reducir el nº de comunicaciones a través del bus de datos (que es extremadamente lento).

Definición 4.2. Una **primitiva** es el objeto básico de visualización. Pueden ser puntos, segmentos, patrones de segmentos, polígonos y patrones de polígonos. En nuestro caso las primitivas a usar serán triángulos.

Definición 4.3. Entenderemos por **malla** una malla de triángulos, que es un conjunto de triángulos que aproximan a una cierta superficie. Cuando nos referimos a la **malla inicial**, hablamos de la aproximación del cuadrado $[0,1] \times [0,1] \subset \mathbb{R}^2$ para un cierto número de triángulos (se calcula una cuadrícula y se dividen por las diagonales para obtener los triángulos).

Una malla se dirá que está **indexada** si para definir los triángulos se proporcionan primero los vértices sin repetirlos, y después se definen los triángulos en base a los índices de los vértices. Por ejemplo, para un cuadrado tendríamos los vértices $(0,0)$ $(1,0)$ $(1,1)$ $(0,1)$, si los numeramos del 0 al 3, el triángulo de la diagonal inferior se definiría como $(0,1,3)$, y el superior $(1,2,3)$. Esta técnica reduce el número de vértices a almacenar y procesar, ya que para el caso normal se deberían definir los vértices para cada triángulo a dibujar.

Definición 4.4. La **representación de fronteras** (boundary representation o B-rep) consiste en la aproximación de un cuerpo topológico o geométrico mediante una malla. De esta manera se aproximan las posibles caras del objeto, pero para ello hay que tener en cuenta la posibilidad de definir una orientación, necesaria para saber cuál es la parte interior y

la exterior de la cara. Aquellos objetos que no sean orientables no podrán visualizarse correctamente en cuanto a iluminación, ya que sus caras interior y exterior coinciden (deberán aproximarse con más de una representación).

Definición 4.5. La acción **teselar** consiste en subdividir una primitiva en primitivas más pequeñas, para así poder aproximar mejor la superficie que se quiere representar. En el caso de OpenGL, para realizar la teselación es necesario especificar los respectivos niveles de teselado:

1. El **nivel de teselado en un lado** (outer) es el número de partes en el que se dividirá el lado para el teselado de la primitiva. Por ejemplo, si el nivel es 3 para un cierto lado, dicho lado se dividirá en 3 partes iguales, que serán los lados de los nuevos triángulos.
2. El **nivel de teselado en el interior** (inner) es el número de partes en el que se dividirá la primitiva hacia el interior. Por ejemplo, si el nivel es 3 para una cierta primitiva, dicha primitiva se dividirá en 3 niveles hacia el baricentro.

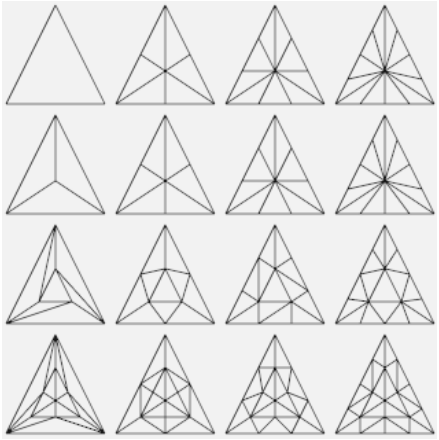


Figura 4.1.: Niveles de teselado del 1 al 4: outer en horizontal (todos los lados por igual) e inner en vertical.

Definición 4.6. Un **shader** es un tipo específico de programa informático que se ejecuta en la GPU. Su uso principal es el cálculo de gráficos. En OpenGL existen los siguientes tipos de shaders:

- Vertex shader: se ejecuta para cada vértice de entrada. Se utiliza para modificar cada vértice.
- Tessellation control shader: tiene de entrada una primitiva y devuelve un patch (conjunto de primitivas), según los niveles de teselado.
- Tessellation evaluation shader: similar al vertex shader pero diseñado para la salida del tessellation control shader, con el objetivo adicional de generar los vértices del teselado mediante coordenadas baricéntricas (proporcionadas por la etapa anterior de forma automática).
- Geometry shader: tiene de entrada una primitiva y puede devolver varias primitivas.

- Fragment shader: después del rasterizado (pasar de gráfico vectorial a píxels), se ejecuta para calcular el color de cada fragmento de triángulo.
- Compute shader: etapa utilizada para realizar cáculos en general. No suele usarse para el renderizado en sí (no se ha utilizado en el proyecto).

Definición 4.7. La **curvatura de Gauss** (K) es una función que a cada punto de una superficie S le asigna un valor de $\mathbb{R}$. Indica el tipo de geometría entorno al punto:

- $K_p = 0$, euclídea, localmente es $\mathbb{R}^2$.
- $K_p > 0$, esférica, localmente es una esfera de radio $R = \frac{1}{\sqrt{K_p}}$.
- $K_p < 0$, hiperbólica, localmente es un espacio hiperbólico de curvatura K_p.

La curvatura de Gauss se ha calculado de la siguiente forma [Wol]:

$$K_{x(u,v)} = \frac{det(x_{uu}, x_u, x_v)det(x_{vv}, x_u, x_v) - [det(x_{uv}, x_u, x_v)]^2}{[|x_u|^2|x_v|^2 - < x_u, x_v >^2]^2}(u, v)$$

que es la curvatura de Gauss en el punto $x(u, v)$, con x carta de la superficie S. Existen muchas maneras distintas de calcularla, pero esta es la que nos favorecerá para su correcta obtención mediante los árboles de expresión.

5. Estudio de la teselación

En este capítulo se pretende diseñar un algoritmo de teselación capaz de aproximar la superficie con cierto nivel de precisión sin utilizar un número elevado de triángulos.

Sin utilizar teselado cualquier superficie se puede representar correctamente pero haciendo uso de una gran cantidad de triángulos pequeños (incluso del orden del millón de triángulos). En dicho caso, se eleva el coste en tiempo y memoria, desaprovechando tales recursos para zonas en las que no se requiera un gran número de primitivas (zonas con poca curvatura de Gauss, en valor absoluto).

El objetivo es evitar dicho caso, es decir, implementar un algoritmo que calcule para cada zona qué resolución es la adecuada para un cierto valor de precisión, partiendo de una malla inicial y subdividiendo dichos triángulos. Para ello se utilizarán las expresiones analíticas de la curvatura de Gauss, proporcionadas por el programa "Procesador" mediante la construcción de árboles de expresión y los derivados respecto ciertas variables.

Para poder representar una zona con un nivel de precisión $\epsilon > 0$ será necesario previamente definir cuándo una malla indexada aproxima a una superficie con dicho nivel de precisión. Se puede entender ϵ como el error que estamos asumiendo para la representación de dicha superficie. Este valor será el que indique principalmente el nivel de teselado a realizar en la zona de la malla en la que se esté trabajando.

Esta técnica se implementará en shaders, para poder ejecutarlo directamente en la GPU, que ofrece mayor rendimiento que la CPU para dicho problema. Es por ello que el programa "Procesador" genera código GLSL (a partir de un lenguaje funcional propio para definir las cartas) para que las funciones asociadas a la parametrización se usen directamente en la GPU, siendo compiladas previamente.

En un principio se iba a realizar en un Geometry Shader pero más tarde se observó que era más acertado el uso de un Tessellation Shader. A parte de estos shaders (Tessellation Control shader y Tessellation Evaluation shader), he utilizado los usuales, Vertex shader y Fragment shader (para dar color), y el Geometry shader para mostrar los vectores de los vértices de la malla, ya que era idóneo para representarlos mediante líneas y en una única llamada por vértice se podían generar hasta los 3 vectores típicos a la vez: tangente, bitangente y normal.

Cabe destacar antes que la idea inicial era desarrollar un algoritmo de división recursiva de los triángulos, el cuál se detendría en el nivel en el que se crea que representa correctamente a la porción de superficie. Sin embargo, el lenguaje GLSL no permite realizar llamadas recursivas, por lo que ha sido necesario buscar alternativas.

5.1. Medidas según la definición

A continuación se desglosan todas las medidas estudiadas, atendiendo a las diferentes definiciones de "buena aproximación a una superficie".

Medida basada en el volumen

Consiste en estimar el volumen de la diferencia entre la malla indexada generada y la superficie original a nivel local. Esta medida está asociada al primer concepto de superficie bien representada:

Definición 5.1 (Superficie bien representada 1)**.** Dada una superficie S y una malla P que la aproxima, se dice que la representa con una precisión de $\epsilon > 0$ si el volumen contenido entre ambos es menor que ϵ.

La medida equivalente para los lados es el área de la sección cuya base es el lado del triángulo y el borde restante es la curva original a aproximar con dicho lado.

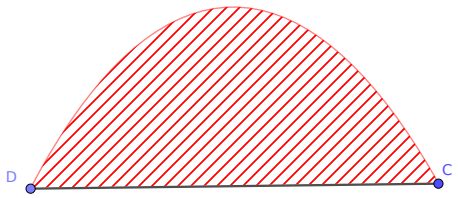


Figura 5.1.: Medida en un lado DC

Sin embargo, para las superficies no compactas esta definición puede no permitir la existencia de un P que la aproxime con una precisión finita.

Además, aparece el problema de la no detección de picos, debido a que al estudiar el volumen, si la altura es grande y la base lo suficientemente pequeña se puede dar el caso de que el volumen quede por debajo de la precisión ϵ y no tesele, aun existiendo dicho pico.

Figura 5.2.: Pico no teselado

Medida basada en el área

Consiste en estimar el área de la superficie original localmente. Esta medida está asociada al segundo concepto de superficie bien representada:

Definición 5.2 (Superficie bien representada 2)**.** Dada una superficie S y una malla P que la aproxima, se dice que la representa con una precisión de $\epsilon > 0$ si el ratio de área $r = \frac{A(S)}{A(P)}$ cumple que $r - 1 < \epsilon$ (siempre se tiene que $r \geq 1$).

La medida equivalente para los lados es $r = \frac{L(\alpha)}{L(l)}$ donde L es la longitud, l el lado del triángulo y α la curva a estimar.

Es una buena alternativa para poder detectar dichos picos, ya que cuando hay uno o varios picos mal aproximados el área original entorno al pico es mucho mayor que la de la superficie generada (el ratio es grande). Además, de esta forma estamos evitando la aparición del problema del farolillo de Schwarz [Epp] debido a que buscamos estimar con una cierta precisión el área original.

Sin embargo, al usar esta medida asumimos que la parametrización a nivel local funciona como una gráfica, como por ejemplo $p(u,v) = (p_x(u,v), p_y(u,v), p_z(u,v)) = (u, v, p_z(u,v))$. Esto no es cierto en la gran mayoría de casos y se puede ver fácilmente con la siguiente parametrización del plano $[0,1]x[0,1]$, embebido en $\mathbb{R}^3$:

$$p(u,v) = (cos(\frac{\pi}{2}u), sin(\frac{\pi}{2}v), 0)$$

Al ser un plano no debería de teselar, pero como detecta que la curva a estimar en el lado del triángulo no está bien aproximada, entonces subdivide:

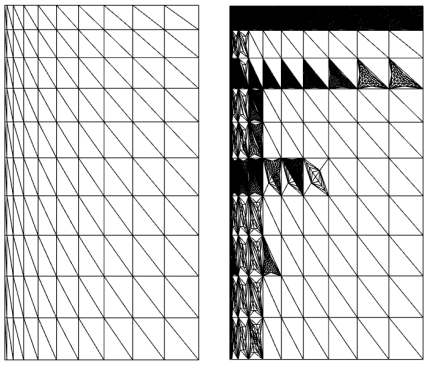


Figura 5.3.: Teselación innecesaria del plano

Medida basada en la curvatura

Consiste en estimar la curvatura de Gauss por área. Esta medida está asociada al tercer concepto de superficie bien representada:

Definición 5.3 (Superficie bien representada 3)**.** Dada una superficie S y una malla P que la aproxima, se dice que la representa con una precisión de $\epsilon > 0$ si para todo triángulo T de la malla se cumple que $K_T A(T) < \epsilon$, donde K_T es el máximo valor de la curvatura de Gauss en valor absoluto en T.

La medida equivalente para los lados es $K_l L(l)$ donde L es la longitud y l el lado del triángulo.

