Universidad de Granada

Departamento de Lenguajes y Sistemas Informáticos

Modelado de Sólidos de Forma Libre Basado en Superficies Paramétricas Triangulares de Grado Bajo

memoria presentada por

Ángel Luis García Fernández

para la obtención del grado de Doctor

dirigida por los doctores

Juan Ruiz de Miras y Francisco Feito Higueruela

Granada, diciembre de 2007

Editor: Editorial de la Universidad de Granada
Autor: Ángel Luis García Fernández
D.L.: Gr. 2626 - 2007
ISBN: 978-84-338-4666-2

2
2
2

3

La memoria “Modelado de Sólidos de Forma Libre Basado en Superficies Paramétricas Triangulares de Grado Bajo” que presenta D. Ángel Luis García Fernández para optar al grado de Doctor ha sido realizada dentro del programa de doctorado “Métodos y Técnicas Avanzadas de Desarrollo de Software” del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada, bajo la dirección de los doctores Juan Ruiz de Miras y Francisco Feito Higueruela, del Departamento de Informática de la Universidad de Jaén.

Granada, diciembre de 2007

El doctorando

Ángel Luis García Fernández

Los directores

Juan Ruiz de Miras

Francisco Feito Higueruela

3

4
4
4

5

Agradecimientos

Mucho tiempo ha pasado desde que empecé a trabajar en el tema de mi tesis. Ahora es el momento de revisar todo ese tiempo y reconocer públicamente a aquellos a los que debo gran parte de lo que soy y donde estoy.

En primer lugar, quiero dar las gracias a mi familia: mis padres Andrés y Antoñita y mis hermanos, Isabel, Antonio, Andrés y Diego, y demás parientes. Ellos han estado siempre a mi lado, a veces sin que me diera cuenta, dándome ánimos para seguir. Me han enseñado desde siempre y cada uno con sus armas (el deporte, la fe, los chistes, las conversaciones hasta las tantas, a veces los enfados...) la responsabilidad, la constancia, el afán de superación, la humildad, la capacidad de sacrificio, la alegría... el ser ante todo una persona. Espero haber sido un alumno aceptable, y perdonadme por las veces que he pagado con vosotros mis malos ratos.

También quiero dar las gracias a mis amigos y compañeros que me han ayudado en lo que han podido. Mi reconocimiento especial a mi amigo Carlos (sigo pensando que eres el mejor programador que conozco... aunque tú digas que conozco a muy poca gente), a mi compañero de despacho, Francisco (ojalá alguna vez me consiga acercar a lo buen profesor que tú eres), a Rosa y Anand (gracias por echarme una mano con el inglés mientras acunábais a vuestro peque) y a los "tres mosqueteros": Juan Gualberto, Juan Antonio y Miguel Ángel. También a mis compañeros del Departamento de Informática de la Universidad de Jaén, que seguro que es el lugar donde más tiempo he pasado estos últimos años; gracias por todas esas veces en que me he sentido a gusto con vosotros.

Parte de este trabajo se realizó en el Instituto Max Planck de Informática de Saarbrücken (Alemania) y en la Texas A&M University de College Station (Estados Unidos). Gracias a los profesores Seidel y Keyser, que me acogieron y ayudaron en la medida de sus posibilidades.

Mi trabajo ha estado parcialmente subvencionado por una beca de formación de personal docente e investigador de la Junta de Andalucía, así como por los proyectos de investigación TIC2001-2099-C03-03 y TIN2004-06326-C03-03, subvencionados por los ministerios de Ciencia y Tecnología y Educación del gobierno de España. Así mismo, he disfrutado de las instalaciones de la Universidad de Jaén todos estos años. Gracias a dichas instituciones por su ayuda.

5

6

Tanto la labor de programación como la preparación del contenido de esta memoria ha sido realizada íntegramente en sistemas GNU/Linux. Gracias a la comunidad de Software Libre por defender sus ideas contra viento y marea.

También quiero agradecer desde estas líneas al profesor Robert Joan Arinyo su exhaustiva revisión del borrador de este documento, así como a los profesores Ioannis Ivrissimtzis, de la Universidad de Durham (Reino Unido) y Fabio Ganovelli, del CNR (Italia) por sus informes y comentarios sobre mi trabajo.

Por último, y no por ello menos importante, quiero agradecer desde estas páginas a mis directores, Juan y Paco, su paciencia conmigo, sus consejos, su disponibilidad para escucharme en mis lamentaciones y acompañarme en mis alegrías, así como todas las oportunidades que me han brindado en estos años para ir avanzando siempre en mi trabajo.

Sé que me dejo cosas en el tintero, pero si las incluyera todas tendría que dividir este libro en al menos dos volúmenes. Gracias a todos.

6

Índice

Lista de Figuras

Lista de Algoritmos

Lista de Tablas

23

1 Introducción

Se presenta aquí este trabajo, describiendo qué lo ha motivado, los objetivos propuestos, los resultados esperados y las aportaciones realizadas, junto con la metodología de trabajo seguida. Por último, se describe de forma breve el contenido de los siguientes capítulos.

1.1 Motivación

El Modelado de Sólidos [Sha02] trata cómo representar y manipular información sobre objetos tangibles existentes o en fase de diseño, de forma que se puedan comprobar a priori características como la resistencia de los materiales, su adaptación al uso para el que se diseñan o su apariencia final después de su fabricación, por ejemplo. Relaciona disciplinas como las Matemáticas, la Física, la Óptica o la Informática, y puede decirse que hoy en día sigue habiendo problemas a los que no se ha dado una solución totalmente satisfactoria por parte de los investigadores de todo el mundo que tratan este tema.

Uno de los campos abiertos en esta disciplina es el modelado de sólidos de forma libre, es decir, aquellos que están delimitados por superficies curvas. Las especiales características de este tipo de sólidos complican su representación frente a los delimitados por caras planas. Es por esto por lo que este trabajo pretende presentar una posible alternativa para el tratamiento de sólidos de forma libre, basándose en la utilización de parches paramétricos (parches triangulares de Bézier cúbicos [Far86]) y Cadenas Simpliciales Extendidas [RF99a].

El modelo de Cadenas Simpliciales Extendidas es relativamente reciente, y está específicamente orientado a la representación de sólidos de forma libre. Su utilidad ha sido probada tanto con sólidos bidimensionales delimitados por cónicas y curvas de Bézier cúbicas [RF97] como con sólidos tridimensionales

23
23

delimitados por parches algebraicos [RF99a, RF99b, RF98, RF02]. Por otra parte, las superficies paramétricas son el estándar de facto en las aplicaciones comerciales de CAD/CAM ya que facilitan en gran medida el modelado y los cálculos sobre las representaciones. Estos hechos motivan la extensión del modelo de Cadenas Simpliciales Extendidas a sólidos delimitados por parches paramétricos como una forma de conjugar las bondades de dicho modelo con la comodidad que aporta ese tipo de superficies; es de esperar que se mantengan las propiedades del modelo de representación y se facilite el tratamiento de las superficies que delimitan los sólidos.

1.2 Objetivos

Como se ha indicado en el apartado anterior, el objetivo fundamental de esta tesis es extender el modelo de Cadenas Simpliciales Extendidas [RF99a] a sólidos de forma libre delimitados por parches paramétricos triangulares, demostrando así que el cambio en el tipo de parches no influye en los algoritmos básicos del modelo.

Este objetivo fundamental se puede descomponer en los siguientes objetivos parciales:

- Extender el modelo de Cadenas Simpliciales Extendidas, que originalmente fue ideado para ser utilizado en sólidos bidimensionales limitados por cónicas y curvas de Bézier y posteriormente ampliado a sólidos tridimensionales delimitados por superifices algebraicas, a modelos tridimensionales delimitados por parches paramétricos triangulares.

- Desarrollar algoritmos para el cálculo de la inclusión de un punto en un sólido y la visualización de modelos.

- Diseñar algoritmos para el cálculo de operaciones booleanas evaluadas y no evaluadas.

