



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 31 de agosto 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	xi
Resumen	xiii
Summary	xv
Introducción	xix
Objetivos	xxi
I. Teorema clásico de Munkres	1
1. Conceptos previos	3
2. Resultados previos	7
2.1. Teoría de recubridores	7
2.2. Teoría de Morse	8
2.3. Hechos utilizados para los teoremas	9
2.4. Teorema de Alisamiento de Asas	13
3. Resultados principales	17
3.1. Enunciados	17
3.2. Demostración del Teorema A	17
3.3. Demostración del Teorema B	21
II. Visualización de superficies	23
4. Conceptos básicos	25
5. Estudio de la teselación	27
5.1. Medidas según la definición	27
5.2. Mejora del teselado	31
5.3. Estimación de las medidas	33
6. Procesador	35
7. Planificación y presupuesto	39
7.1. Planificación temporal inicial	40
7.2. Diferencias con la planificación real	41
7.3. Presupuesto	41

8. Análisis y diseño	43
8.1. Especificación de requisitos	43
8.2. Diagramas	44
8.3. Principales estructuras de datos	46
8.4. Principales desarrollos algorítmicos	46
9. Implementación y pruebas	49
A. Instalación del software	59
A.1. Requisitos previos	59
A.2. Instalación	59
A.3. Errores de compilación	60
B. Guía de uso del programa	61
Conclusiones	65
Bibliografía	69

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 conlleva 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 difeomorfismos. 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.

La visualización de la superficie se realizará mediante una aproximación con mallas indexadas (conjunto de triángulos), cuyos vértices siempre estarán en la superficie objetivo. 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 visualice correctamente, la definición escogida estará orientada a cómo la visión humana la percibe. Aunque nos centremos en esta definición, trivialmente el límite de la malla indexada será la superficie a representar.

El estudio de la demostración del teorema nos ha aportado varios recursos bastante interesantes, tales como “el truco del Toro” o el uso del grafo asociado a una función de Morse para obtener difeomorfismos de partes de la superficie a discos, anillos, “pantalones” o “pantalones cruzados”.

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).

Resumen

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.

The initial main goals are to implement and design an efficient program for visualize homotopies, and proof the following theorem as a corollary of the 2 results cited on Allen Hatcher's article [Hat13]: For every topological 2-manifold exists only one smooth structure, except diffeomorphisms. 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 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 [Chao]. 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 for 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 (using on shader) 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 include in the shaders that will need it (tessellation shaders).

For the implementation of the “processor” I used a base code, provided in 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 [Aie], 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 same 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 approximation 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

Summary

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 por 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).

Nosotros estudiaremos la existencia y unicidad, salvo difeomorfismos, de una estructura diferenciable para una variedad topológica 2-dimensional. Este es un teorema clásico de la topología diferencial, cuyas demostraciones previas requieren de un conocimiento amplio en el campo y su realización es extensa. Una de sus demostraciones la realiza James Munkres en su artículo “Obstructions to Imposing Differentiable Structures” [Mun64], aunque él mismo indica en la introducción que S. S. Cairns lo había demostrado para el caso sin borde con la ayuda del teorema de triangulación de E. E. Moise, también indicado en el artículo de Ciprian Manolescu [Man14].

Sin embargo, Allen Hatcher en el artículo “The Kirby Torus Trick for Surfaces” [Hat13] enuncia y demuestra 2 resultados cuyo corolario trivial es dicho teorema.

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 ha orientado el estudio al uso de mallas, es decir, aproximar una superficie mediante un conjunto de triángulos, para renderizar mediante rasterizado. El estudio en sí consistirá en detectar en qué zonas será necesaria una mayor cantidad de triángulos para aproximarla mejor.

Antes de iniciar el estudio del teselado (subdivisión de triángulos) será necesario definir un lenguaje propio para poder indicar explícitamente las cartas de la superficie. Se realizará por medio de un “procesador” que dará como salida un código GLSL, que 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).

Inicialmente se propuso el uso del Geometry shader para implementar un algoritmo de tese-

lado 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 apartado de **Implementación y pruebas**, decidí utilizar el Tessellation shader, que es más reciente y 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 e 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.
- En cuanto a la parte informática, se han añadido los capítulos referentes al desarrollo de una aplicación (capítulos 7, 8 y 9), aunque cabe destacar que ha tenido bastante peso el estudio de la teselación, por lo que algunos diagramas no se indicarán si no son relevantes. 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 [[Chao0](#)].
- Para la implementación del programa, la documentación de OpenGL en la Wiki de Khronos [[Khr](#)] y la de Dear ImGui para la interfaz [[?](#)].

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. Además, se ha completado con teoría de Morse, sin escribir 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 Moorse (sólo función altura) y sus puntos críticos.
 - 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.

Objetivos

- El software visualiza de manera eficiente las cartas indicadas, permitiendo animaciones fluidas y mejorando la gestión de recursos frente al uso de una malla de triángulos uniforme. 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 [Chao0], [Chh17] y [VC07], 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. La restricción de su imagen aporta un homeomorfismo.

Definición 1.3. Un **sistema coordenado** sobre S es un embebimiento $h : \mathbb{R}^2 \rightarrow S$.

Definición 1.4. Sea S un espacio topológico Hausdorff, un **atlas 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 $h_i(\mathbb{R}^2) \cap h_j(\mathbb{R}^2) \neq \emptyset$ entonces $h_j^{-1} \circ h_i : h_i^{-1}(h_i(\mathbb{R}^2) \cap h_j(\mathbb{R}^2)) \rightarrow h_j^{-1}(h_i(\mathbb{R}^2) \cap h_j(\mathbb{R}^2))$ es un difeomorfismo.

Definición 1.5. Sea S un espacio topológico Hausdorff, una **estructura diferenciable 2 – dimensional** sobre S es un atlas 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 **inmersión** es una aplicación diferenciable entre variedades diferenciables cuya derivada es inyectiva en todo punto.

Definición 1.8. Una **celda** C es un subconjunto abierto de un espacio topológico X tal que C homeomorfa a la bola unidad n -dimensional, que se puede extender de forma continua al borde. Dicho homeomorfismo se denomina **aplicación celda**. Si esta aplicación es diferenciable, se dice que la celda es diferenciable.

Definición 1.9. Una **triangulación diferenciable** de una variedad X es un conjunto de celdas diferenciables tal que la unión de todas ellas es X y cuyas intersecciones son únicamente los lados, es decir, los interiores son disjuntos 2 a 2.

1. Conceptos previos

Definición 1.10. Sean f y g homeomorfismos entre los espacios topológicos X e Y . Una **isotopía** es una homotopía entre f y g , $H : X \times [0, 1] \rightarrow Y$, con:

1. $H_0 = f$.
2. $H_1 = g$.
3. $\forall t \in [0, 1]$, H_t es un homeomorfismo.

Definición 1.11. Sean f y g embebimientos entre las variedades N e M . Una **isotopía de embebimientos** es un homeomorfismo $H : M \times [0, 1] \rightarrow M \times [0, 1]$ cumpliendo:

1. $H(y, 0) = (y, 0) \forall y \in M$.
2. $H(f(x), 1) = (g(x), 1) \forall x \in N$.
3. $H(M \times \{t\}) = M \times \{t\} \forall t \in [0, 1]$.

Equivalentemente podemos decir que H es la isotopía de Id_M en $g \circ f^{-1}$ donde tenga sentido.

Definición 1.12. Sea M una variedad diferenciable y $f : M \rightarrow \mathbb{R}$ una función diferenciable en M . Un punto $p \in M$ se dice que es un **punto crítico** de f si $Df(p) = 0$ en $T_p M$. A su imagen por f , $f(p)$, se le dice **valor crítico** de f .