Si utilizamos como medida sólo la curvatura, que no depende de la parametrización de la superficie, teselaríamos de igual forma en un triángulo grande T_1 y en uno pequeño T_2 si $K_{T_1} = K_{T_2}$, obteniendo subtriángulos distintos. Esto se debe a que la parametrización elegida deforma la malla inicial cambiando de forma irregular el tamaño de los triángulos. Un claro ejemplo es la reducción de la base de los triángulos a medida que nos acercamos a los polos en una esfera con la parametrización usual. Se puede observar en la siguiente imágen, donde el color indica el área diferencial de la parametrización.

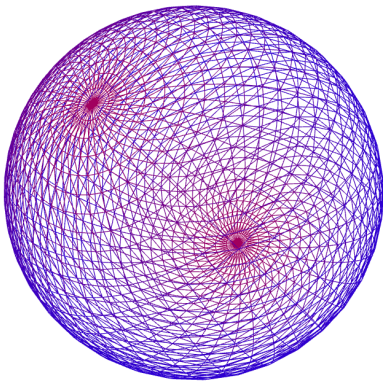


Figura 5.4.: Deformación heterogénea de la malla

Para evitarlo, multiplicamos por el área del triángulo, así que para el caso anterior el triángulo T_1 tendría un valor de dicha medida mayor que el triángulo T_2, por lo que sería necesario teselar más.

También tiene un punto negativo, y es que genera una excesiva teselación en zonas con una cruvatura de Gauss muy elevada (en valor absoluto), por lo que puede crear efectos antinaturales si hay zonas con poca curvatura cerca, ya que hay un cambio brusco de teselado:

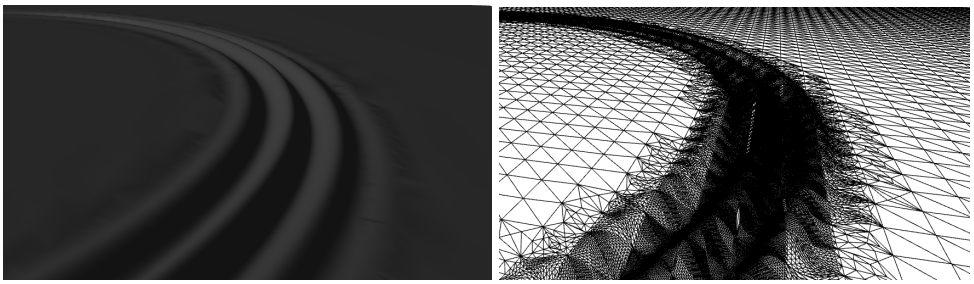


Figura 5.5.: Teselado basado en K.

Esto se puede corregir elevando el valor absoluto de la curvatura de Gauss a un exponente $\alpha > 0$, en el que si $\alpha < 1$ la teselación será más uniforme, manteniendo sin teselar las zonas planas, y si $\alpha \geq 1$ se le dará más inportancia a las zonas curvas. En la práctica se recomienda usar $0.1 < \alpha < 0.5$ para un buen equilibrio calidad/rendimiento.

Las siguientes imágenes muestran el uso de esta funcionalidad, requeriendo el mismo número de triángulos que en las figuras anteriores:

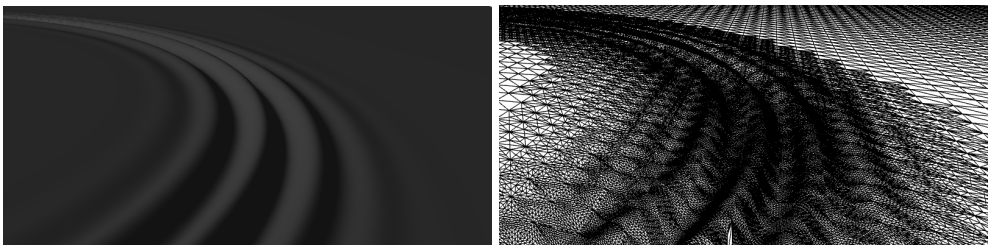


Figura 5.6.: Teselado con exponente α.

5.2. Mejora del teselado

En esta sección vamos a estudiar cómo mejorar el proceso de teselado para mostrar buenos resultados sin tener un gran impacto en el rendimiento. En general, nuestro cerebro reconstruye el objeto visualizado atendiendo a los bordes detectados y a las texturas, además de obviar los elementos que están fuera del campo de visión y aquellos ocultos. La siguiente comparación es un buen ejemplo para corroborarlo:

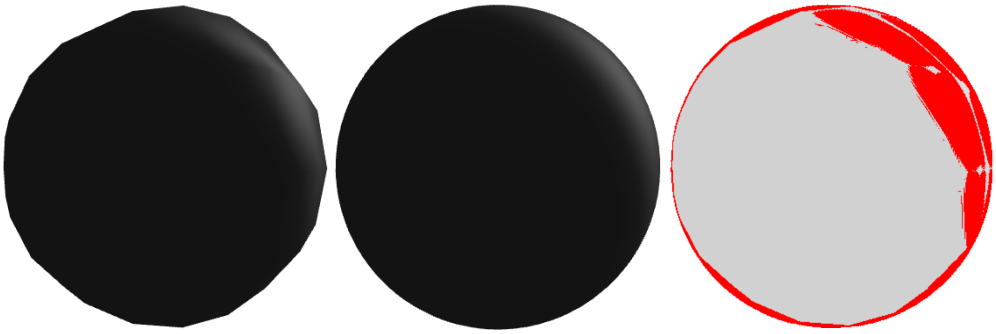


Figura 5.7.: Calidad visual según los bordes y texturas

La primera figura está aproximada con una cantidad muy inferior de primitivas con respecto a la segunda, y la última es la diferencia entre las dos primeras, para así ver la importancia de la correcta representación de bordes y texturas. Para el caso de la visualización de superficies tenemos que:

1. Visualizar una superficie a diferentes **distancias** equivale a visualizar la misma superficie pero con distinto tamaño, es por ello que el umbral utilizado para la definición de buena aproximación puede ser relativo a la distancia de la cámara al triángulo visualizado. De esta forma, para zonas con características similares y a diferentes distancias, el teselado será mayor en las que estén más cerca de la cámara, funcionando de manera similar a un gráfico vectorial escalable (SVG), pero en $\mathbb{R}^3$. Además, podemos complementarlo añadiendo una distancia máxima de teselado, para evitar teselar zonas alejadas en superficies aparentemente no compactas.

2. Los únicos **bordes** posibles son aquellas zonas donde el vector de visualización del punto (con origen la cámara y destino el punto a comprobar, normalizado) es perpendicular a la normal en el punto. Si reducimos el nivel de teselado en zonas que no sean bordes, teniendo cuidado en las zonas con mayor iluminación, mantedremos la misma calidad visual reduciendo el número de primitivas usadas. En la práctica se ha utilizado un factor de reducción de $\frac{2}{3}$.

3. Ver la **textura** de una superficie equivale a ver cómo afecta la iluminación a la misma, por lo que en nuestro caso una zona con textura visible es aquella donde el factor de iluminación difusa es mayor que 0. Además, podemos reducir el nivel de teselado como en el apartado anterior en caso de que la iluminación sea tenue, es decir, mayor que 0 pero menor que un umbral $\alpha > 0$ (en la práctica se ha utilizado $\alpha = 0.5$). El factor de iluminación difusa se calcula de la siguiente forma:

$$< N(u,v), \frac{P_L - p(u,v)}{|P_L - p(u,v)|} >$$

   Con p carta, $N(u,v)$ la normal en $p(u,v)$ y P_L la posición de la luz actual.

4. Un elemento está **fuera del campo de visión** si el vector frontal de la cámara y el vector de visualización del punto cumplen ciertas restricciones, cuya complejidad dependerá de la definición de campo de visión escogida. Simplificando el problema, podemos decir que esto sucede si su producto escalar es cercano al 0, bajo un cierto umbral $\delta > 0$ que definirá el campo de visión con forma de cono. Dado un ángulo de visión α, el umbral se puede calcular como $cos(\frac{\alpha}{2})$, obtenido a partir de la expresión del producto escalar siguiente, donde p es el vector frontal de la cámara y q el de visualización del punto (normalizados):

$$< p,q >= |p||q|cos(\angle(p,q)) = cos(\angle(p,q))$$

   Para tener una visión de 75º sería necesario un umbral de 0.793 aproximadamente. En la prácitca se calcula automáticamente según el FoV en el eje X y el eje Y (se calcula el FoV de la diagonal, para encerrar la "pirámide de visión" en un "cono de visión" mayor).

5. Si la superficie es orientable y su complemento tiene 2 componentes conexas, entonces el punto p está **oculto por superposición** si el producto escalar del vector de visualización de p con la normal en p es mayor que 0. Para evitar el solapamiento con la definición de borde, se puede proporcionar un umbral $\epsilon > 0$, comprobando si dicho producto es mayor que ϵ en vez del 0.

Aplicando los cuatro primeros apartados podemos mejorar el rendimiento del programa para cualquier superficie, independientemente de sus características topológicas y geométricas.

En cuanto al último apartado, el aumento de rendimiento puede ser bastante grande según el tipo de superficie, pero se trata de una situación más específica y no siempre se va a poder utilizar. Es por ello que en el programa se da a elegir al usuario si quiere activar o no dicha funcionalidad (desactivada por defecto).

También se intentó implementar un post-procesado de los vértices, moviéndolos según la función de curvatura de Gauss (K), para reposicionar los vérices en zonas de las superficies críticas (extremos de la función K), pero su cálculo era costoso y al realizarse para cada vértice se generaba un gran impacto en el rendimiento. Además, los resultados obtenidos no eran muy diferentes al mismo teselado sin dicha funcionalidad, aunque puede ser interesante para otro tipo de problemas (se comentará en las **Conclusiones y vías futuras**).

5.3. Estimación de las medidas

Debido al cálculo costoso que conlleva la obtención de dichas medidas en el interior del triángulo, el problema se ha reducido a calcular el nivel de teselado en las medianas del triángulo y posteriormente tomando el valor medio, para así evitar grandes diferencias de teselado con respecto a los lados del triángulo. Además, esto facilita compartir ciertos cálculos con los lados y así mejorar el rendimiento.

Para calcular la curvatura de Gauss por área y la iluminación en un segmento es necesaria una aproximación puesto que dichos valores normalmente no serán constantes. Para ello realizamos muestras equidistantes en el segmento a estudiar, de forma determinista para que triángulos adyacentes tengan el mismo nivel de teselado en el lado compartido.

Una vez tomadas las muestras, calculamos el valor de la curvatura de Gauss en valor absoluto y del producto escalar para la iluminación difusa y nos quedamos con el máximo para cada uno, contemplando así el peor caso posible. Se ha utilizado el máximo en vez de cualquier otra función para así detectar mejor la presencia de “picos” y otras zonas extremadamente curvadas.

6. Procesador

En este capítulo nos centramos en el diseño de un lenguaje sencillo para representar las cartas que definen a la superficie, que es un punto clave de la aplicación.

El procesador del lenguje tendrá como salida lenguaje GLSL para compilarlo como un shader y así transformar la malla inicial (triangulación del conjunto $[0,1] \times [0,1] \subset \mathbb{R}^2$), dando forma a la superficie. Paralelamente construirá el árbol de expresión de cada parametrización, permitiendo el cálculo de las derivadas parciales, necesarias para la obtención de las normales y la curvatura de Gauss.