1.3 Resultados esperados

Los resultados esperados al comenzar este trabajo eran los siguientes:

- La formalización del concepto de celda de forma libre, pilar básico del modelo de Cadenas Simpliciales Extendidas, utilizando parches paramétricos triangulares.

- Un diseño robusto del algoritmo para el estudio del test de inclusión de un punto en un sólido representado utilizando una cadena simplicial extendida.

- Aplicaciones del algoritmo para el cálculo del test de inclusión habituales en Modelado de Sólidos, como la visualización mediante trazado de rayos.

- Diseño de algoritmos que permitan obtener el resultado de la aplicación de operaciones booleanas sobre sólidos representados por cadenas simpliciales extendidas.

1.4 Aportaciones

La principal aportación que se pretende obtener de este trabajo es un esquema formal para la representación y manipulación de sólidos tridimensionales de forma libre delimitados por parches paramétricos triangulares, que permita modelar sólidos independientemente de su aspecto y no se ciña sólo a la representación de la frontera, sino también a la del volumen de los mismos.

Al representar no sólo la frontera, sino también el volumen de los sólidos, se deja abierta la puerta a aplicaciones vinculadas a la representación de volumen, como cálculos de propiedades, generación de prototipos utilizando impresoras 3D, detección de colisiones, etcétera.

1.5 Cronología de trabajo

Los trabajos preliminares comenzaron en el año 2001, con la adaptación del modelo de Cadenas Simpliciales Extendidas a parches paramétricos [GRF03].

Posteriormente, el algoritmo para el cálculo del test de inclusión de un punto en un sólido representado con el nuevo modelo centró los esfuerzos de diseño e implementación [GRF04a].

Finalmente, el algoritmo para el cálculo de la inclusión desarrollado permitió enfocar el trabajo hacia el tratamiento de las operaciones booleanas entre sólidos de forma libre [GRF05]. Al trabajar en esta dirección surgió de forma natural el cálculo del recorte de los parches triangulares utilizados para representar los sólidos, lo que hizo necesario desarrollar una solución adecuada al modelo de representación utilizado ante la ausencia de trabajos similares en la bibliografía estudiada [GRF06].

1.6 Estructura de este documento

El resto de este documento desarrolla los siguientes contenidos:

- El Capítulo 2 se dedica al Modelado de Sólidos en general, estudiando métodos de modelado de referencia y su posible aplicación a sólidos de forma libre.

- El Capítulo 3 describe el modelo de Cadenas Simpliciales Extendidas: sus antecedentes, los elementos en los que se basa y las operaciones que se pueden realizar entre modelos.

- En el Capítulo 4 se trata la adaptación del modelo de Cadenas Simpliciales Extendidas a sólidos delimitados por parches triangulares de Bézier.

- El Capítulo 5 se destina al algoritmo para el cálculo del test de inclusión de un punto en un sólido de forma libre representado mediante una cadena simplicial extendida en general, centrándose después en el caso de un sólido delimitado por parches triangulares de Bézier.

- El Capítulo 6 versa sobre la representación y tratamiento de las operaciones booleanas regularizadas entre sólidos modelados utilizando cadenas simpliciales extendidas.

- Por último, el Capítulo 7 se ocupa en la exposición de conclusiones y propuestas de trabajos futuros en este campo de investigación.

- El Apéndice A hace una descripción somera de la aplicación software en la que se han implementado los algoritmos que se presentan en este trabajo.

- En el Apéndice B se incluye un amplio resumen en inglés, en cumplimiento de uno de los requisitos para la obtención de la mención de Doctorado Europeo.

28

28
28

29

2 El Modelado de Sólidos

Este capítulo se centrará en el Modelado de Sólidos como ámbito en el que se engloba este trabajo, prestando especial atención al modelado de sólidos de forma libre. En primer lugar se estudiará el concepto de Modelado de Sólidos a través de las definiciones de varios autores para, a continuación, estudiar algunos de los más comunes sistemas de modelado desarrollados a lo largo del tiempo y su aplicación al modelado de sólidos de forma libre. Finalmente, se discuten propuestas de otros investigadores especialmente pensadas para el modelado de sólidos de forma libre.

2.1 Concepto de Modelado de Sólidos

El Modelado de Sólidos es una disciplina relativamente joven. Su inicio y desarrollo han estado siempre ligados al diseño y fabricación asistida por computador (CAD/CAM), como un medio para resolver la necesidad de modelar no sólo la superficie del sólido, sino también su interior y sus propiedades. Esta necesidad se podría regir por estas tres condiciones [Sha02]:

1. Cualquier modelo ha de corresponderse con un objeto físico existente o susceptible de existir.

2. Los modelos construidos han de representar sin ambigüedades el objeto modelado.

3. El esquema de representación debería permitir todas y cada una de las operaciones geométricas que se puedan realizar sobre el objeto físico.

29
29

Según estas condiciones, un mismo objeto físico puede ser modelado de distintas formas válidas, y todas ellas han de ser consistentes respecto al sólido.

Pero ¿qué es el Modelado de Sólidos?. En los siguientes párrafos se recogen algunas definiciones tomadas de la bibliografía estudiada:

> *"El Modelado de Sólidos es un conjunto consistente de principios para el modelado matemático y por computador de sólidos tridimensionales"*
>
> *[Sha02]*

> *"El Modelado de Sólidos es un conjunto en rápida expansión de teoría, algoritmos y técnicas software que tratan la representación, diseño, visualización y análisis de modelos tridimensionales"*
>
> *[RTA93]*

> *"El objetivo del Modelado de Sólidos es representar, manipular y razonar, utilizando computadoras, sobre la forma tridimensional de los objetos sólidos físicos"*
>
> *[HR96]*

En otras referencias estudiadas no se encuentra una defición concreta, sino que recurren a su finalidad y sus campos de aplicación para dar una idea intuitiva del Modelado de Sólidos, y tomarla como punto de partida para explicar las características que ha de cumplir un sistema de modelado y analizar las técnicas conocidas hasta ese momento [FvDFH92, Req80]. Otros autores prefieren hablar de Modelado Geométrico, definiéndolo como:

> *"Colección de métodos matemáticos interrelacionados, aunque poco integrados, que utilizamos para describir la forma de un objeto o expresar algún proceso físico en términos de una metáfora geométrica apropiada"*
>
> *[Mor97]*

> *"El Modelado Geométrico se ocupa de la representación y manipulación de objetos geométricos en un computador"*
>
> *[Rap95]*

En general, el concepto de Modelado Geométrico es un poco más amplio que el de Modelado de Sólidos, aunque ambos están íntimamente relacionados, puesto que la geometría del objeto que se quiere representar es algo que debe contener un modelo de un sólido [Män88]. Según Rappoport [Rap95], el Modelado Geométrico incluye la base de lo que él denomina Computación Geométrica, que comprende las áreas teóricas y de aplicación de la Informática referentes a la geometría y la visualización: gráficos y animación por computador, diseño y fabricación asistido por computador (CAD/CAM), robótica, visión artificial, geometría discreta computacional y diseño geométrico asistido por computador (CAGD). Otros autores [Mor97] afirman que el Modelado Geométrico comprende el diseño geométrico asistido por computador (CAGD), geometría algebráica, geometría computacional y modelado de sólidos, entendido como el esquema de representación CSG.

El objetivo fundamental del Modelado de Sólidos es facilitar, y en algunos casos posibilitar, el diseño, estudio y/o fabricación de objetos físicos a través de la representación y manipulación de su geometría y sus propiedades en soporte informático. Para ello, se recurre a la construcción de *modelos* de dichos objetos.