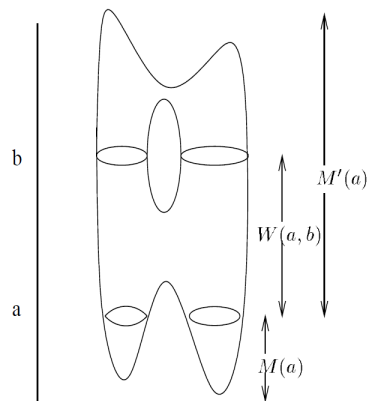
Definición 1.13. Sea M una variedad diferenciable y $f : M \rightarrow \mathbb{R}$ una función diferenciable en M . Un punto crítico $p \in M$ se dice que es **no degenerado** si $H_\phi(f)(p)$ es regular para cualquier parametrización ϕ centrada en p , donde $H_\phi(f) = H(f \circ \phi)$ la matriz Hessiana de $f \circ \phi$.

El **índice** de dicho punto es la dimensión del mayor subespacio de $T_p M$ 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 1.14. Sea f una función diferenciable en una **superficie** S , se dice que es una **función de Morse** si todos sus puntos críticos son no degenerados.

Definición 1.15. Sea f una función de Morse, dados $a < b$ valores regulares de f en \mathbb{R} definimos:

- $V(a) = f^{-1}(a)$, se puede ver como una curva de nivel.
- $M(a) = f^{-1}((-\infty, a])$, el conjunto que hay “debajo” de la curva de nivel cuyo valor es a .
- $M'(a) = f^{-1}([a, \infty))$, el conjunto que hay “encima” de la curva de nivel cuyo valor es a .
- $W(a, b) = f^{-1}([a, b])$, el conjunto contenido entre 2 curvas de nivel.



2. Resultados previos

2.1. Teoría de recubridores

A continuación se enuncian los resultados utilizados de la Teoría de Recubridores.

Teorema 2.1 (de Levantamiento de aplicaciones). *Sea (\tilde{X}, π) recubrimiento de X , Y espacio topológico y $f : Y \rightarrow X$ 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. $\exists ! \tilde{f} : Y \rightarrow \tilde{X}$ 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 S superficie y $f : S \rightarrow \mathbb{R}$ función de Morse. Tomamos $a < b$ valores regulares tal que $W(a, b)$ no contiene 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 demuestra en el artículo “Classification of Surface via Morse Theory” [Chao], el teorema 8. \square

Teorema 2.3. Sea S superficie y $f : S \rightarrow \mathbb{R}$ función de Morse. Sea p un punto crítico y $a < b$ valores regulares tal que $W(a, b)$ no contiene 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 .
- Si el índice de p es 1, $M(b)$ es difeomorfo a $M(a)$ junto con un rectángulo pegado en dos segmentos disjuntos 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.

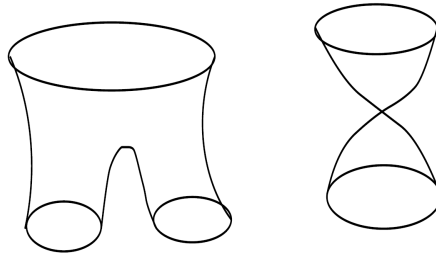


Figura 2.1.: Pantalones normales y cruzados

Demostración. El primer punto se demuestra en el artículo “Classification of Surface via Morse Theory” [Chao], el teorema 13. El punto siguiente se demuestra en el mismo artículo, el teorema 16. \square

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 se escapa de los límites del proyecto, en cuyos casos se aportarán demostraciones que suponen ciertos aquellos elementos propios de la 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. \square

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.

Podemos tomar $f : S \rightarrow \mathbb{R}$ función de Morse apropiada, es decir, los inversos de compactos son compactos y todos sus puntos críticos están a distintos niveles (es posible porque están aislados). 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.

2. Resultados previos

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 triangulación diferenciable dividiendo de forma adecuada los polígonos. \square

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

Demostración. Dividimos T'_S 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'_S 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 (existe un C_j entre las piezas que representan los vértices). Existe una aplicación cociente $q : T'_S \rightarrow G$ que lleva cada punto del entorno de una circunferencia C_j a la correspondiente proyección sobre el arco de G que representa a C_j . Por consiguiente, los C_j van a los vértices del grafo G .

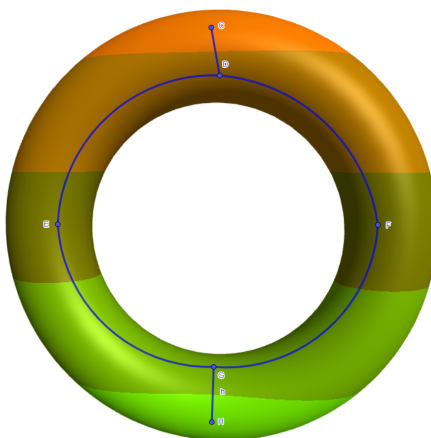


Figura 2.2.: Ejemplo de grafo asociado para la función de Morse “altura” que parte del Toro.

La aplicación $q_* : \Pi_1(T'_S) \rightarrow \Pi_1(G)$ es un homomorfismo sobreyectivo con inversa a la izquierda, viendo q como una homotopía. Por tanto, $\Pi_1(G)$ es un cociente de $\Pi_1(T'_S)$, 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'_S 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. \square

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 $\Pi_1(T_S)$, 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_S sea una sucesión de anillos pegados diferenciablemente, así que T_S es difeomorfo a T con la estructura usual o a una botella de Klein, pero no puede ser éste último por ser $\Pi_1(T_S)$ abeliano (ya que el grupo fundamental de la botella de Klein no es abeliano y no es posible siquiera un homeomorfismo entre ellos). \square

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 $g : (D^1 \times \mathbb{R})_E \rightarrow (D^1 \times \mathbb{R})_U$, con U la estructura usual, que además es la identidad entorno a $\partial D^1 \times \mathbb{R}$.

Demostración. Tomamos la proyección $\pi : (D^1 \times \mathbb{R})_S \rightarrow \mathbb{R}$ que ya es diferenciable en un entorno del borde. Si vemos $(D^1 \times \mathbb{R})_S$ como una variedad sabemos que existe una función de Morse $f : (D^1 \times \mathbb{R})_S \rightarrow \mathbb{R}$, 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} .

2. Resultados previos

La nueva función f no tiene puntos críticos, por lo que puede ser la segunda componente de un difeomorfismo $g : (D^1 \times \mathbb{R})_S \rightarrow D^1 \times \mathbb{R}$, 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. \square

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 $g : D_E^2 \rightarrow D_U^2$, 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_S^2 y como es diferenciable, la extendemos a una función de Morse propia $f : D_S^2 \rightarrow (0, 1]$, cuyos puntos críticos están en el interior y $f^{-1}(1) = \partial D_S^2$.

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 $g : D_S^2 \rightarrow D^2$ 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_S^2 . \square

2.4. Teorema de Alisamiento de Asas

Teorema 2.4. (de “alisamiento de asas”) Sea S una **variedad** diferenciable, entonces:

1. Un embebimiento $\mathbb{R}^2 \rightarrow S$ puede isotoparse a un embebimiento diferenciable en un entorno compacto del origen. Además, **la isotopía coincide con la identidad** fuera de un entorno compacto que contiene al anterior.
2. Un embebimiento $D^1 \times \mathbb{R} \rightarrow S$ 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 coincide con la identidad fuera de un entorno compacto de $(D^1 \times 0) \cup (\partial D^1 \times \mathbb{R})$** .
3. Un embebimiento $D^2 \rightarrow S$ que es diferenciable entorno a ∂D^2 puede isotoparse a un embebimiento diferenciable en todo D^2 . La isotopía **coincide con** 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 el “truco 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 $q : T' \rightarrow \mathbb{R}^2$ 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 $h : \mathbb{R}^2 \rightarrow S$ el embebimiento del enunciado, por el cual, haciendo “pull-back”, 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

2. Resultados previos

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 $g : T_{E_2} \rightarrow T$. 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 $\hat{g} : \mathbb{R}^2_{E_1} \rightarrow \mathbb{R}^2$ 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.