A continuación se muestra un código de ejemplo que ilustrará la forma de una parametrización utilizando el lenguaje deseado:

```
//– Ejemplo para la parametrización de la esfera –//
// Definir las costantes (PI es una cte predefinida)
PI2 : real = 2*PI;

// Definir funciones adicionales (sin y cos están predefinidas)
compx(u, v : real) : real = cos(v*PI)*cos(u*PI2);
compy(u, v : real) : real = cos(v*PI)*sin(u*PI2);
compz(u, v : real) : real = sin(v*PI);

// Definir la función principal
f(u, v : real) : vec3 = vec3(compx(u,v), compy(u,v), compz(u,v));

// Usar otra función para redefinir el dominio, en vez del cuadrado [0,1] × [0,1]
g(u, v : real, t0, t1 : real) : vec3 = f(t0 * (u-0.5), t1 * (v-0.5));

// Para pintar 'g', debe devolver un tipo 'vec3' y los dos primeros argumentos
// deben ser reales (u y v). El resto serán parámetros de tiempo (para homotopías)
plot g;
```

Especificación BNF

Se describirá en esta sección de forma rigurosa el lenguaje que recibirá como entrada el programa "procesador":

```
<Programa>                ::=  <lista_sentecias><sentencia_plot>;

<lista_sentecias>         ::=  <lista_sentecias><sentencia>;
                           |   <sentencia>;

<sentencia>               ::=  <sentencia_declar_valor>
                           |   <sentencia_declar_fun>

<sentencia_declar_valor>  ::=  <identificador>: <identificador>= <expresion>

<sentencia_declar_fun>    ::=  <identificador>(<lista_param>) : <identificador>= <expresion>

<sentencia_plot>          ::=  plot <lista_ident>

<expresion>               ::=  (<expresion>)
                           |   if <expresion>then <expresion>else <expresion>
                           |   <expresion><op_binario><expresion>
                           |   <op_unario><expresion>
                           |   <identificador>(<lista_expresiones>)
                           |   <expresion>[<expresion>]
                           |   <identificador>
                           |   <constante>

<op_binario>              ::=  + | - | * | / | and | or | >| <| >= | <= | == | != | ^

<op_unario>               ::=  - | !

<lista_expresiones>       ::=  <lista_expresiones>,<expresion>
                           |   <expresion>

<lista_param>             ::=  <lista_param>, <lista_ident>: <identificador>
                           |   <lista_ident>: <identificador>

<lista_ident>             ::=  <lista_ident>, <identificador>
                           |   <identificador>

<identificador>           ::=  <letra><cadena>
                           |   <letra>

<cadena>                  ::=  <cadena><letra>
                           |   <cadena><numero>
                           |   <letra>
                           |   <numero>
```

<constante>	::=	<numero_entero>
	|	<numero_real>
	|	<bool>
<numero_real>	::=	<numero_entero>
	|	<numero_entero>.<numero_entero>
<numero_entero>	::=	<numero_entero><numero>
	|	<numero>
<bool>	::=	true | false
<letra>	::=	a | ... | z | A | ... | Z
<numero>	::=	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Además, está permitido llamar a la derivada parcial de una función ya definida. Por ejemplo, supongamos que hemos definido "f(u,v : real) : vec3", entonces podemos definir después "g(u,v : real) : vec3 = fPuPv(u,v);", donde "Pu" significa parcial respecto de la variable "u" y "Pv" la parcial respecto de "v" (conmutativo). Por tanto, podemos utilizar como llamada a una función el identificador "<f><sucesión de Px_i>", con "<f>" función base ya definida y "x_i" variables suyas.

7. Planificación y presupuesto

7.1. Planificación temporal inicial

<table>
<tr><td colspan="4">Planificación temporal en fases</td></tr>
<tr><td>Nº</td><td>Nombre</td><td colspan="2">Tareas</td></tr>
<tr><td>1</td><td>Inicial</td><td colspan="2">- Estudio inicial del problema</td></tr>
<tr><td>2</td><td>Implementación básica</td><td>Procesador:

- Definición del lenguaje (BNF)

- Implementar primera versión del procesador a partir del código de la asignatura Procesadores de Lenguajes, que sólo detecta errores.
- Complementar el procesador para traducir a código GLSL (generación de árbol sintáctico), para usarlo en el vertex shader.</td><td>Visualizador:

- Toma de contacto con OpenGL, el lenguaje GLSL e ImGui.
- Construir un programa de visualización a partir de código base, haciendo uso de los shaders básicos. La normal será la del triángulo.</td></tr>
<tr><td>3</td><td>Conexión inicial</td><td colspan="2">- Conectar apropiadamente los programas, para visualizar la superficie definida con la parametrización.
- Estudiar el correcto funcionamiento de ambas aplicaciones hasta ahora. Corregir en caso de ser necesario.</td></tr>
<tr><td>4</td><td>Completar procesador</td><td colspan="2">- Implementar los algoritmos que calculan los árboles de las derivadas parciales de las expresiones. Añadir seguidamente la generación de funciones típicas, como la normal, el área diferencial y la curvatura de Gauss.</td></tr>
<tr><td>5</td><td>Conexión avanzada</td><td colspan="2">- Utilizar la función normal en los puntos, en vez de las normales de los triángulos. Además, mostrar la normal como un segmento de color distinto al de la superficie.
- Mostrar mediante colores el valor del área diferencial y de la curvatura de Gauss en cada punto.
- Estudiar el correcto funcionamiento de ambas aplicaciones hasta ahora. Corregir en caso de ser necesario.</td></tr>
<tr><td>6</td><td>Implementación avanzada</td><td colspan="2">- Implementar la teselación uniforme mediante el uso del geometry shader.
- Completar con una gran variedad de ejemplos.
- Estudiar el correcto funcionamiento del teselado.</td></tr>
<tr><td>7</td><td>Estudio de la mejor medida</td><td colspan="2">- Elegir ciertas estadísticas del renderizado para comprobar si los resultados son buenos.
- Estudiar varias medidas para controlar el nivel de teselado.</td></tr>
<tr><td>8</td><td>Estética</td><td colspan="2">- Completar la interfaz del usuario.</td></tr>
<tr><td>9</td><td>Revisión</td><td colspan="2">- Estudiar el correcto funcionamiento de ambas aplicaciones hasta ahora. Corregir y añadir elementos si fuese necesario.</td></tr>
</table>

7.2. Diferencias con la planificación real

- Después de implementar el algoritmo de derivación (fase 4), fue necesario añadir un algoritmo de simplificación de expresiones, para evitar cálculos triviales (sumar/multiplicar 0 y multiplicar/dividir por 1).
- Tras el estudio del geometry shader (fase 6), se observó que no era tan adecuado para el proyecto como el propio tessellation shader, así que se inició una nueva fase de estudio para dicho shader.
- Durante la fase de estudio de la mejor medida (fase 7), por la naturaleza del procesador del lenguaje, este era fácilmente adaptable para permitir llamadas a derivadas parciales dentro del lenguaje (de funciones ya definidas). Es por ello que se desarrolló paralelamente esta funcionalidad.
- Finalmente fue necesario introducir una fase de optimización de código, en especial para los shaders (GLSL) para sacar el máximo partido al cáculo vectorial y evitar realizar cálculos repetitivos en la GPU.

7.3. Presupuesto

El presupuesto se ha calculado en base al sueldo medio de un ingeniero informático, junto con el coste de desarrollo temporal del proyecto. No se añaden gastos de licencias de software puesto que el software es libre en su totalidad (ImGui y Mesa, similar a OpenGL).

Sueldo medio	18000 €/año
Horas anuales	1800 h/año
Precio de la hora	10 €/h

Tiempo estimado	60 días
Horas al día	6 h/día
Total horas	360 h
Presupuesto horas	3600 €

Aunque el hardware usado estaba ya adquirido, se incluirá como posible gasto extra:

Precio actual Lenovo v110-15isk 80tl 8 GB RAM, 500 GB HDD	460 €
Gasto estimado de material (electricidad, desgaste, etc)	40 €
Total presupuesto	**4100** €

8. Análisis y diseño

En este capítulo se especificará toda aquella información referente a la estructura del programa y los requisitos del mismo, aunque está mayormente enfocado a la definición de los algoritmos desarrollados.

8.1. Especificación de requisitos

Requisitos funcionales

1. Se podrán visualizar varias parametrizaciones a la vez, donde cada una representará una carta de una superficie específica, con el objetivo de representar homotopías e isotopías entre superficies.
   a) El sistema debe permitir visualizar cualquier parametrización que se le indique, siempre que cumpla con la estructura del lenguaje definido.
   b) Cada parametrización podrá admitir parámetros de "tiempo", para así modificar la porción de superficie que representa y así poder visualizar homotopías e isotopías.
2. El programa contará con una interfaz clara, sencilla y completa.
   a) El usuario tendrá la posibilidad de indicar manualmente los parámetros adicionales de las cartas (t_i). Además se incluirá la opción de que cada t_i se mueva de forma automática, para así generar animaciones fluidas.
   b) El usuario podrá indicar ciertos parámetros del cálculo de la malla de la superficie, como:
      1) El tamaño de la malla inicial con la que se visualizará cada carta de la superficie.
      2) La precisión con la que se quiere representar la superficie actual.
   c) El usuario podrá indicar si quiere visualizar ciertos atributos de la superficie, como:
      1) La curvatura de Gauss, asignando un color para la curvatura negativa y otro para la positiva, dependiendo de un parámetro de escala para resaltar las zonas.
      2) El área diferencial de la parametrización, junto con un umbral y un factor de escala que se podrán modificar.
      3) Se podrán visualizar los vectores tangente, bitangente y normal de cada vértice generado.
   d) El usuario tendrá la posibilidad de modificar los valores referentes a la iluminación.
      1) Los coeficientes del modelo de iluminación Phong.

2) El color del fondo de la escena y el color base del objeto visualizado.

3. El programa no renderizará nuevos frames si no se requieren nuevos cálculos, es decir, si no se detectan cambios en la entrada y la escena está estática.

Requisitos no funcionales

1. El programa debe renderizar las superficies con un tiempo de respuesta bajo, pensando en dispositivos con una GPU común, como por ejemplo una gráfica integrada.
   a) El programa adaptará su rendimiento según el estado del propio programa.

8.2. Diagramas

En esta sección se ilustran ciertos diagramas que facilitan el entendimiento del programa. Los siguientes diagramas están relacionados con la acción "compilar", que es la que implica mayor comunicación entre entidades, donde cabe destacar que "lastParam.in" es donde se encuentra la parametrización de la superficie en texto, "error.log" es donde se almacena la salida de error, "temp" donde se indican el nº de cartas a visualizar (y el nº máximo de parámetros) y "functions.s" el código GLSL para los shaders.

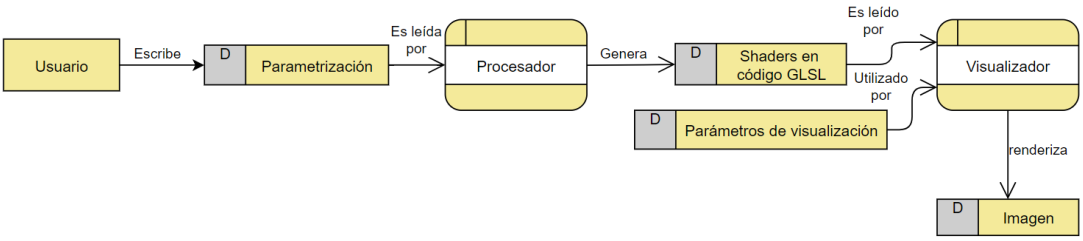


Figura 8.1.: Diagrama de flujo de datos tras ejecutar la acción "compilar".

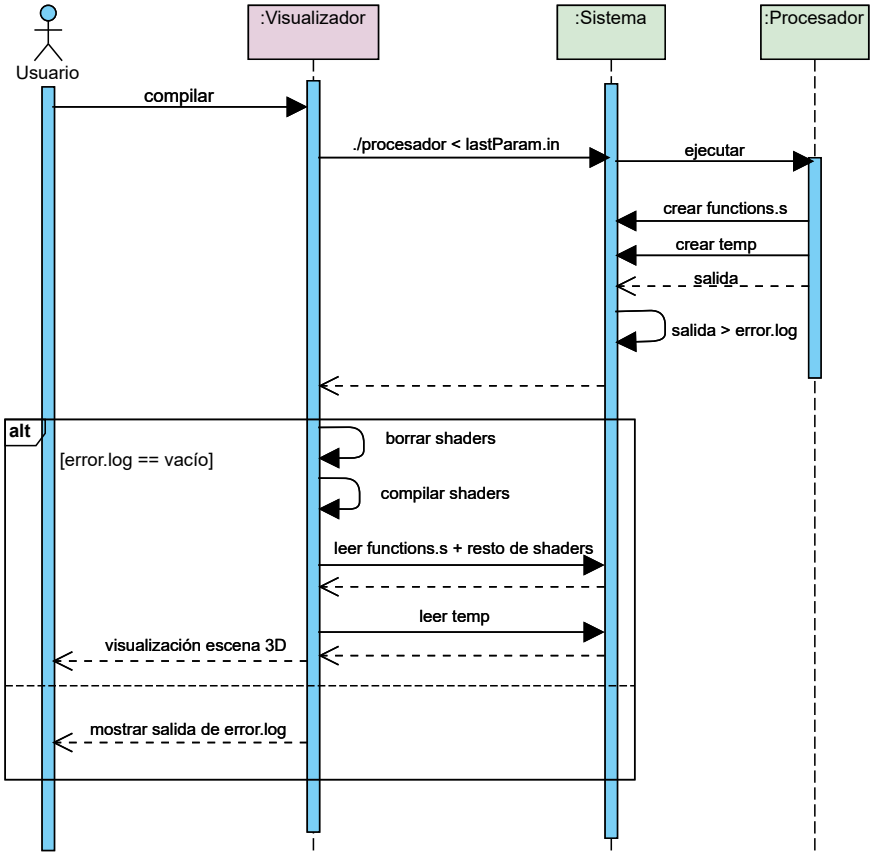


Figura 8.2.: Diagrama de interacción de la acción “compilar”.