Un ***modelo*** es una representación analítica y abstracta, que ha de proporcionar la suficiente información para resolver el problema que ha motivado su elaboración. En muchos casos se realizan simulaciones sobre un modelo de un objeto que todavía no existe físicamente para comprobar su comportamiento (resistencia, flexibilidad, ligereza, aerodinámica, conductibilidad del calor, la electricidad o la luz, ...) antes de iniciar su fabricación, o bien sí existe y no es posible o resulta muy costoso realizar el estudio del objeto *in situ*. Otras veces se requiere calcular propiedades volumétricas, como el centro de masas o la densidad. También puede ser necesario estudiar los efectos de posibles modificaciones en objetos ya existentes, para comprobar las posibles interacciones (colisiones, producción de calor, ...) entre dos o más objetos, o bien para estudiar procesos que en el mundo real tardarían mucho tiempo o resultarían peligrosos (estudio del proceso de oxidación de metales, de la erosión, de la desintegración de materiales radiactivos, ...). Todas estas situaciones son ejemplos en los que se puede aplicar el Modelado de Sólidos.

2.2 Concepto de sólido

En 1980, Requicha [Req80] presentó un marco conceptual para el Modelado de Sólidos que fue masivamente aceptado y hoy en día sigue considerándose plenamente válido. En su artículo, los objetos físicos del mundo real son modelados como sólidos abstractos utilizando subconjuntos especiales del espacio euclídeo $\mathbb{E}^3$, descritos mediante esquemas de representación. Todo el trabajo en el soporte informático se hace a partir de las descripciones, también llamadas modelos, generadas aplicando dichos esquemas.

En el mismo artículo se establecen las condiciones que debe cumplir un subconjunto cualquiera de $\mathbb{E}^3$ para que sea considerado un sólido abstracto:

- **Rigidez**: implica que su forma no depende de la posición ni la orientación del mismo.

- **Homogeneidad dimensional**: tiene que tener interior, y su frontera no puede tener partes aisladas o "colgando" (figura 2.1).

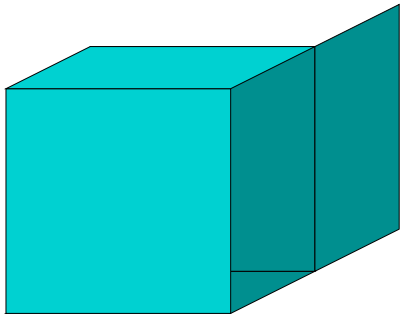


Figura 2.1 Ejemplo de un subconjunto no homogéneo

- **Finitud**: debe ocupar una porción finita del espacio.

- **Clausura bajo movimientos rígidos (traslaciones y rotaciones) y algunas operaciones booleanas que añadan o eliminen material**: el resultado debe seguir siendo un sólido abstracto.

- **Descriptibilidad finita**: debe ser representable en un computador, por tanto, ha de permitir su descripción con un número limitado de primitivas (polígonos, parches, ...).

- **Determinismo en la frontera**: la frontera diferencia claramente lo que es el interior del sólido abstracto de lo que es el exterior.

Requicha afirma que las entidades que mejor se ajustan a estas condiciones son las que él denomina *r-sets*: subconjuntos de $\mathbb{E}^3$ limitados (ocupan una parte finita del espacio), cerrados (contienen a su frontera), regulares (iguales a la clausura de su interior) y semianalíticos (descriptibles mediante combinaciones booleanas de desigualdades del tipo $f_i(x,y,z) \geq 0$, siendo f_i funciones analíticas: admiten desarrollo en series de Taylor alrededor de cualquier punto del espacio).

2.2.1 Definición de sólido de forma libre

Los sólidos de forma libre son un tipo concreto de sólidos que tienen como característica especial el hecho de que su frontera no está constituida por caras planas, sino por superficies curvas. Algunos de estos sólidos se pueden representar fácilmente de manera analítica, como las esferas, los cilindros, los toros, etcétera. Otros, sin embargo, tienen un aspecto totalmente irregular que hace imposible su definición analítica, por lo que si se desea modelarlos, es necesario recurrir a algún método de modelado que proporcione una representación válida.

A los elementos no planos que actúan como delimitadores de los sólidos de forma libre se les denominará ***elementos de forma libre*** [Rui01], en contraposición a los elementos planos que delimitan a los sólidos poliédricos.

2.3 Esquemas de representación de sólidos

Siguiendo con el formalismo de Requicha [Req80], un esquema de representación de sólidos **s** es un conjunto de elementos léxicos y las reglas sintácticas para combinarlos en una descripción. El esquema así definido establece una correspondencia entre dos conjuntos: el conjunto formado por todos los sólidos abstractos **M**, y el conjunto de todas las descripciones sintácticamente correctas **R** (figura 2.2).

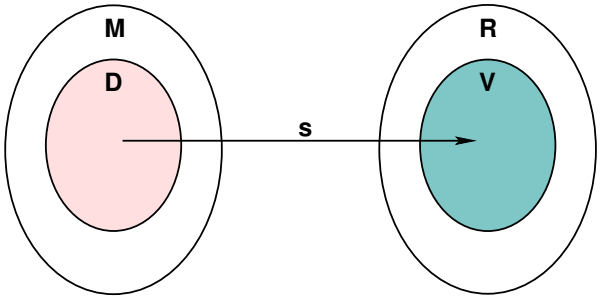


Figura 2.2 Diagrama de la correspondencia que define un esquema de representación de sólidos **s** según Requicha

En la correspondencia que define **s** se distinguen los subconjuntos **D** y **V**. El primero de ellos, **D**, se corresponde con el conjunto de sólidos representables mediante el esquema **s**, y se le denomina *dominio de* ***s***; el segundo conjunto, **V**, se corresponde con el conjunto de descripciones sintácticamente correctas según **s** que también son semánticamente correctas, esto es: se corresponden con un sólido.

Se dice que una representación $r \in \mathbf{V}$ es no ambigua o completa si se corresponde con un único elemento $m \in \mathbf{D}$. Así mismo, r es única si m no se puede representar de otra manera bajo el esquema **s**. Si todos los elementos de **V** son representaciones no ambiguas, el esquema **s** también lo es, e igualmente, si todos los elementos de **V** son representaciones únicas, **s** también será un esquema único. Un esquema único no es siempre completo, y viceversa: un esquema completo no tiene que ser necesariamente único.

Anteriormente se ha recordado que el Modelado de Sólidos surge por una necesidad de obtener mediante computadores información sobre objetos existentes o susceptibles de existir. Dicha información será el resultado de una serie de operaciones sobre representaciones de los sólidos construidas según un esquema de representación. En muchos casos, un esquema de representación de sólidos puede resolver las necesidades de información sobre los objetos a estudiar, pero no siempre es así, pudiendo hacerse necesario combinar dos o más esquemas para satisfacer todas las necesidades del problema de aplicación.

Requicha también establece una serie de propiedades deseables para que los esquemas de representación puedan proporcionar la mayor cantidad posible de

información. Estas propiedades se dividen en dos grupos: propiedades formales y propiedades informales.

2.3.1 Propiedades formales

Las propiedades formales de un esquema de representación de sólidos se refieren a las características de los conjuntos de la figura 2.2. Se definen de la siguiente forma:

- **Dominio**: el dominio de un esquema **s** es el conjunto de sólidos representables utilizando este, y se representa mediante el subconjunto **D** en la figura 2.2. Cuanto más próximo sea este conjunto al conjunto de sólidos abstractos **M**, mayor expresividad tendrá el esquema, y por tanto, mayor será su utilidad.

- **Validez**: todo esquema de representación ha de disponer de un mecanismo para comprobar la validez de las representaciones, o lo que es lo mismo, una forma de discriminar el subconjunto $\mathbf{V} \subseteq \mathbf{R}$ de la figura 2.2. Algunas formas de conseguirlo son: diseñar algoritmos de comprobación, establer restricciones de validez en el proceso de construcción de modelos, o bien crear un esquema de representación que siempre produzca modelos válidos, verificándose entonces que $\mathbf{V} \equiv \mathbf{R}$.

- **Completitud**: una representación $r \in \mathbf{V}$ es completa si se corresponde con un único elemento $m \in \mathbf{D}$. Un esquema de representación es completo si todas las representaciones posibles utilizando dicho esquema son completas.

- **Unicidad**: cada representación distinta se corresponde con un sólido diferente. Esto permite comparar sólidos simplemente mediante el estudio sintáctico de sus representaciones, sin necesidad de realizar una labor de comprensión de las mismas. Como consecuencia, dicha comparación se puede plasmar más o menos fácilmente en un algoritmo ejecutable en un computador.