$$\begin{array}{ccccc} (\mathbb{R}^2 / \mathbb{Z}^2)_{E_1} & \xrightarrow{g} & \mathbb{R}^2 / \mathbb{Z}^2 & \xrightarrow{L} & \mathbb{R}^2 / \mathbb{Z}^2 \\ \pi_1(0, T_{E_1}) & \xrightarrow{g_*} & \pi_1(0, T) & \xrightarrow{L} & \pi_1(0, T) \end{array}$$

De esta forma construimos un difeomorfismo $\hat{g} : \mathbb{R}^2_{E_1} \rightarrow \mathbb{R}^2$ como el levantamiento de g fijando el origen, que de forma natural es doblemente periódico.

$$\begin{array}{ccc} \mathbb{R}^2 & \xrightarrow{\hat{g}} & \mathbb{R}^2 \\ \pi \downarrow & \searrow g \circ \pi & \downarrow \pi \\ (\mathbb{R}^2 / \mathbb{Z}^2)_{E_1} & \xrightarrow{g} & \mathbb{R}^2 / \mathbb{Z}^2 \end{array}$$

Identifiquemos \mathbb{R}^2 con el interior del disco unidad de \mathbb{R}^2 mediante una reparametrización radial que es la identidad entorno al 0, haciendo “pull-back” para cada disco de \mathbb{R}^2 para así obtener la estructura diferenciable inducida por E_1 , que llamaremos E . Entonces aplicando esta identificación en el dominio y la imagen de \hat{g} obtenemos $G : D_E^2 \rightarrow D^2$ automorfismo diferenciable, que sigue siendo \hat{g} entorno al 0 y tiende a ser la identidad en el borde (por la periodicidad $\|\hat{g}(x) - x\|$ 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 $G : \mathbb{R}_E^2 \rightarrow \mathbb{R}^2$, 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 es la identidad fuera), 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 $h : D^1 \times \mathbb{R} \rightarrow S$ 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 $q : D^1 \times \mathbb{R} \rightarrow (D^1 \times D^1) - (0 \times \partial D^1)$ que es la identidad entorno $D^1 \times 0$. El comportamiento del homeomorfismo q se muestra en la siguiente figura:



Definimos $G : ((D^1 \times D^1) - (0 \times \partial D^1))_{E_1} \rightarrow (D^1 \times D^1) - (0 \times \partial D^1)$ por $G = q \circ g \circ q^{-1}$, que como es la identidad entorno al borde del dominio, se puede extender a $G : \mathbb{R}_{E_1}^2 \rightarrow \mathbb{R}^2$. No hay problema en los dos puntos de $0 \times \partial D^1$ ya que por como se define q , en ambos tiene límite y es la identidad. 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 $\partial D^1 \times \mathbb{R}$.

2. Resultados previos



Podemos adaptar el truco de Alexander definiendo una isotopía G_t de homeomorfismos 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 $h : D^2 \rightarrow S$ 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 2.1. *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. Se restringe h a $B_p \subset W$, una bola cerrada con centro p . Sabemos que existe $f : D^2 \rightarrow B_p$ difeomorfismo y $g = h \circ f$ se puede extender continuamente a \mathbb{R}^2 (al punto q fuera de D^2 se le asigna el mismo valor que al punto de intersección de la circunferencia con el segmento del origen a q). Aplicamos el teorema anterior a $g : \mathbb{R}^2 \rightarrow S$ de manera que el entorno del origen V donde se altera g sea menor que la bola cerrada unidad, obteniendo así una isotopía g_t .

Definimos la isotopía deseada como $h_t(x) = (g_t|_{D^2} \circ f^{-1})(x)$ si $x \in B_p$ y $h_t(x) = h(x)$ si $x \notin B_p$. Está bien definida puesto que $g_t|_{D^2} \circ f^{-1}$ es la identidad fuera de un entorno de p contenido en B_p y simplemente extendemos por la identidad. Además, de forma natural $h_0 = h$ por ser $g_0 = g$ y h_1 es diferenciable en un entorno de p contenido en B_p por serlo g_1 en un entorno del origen.

□

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 $\{h_i | 1 \leq i \leq N\}$ con $N \in \mathbb{N}$ si es finito o $N = \infty$ si no lo es. Vamos a construir por inducción una estructura diferenciable en el conjunto $U_n = \cup_{i \leq n} h_i(\mathbb{R}^2)$, que por ser un sistema coordinado su límite debe de ser S , probando así el resultado. Cabe destacar que cada U_i 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 uno (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. $B(p, \epsilon_p)$ es abierto y queremos obtener \hat{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 \hat{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 $h_n : \mathbb{R}^2 \rightarrow h_n(\mathbb{R}^2)$ es isotópica a un homeomorfismo $\hat{h}_n : \mathbb{R}^2 \rightarrow h_n(\mathbb{R}^2)$ 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.

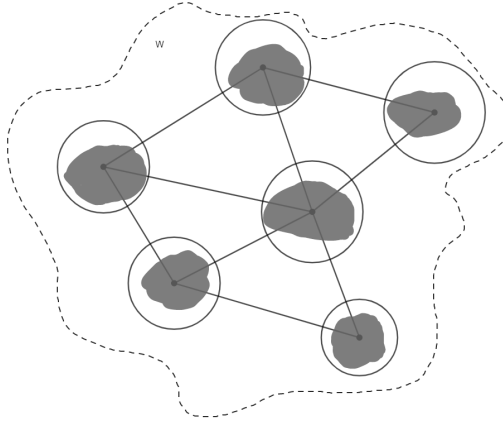


Figura 3.2.: O_p para cada vértice.

2. Tenemos por el paso anterior un $h_n : \mathbb{R}^2 \rightarrow h_n(\mathbb{R}^2)$ isotópico al original en las con-

diciones explicadas, y que es diferenciable como aplicación $W \rightarrow U_{n-1}$ 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 $W \rightarrow U_{n-1}$ 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:

- 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 .
- 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 .
- 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 $h_n : \mathbb{R}^2 \rightarrow h_n(\mathbb{R}^2)$ deseado.

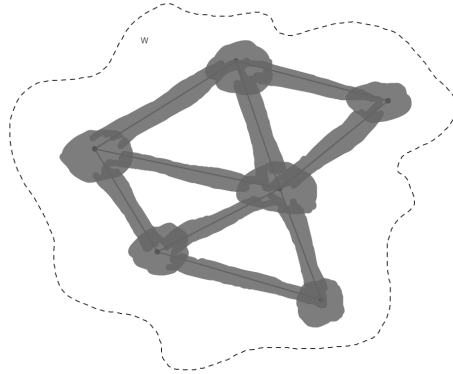


Figura 3.4.: Entorno del 1-esqueleto donde h_n es diferenciable.

3. Resultados principales

3. El tercer paso es similar a los anteriores, pero ahora usando el apartado 3 del Teorema de Alisamiento de Asas. Lo que hacemos es considerar para cada triángulo T de la triangulación de W un dominio de Jordan D_T satisfaciendo:

- a) $D_T \subset \mathring{T}$.
- b) $h_n : W \rightarrow U_{n-1}$ 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 $W \rightarrow U_{n-1}$ 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).

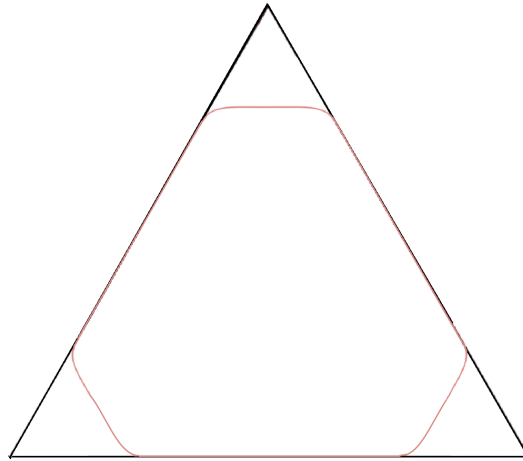


Figura 3.5.: Curva de Jordan cercana al triángulo

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 $D_T \subset \hat{T}$ 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 dominios D_T son disjuntos 2 a 2.