Además se muestran los bocetos iniciales de la interfaz de usuario, incluyendo únicamente las agrupaciones de elementos, ya que la distribución de botones y otras entidades se desconocía a priori (no se sabía cuáles serían necesarios ni en qué forma, hasta un punto más avanzado del proyecto).

Figura 8.3.: Boceto de la interfaz con los elementos principales.

Figura 8.4.: Boceto de la ventana de edición de código.

Cabe destacar que al final la interfaz en su totalidad está compuesta por ventanas móviles y contraíbles, aportando mayor flexibilidad y comodidad al usuario, sobretodo si quiere ver la escena en pantalla completa.

8.3. Principales estructuras de datos

La estructura de datos más interesante y que destaca fuera de lo normal es la de nodo de expresión (nodo), utilizada en el programa "procesador" para representar los árboles de expresión. Es un "struct" con los siguientes componentes:

- char *lex; representa el lexema del nodo: el nombre de la función a la que llama, el lexema del operador, el nombre de la variable o la cte.
- struct nodo *children[10]; los hijos del nodo (10 máximo, pero ampliable si fuese necesario).
- int nchild; el nº de descendientes que realmente tiene el nodo.
- tipoNodo tipo; indica el tipo de nodo de expresión: cte, variable, operador, función, índice, paréntesis o condicional (if).
- tipoNodoArray subTipo, indica el tipo de dato que maneja el nodo, es decir, si utiliza escalares, vectores o matrices, junto con sus respectivas dimensiones (son siempre cuadradas, por similitud al lenguaje GLSL).

8.4. Principales desarrollos algorítmicos

Aunque en el capítulo 5 de esta segunda parte se ha descrito con detalle el estudio del teselado, que está directamente ligado con el algoritmo de teselación, en esta sección se hablará del resto de desarrollos algorítmicos, los cuales están mayormente relacionados con el procesador del lenguaje (programa "Procesador").

Dicho procesador se ha implementado mediante un generador de analizadores léxicos tipo 'lex' (flex) junto con un generador de analizadores sintácticos y semánticos tipo 'yacc' (Bison). La especificación BNF del lenguaje se desarrolla en el capítulo 6 "Procesador". La implementación de las expresiones regulares, la gramática y las acciones que se ejecutan al reducir una regla, se han realizado tal y como se procedió en la asignatura de Procesadores de Lenguajes. No se indicarán explicitamente tales implementaciones puesto que la especificación BNF es suficiente para comprender el lenguaje diseñado.

Generación de árboles de expresión

Se generan de forma natural al realizar el análisis sintáctico con el procesador. Cuando se reduce la expresión por una regla, en dicha regla se crea un nodo de expresión nuevo y se le asigna a la variable a la que se reduce, con los hijos y características correspondientes.

Obtener el árbol de la derivada parcial de una expresión

El software implementado es capaz de hacer derivación simbólica, útil para la generación automática de funciones típicas de una superficie, como la normal o la curvatura de Gauss.

Dado un árbol de expresión y una variable, el cálculo del árbol de la derivada parcial respecto de dicha variable se realiza de manera recursiva, aplicando las reglas usuales de derivación.

El caso más complejo, que cabe destacar, es cuando se trata de una llamada a otra función ya definida por el usuario, cuyos parámetros contienen la variable a derivar. Entonces se aplica la regla de la cadena para derivadas parciales, es decir:

$$\frac{\partial}{\partial u}(f(exp_1, ..., exp_n)) = \sum_{i=1}^{n} \frac{\partial(exp_i)}{\partial u} \frac{\partial f}{\partial x_i}(exp_1, ..., exp_n))$$

Pero es necesario definir la función $\frac{\partial f}{\partial x_i}$ $\forall i$, así que se comprobaría si existen ya dichas funciones y en caso negativo se definirán (se deriva su árbol de expresión asociado y se escribe en una función con nombre fPx_i).

Simplificación de las expresiones

La simplificación de una expresión consiste en quitar paréntesis redundantes y eliminar aquellas operaciones evitables, como sumar 0 o multiplicar por 1. Esto mejora la legibilidad del código de salida.

Dado un árbol de expresión, su simplificación se realiza de forma recursiva. Si denominamos $simplificar(n)$ a la función que devuelve el árbol de expresión simplificado, con n el nodo raíz, entonces el algoritmo es el siguiente:

1. Si n tiene hijos n_i, ejecutar $simplificar(n_i)$.
2. Si n es un nodo de paréntesis y su hijo no es un nodo de operación, sustituye n por su propio hijo (quita los paréntesis).
3. Si n es un nodo de operador y algún hijo es 0 o 1, se dice que puede ser simplificable:
   - Si el operador es unario, entonces:
     - Si el nodo hijo es 0 y el operador es $-$, se sustituye el nodo padre por el nodo hijo (quitar el signo).
     - Si el operador es !, entonces se sustituye el nodo padre por el contrario (aplica la negación).
   - Si el operador es binario, entonces (entendemos $i, j \in \{1, 2\}$, $i \neq j$):
     - Si el nodo hijo n_i es 0 y el operador es $+$, se sustituye el nodo padre por el nodo hijo n_j.
     - Si el nodo hijo n_2 es 0 y el operador es $-$, se sustituye el nodo padre por el nodo hijo n_1.
     - Si el nodo hijo n_1 es 0 y el operador es $-$, se elimina el nodo hijo n_2 (queda como operador unario).
     - Si el nodo hijo n_i es 0 y el operador es $*$, se sustituye el nodo padre por el nodo cte 0.
     - Si el nodo hijo n_i es 1 y el operador es $*$, se sustituye el nodo padre por el nodo hijo n_j.
     - Si el nodo hijo n_1 es 0 y el operador es $/$, se sustituye el nodo padre por el nodo cte 0.

- Si el nodo hijo n_2 es 1 y el operador es /, se sustituye el nodo padre por el nodo hijo n_1.

A continuación se ilustra el árbol asociado a la expresión $u\hat{}2 * v + 1$, junto con el árbol de su derivada parcial respecto de u y su posterior simplificación (sólo teniendo en cuenta los operadores -, +, * y /). La potencia con exponente 1 no se simplifica puesto que no se ha implementado (principalmente porque tras derivar siempre se escribe como un flotante), aunque en un futuro se podría mejorar, junto con técnicas de simplificación más complejas, para así reducir el tiempo de compilación de los shaders y realizar menos cálculos costosos.

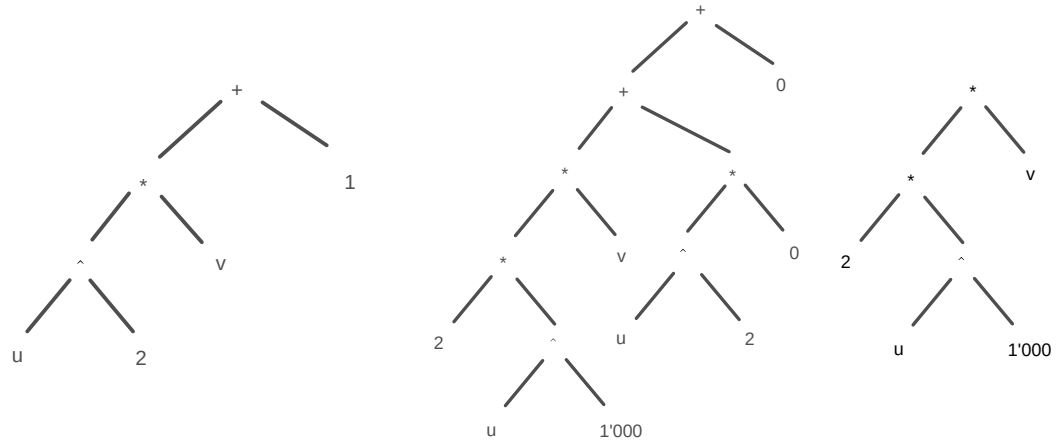


Figura 8.5.: Árbol de la expresión, su parcial respecto u y el mismo pero simplificado.

Auto-umbral

Para mayor comodidad a la hora de estudiar el comportamiento del programa, implementé un algoritmo sencillo para encontrar de manera automática el umbral de la medida para el que se mostrarán n triángulos, donde n es un número natural mayor que el nº de primitivas de la malla inicial. El objetivo era comparar las distintas versiones del programa cuando muestran casi el mismo nº de triángulos.

El funcionamiento del algoritmo es sencillo. Dados los siguientes elementos:

- n_f número objetivo.
- n_0 el número de primitivas actual.
- ϵ_0 el umbral actual.
- $\lambda > 0$ una constante cuya finalidad es similar a un "Learning Rate" o ratio de aprendizaje, es decir, el tamaño de los pasos con los que se aproximará al objetivo.

Entonces el nuevo umbral ϵ se calcula de la siguiente manera:

$$\epsilon = \epsilon_0 + \lambda(\frac{n_0}{n_f} - 1)$$

De esta forma, si el nº objetivo es muy grande, el cociente está muy cercano a 0, por lo que la componente a sumar al umbral ϵ_0 está cerca del $-\lambda$ (reduce enormemente el umbral). Si por el contrario el nº objetivo es muy pequeño, el cociente será mucho más grande que 1 y por tanto el incremento será positivo (el umbral subirá para reducir el nº de subdivisiones). En caso de que el nuevo ϵ esté fuera del intervalo $[\epsilon_{min}, \epsilon_{max}]$, se quedará en el extremo más cercano.

9. Implementación y pruebas

Tipo de shader para el teselado

En esta sección se justificará el uso del tesellation shader para el estudio del teselado de una superficie.

Geometry shader

La ventaja de este tipo de shader es que es muy flexible a la hora de generar nuevas primitivas, ya que permite añadir primitivas totalmente disconexas.

En primer lugar opté por describir de forma explícita un cierto número de niveles de la función recursiva deseada, 3 niveles concretamente. Los resultados eran aceptables pero se podía exceder el límite de vértices, quedando así una superficie incompleta. Además, el tiempo de compilación crecía de forma exponencial a medida que se añadían más niveles (para 2 niveles era de 10-15s y para 3 no concluía).

Puesto que este método era costoso temporalmente y tenía muchas limitaciones, decidí implementar un algoritmo similar pero en un bucle, cuyo esquema es el siguiente:

1. Dado un lado, dividirlo en tantos segmentos como sea necesario, atendiendo a una cierta medida. Dicha medida sólo depende de las características del lado, para que el pegado sea el correcto con los triángulos adyacentes.
2. Se realiza una división hacia el baricentro, de forma similar al punto anterior.
3. Con los vértices de los dos puntos anteriores se genera una malla, es decir, para cada lado, se genera otro lado paralelo para cada subdivisión hacia el baricentro, manteniendo proporcionalmente las subdivisiones del lado original.
4. Se genera una tira de triángulos cuyos vértices sean los de la malla anterior.

Con este método los problemas anteriores se solventan en gran medida, pero dicho algoritmo es semejante al del Tessellation shader, por lo que era natural estudiar el funcionamiento en tal shader.

Finalmente, los inconvenientes observados han sido los siguientes:

- El número de vértices de salida está limitado por una constante predefinida, con valor GL_MAX_GEOMETRY_OUTPUT_VERTICES=256 (depende del hardware), es decir, como mucho se puede devolver una tira de 254 triángulos, independientemente del tamaño del triángulo original.
- El shader tarda en compilarse de media entre 3 y 5 segundos.