2.3.2 Propiedades informales

Las propiedades informales caracterizan en mayor medida la usabilidad de los esquemas de representación, y no son tan fácilmente valorables como las formales:

- **Concisión**: entendida como el tamaño de las representaciones en memoria. En general es deseable que las representaciones ocupen poco espacio, de manera que se puedan almacenar y manejar en la memoria de un computador fácilmente.

- **Facilidad de creación de modelos**: las características del esquema de representación han de posibilitar que la creación de modelos en una aplicación basada en él sea sencilla.

- **Eficacia en el contexto de aplicación**: las representaciones obtenidas han de permitir la obtención de algoritmos correctos, eficientes y robustos.

2.4 Esquemas clásicos de representación de sólidos

El Modelado de Sólidos es un campo de trabajo en continua evolución. No son pocos los algoritmos propuestos cada año en las revistas especializadas, así como los análisis y revisiones de algoritmos y esquemas de modelado ya existentes que tratan de mejorar las prestaciones de los hasta ahora conocidos. No obstante, hay una serie de esquemas que por unas u otras razones se han mostrado especialmente útiles, formando un núcleo de métodos "clásicos" para el modelado de sólidos. Aquí se van a reseñar brevemente algunos de ellos, considerando su posible aplicación al modelado de sólidos de forma libre, que son el objeto de este trabajo.

2.4.1 Instanciación de primitivas

La instanciación de primitivas [Män88, MNK90, SV95] se basa en la utilización de un amplio conjunto de modelos predefinidos de piezas sencillas, algunas de cuyas características geométricas son modificables a través de unos parámetros. Para realizar un diseño utilizando este método, basta con ir seleccionando las primitivas una a una, indicando su posición, orientación y los valores de los

parámetros para concretar su forma definitiva. En la figura 2.3 se muestra un ejemplo de primitiva parametrizable y una posible instancia de la misma.

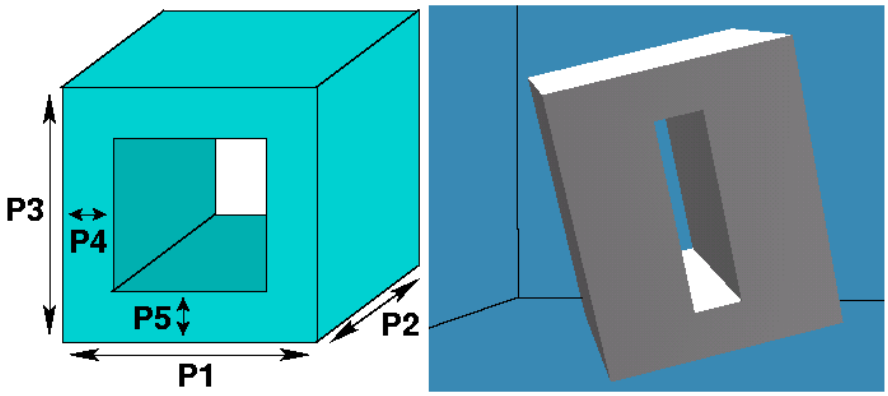


Figura 2.3 Primitiva con cinco parámetros, y un ejemplo de instanciación

En la mayoría de los casos es muy sencillo y rápido crear modelos con este método, aunque el grado de expresividad está limitado por el número de primitivas y por la inexistencia de medios para combinar las instancias. Además, hay que tener especial cuidado con la parametrización, puesto que puede dar lugar a sólidos no válidos; es preciso indicar claramente lo que significa cada parámetro, como por ejemplo en la figura 2.3, donde no queda suficientemente claro si P4 y P5 influyen en el tamaño del hueco, o sólo influyen en la posición del mismo. Además, hay que controlar que no se proporcionen valores inválidos a los parámetros: dada la primitiva de la figura 2.3, y suponiendo que P4 indica el grosor de las paredes laterales, tal y como se ve en la instanciación, dicho parámetro no puede tener un valor menor de cero ni mayor de la mitad de P1; del mismo modo, y suponiendo que P5 define el grosor de las paredes superior e inferior, dicho parámetro no puede tener un valor menor de cero ni mayor de la mitad de P3. Esta técnica es útil en entornos de diseño industrial donde se trabaja a partir de piezas estándar como tornillos, tuercas, tubos, engranajes, etc. para construir nuevos objetos, y donde este tipo de técnica puede llegar a ser muy eficaz.

Un inconveniente de esta técnica es que el cálculo de propiedades del sólido, como pueden ser el volumen o la densidad, no puede en general hacerse de manera uniforme para cada primitiva, sino que junto con la definición de cada una de ellas que se incorpora a la aplicación de modelado, se han de incluir los algoritmos para el cálculo de sus propiedades. Esto hace que sea costoso

el desarrollo de un conjunto amplio de primitivas que proporcionen suficiente expresividad al sistema.

Un enfoque similar es el modelado basado en características [BJ93], donde los sólidos están definidos por la combinación de una serie de elementos denominados características. Este sistema de modelado contempla tres formas de construir los modelos:

1. Modelos de características de frontera: en este tipo de modelos, las características que se combinan pueden ser conjuntos de caras que forman volúmenes cerrados, y que se denominan *características de volumen*, o bien conjuntos de caras que no forman volúmenes cerrados, llamados *características de superficie*.

2. Modelos de características CSG: las características que se combinan son primitivas similares a las descritas anteriormente, y se utilizan los operadores de conjuntos clásicos en el proceso de construcción del modelo [RV77].

3. Modelos de características de semiespacios: los modelos se construyen mediante la combinación de semiespacios. Un semiespacio o conjunto de semiespacios define una característica en este caso.

En el modelado basado en características, el usuario selecciona las características a utilizar, y establece los valores adecuados para cada uno de los parámetros que definen la característica, como pueden ser el grado de curvatura, longitud de una protrusión, etcétera, así como la forma en que se unirá con el resto de características para formar el modelo del sólido objetivo.

Aplicabilidad a sólidos de forma libre

La instanciación de primitivas no es en general una solución aceptable para modelar sólidos de forma libre, puesto que implicaría tener una primitiva correspondiente a cada sólido que quisiéramos representar. Esto podría ser válido en aquellos casos en los que el conjunto de sólidos a modelar fuera conocido y su cardinalidad hiciera factible el desarrollo de una aplicación de estas características.

La aplicación del modelado basado en características a la representación de sólidos de forma libre [vdBBV02] está condicionada, al igual que la instanciación simple de primitivas, a la cantidad de patrones disponibles para su instanciación, siendo recomendable incorporar en el sistema de modelado las herramientas necesarias para que el diseñador pueda añadir nuevas características de forma libre, definiendo su aspecto y sus parámetros, así como la forma en la que se une con otras características. Todo esto hace que, aunque pueda ser cómodo e intuitivo trabajar a partir de patrones ya existentes, el diseñador sienta que pierde eficiencia en su trabajo al tener que crear nuevos patrones.

2.4.2 Esquemas basados en división espacial

Los sistemas de partición espacial se basan en la descomposición del espacio ocupado por el sólido que se desea modelar en elementos más sencillos , aplicando al modelado un enfoque *divide y vencerás*. Dependiendo del tipo de descomposición que se aplique, la construcción puede presentar pérdidas de precisión.

2.4.2.1 Enumeración

La enumeración espacial [KCY93, Män88] es un esquema de modelado que considera el espacio dividido en infinitas celdas paralelas a los planos coordenados, y de un tamaño fijo. Estas celdas reciben el nombre de *voxels* (de la expresión en inglés "volume elements", que significa "elementos de volumen"). Para modelar un sólido será por tanto necesario indicar las celdas que están ocupadas por el volumen del mismo. Ejemplos de técnicas para determinar la inclusión de un voxel en el sólido son el estudiar la inclusión del centro del voxel, el grado de ocupación del volumen del voxel por el sólido o la inclusión de los vértices del voxel.