Todas las isotopías de los pasos anteriores coinciden por extensión continua con el homeomorfismo $h_n : \mathbb{R}^2 \rightarrow h_n(\mathbb{R}^2)$ 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 $\mathbb{R}^2 \rightarrow h(\mathbb{R}^2)$ coincidentes con h_n en $\mathbb{R}^2 - W$.

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

3.3. Demostración del Teorema B

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

Demostración. Sea $f : S \rightarrow S'$ homeomorfismo entre variedades diferenciales 2-dimensionales, se puede utilizar el **Hecho 2**, que nos aporta una triangulación diferenciable de S . Por definición de triangulación diferenciable, tenemos que la aplicación celda φ_n es un difeomorfismo. Vamos a proseguir de forma similar a la demostración del Teorema A, pero esta vez la función a isotopar es $g_n = f \circ \varphi_n : \mathbb{R}^2 \rightarrow f(\varphi(\mathbb{R}^2)) \subset S'$ homeomorfismo. Sea $W_n = \varphi_n^{-1}(S) = \varphi_n^{-1}(S \cap \varphi_n(\mathbb{R}^2))$ abierto de \mathbb{R}^2 , vamos a isotopar f en 3 etapas:

1. Para todos los vértices de la triangulación de S , vistos en W_n mediante φ_n^{-1} , tomamos una bola $B(p, \epsilon)$ de tal forma que sus cierres no se corten 2 a 2 y estén contenidos en W_n . Acto seguido podemos proceder de forma idéntica al apartado 1 de la demostración del Teorema A, obteniendo como resultado un homeomorfismo $\hat{g}_n : W_n \rightarrow g_n(W_n)$ isotópico a g_n , que es diferenciable en un entorno de cada vértice de la triangulación $O_p \subset W_n$, cuyos cierres no se cortan y quedan dentro de W_n y además \hat{g}_n coincide con g_n en un entorno de cada vértice mayor al anterior (cuyos cierres tampoco se cortan y están contenidos en W_n).

Si deshacemos el cambio con φ_n , obtenemos que $f : \hat{g} \circ \varphi_n : \varphi_n(W_n) \rightarrow g_n(W_n)$ homeomorfismo es isotópica a $\hat{f} = \hat{g} \circ \varphi_n : \varphi_n(W_n) \rightarrow g_n(W_n)$ homeomorfismo, cumpliendo lo descrito pero para $\varphi_n(W_n)$, por ser φ_n un difeomorfismo. Como \hat{f} coincide con f en el borde de $\varphi_n(W_n)$, la isotopía se puede extender a $S \rightarrow S'$. Mantenemos el nombre f para la nueva \hat{f} .

Realizamos de forma incremental este proceso, hasta conseguir una isotopía a una función que sea diferenciable en un entorno de todos los vértices de la triangulación de S que además queda fija en un entorno mayor (cuyos cierres no se cortan).

3. Resultados principales

2. Para todo lado l de la triangulación en W_n definimos un entorno R_l que es difeomorfo a un rectángulo, que a su vez es difeomorfo a $D^1 \times \mathbb{R}$, para así poder aplicar el apartado 2 del Teorema de Alisamiento de Asas. Para ello debe cumplir:

- a) R_l corta a l en una curva compacta y es disjunto con cualquier otro lado de la triangulación diferenciable. En particular R_l no contiene ningún vértice de la triangulación en W_n .
- b) Si p_1 y p_2 son los vértices extremos de l , una componente del borde está contenida en O_{p_1} y otra en O_{p_2} , con O_p entorno de p donde es diferenciable g_n , es decir, definimos R_l de manera que g_n sea diferenciable en 2 componentes de su borde (∂R_l) .
- c) Los cierres de los R_l en \mathbb{R}^2 son disjuntos 2 a 2 y están contenidos en W_n .

Estamos en las condiciones necesarias para aplicar el apartado 2 del Teorema de Alisamiento de Asas, dando lugar a una isotopía a $\hat{g} : W_n \rightarrow g(W_n)$ homomorfismo diferenciable en un entorno de cada lado de la triangulación en W_n que coincide con g fuera de un entorno mayor al anterior. Procedemos de igual forma que en el apartado 1 de esta demostración, obteniendo una isotopía de f a $\hat{f} : S \rightarrow S'$ homeomorfismo diferenciable en un entorno de cada lado de la triangulación diferenciable de S , que coincide con f fuera de un entorno mayor que el anterior.

3. En este último apartado queremos aplicar el punto 3 del Teorema de Alisamiento de Asas. Como la triangulación de S es diferenciable, sabemos que el interior de cada triángulo es difeomorfo al disco unidad de \mathbb{R}^2 , por lo que para utilizar un dominio de Jordan D_T para cada T triángulo de la triangulación de W_n de forma que no se corten 2 a 2, tomamos un conjunto más pequeño que el interior del triángulo (se puede tomar el mismo conjunto pero multiplicado por un factor de escala ligeramente menor que 1).

Tenemos que g_n es diferenciable en un entorno de ∂D_T y coincide con la g original fuera de un entorno del borde mayor que el anterior. De igual forma que en el apartado 3 de la demostración del Teorema B obtenemos una isotopía de g a $\hat{g} : W_n \rightarrow g_n(W_n)$ homomorfismo que es diferenciable en la triangulación de W_n (en los vértices, los lados y en el interior de cada triángulo). Siguiendo el esquema de los apartados anteriores obtenemos que f es isotópica a $\hat{f} : S \rightarrow S'$ homeomorfismo diferenciable en toda la triangulación diferenciable de S (vértices, lados e interiores). Como consecuencia \hat{f} es, de forma natural, un difeomorfismo, concluyendo así 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. Dichos conceptos se han obtenido de las referencias [Khr] y [dV], junto con el conocimiento adquirido de la asignatura Informática Gráfica.

Definición 4.1. 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 4.2. Entenderemos por **mall**a 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).

Definición 4.3. 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.

Definición 4.4. 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.

Definición 4.5. 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.

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.
- Geometry shader: tiene de entrada una primitiva y devuelve varias primitivas.
- Fragment shader: después del rasterizado (pasar de gráfico vectorial a píxels), se ejecuta para calcular el color de cada fragmento de triángulo.

4. Conceptos básicos

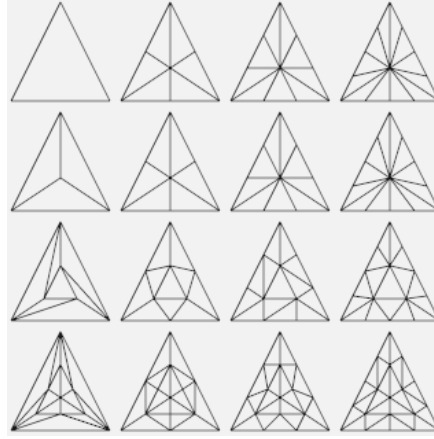


Figura 4.1.: Niveles de teselado del 1 al 4: outer en horizontal (todos los lados por igual) e inner en vertical.

- Compute shader: etapa utilizada para realizar cálculos 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]:

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

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 describe el estudio realizado para teselar de manera eficiente un triángulo. Esta técnica se implementará en shaders, para poder ejecutarlo directamente en la GPU, que ofrece mayor rendimiento que la CPU para dicho problema. 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.

Cabe destacar antes que la idea inicial era desarrollar un algoritmo de división recursiva de los triángulos, el cuál se detiene en el nivel en el que se cree que representa correctamente a la porción de superficie. Sin embargo, el lenguaje Glsl no permite realizar llamadas recursivas, por lo que era 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 el poliedro generado 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.