Tessellation shader

Este shader tiene un pequeño problema y es que la subdivisión está predefinida (equal, fractional_odd o fractional_even spacing), por lo que los vértices generados en la fase de control del Tessellation shader tienen un esquema fijo, para un número de subdivisiones dado. No es un gran inconveniente puesto que en la fase de evaluación se pueden variar libremente dichos vértices, siempre que se haga con cuidado (en esta fase los vértices se generan mediante coordenadas baricéntricas).

Además, la versión de OpenGL necesaria es superior al del Geometry shader (4.0 frente a 3.2).
Las ventajas con respecto al Geometry shader son las siguientes:

- El shader tarda en compilarse de media menos de $1s$.
- El número de vértices de salida no está tan limitado (GL_MAX_PATCH_VERTICES= 36477 frente a 256).
- Está pensado para realizar directamente el algoritmo de teselación, por lo que no hay que implementarlo.

Al tener implementado el algoritmo de teselación, el estudio se reduce a encontrar una medida que nos indique cómo de buena es la representación de la superficie. Como la teselación de un triángulo se indica por cada lado (outer tessellation factor) y en el interior (inner tessellation), para cada tipo de medida hay que proporcionar una adicional para los lados, para que la teselación sólo dependa de lo que sucede en ellos y de esta forma pegue correctamente con el triángulo adyacente. Dicho estudio ya se ha realizado en el apartado **Estudio de la teselación**, que es el capítulo 5 de la parte II.

Optimización del código GLSL

Puesto que queremos que el programa renderice la escena en tiempo real, es necesario simplificar todo lo posible el código de los shaders, sobre todo del fragment shader (se va a ejecutar para cada vértice generado en el teselado). Para ello se han tenido en cuenta los siguientes hechos (obtenidos principalmente de la Wiki de Khronos [Khr], en la sección GLSL Optimizations):

- Evitar el uso de saltos condicionales y de bucles. En caso de ser necesario algún bucle, usar los de la forma “for(int i=0; i<n; i++)” con n entero constante, para permitir el desenrollado del bucle por el compilador.
- Utilizar “Swizzle” en vez de asignar vectores componente a componente.
- Utilizar “MAD”, es decir, usar siempre que sea posible la multiplicación por el inverso en vez de la división (para aquellos valores “uniform” o constantes) y desarrollar las expresiones como una sumatoria de productos. Por ejemplo, utilizar $a * 0.5 + b * 0.5$ en vez de $(a + b)/2$.
- Utilizar “dot” para agrupar varias expresiones en una. De igual forma utilizar todo el cálculo vectorial y matricial posible para resumir las operaciones.

- Si una expresión es común para un mismo frame, calcularlo en la CPU y asignarlo como otro "uniform" para quitarle carga a la GPU, ya que se ejecutaría para cada primitiva o vértice (dependiendo de en qué shader se realize el cálculo).

Es cierto que el compilador ya realiza de manera automática algunas de estas optimizaciones, pero su puesta en práctica puede reducir ligeramente el tiempo de compilación.

Procesador

Para dicho programa se ha implementado el análisis léxico usando un generador de analizadores de léxico tipo 'lex' (flex), a partir de las expresiones regulares que definen los tokens del lenguaje. También se ha implementado un analizador sintáctico LALR mediante un generador de analizadores sintácticos tipo 'yacc' (Bison), partiendo de la gramática del lenguaje, junto con las acciones semánticas que comprueban dichos errores (los errores semánticos) y generan los árboles de expresión.

Las expresiones regulares, la tabla de tokens y la gramática se han definido a partir de la especificiación BNF, ya citada en el desarrollo del proyecto (parte II, capítulo Procesador). La implementación del "procesador" no se ha realizado desde cero, es decir, se ha partido de una implementación parcial que se aportaba en la asignatura de Procesadores de lenguajes. Dicha implementación, tras ser adaptada, sólo realizaba la traducción directa del código de entrada, junto con la comprobación de errores (léxicos, sintácticos y semánticos). La parte restante corresponde con la generación del árbol de expresión y calcular los árboles derivados, además de adecuar la salida correctamente para que se pueda utilizar como parte de un shader.

Visualizador

En un principio se propuso utilizar como código de partida el proporcionado en la asignatura de Informática Gráfica, pero se observó que tenía muchas funcionalidades que no se llegarían a utilizar y cuya adaptación requeriría más tiempo. Por ello se decidió empezar de cero, pero utilizando como "esqueleto" un código de ejemplo de la referencia Learn OpenGL [dV], que aportaba simplemente la visualización de un objeto en una escena 3D con una fuente de luz y una cámara sencilla.

Interfaz

Para la implementación de la interfaz se ha utilizado la librería de código libre "Dear ImGui" [Cor], junto los complementos "ImGui Quat" [Mor] para visualizar el vector de luz y modificarlo, e "ImGui Dialog" [Cui] para mostrar un diálogo con un explorador de archivos y así poder seleccionar la parametrización deseada.

Generación de informes

Para conocer el rendimiento de la aplicación se han medido los fotogramas generados por segundo y las primitivas que hay presentes en la escena, incluyendo las que son producto del teselado.

La medición del nº de primitivas se ha realizado mediante el uso de la estructura "query"de OpenGL, la cual se actualiza cada vez que renderiza la escena. El nº de fotogramas por segundo se ha calculado manualmente, contando los fotogramas generados en cada segundo.

Además, para mayor comodidad se ha añadido una opción para realizar dichas mediciones de forma automática y escribir los resultados en un archivo cuyo nombre dependerá de la parametrización actual y el modo de visualización. Por defecto realiza 22 medidas (22 segundos). Es ligeramente mayor que 20 para después quitar la primera y/o última medida, ya que están influenciadas por la interacción del usuario con la interfaz.

Estas medidas son suficientes para observar si la aplicación está aprovechando correctamente los recursos para obtener una buena visualización de la superficie, ya que el objetivo es que funcione en tiempo real ofreciendo la mínima carga posible al sistema.

Resultados

En esta sección se mostrarán los datos medidos en la aplicación para comparar las técnicas usadas. Se han seleccionado 2 superficies de las disponibles como ejemplos, “gaussiana.in” y “waves.in”. Los parámetros se han elegido de manera que la visualización sea similar entre los distintos modos de renderizado, minimizando el nº de triángulos de la escena para cada modo, que es el objetivo de estudio: ofrecer un nivel de aproximación específico, maximizando el rendimiento.

gaussiana.in					
	FPS				
Modo	Min	Max	Media	Mejora (fps)	Mejora (%)
No tess.	205	250	231,54	−	−
Normal tess.	253	278	267,63	+36,09	+15,58
Improve1	242	282	266,63	+35,09	+15,15
Improve2	**255**	**284**	**273,72**	**+42.18**	**+18,21**
	Triángulos				
Modo	Min	Max	Media	Mejora (tri.)	Mejora (%)
No tess.	20000	20000	20000,00	−	−
Normal tess.	10547	11802	11165,5	−8834,50	−44,17
Improve1	8267	10232	9100,00	−10900,00	−54,5
Improve2	**6642**	**10120**	**8212,68**	**−11787,31**	**−58,93**

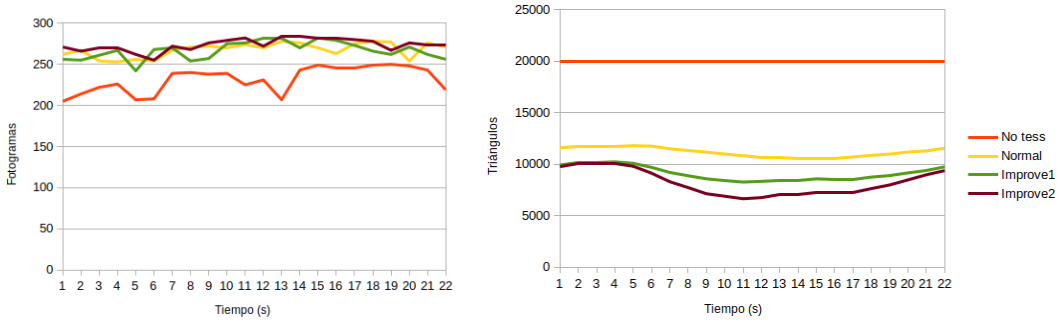


Figura 9.1.: Gráfica de la animación de rotación en gauss.in

waves.in					
	FPS				
Modo	Min	Max	Media	Mejora (fps)	Mejora (%)
No tess.	21	24	21,81	–	–
Normal tess.	53	63	58,27	+36,45	+167,08
Improve1	**63**	72	64,68	+42,86	+196,45
Improve2	62	**87**	**69,86**	**+48.04**	**+220,20**
	Triángulos				
Modo	Min	Max	Media	Mejora (tri.)	Mejora (%)
No tess.	1000000	1000000	1000000,00	–	–
Normal tess.	456480	637537	550527,77	−449472,22	−44,94
Improve1	197244	224222	212327,00	−787673,00	−78,76
Improve2	**194703**	**223824**	**210390,31**	**−789609,68**	**−78,96**

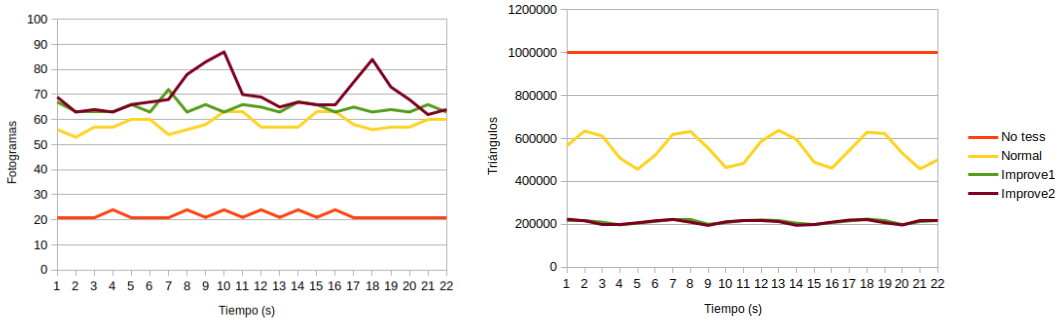


Figura 9.2.: Gráfica de la animación de rotación en waves.in

Las siguientes imágenes corresponden con la parametrización "gaussiana.in" con los distintos modos de visualización, donde para cada modo se ha buscado la configuración óptima, es decir, aquella en la que se ve visualmente bien y el nº de triángulos es mínimo. Por tanto, las 4 imágenes debería ser exactamente iguales en modo relleno, en modo malla se verán las distintas distribuciones de triángulos subyacentes:

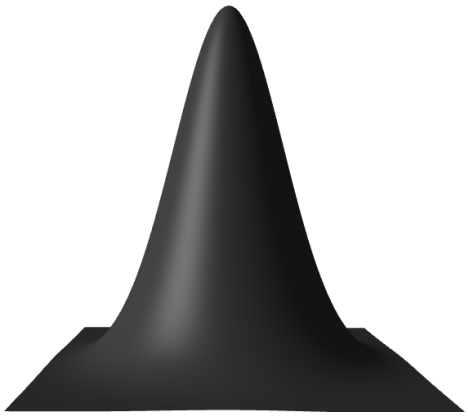


Figura 9.3.: gaussiana.in sin teselar

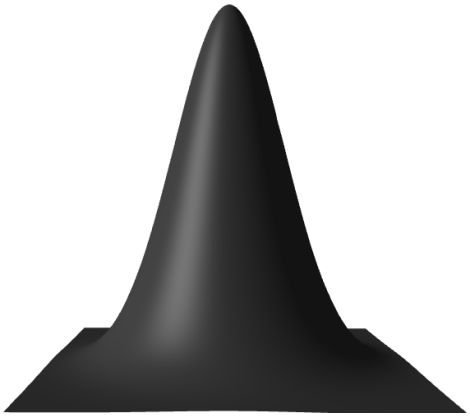


Figura 9.4.: gaussiana.in teselado normal

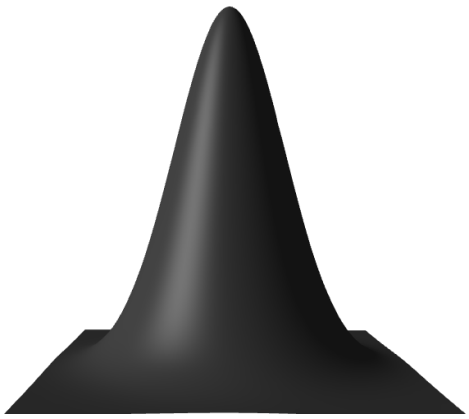


Figura 9.5.: gaussiana.in improve1

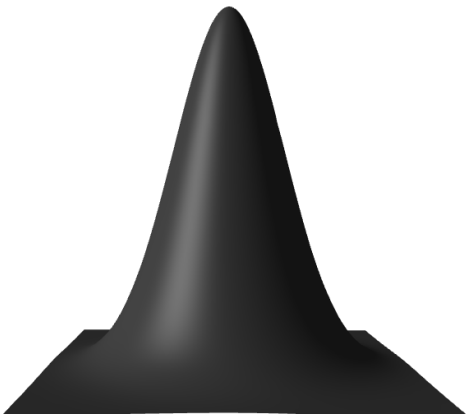


Figura 9.6.: gaussiana.in improve2

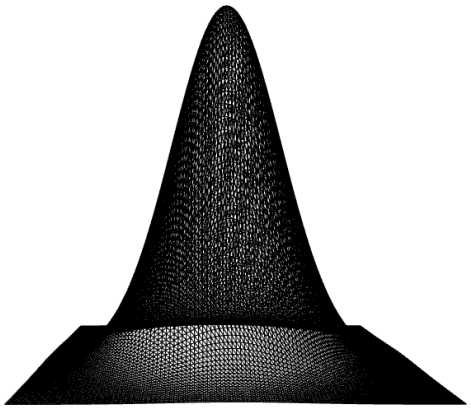


Figura 9.7.: gaussiana.in sin teselar

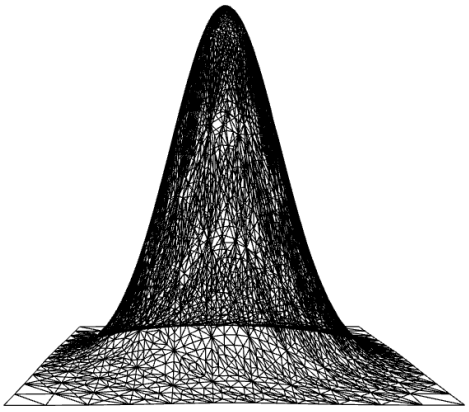


Figura 9.8.: gaussiana.in teselado normal

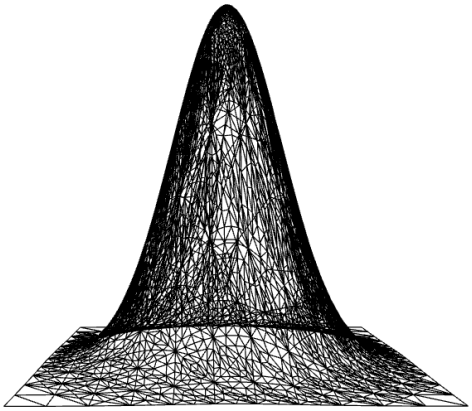


Figura 9.9.: gaussiana.in improve1

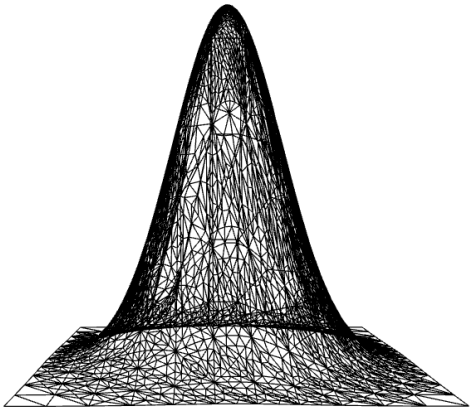


Figura 9.10.: gaussiana.in improve2

Seguidamente se muestran las imagenes correspondientes a la parametrización "waves.in". Sin embargo, debido a la complejidad de la superficie base (plano horizontal perturbado con un coseno), para el modo "sin teselado" es imposible llegar a una buena representación incluso utilizando una malla inicial de 1 millón de triángulos, debido a la aparición de efectos ópticos, por la uniformidad de la malla inicial (parece haber más de un foco mientras que sólo hay uno).

También es posible observar que al pasar de "improve1" a "improve2" puede aparecer algún segmento sin teselar. Se debe a que detecta que es un segmento oculto, mediante un cálculo basado en los extremos del segmento, pero como ya se comentó anteriormente esta funcionalidad está pensada para superficies específicas.

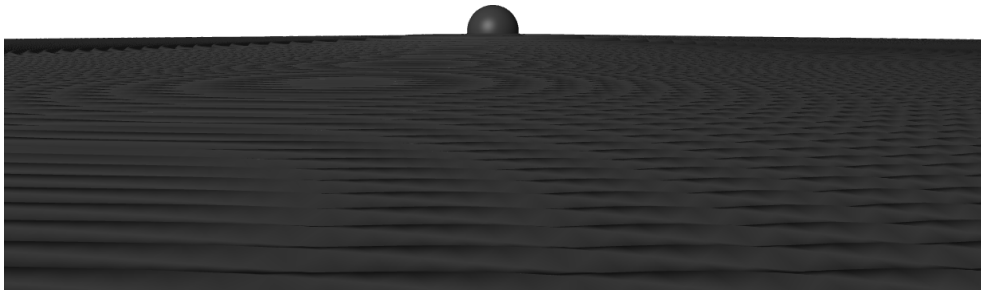


Figura 9.11.: waves.in sin teselar

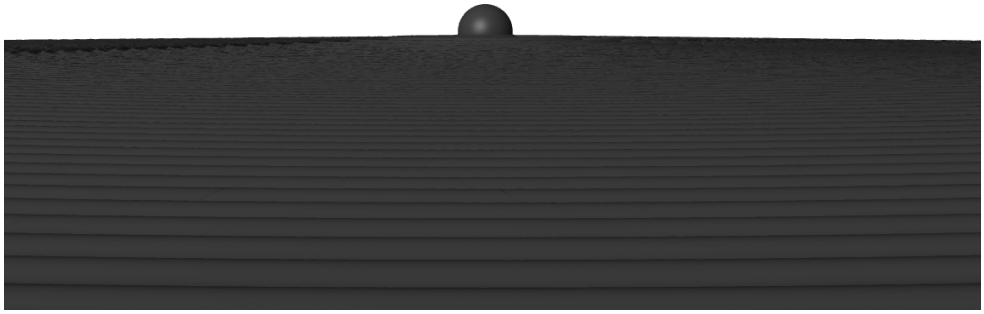


Figura 9.12.: waves.in teselado normal

Figura 9.13.: waves.in improve1

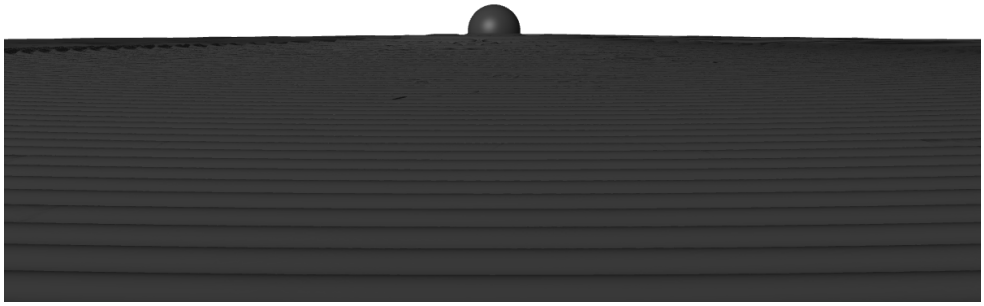


Figura 9.14.: waves.in improve2

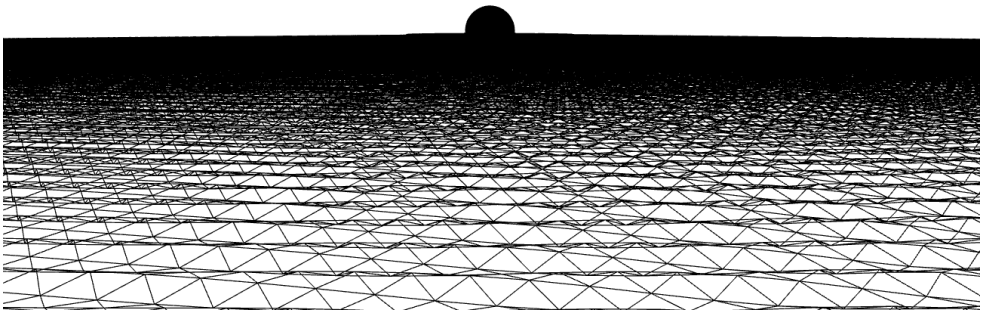


Figura 9.15.: waves.in sin teselar

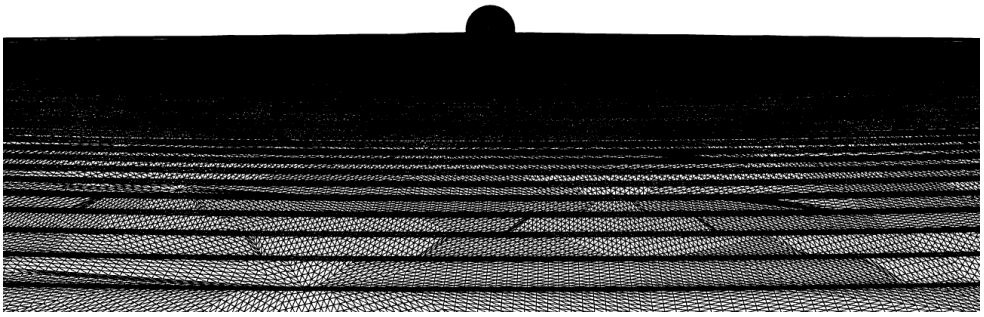


Figura 9.16.: waves.in teselado normal

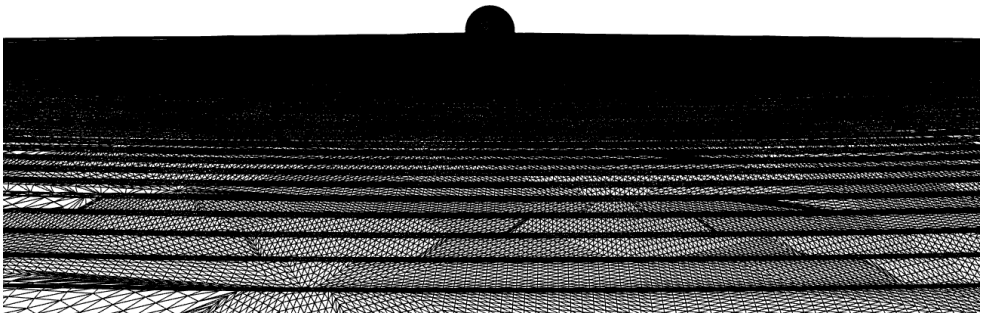


Figura 9.17.: waves.in improve1

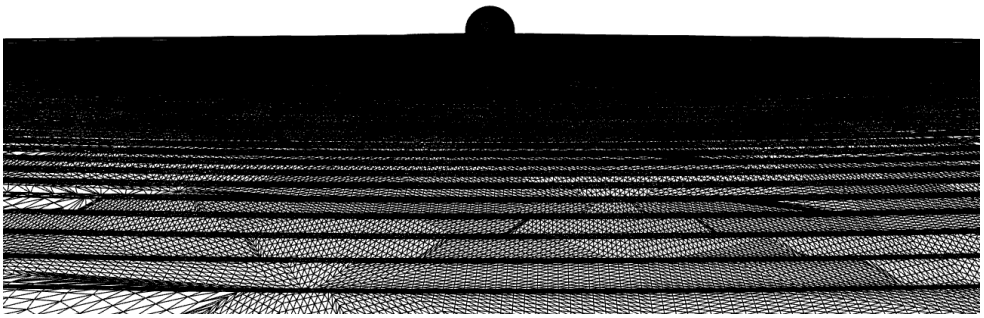


Figura 9.18.: waves.in improve2

Conclusiones y vías futuras

Conclusiones

Con este trabajo se ha intentado transmitir la importancia del teorema sobre variedades topológicas 2-dimensionales, comprendiendo la dificultad que conlleva su demostración y la gran cantidad de elementos utilizados de la teoría de Morse.