Este es un esquema que, salvo en el caso de sólidos poliédricos con caras paralelas a los planos coordenados, produce modelos con pérdidas. El grado de precisión alcanzable dependerá del tamaño de voxel que se utilice: a menor tamaño de voxel, mayor precisión, pero al mismo tiempo el tamaño de las representaciones será mayor, pudiendo afectar a aspectos como la eficiencia de los algoritmos, o el espacio total en memoria necesario para la aplicación de modelado. Así mismo, la representación resultante también variará según el algoritmo que se haya utilizado para elegir los voxels que componen el modelo. En la figura 2.4 se pueden ver dos imágenes procedentes de dos modelos de

enumeración correspondientes al mismo sólido y construidos con distinta resolución. Se puede apreciar cómo el modelo de la izquierda, que utiliza voxels más grandes, pierde más detalles que el modelo de la derecha, que tiene un tamaño de voxel menor. Esta pérdida de información es más evidente en las zonas que, como la cara del gato de la figura 2.4, poseen detalles muy pequeños.

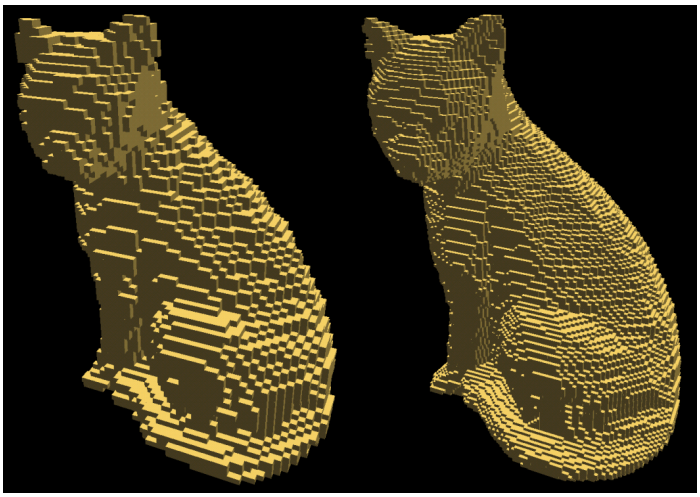


Figura 2.4 Dos modelos de enumeración con distinta resolución

Aplicabilidad a sólidos de forma libre

La enumeración espacial presenta un serio problema a la hora de representar sólidos de forma libre: al representar los objetos mediante elementos delimitados por caras planas, sólo puede aproximar las superficies curvas de los sólidos, y si se quiere reducir (que no eliminar) la pérdida de precisión hay que aumentar mucho el nivel de detalle, con el costo añadido de memoria y velocidad de procesamiento que esto supone. Esto hace que no sea la mejor elección para modelar sólidos de forma libre.

2.4.2.2 Octrees

La representación mediante árboles octales, también llamados *octrees* [Mea82, Män88] también considera el espacio dividido en celdas paralelas a los planos coordenados, al igual que la enumeración espacial, con la diferencia de que las celdas pueden tener distintos tamaños, múltiplos del tamaño mínimo de voxel. El punto de partida para la construcción de un octree para un sólido

es un cubo que lo englobe totalmente. Ese cubo se divide en ocho octantes, que pueden estar totalmente dentro, totalmente fuera, o parcialmente dentro del sólido. Aquellos octantes que no pueden clasificarse como dentro o fuera del sólido se vuelven a dividir, repitiendo recursivamente el proceso hasta que cada uno de los cubos está completamente dentro o completamente fuera del sólido, o se ha llegado al tamaño mínimo de voxel que indica la resolución de la representación.

Conforme se va realizando el proceso de división descrito anteriormente, se va creando un árbol octal, en el que los nodos hoja son etiquetados como *blancos* o *negros*, dependiendo de si sus octantes correspondientes se encuentran completamente fuera o completamente dentro del sólido a representar, respectivamente. Los nodos intermedios del árbol, que no se pueden clasificar como blancos o negros, son etiquetados como *grises*, y tienen a su vez ocho hijos, correspondientes a los ocho trozos en que se han dividido. En la figura 2.5 se puede ver un ejemplo de octree con tres niveles junto con una imagen del modelo representado.

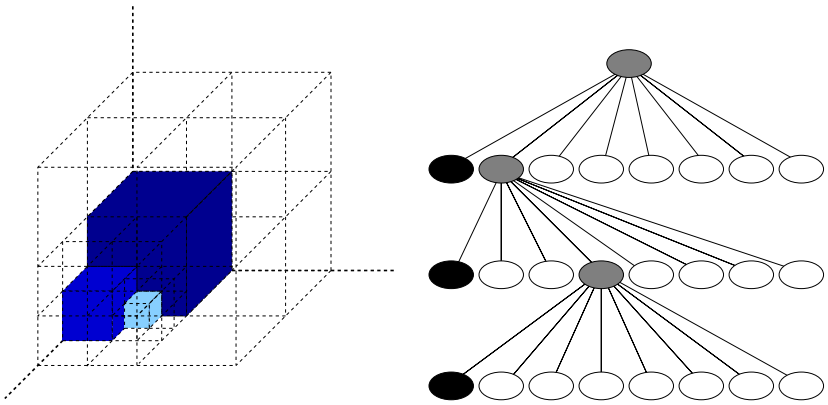


Figura 2.5 Ejemplo de la subdivisión en octantes en la construcción de un octree (a), y árbol resultante (b)

Los octrees de sólidos que no tengan caras planas o estas no sean paralelas a los planos coordenados, serán representaciones aproximadas, al igual que ocurre con la enumeración espacial. Sin embargo, el esquema de octrees tiene como ventaja respecto a aquél un menor consumo de memoria gracias a que sólo se subdivide un volumen en octantes cuando es necesario, reduciendo así el número de elementos sobre los cuales hay que almacenar información.

Los algoritmos sobre octrees se reducen en la mayoría de los casos a recorridos del árbol, lo que los hace robustos y eficientes [RUL00]. Las operaciones booleanas entre modelos se realizan entre nodos del mismo nivel, pudiendo ser necesario que algún nodo sea subdividido con el fin de tener el mismo nivel de detalle que los nodos del árbol del otro modelo.

Se ha trabajado mucho sobre los octrees con el fin de mejorar sus prestaciones. Algunas de las variantes de octree más conocidas son los octrees lineales y los octrees extendidos.

Los octrees lineales [Gar82] identifican cada voxel con un número octal; el número de cifras indica el nivel del árbol en el que se encuentra el voxel, y cada cifra indica cuál de las ocho ramas que parten del nodo superior hay que recorrer para llegar al voxel. De esta forma, sólo se almacenan los códigos correspondientes a los voxels que ocupa el sólido, obteniendo una representación más compacta.

Los octrees extendidos [BN85, BN90] permiten representar cualquier sólido poliédrico sin pérdidas, debido a que incluyen tres nuevos tipos de nodo hoja: nodo cara, nodo arista y nodo vértice. Cada uno de ellos se caracteriza porque está parcialmente ocupado por el sólido, conteniendo parte de una cara, parte de una arista, o un vértice, respectivamente. La representación de estos nodos incluye, además de su tipo, información sobre el elemento que contiene, como la intersección de la cara o la arista con las caras del voxel, o la posición del vértice dentro del mismo. Estas modificaciones hacen que la implementación de los algoritmos sea más compleja cuando deben tratarse las hojas del árbol.

Entre las variantes más actuales se incluyen los SP-octrees [CT02]. Un SP-octree almacena información adicional en sus nodos intermedios referente a los planos que contienen las caras del sólido a representar, permitiendo así aplicaciones como la transmisión progresiva de modelos [CTV03]. Los algoritmos que operan sobre esta estructura no necesitan siempre llegar hasta el nivel de los nodos hoja. Así mismo, se elimina información redundante de los nodos hoja respecto a los octrees extendidos, puesto que esta información se almacena en los nodos superiores del árbol.

Un inconveniente importante de los octrees se encuentra en la aplicación de transformaciones tales como rotaciones o escalados, que requieren en muchas ocasiones la reconstrucción del árbol.