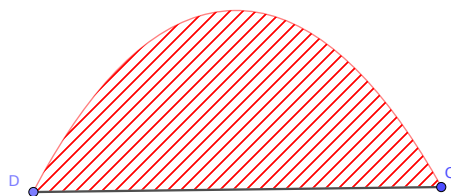


Figura 5.1.: Medida en un lado DC

5. Estudio de la teselación

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 $r = \frac{L(\alpha)}{L(I)}$ 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 $[0, 1] \times [0, 1]$, 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:

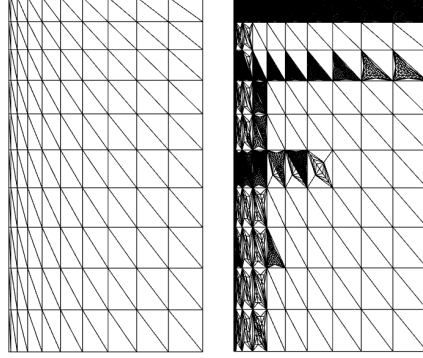


Figura 5.3.: Teselación innecesaria del plano

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 del politopo 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 $K_l L$ 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 imagen, donde el color indica el área diferencial de la parametrización.

5. Estudio de la teselación

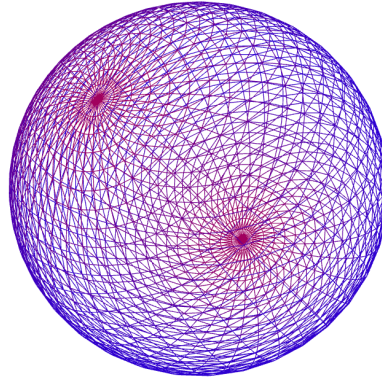


Figura 5.4.: Deformación heterogénea de la malla

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.

Puesto que la escena es dinámica y es posible mover el punto de visión, se puede utilizar el área relativa al “ojo” en vez del área real, es decir, variar la teselación necesaria según la distancia del ojo a la superficie. Esto permite que a medida que nos acerquemos a la superficie la teselación aumente, funcionando de manera similar a un gráfico vectorial escalable (SVG), pero en \mathbb{R}^3 .

También tiene un punto negativo, y es que genera una excesiva teselación en zonas con una curvatura 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:

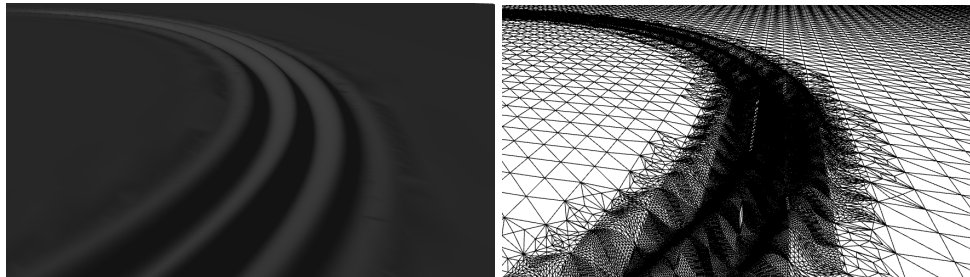


Figura 5.5.: Teselado basado en K.

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 importancia 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, requiriendo el mismo número de triángulos que en las figuras anteriores:

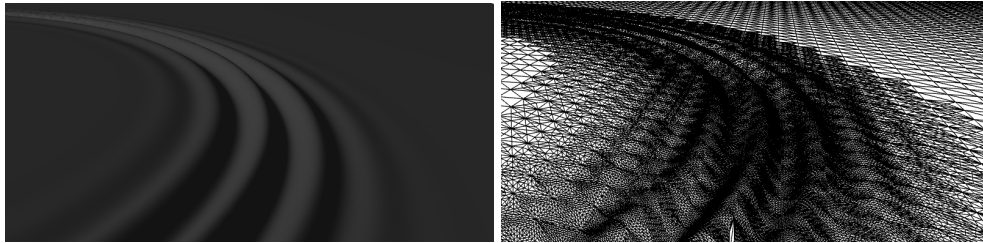


Figura 5.6.: Teselado con exponente α .

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:

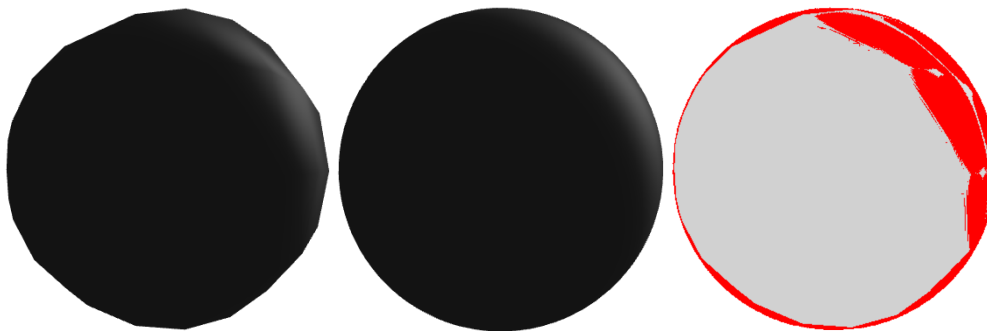


Figura 5.7.: Calidad visual según los bordes y texturas

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

5. Estudio de la teselación

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 la que está más cerca de la cámara. Además, podemos complementarlo añadiendo una distancia máxima de teselado, para evitar teselar zonas alejadas en superficies 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, mantendremos 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:

$$\langle N(u, v), \frac{P_L - p(u, v)}{|P_L - p(u, v)|} \rangle$$

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:

$$\langle p, q \rangle = |p||q|\cos(\angle(p, q))$$

Para tener una visión de 75° sería necesario un umbral de 0.793 aproximadamente. En la práctica 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 vértices 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 con tamaño mucho menor que el del triángulo inicial.

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 lenguaje 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 constantes (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  $[0, 1] \times [0, 1]$ 
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_sentencias><sentencia_plot>;
<lista_sentencias>	::=	<lista_sentencias><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, hemos definido " $f(u,v : \text{real}) : \text{vec3}$ " y estamos definiendo " $g(u,v : \text{real}) : \text{vec3} = fP_uP_v(u,v);$ ", donde " P_u " significa parcial respecto de la variable " u " y " P_v " la parcial respecto de " v " (conmutativo). Por tanto, podemos utilizar como llamada a una función el identificador " $\langle f \rangle \langle \text{sucesión de } P x_i \rangle$ ", con " f " función base ya definida.

7. Planificación y presupuesto

7.1. Planificación temporal inicial

Planificación temporal en fases			
Nº	Nombre	Tareas	
1	Inicial	- Estudio inicial del problema	
2	Implementación básica	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.	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.
3	Conexión inicial	- 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.	
4	Completar procesador	- 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.	
5	Conexión avanzada	- 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.	
6	Implementación avanzada	- Implementar la teselación uniforme mediante el uso del geometry shader. - Completar con una gran variedad de ejemplos. - Estudiar el correcto funcionamiento del teselado.	
7	Estudio de la mejor medida	- Elegir ciertas estadísticas del renderizado para comprobar si los resultados son buenos. - Estudiar varias medidas para controlar el nivel de teselado.	
8	Estética	- Completar la interfaz del usuario.	
9	Revisión	- Estudiar el correcto funcionamiento de ambas aplicaciones hasta ahora. Corregir y añadir elementos si fuese necesario.	

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 cálculo 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, normal y binormal 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.

8. Análisis y diseño

- 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. El siguiente diagrama es el de interacción de la acción “compilar”, que es la que implica mayor comunicación entre entidades.