Se ha desgranado la demostración aportada por Allen Hatcher en su artículo "The Kirby Torus Trick for Surfaces", de manera que esté al nivel de un estudiante, pero siempre que se supongan ciertos los resultados utilizados de la teoría de Morse. En caso de querer completarlo podría requerir todo un trabajo exclusivamente para ello.

La demostración de los resultados del artículo de Allen Hatcher no es complicada ni extensa si nos basamos en los "Hechos" que él aporta. Estos hechos son los que verdaderamente requieren un amplio conocimiento de la teoría de Morse. La herramienta trasladada por Allen Hatcher, la técnica del toro de Kirby, junto con tales Hechos han sido la clave para la demostración del "Teorema de Alisamiento de Asas", que a su vez es la base de las demostraciones de los teoremas A y B.

He podido apreciar toda una teoría completamente nueva, que daría para una asignatura, y es por esto que he intentado ilustrarla con el programa que he diseñado e implementado.

Además, dicho programa ha aportado resultados bastante buenos si lo comparamos con el mismo programa sin utilizar teselado. El camino para encontrar la medida utilizada ha requerido mucho tiempo y esfuerzo, principalmente en el análisis de otras medidas previas, cuyas características nos han guiado a la medida actual. También se ha dedicado mucho tiempo al estudio e impementación de algoritmos de teselado para el Geometry shader, que finalmente no se han utilizado, puesto que el Tessellation shader ya lo realiza de manera eficiente, pero no ha sido en vano, ya que ha facilitado el entendimiento dl funcionamiento del proceso de teselado.

Por tanto, en el sentido matemático puedo decir que se han alcanzado los objetivos, tal y como se propusieron inicialmente, aunque implícitamente no hayamos abordado el caso de variedades con borde.

Para la parte informática puedo decir que también se han alcanzado, e incluso se han conseguido otros objetivos que en un principio no estaban presentes, como el uso por parte del usuario de derivadas parciales y la visualización de algunas características de una función de Morse predefinida (la función altura).

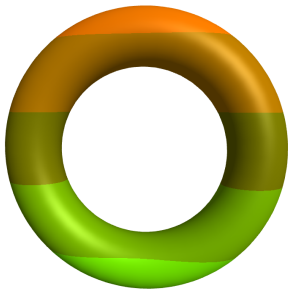


Figura 9.19.: Niveles de la función altura en el toro.

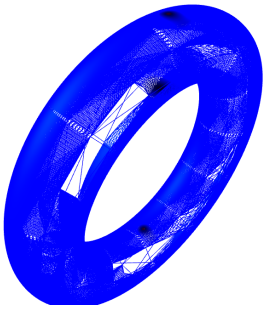


Figura 9.20.: Puntos críticos de la función altura en el toro (zonas oscuras).

Se pueden ver los puntos críticos en color negro, aunque sólo se pueden observar 3 de los 4 existentes para el toro ya que al ser una superficie opaca sólo se pueden ver los ocultos en modo malla y dependiendo de la posición de la cámara (se ha intentado capturar la escena en la que aparezcan el mayor nº de puntos críticos posible).

Vías futuras

Resultados importantes

Los hechos definidos por Allen Hatcher en su artículo son de gran interés y pueden aportar mucha información para el suavizado de estructuras topológicas. Aunque es cierto que tales resultados están orientados específicamente para la demostración de los 2 resultados, se podrían utilizar para simplificar otras demostraciones del campo de la topología diferencial, siempre que traten variedades topológicas 2-dimensionales.

Teselación Phong

Para complementar el modelo de iluminación Phong, sería interesante probar a añadir 1 o 2 niveles de teselado en el que se implemente la teselación Phong [BABo8]. Los nuevos vértices generados en dichos niveles muy probablemente no estarán en la superficie, es decir, perdemos una característica bastante importante de la malla de triángulos generada.

Sin embargo, es la idónea para la iluminación Phong, por lo que si sólo se añaden unos pocos vértices los resultados visuales podrían mejorar, porque desparecerían en algunos casos las "zonas de transición de luz" (donde se aprecian colores planos aunque la transición entre ellos sea continua).

Se podría implementar en el Geometry shader, puesto que al ser una etapa posterior al Tessellation shader sería como añadir uno o varios niveles más de teselación. Es posible que la ganancia en calidad no fuese suficiente en comparación con la pérdida de eficiencia, pero para ello sería necesario estudiarlo a fondo.

Transform feedback

El estudio de la teselación podría continuar por la técnica de "Transform feedback", que permite procesar de nuevo la malla generada como salida tras el teselado. Se podría implementar un teselado incremental, de manera similar a una función recursiva, es decir, subdividir triángulos hasta que se llegue al punto deseado (GLSL no permite recurrencias de forma natural). Intuitivamente parece que consumirá bastantes recursos, pero puede que gestione mejor los recursos disponibles, incluso se podrían eliminar etapas de la secuencia de renderizado (los Tessellation shaders no harían falta, bastaría con el Geometry shader, aunque habría que estudiar la eficiencia).

Funciones de Morse

Una funcionalidad que se podría añadir al programa en un futuro, es la visualización de una función de Morse para una superficie dada. Aprovechando que podemos definir una superficie mediante cartas, podemos mostrar datos interesantes de cualquier función de Morse que tenga como dominio dicha superficie, tales como puntos críticos e "isobaras". Los puntos críticos son fácilmente calculables teniendo en cuenta que si para una carta ϕ y una función de Morse f, tomando $g(u,v) = f \circ \phi(u,v)$, si tiene derivadas parciales iguales a 0 en (u,v) y $p = \phi(u,v)$ es un valor regular de ϕ, entonces p es un punto crítico de f.

Teniendo en cuenta el lema 2.1 del apartado de Teoría de Morse (parte I), se puede decir que nuestro programa actualmente es capaz de visualizar cualquier función de Morse si a las cartas a representar se les aplica previamente un embebimiento adecuado en $\mathbb{R}^3$ (ya que la función de Morse coincidiría con la función altura), aunque la superficie también se visualizaría a través de dicho embebimiento, aplicando una deformación drástica a la superficie. Es decir, la teoría apoya la posibilidad de representar cualquier función de Morse con domino una superficie de $\mathbb{R}^3$, pero visualmente sería incomprensible.

Para mejorar la visualización de los puntos críticos se podría añadir la funcionalidad de teselar sólo cerca de los puntos críticos, cuando se estén visualizando, para que se detecten con mayor precisión. Adicionalmente se podría renderizar la superficie controlando su opacidad, para así poder ver todos los puntos críticos a la vez, aunque estén en caras ocultas.

Portabilidad

En cuanto a la portabilidad se podría estudiar implementarlo para sistemas operativos Windows o incluso hacer una versión web con WebGL.

Superficies estáticas

A priori, el programa implementado sirve para visualizar con cierto nivel de precisión una superficie u homotopía, pero haciendo un correcto uso de la secuencia de renderizado se podría aprovechar para generar objetos $3D$ estáticos, es decir, dada una parametrización de entrada, obtener como salida una malla indexada en la que los triángulos están concentrados en aquellas zonas más complejas de la superficie. De esta forma podríamos cargar en un

futuro dicha superficie sin tener que evaluar en cada frame la parametrización.

Para completar lo anterior se podría estudiar el generar junto a dicha malla indexada un mapa de normales, de esta manera la superficie se visualizaría mucho mejor en cuanto a iluminación sin empeorar drásticamente el rendimiento. Esta funcionalidad sería idónea para generar mallas indexadas que aproximen a una cierta superficie y que se utilizarán en una escena $3D$. También se podría completar con otras técnicas de teselado para aumentar el rendimiento cuando haya varios objetos en la escena y estén a distintas distancias (visualizar el objeto real cuando esté cerca y niveles de teselado inferiores cuando supere una cierta distancia).

En este caso si sería interesante estudiar a fondo el post-procesado de vértices, sobretodo para reducir los triángulos utilizados en zonas similares al plano. Por ejemplo, si queremos mostrar una función que perturba el plano en zonas aisladas, como la parametrización de ejemplo "waves2.in", se utilizarían muy pocos triángulos en las zonas totalmente planas y el resto se usarían en las zonas restantes, independientemente del tamaño de la malla inicial (actualmente las zonas planas están trianguladas según la malla inicial).

A. Instalación del software

A.1. Requisitos previos

Los requisitos mínimos para poder compilar el proyecto son los siguientes:

- SO: Ubuntu 18.04 LTS o superior (o distribuciones similares).
- GPU: compatible con versión de OpenGL 4.4 o superior (para poder usar el tessellation shader, entre otros). Se puede ver la versión instalada (después de realizar el proceso de instalación que se indicará) ejecutando el comando: glxinfo | grep "core profile version string"
- Dependencias: "make", para poder usar el makefile, y "apt", para gestionar paquetes.

A continuación se muestran las dependencias del proyecto, aunque el propio makefile las comprobará y actualizará:

- Procesador:
  - gcc.
  - flex.
  - bison, versión 3.5 o superior.
- Visualizador de Superficies (además de las de Procesador):
  - g++, versión 11 o superior.
  - libglapi-mesa, mesa-utils y mesa-common-dev, que es una implementación de código abierto de OpenGL.
  - libglfw3 y libglfw3-dev, para que la aplicación pueda gestionar las ventanas del sistema.

A.2. Instalación

Para instalar el programa basta con clonar el repositorio de GitHub [FdlH] y ejecutar la orden "make install" desde la terminal, en el directorio raíz del programa. Realizará la comprobación de las dependencias de forma automática y en caso de no estar instaladas solicitará la confirmación de su instalación (a la última versión). Después compilará automáticamente el programa.

Internamente utiliza "apt-get upgrade", excepto para Bison, que será instalado haciendo uso de recursos locales, con la versión exacta 3.5.

A.3. Errores de compilación

En caso de que aparezcan errores al compilar, compruebe detenidamente que se cumplen los **Requisitos previos**. Es posible que algunas dependencias no las instale el comando "apt", por lo que se recomienda actuar como se indica en esta respuesta [SB] (para permitir la instalación de todos los paquetes).

B. Guía de uso del programa

En este capítulo se explicará brevemente cómo utilizar el programa. Aun así, la interfaz del programa se ha intentado diseñar lo más sencilla y clara posible, mostrando adicionalmente cuadros de texto si se mantiene el ratón sobre ciertos elementos.

Si sólo quiere hacer uso del programa "procesador", ejecute el comando:

make read FILE=<archivo>

pero desde el directorio "processor'. Esta acción devolverá la traducción de "<archivo>" al fichero "../shaders/functions.s" y la salida de error en "../error.log".

Si queremos utilizar el programa completo, primero instalaremos el programa tal y como se indica en el apéndice de **Instalación del software**. Una vez instalado, se abrirá siempre con el comando "make", "make execute" o equivalentemente "./bin/program". Teniendo abierto el programa se mostrará siempre la última parametrización compilada con éxito. En el lateral izquierdo aparecerá un elemento de la interfaz, el menú, donde se podrá modificar:

- Parametrización:
  - Seleccionar una parametrización ya existente, crearla o compilarla. También se podrá editar la actual, en cuyo caso aparecerá una ventana de edición de la propia interfaz. Dicha ventana también aparecerá en caso de que hayan errores léxicos, sintácticos o semánticos en la parametrización, con la salida del error desplegada. Como ejemplos se aportarán parametrizaciones de una gran variedad de superficies, ubicadas en el directorio "manifolds".
  - Cambiar el tamaño de la malla de partida, confirmando con la tecla "Enter".
  - Visualizar la ventana con los parámetros temporales, con un tick que indica si está activa la ventana o no. Estará semi-visible si la superficie no tiene parámetros temporales. Dicha ventana mostrará los parámetros temporales en orden, permitiendo moverlos manualmente o generar una animación:
    - Sin: movimiento sinusoidal entre el valor 0 y 1. Ideal para animaciones oscilantes.
    - Lineal: movimiento lineal del valor 0 al 1, volviendo instantáneamente al 0. Utilizado para movimientos lineales respecto al tiempo, que junto con el uso de funciones periódicas se puede generar la sensación de movimiento infinito (como los ejemplos wavesX.in).
- Visualización del objeto:
  - Invertir normales (si no se quiere modificar la parametrización).
  - Visualizar en modo malla ("Polygon mode").
  - Activar/desactivar la auto-rotación (rotación orbital entorno al punto hacia el que mira la cámara).