Aplicabilidad a sólidos de forma libre

La representación mediante octrees presenta problemas similares a los de la representación por enumeración espacial a la hora de su aplicación a sólidos de forma libre, ya que no se puede representar de manera exacta el volumen del sólido con los elementos discretos (voxels) de los que se dispone. No obstante, el coste que supone aumentar el detalle de la representación para reducir la pérdida de precisión es menor que en el caso de la enumeración espacial. Con todo, este esquema tampoco es la mejor elección para modelar sólidos de forma libre.

Brunet y Ayala presentaron una extensión del modelo de octrees orientado a la representación de superficies de forma libre [BA87]. Esta extensión parte de parches paramétricos bicúbicos, que son descompuestos en parches bicuadráticos para calcular su posible intersección con los nodos del árbol, y así poder determinar qué nodos se deben refinar en el proceso de construcción del octree. Sin embargo, esta técnica no ha alcanzado popularidad debido a la complejidad extra que supone su implementación.

2.4.2.3 Descomposición espacial

En este caso [Män88, Woo03], los modelos se componen a partir de celdas más complejas que las descritas anteriormente. La idea es que se disponga de una variedad de celdas suficiente como para que se puedan representar sin pérdidas los sólidos que se desean modelar, y un operador de unión que establezca las relaciones topológicas entre ellas.

Es recomendable que las celdas sean lo suficientemente simples como para que se puedan representar fácilmente tanto mediante expresiones implícitas como paramétricas; típicamente, la topología de las celdas ha de ser equivalente a una esfera, es decir, sin agujeros. Al crear las combinaciones, las celdas permanecen semidisjuntas; es decir: pueden compartir los puntos de la frontera, pero no los puntos interiores.

El objetivo fundamental es conseguir una descomposición en celdas de tal forma que la evaluación de algoritmos sobre el modelo se reduzca de alguna forma a la evaluación del mismo en las celdas individuales y la unión de los resultados. Evidentemente, tanto la evaluación del algoritmo en las celdas como la unión de los resultados deben implicar una dificultad menor que el trabajo

con el modelo del sólido completo. En el mundo CAD/CAM interesa especialmente que las celdas se correspondan con volúmenes susceptibles de poder ser producidos por la maquinaria de fabricación estándar.

El hecho de que las celdas sean disjuntas puede complicar el modelado hasta límites insospechados. Además, operar con modelos descompuestos en celdas es a veces muy complejo, ya que las relaciones topológicas entre las celdas varían de unos modelos a otros debido a que no tienen una estructura fija.

Aplicabilidad a sólidos de forma libre

La descomposición espacial puede ser una opción válida para modelar este tipo de sólidos, pero el conjunto de celdas disponibles para formar los modelos condiciona fuertemente el grado de expresividad del sistema. En general, no se podría modelar cualquier sólido de forma libre, puesto que esto requeriría disponer de un número infinito de celdas para combinar, lo cual hace que en la práctica sea inviable construir un modelador de propósito general basado en esta técnica.

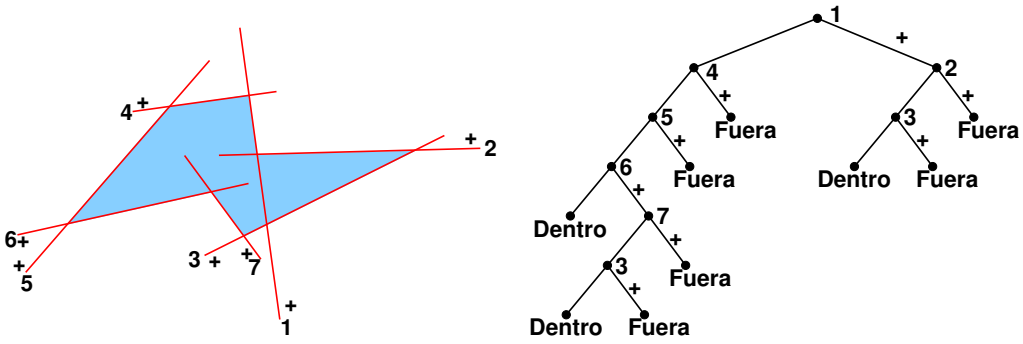
2.4.2.4 BSP

El esquema de representación BSP [TN87] describe los sólidos mediante la combinación de los semiespacios definidos por un conjunto de planos. La estructura de datos clásica para esta representación es un árbol binario, de forma que cada nodo almacena la definición de un plano, y cada una de las ramas que parten de él se corresponde con uno de los dos semiespacios (uno positivo y el otro negativo) que el plano determina.

El punto de partida para la construcción de un árbol BSP es el conjunto de caras del sólido. Mediante algún algoritmo, se selecciona el plano que contiene una de las caras como raíz del árbol, y se clasifica el resto de caras con respecto a los semiespacios que este plano crea. Si alguna de las caras es intersectada por el plano, se divide por el segmento de intersección, y se clasifica cada parte del mismo modo que el resto de caras. A continuación, se repite el proceso para cada subconjunto de caras, construyendo así la estructura de datos.

En la figura 2.6 se muestra un ejemplo en dos dimensiones de un árbol BSP. Un sólido 2D es un polígono, y los elementos divisores del espacio son las rectas que contienen las aristas del mismo. En el ejemplo de la figura 2.6,

a cada recta se le ha asignado un número para su identificación, y se indica con un signo + el semiespacio positivo que ésta define. Se puede observar que en la primera división, realizada con la recta 1, se intersecta el lado contenido por la recta tres; por este motivo, el lado se divide en dos, añadiéndose cada parte a un lado del árbol, y por eso aparecen dos nodos en los que el elemento divisor es la recta tres.



a b

Figura 2.6 Sólido bidimensional (a), y un posible árbol BSP que lo representa (b)

Los algoritmos sobre árboles BSP se basan, al igual que los algoritmos sobre octrees, en recorridos por el árbol, y en general son sencillos, robustos y eficientes. El problema principal que se plantea con esta estructura es el proceso de creación, puesto que una inadecuada elección de los planos divisores puede dar lugar a árboles no balanceados (como el de la figura 2.6) que por tanto reducen la eficiencia en su utilización, o bien pueden dar lugar a que haya multitud de caras que sea necesario dividir, con lo que la creación del árbol puede llegar a ser muy costosa en tiempo y memoria, y no compense la utilización de esta estructura [OSF03a, OSF03b].

El esquema de representación de árboles BSP sólo es aplicable a sólidos delimitados por caras planas, y no tiene pérdidas de precisión.

Aplicabilidad a sólidos de forma libre

El esquema BSP tiene un problema similar a los octrees y la enumeración espacial: la definición de los modelos se hace mediante la clasificación del espacio utilizando planos, por lo que no se pueden representar con exactitud objetos con superficies curvas.

2.4.3 CSG

La Geometría Constructiva de Sólidos (CSG) [RV77, Req80, Män88] es uno de los esquemas de representación más utilizados en el entorno CAD por su robustez y facilidad de utilización. Con CSG, los sólidos se representan como el resultado de la aplicación de una serie de operaciones booleanas (unión, intersección, complemento y diferencia) sobre un conjunto de instancias de primitivas simples. Estas primitivas representan sólidos básicos, tales como cilindros, prismas, conos, etcétera, y como tales, son cerradas y regulares. Cada instancia de una primitiva lleva asociada una transformación geométrica, siendo las secuencias de escalados, rotaciones y traslaciones las más comúnmente usadas en las aplicaciones de modelado basadas en este esquema.