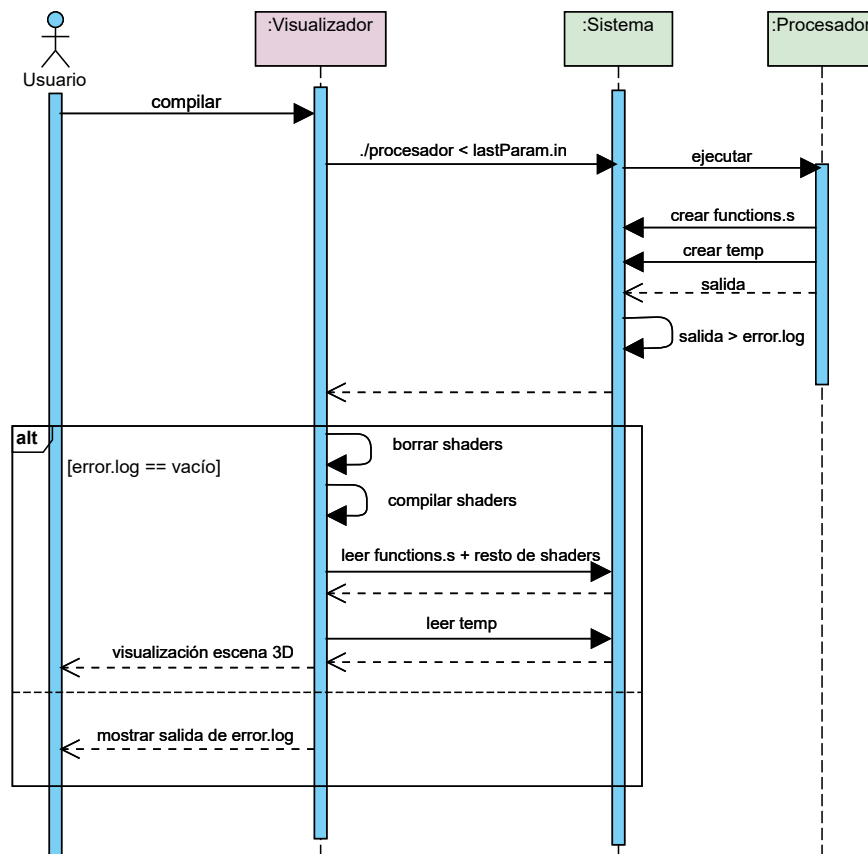


Figura 8.1.: Diagrama de interacción de la acción “compilar”.

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 8.2.: Boceto de la interfaz con los elementos principales.



Figura 8.3.: 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

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

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:

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

Pero es necesario definir la función $\frac{\partial f}{\partial x_i} \forall i$, así que se comprobaría si existe 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 $simplificar(n)$ 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 .

9. Implementación y pruebas

Tipo de shader para el teselado

En esta sección se justificará el uso del tessellation 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 desconexas.

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.

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 1. 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 realice el cálculo).

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

Procesador

La implementación del “procesador” nos se ha realizado desde cero, 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 LearnOpenGL [dV], que aportaban 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” [Aie] para mostrar un diálogo con un explorador de archivo y así poder seleccionar la parametrización deseada.

Generación de informes

Para medir 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, a partir de la latencia entre fotogramas (cuando se acumula más de 1 segundo de latencia, devuelve el nº de fps en ese 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

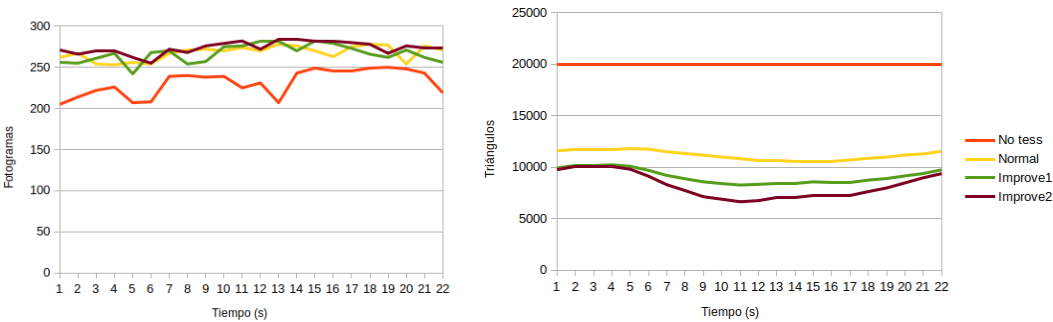


Figura 9.1.: Gráfica de la animación de rotación en gauss.in

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

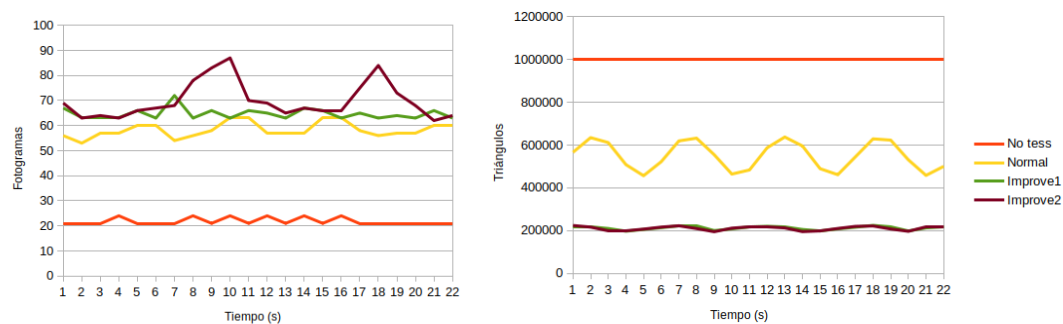


Figura 9.2.: Gráfica de la animación de rotación en waves.in

9. Implementación y pruebas

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 n° de triángulos es mínimo. Por tanto, las 4 imágenes debería ser exactamente iguales:

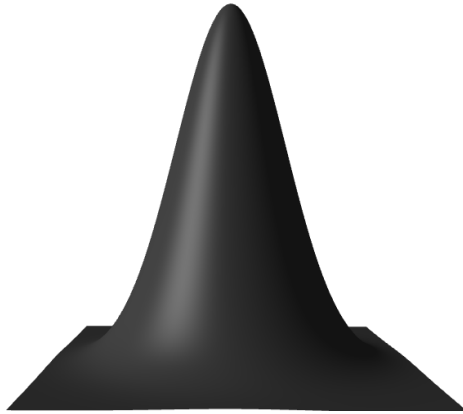


Figura 9.3.: gaussiana.in sin teselar

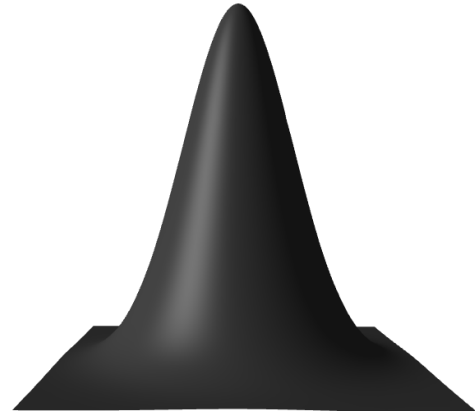


Figura 9.4.: gaussiana.in teselado normal

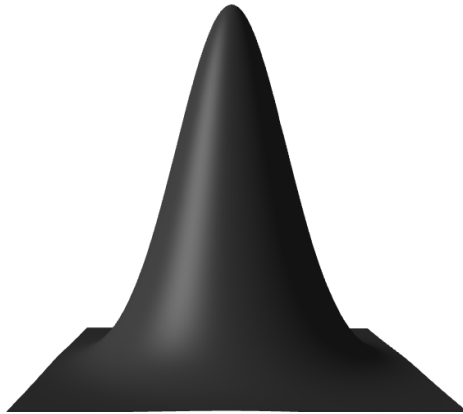


Figura 9.5.: gaussiana.in improve1

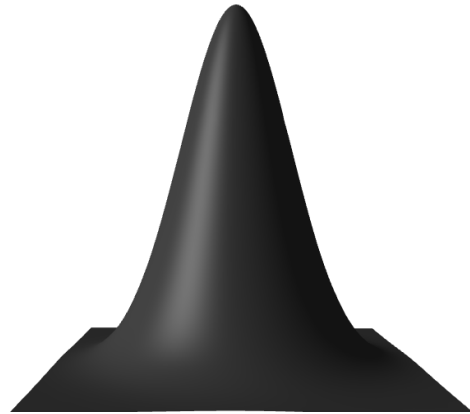


Figura 9.6.: gaussiana.in improve2

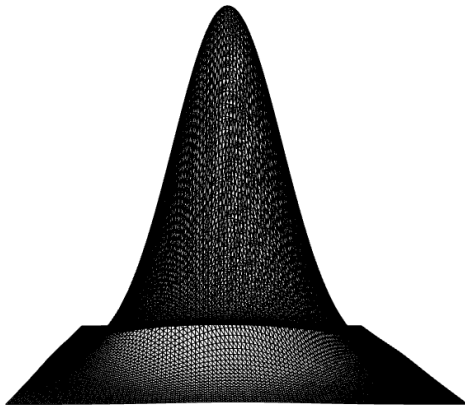


Figura 9.7.: gaussian.in sin teselar

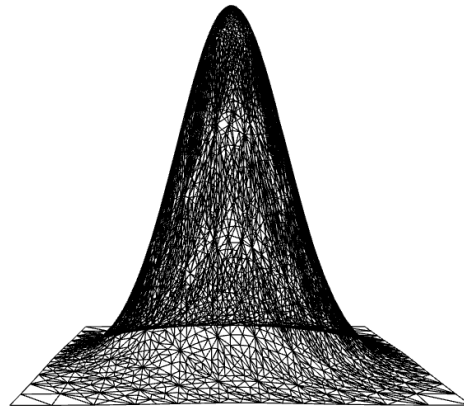


Figura 9.8.: gaussian.in teselado normal

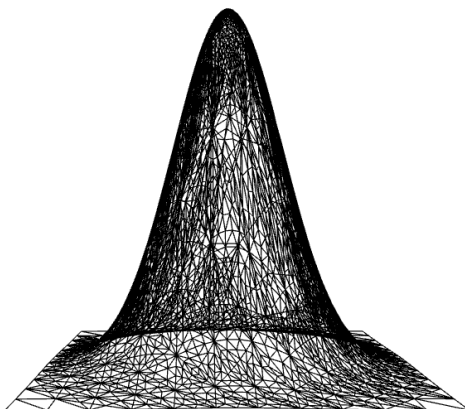


Figura 9.9.: gaussian.in improve1

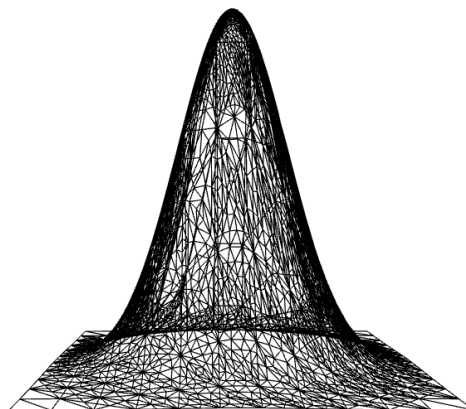


Figura 9.10.: gaussian.in improve2

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.