  - Ver los vectores tangentes, bitangentes y normales a los puntos (ya sean los de la malla inicial o de todos los generados).
  - Cambiar modo del color de la superficie, ya sea el color base, la curvatura de Gauss, el área diferencial, la altura o los puntos críticos de ésta vista como función de Morse. Una vez seleccionado un modo, aparecerán coeficientes que permitirán ajustar correctamente la visualización a la superficie actual.

- Opciones del teselado:
  - Desactivar/activar el teselado.
  - Indicar la precisión a la que se desea llegar con el teselado.
  - Opciones avanzadas: modificar aquellos coeficientes específicos del teselado, como el tipo de mejora de rendimiento a usar ("improve" normal o específica), la distancia de teselado, el umbral para detectar bordes y el exponente aplicado a la curvatura de Gauss. Todos ellos se inician con un valor por defecto.

- Iluminación: es posible cambiar los coeficientes del modelo de iluminación "Phong" y ver el vector de dirección de la luz actualmente. La luz no es direccional, el vector indica la dirección de la luz con respecto al origen $(0, 0)$.

- Estadísticas: muestra los fotogramas por segundo y la latencia medias, junto con el número de primitivas generadas tras la fase del geometry shader (después del tessellation shader). Permite además grabar los datos y los almacena de manera automática tras 22 segundos (antes si pulsamos "Stop" y seguidamente "Save info") en un fichero en el directorio raíz, con nombre dependiente de la parametrización y configuración actual.

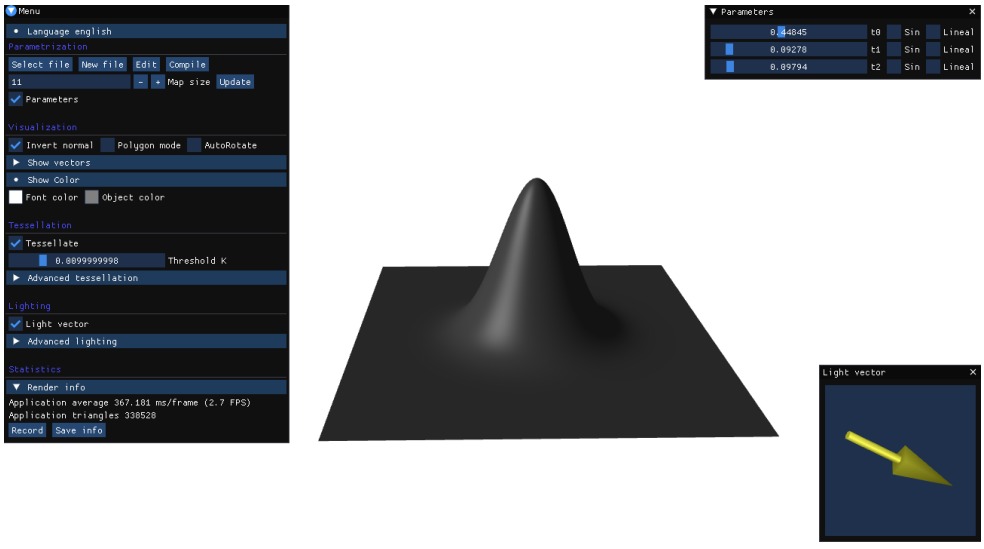


Figura B.1.: Ventanas iniciales.

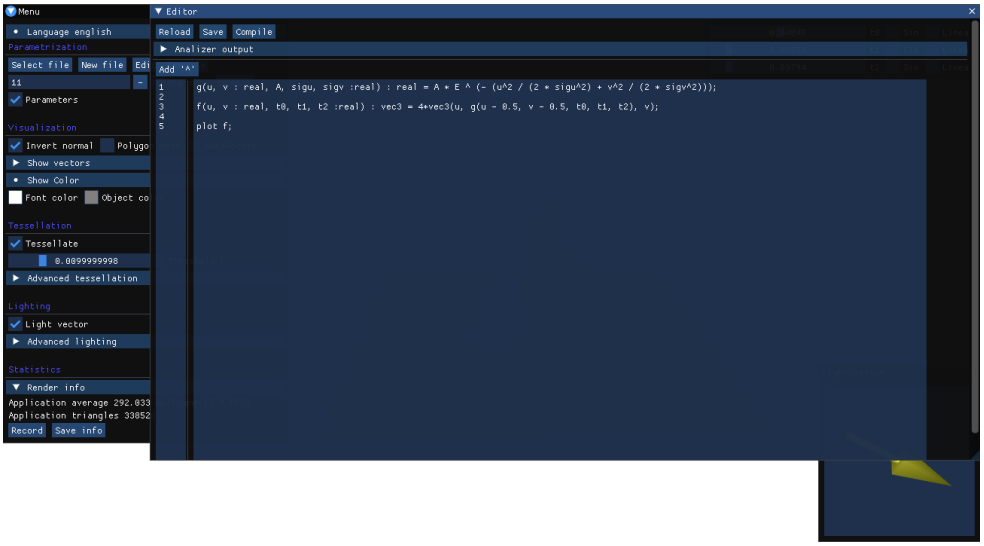


Figura B.2.: Ventana de edición de código.

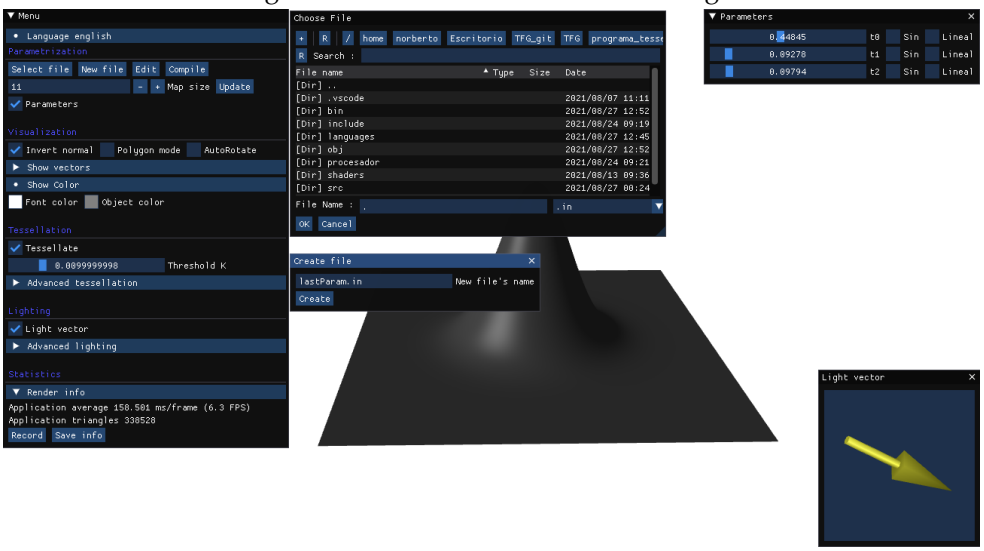


Figura B.3.: Ventanas de selección y creación de archivos.

Para mayor comodidad, se han incorporado atajos de teclado para ciertas acciones:

- 'LCtrl' + 'R': activa/desactiva el modo de rotación automática.
- 'LCtrl' + 'P': cambia entre los modos de visualización malla y relleno.
- 'LCtrl' + 'N': activa/desactiva la visualización de normales.
- 'LCtrl' + 'L': ejecuta el "procesador" y compila los shaders.

Además, existen varios controles del ratón para manejar la cámara:

- Botón izquierdo: si se mantiene pulsado y arrastramos, la cámara realizará una traslación en la dirección contraria.
- Botón derecho: si se mantiene pulsado y arrastramos, la cámara realizará una rotación (cámara orbital) en la dirección contraria.
- Rueda: controla el zoom.
- Botón de la rueda: reinicia la posición de la cámara.

Bibliografía

A continuación se listan todas las referencias por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

[BABo8] Tamy Boubekeur, Marc Alexa, and TU Berlin. Phong Tessellation. *ACM Transactions on Graphics*, 2008. Artículo sobre la teselación Phong, mejorando los resultados cuando se utiliza su modelo de iluminación. [Citado en pág. 66]

[Chh17] Rithivong Chhim. Morse Theory and Applications. *CIMAT*, 2017. Material complementario de la teoría de Morse, con el objetivo de encontrar resultados que trabajen con difeomorfismos. [Citado en pág. 3]

[CKoo] Ajit Champanerkar, Abhijit andKumar and S. Kumaresan. Classification of Surfaces via Morse Theory. *Expositiones Mathematicae*, 18(1), 2000. Conceptos interesantes de la teoría de Morse, centrado generalmente en funciones diferenciables, salvo algún resultado en específico. [Citado en págs. xv, xxi, 3, 8, and 9]

[Cor] Aymar Cornut. Dear Imgui. https://github.com/ocornut/imgui. Repositorio oficial de la librería "Dear ImGui", con la que se ha implementado la interfaz del programa. Accedido el 28 de agosto de 2021. [Citado en págs. xvii, xxi, and 57]

[Cui] Stephane Cuillerdier. Imgui Dialog. https://github.com/aiekick/ImGuiFileDialog/tree/7b96209e5e96caff9cfeaf3e67dc7cb7d686ae74. Complemento de la interfaz para visualizar un explorador de archivos y así elegir un fichero de parametrización. Accedido el 28 de agosto de 2021. [Citado en págs. xvii and 57]

[dV] Joey de Vries. Learn Opengl. https://learnopengl.com/. Ejemplos de uso de shaders, especialmente el Geometry shader. Accedido el 28 de agosto de 2021. [Citado en págs. 27 and 57]

[Epp] David Eppstein. Schwarz Lantern. https://en.wikipedia.org/wiki/Schwarz_lantern. Recurso online en el que se explica e ilustra el problema del Farolillo de Schwarz. Accedido el 28 de agosto de 2021. [Citado en pág. 33]

[FdlH] Norberto Fernández de la Higuera. Visualizador de Superficies. https://github.com/norbertoalfa/TFG. Repositorio de GitHub del proyecto donde se encuentra la implementación del programa. Accedido el 30 de agosto de 2021. [Citado en pág. 69]

[Hat13] Allen Hatcher. The Kirby Torus Trick for Surfaces. *arXiv.org*, 2013. Artículo principal, en el que se demuestran 2 resultados cuyo colorario directo es el teorema central del proyecto. [Citado en págs. xiii, xv, xvii, xix, and xxi]

[Khr] Khronos Wiki. https://www.khronos.org/opengl/wiki/. Toda la información referente a OpenGL. Accedido el 28 de agosto de 2021. [Citado en págs. xvi, xvii, xxi, 27, and 56]

[Man14] Ciprian Manolescu. Triangulations of Manifolds. *Notices of the International Congress of Chinese Mathematicians*, 2(2):21–23, 2014. Información relevante sobre el trayecto histórico de la triangulación de variedades, en concreto, la obtención de estructuras diferenciables. [Citado en pág. xix]

[Mor] Michele Morrone. Imgui Quat. https://github.com/BrutPitt/imGuIZMO.quat. Complemento de la interfaz para visualizar el vector de dirección de la luz. Accedido el 28 de agosto de 2021. [Citado en págs. xvii and 57]

[Mun64] James Munkres. Obstructions to Imposing Differentiable Structures. *Illinois J. Math.*, 8(3):361–376, 1964. Demostración del teorema aportada por Munkres, aunque cita su previa demostración por S. S. Cairns haciendo uso del teorema de triangualción de E. E. Moise, para el caso sin borde. [Citado en pág. xix]

[SB] Basharat Sialvi and Kevin Bowen. Errores Ccomunes de apt. https://askubuntu.com/questions/140246/how-do-i-resolve-unmet-dependencies-after-adding-a-ppa/142808. Recolección de errores comunes del comando "apt", con sus correspondientes soluciones. Accedido el 3 de septiembre de 2021. [Citado en pág. 70]

[VC07] Juan Guillermo Vélez C. Introducción Al Estudio De Las Estructuras Geométricas. *Universidad de Los Andes*, pages 40–41, 2007. Nociones básicas de las estructuras geométricas, principalmente la definición de triangulación diferenciable. [Citado en pág. 3]

[Wol] Gaussian Curvature. https://mathworld.wolfram.com/GaussianCurvature.html. Obtención de una fórmula (10) que indica el cálculo de la curvatura de Gauss K fácilmente usable en el ámbito informático. Accedido el 28 de agosto de 2021. [Citado en pág. 29]