Puesto que el resultado de una operación booleana aplicada sobre sólidos regulares puede no ser regular (figura 2.7b), Requicha [RV77] definió las operaciones booleanas regularizadas basándose en los conceptos de clausura e interior de un conjunto [Kur73]:

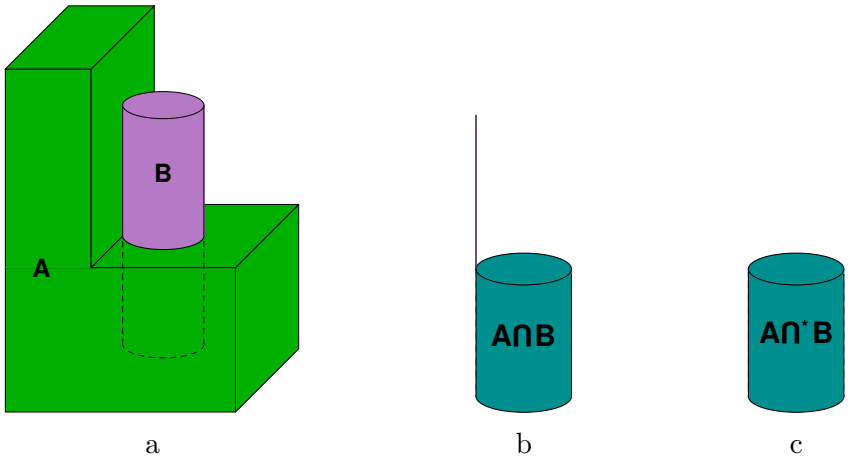


Figura 2.7 Dos sólidos regulares (a), y los resultados de su intersección sin regularizar (b) y regularizada (c)

- Dado un conjunto **S** perteneciente a $\mathbb{E}^3$, el ***interior*** de **S** está formado por todos aquellos puntos que en un entorno arbitrariamente pequeño sólo contienen puntos de **S**.

- Dado un conjunto **S** perteneciente a $\mathbb{E}^3$, la ***clausura*** de **S** está formada por todos aquellos puntos de $\mathbb{E}^3$ tales que en un entorno arbitrariamente pequeño contienen algún punto de **S**.

Así pues, las operaciones booleanas regularizadas se definen de la siguiente forma [RV77]:

$$\begin{aligned} X \cap^* Y &= clausura(interior(X \cap Y)) \\ X \cup^* Y &= clausura(interior(X \cup Y)) \\ X -^* Y &= clausura(interior(X - Y)) \\ complemento^* X &= clausura(interior(complementoX)) \end{aligned}$$

Estas definiciones garantizan que el resultado de la operación será un sólido válido, puesto que será un conjunto cerrado y regular (figura 2.7c).

La estructura de datos clásica utilizada en los modelos CSG es un árbol binario que almacena en sus hojas los datos de las instancias utilizadas en la construcción del modelo, y en sus nodos intermedios operaciones booleanas o transformaciones rígidas (traslaciones y rotaciones) que se aplican a los subárboles que están por debajo de dichos nodos (figura 2.8). Esta estructura facilita la implementación de algoritmos como el cálculo de operaciones booleanas entre modelos o el test de inclusión de puntos en el sólido utilizando una estrategia *divide y vencerás*.

El esquema de representación CSG es conciso, siempre produce modelos de sólidos válidos, no da lugar a ambigüedades, pese a que para un mismo modelo pueden existir varios árboles que lo definan, y es fácil de utilizar; de hecho, ni siquiera es necesaria una aplicación gráfica interactiva, puesto que se puede utilizar un lenguaje de descripción sencillo basado en la estructura de árbol CSG. No obstante, el grado de expresividad que se puede alcanzar depende del número de primitivas disponibles. Además, algunas operaciones como la visualización mediante la frontera son difíciles de realizar, puesto que no se almacena información explícita de la frontera de las primitivas ni de los resultados de las operaciones booleanas, como por ejemplo son las curvas o

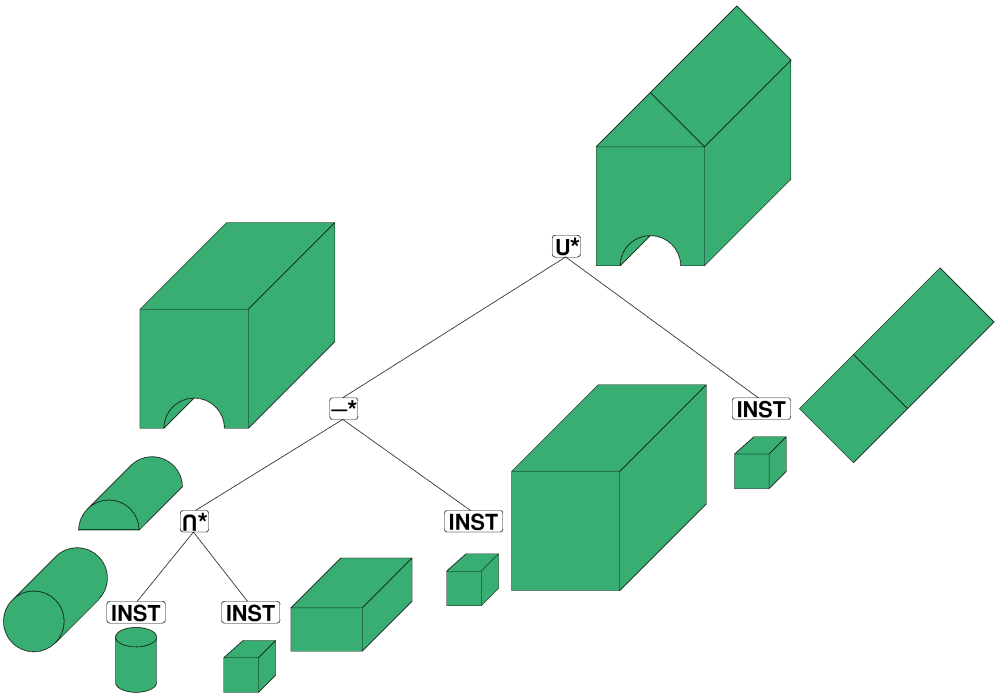


Figura 2.8 Un posible árbol CSG para un sólido. Las figuras debajo de los nodos instanciación son las primitivas. A los lados de los nodos aparecen los resultados parciales

segmentos de intersección, o nuevos vértices que puedan aparecer, y tampoco existe información sobre las relaciones topológicas entre los elementos que componen la frontera del sólido modelado.

Aplicabilidad a sólidos de forma libre

Los modeladores basados en el esquema CSG clásico [RV77] permiten modelar sólidos de forma libre dentro de las limitaciones impuestas por el número de primitivas disponibles, pero su expresividad no es suficiente como para representar cualquier sólido de forma libre. Además, algunas operaciones, como el cálculo de la frontera del sólido, pueden llegar a ser muy complejas por el número de operaciones booleanas utilizadas en su construcción. Sin embargo, la forma de trabajar utilizando instanciación de primitivas y operaciones booleanas es aceptada como una de las más cómodas para los usuarios de aplicaciones de modelado, y la mayoría de los esquemas de representación propuestos en la bibliografía para sólidos de forma libre estudian el diseño e implementación

de algoritmos que permitan trabajar utilizando el estilo CSG [Key00, KM96, BKZ01, Lin00, RS97].

2.4.4 Barrido

El barrido [Mor97, AMBJ06] es una técnica sencilla que permite construir figuras relativamente complejas fácilmente, lo que la ha convertido en una de las técnicas más populares en el entorno CAD. Un sólido de barrido está formado por el conjunto de puntos barridos por un elemento generador que se va desplazando a lo largo de una trayectoria. En la figura 2.9 se muestran algunos ejemplos de sólidos de barrido.