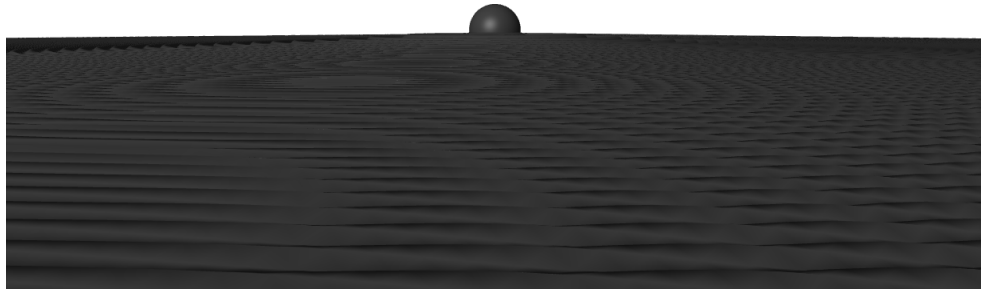


Figura 9.11.: waves.in sin teselar

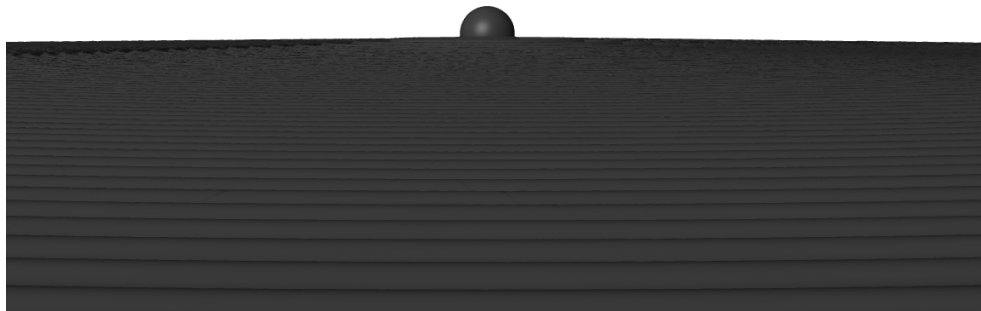


Figura 9.12.: waves.in teselado normal

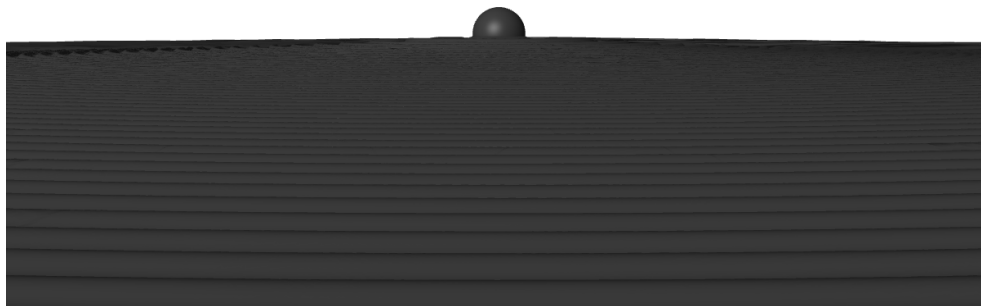


Figura 9.13.: waves.in improve1

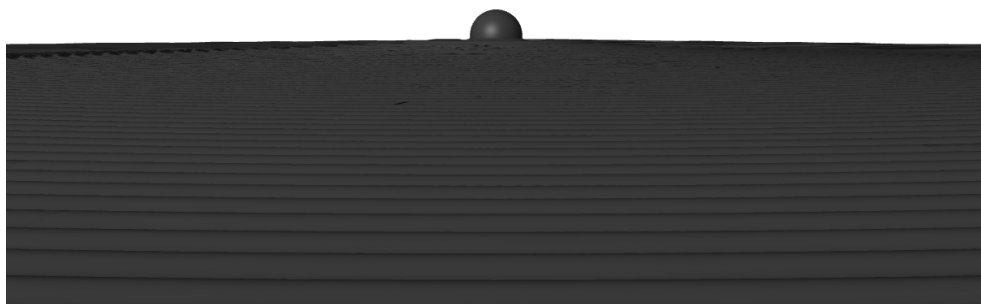


Figura 9.14.: waves.in improve2

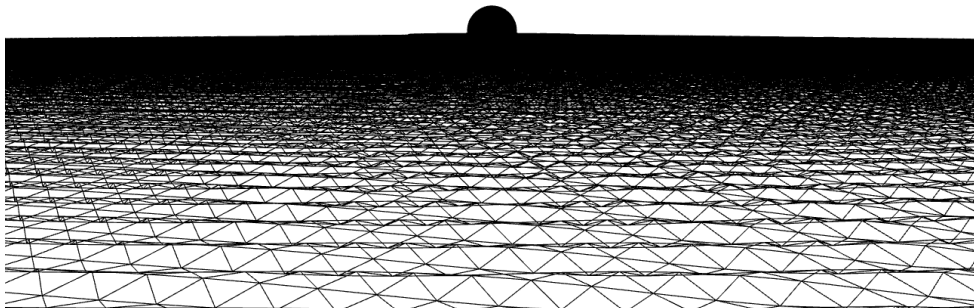


Figura 9.15.: waves.in sin teselar



Figura 9.16.: waves.in teselado normal

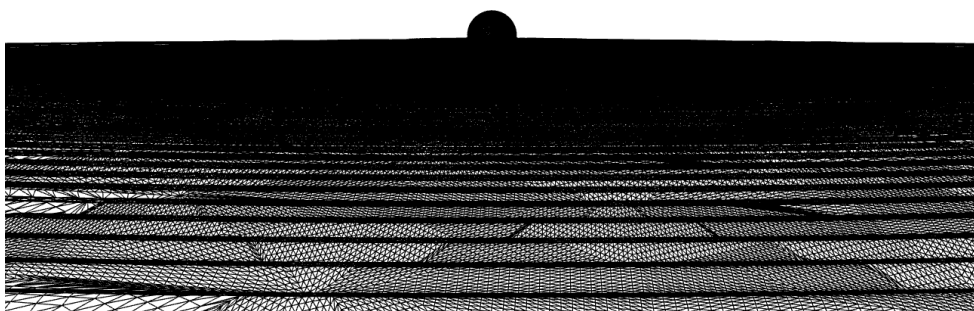


Figura 9.17.: waves.in improve1

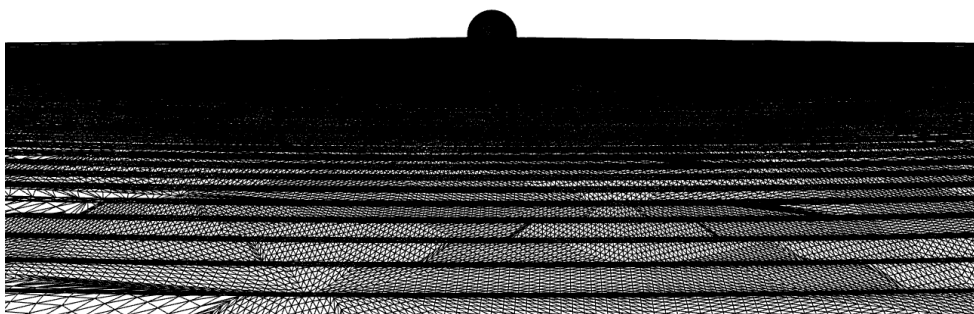


Figura 9.18.: waves.in improve2

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).
- 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 [\[FdIH\]](#) 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 estén actualizadas para el comando “apt” y sea necesario instalarlo manualmente desde sus respectivas páginas oficiales.

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 “procesador”. 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:

- 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 “variedades”.
 - Cambiar el tamaño de la malla de partida, requiriendo su posterior actualización manual (botón contiguo).
 - 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”).

B. Guía de uso del programa

- Activar/desactivar la auto-rotación (rotación 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.

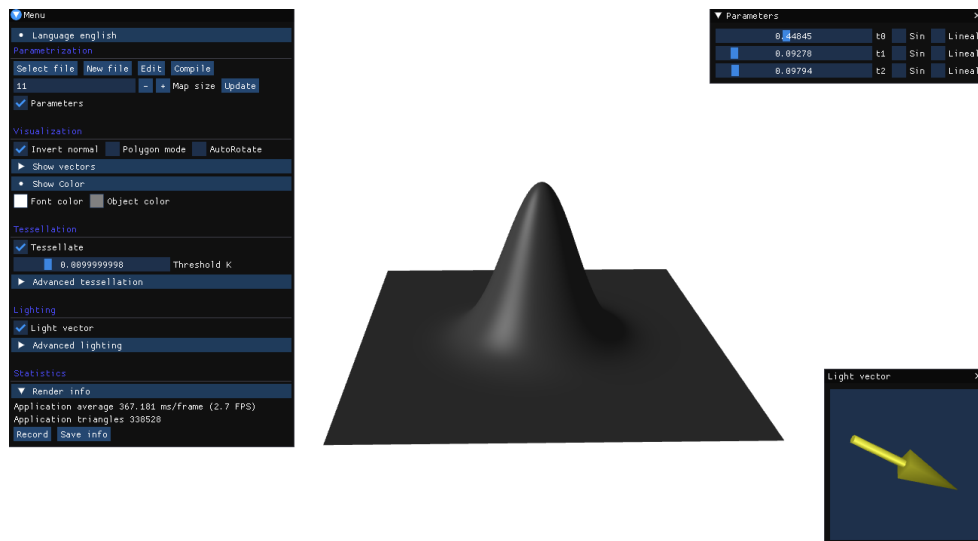


Figura B.1.: Ventanas iniciales.

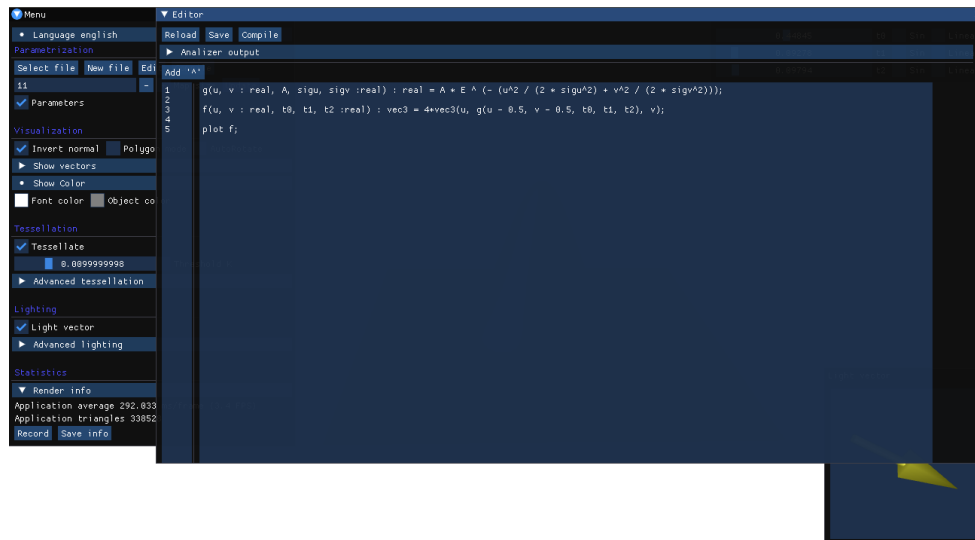


Figura B.2.: Ventana de edición de código.



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

B. Guía de uso del programa

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.

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, junto con el “Teorema de Alisamiento de Asas”, son los que verdaderamente requieren un amplio conocimiento de la teoría de Morse. La herramienta aportada por Allen Hatcher, el “truco del Toro”, ha sido la clave para la demostración del “Teorema de Alisamiento de Asas”.

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 implementación 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 ha sido en vano, ya que ha facilitado el entendimiento del 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).

Vías futuras

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.

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).

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 y 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 $p = \phi(u, v)$ y p es un valor regular de ϕ , entonces p es un punto crítico de f .

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

A priori, el programa implementado sólo 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 (visualizar el objeto real cuando esté cerca y la malla inicial 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 complementarias, independientemente del tamaño de la malla

inicial (actualmente las zonas planas están trianguladas según la malla inicial).

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

- [Aie] Aiekick. 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 51]
- [Chao0] Abhijit Champanerkar. 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, xx, 3, and 8]
- [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]
- [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 and 51]
- [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. 25 and 51]
- [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. 28]
- [FdIH] Norberto Fernández de la Higuera. Visualizador de Superficies. <https://github.com/norbertofdz/TFG>. Repositorio de GitHub del proyecto donde se encuentra la implementación del programa. Accedido el 30 de agosto de 2021. [Citado en pág. 59]
- [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 xx]
- [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, xx, 25, and 50]
- [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 51]
- [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 triangulación de E. E. Moise, para el caso sin borde. [Citado en pág. xix]

Bibliografía

- [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. 26]