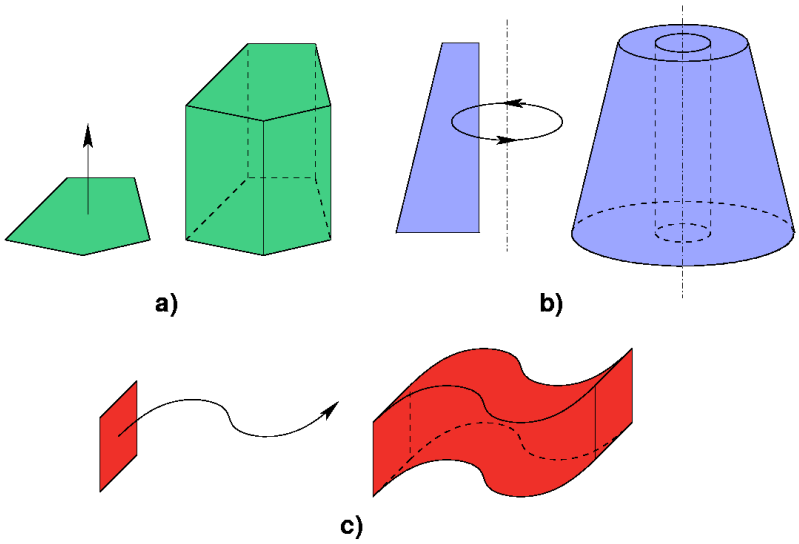


Figura 2.9 Sólidos de barrido. a) Extrusión. b) Sólido de revolución. c) Barrido general

En su forma más básica, el generador es una sección del sólido a generar, y la trayectoria un segmento perpendicular al plano que contiene dicha sección; estos barridos se denominan traslacionales o extrusiones (figura 2.9a). Si la trayectoria es una rotación alrededor de un eje, el barrido se denomina rotacional, y genera un sólido de revolución (figura 2.9b). Se pueden obtener sólidos más complejos si la trayectoria es una curva arbitraria, o si el generador sufre algún tipo de transformación en su forma, tamaño u orientación durante el recorrido de la trayectoria (figura 2.9c).

Una ventaja importante de esta técnica es que en general no es necesario almacenar mucha información para definir un sólido. Para ello es necesario que tanto el generador como la trayectoria sean descriptibles analíticamente, aunque suele ser difícil comprobar la validez de los sólidos generados, y puede llegar a ser un problema la determinación de las fronteras para tareas como la visualización o la implementación de operaciones booleanas entre sólidos de barrido. Así mismo, puede llegar a ser muy complejo encontrar métodos para calcular las propiedades del sólido, ya que dependen en muchos casos de las características concretas del sistema de construcción de cada uno. Por ejemplo: el cálculo del volumen de un sólido de extrusión como el de la figura 2.9a es mucho más sencillo en términos generales que el cálculo del volumen de un sólido generado mediante un barrido siguiendo una trayectoria curvilínea arbitraria como el de la figura 2.9c.

Los barridos son muy útiles en ámbitos relacionados con la robótica y la fabricación asistida por computador, ya que permiten estudiar colisiones entre partes móviles, la acción de las cuchillas de las máquinas fresadoras (la cantidad de material que eliminen de una pieza será igual a la intersección de la pieza con el volumen barrido por las cuchillas), etcétera.

Aplicabilidad a sólidos de forma libre

Muchos sólidos de forma libre son representables mediante barridos más o menos complejos, pero al igual que ocurre con otras técnicas, la expresividad está limitada por el método de construcción. Además, cuanta más libertad se dé al barrido con el fin de aumentar la expresividad, mayor será la dificultad para realizar cualquier tipo de cálculo sobre el modelo.

2.4.5 B-Rep

Las representaciones basadas en la frontera (en inglés *boundary representation* o B-rep) definen los sólidos a partir de la superficie que los delimita [Req80, Män88, FvDFH92, Sha02]. Esta superficie se divide en elementos discretos representables fácilmente, denominados caras si son planos, o parches si son curvos. Las caras están delimitadas mediante segmentos, denominados aristas, que a su vez están limitados por vértices, y los parches se pueden definir mediante una serie de puntos de control y una formulación implícita o explícita. Los modelos B-rep limitados por caras planas se denominan poliedros; a partir

de ahora, nos referiremos a sólidos poliédricos, puesto que todas las consideraciones sobre B-rep se pueden extender más o menos fácilmente a sólidos delimitados por parches curvos. En la figura 2.10 se muestra un sólido poliédrico y la descomposición de su frontera en caras, aristas y vértices.

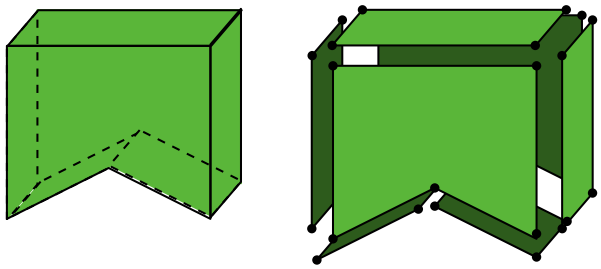


Figura 2.10 Sólido poliédrico y descomposición en elementos delimitadores

El esquema B-rep se basa en el teorema de la curva de Jordan [Jor87, Tve80], que afirma que una curva cerrada y homeomorfa a una circunferencia en $\mathbb{R}^2$ divide al plano en dos subconjuntos disjuntos, uno interior y otro exterior, delimitados por dicha curva. Aplicado a tres dimensiones, se puede enunciar de la siguiente forma: dada una superficie homeomorfa a una esfera en $\mathbb{R}^3$, ésta divide el espacio en dos subconjuntos disjuntos, uno interior y otro exterior, delimitados por dicha superficie. Un modelo B-rep de un sólido regular está formado por tanto no sólo por el volumen interior a la frontera del sólido, sino también por la frontera de dicho sólido.

Una de las principales necesidades a la hora de diseñar un esquema de representación basado en la frontera es proporcionar los medios para saber si un conjunto de caras forma realmente una superficie cerrada que defina un sólido. Para hacer esta comprobación más sencilla, todo esquema de representación basado en la frontera incluye no sólo información sobre las caras (incluyendo su orientación), sino también sobre cómo éstas se relacionan entre sí y con los vértices y aristas. A partir de estas relaciones topológicas, se pueden establecer condiciones de validez y algoritmos que las comprueban de manera eficiente y robusta.

Las condiciones de validez que se impongan a un modelo de fronteras dependerán del grado en que se restrinja el dominio de la representación. Así, si se restringe el dominio a los sólidos 2-variedad (sólidos tales que la vecindad

de cada punto de su frontera es homeomorfa con un disco en el plano), las condiciones de validez son las siguientes:

- Cada arista está limitada por dos vértices.
- Cada arista es compartida por dos caras.
- Las caras que comparten una arista tienen la misma orientación (el exterior de ambas caras se ecuentra al mismo lado, o lo que es lo mismo, sus vectores normales apuntan hacia el exterior del sólido).
- Las aristas sólo se intersectan en los vértices.
- Las caras sólo se intersectan en las aristas.

Las tres primeras condiciones se pueden comprobar más o menos fácilmente a partir de las relaciones topológicas en la estructura de datos. Sin embargo, la comprobación de las dos últimas condiciones es una tarea más compleja; una condición necesaria, aunque no suficiente, que permite descartar conjuntos de caras susceptibles de ser considerados representaciones válidas, es la característica de Euler de la superficie, que identifica la relación entre los vértices, aristas y caras de una superficie S:

$$\chi(S) = V - E + F \tag{2.1}$$

siendo V el número de vértices, E el número de lados, y F el número de caras. Para un sólido 2-variedad sin agujeros, este valor es 2; si el resultado de este cálculo utilizando la información de un conjunto de caras es distinto de 2, dicho conjunto de caras no es una representación B-rep válida.

Si el sólido tiene agujeros o está compuesto de varias partes, la condición basada en la característica de Euler es:

$$V - E + F - H = 2(C - G) \tag{2.2}$$

siendo V el número de vértices, E el número de lados, F el número de caras, H el número de agujeros en las caras que no traspasan el sólido, C el número de componentes separadas que forman el sólido, y G el número de agujeros que atraviesan el sólido.

Tal y como se ha comentado antes, la fórmula de Euler por sí sola expresa una condición necesaria, pero no suficiente, para que una superficie limite un sólido, puesto que indica una condición topológica, no geométrica. Por otra parte, los algoritmos para encontrar agujeros y componentes aisladas en los candidatos a sólidos implican una dificultad añadida a esta comprobación.

Los puntos fuertes de las representaciones basadas en B-rep suelen ser la visualización y la edición, puesto que al almacenar la información de la frontera de forma explícita se hace relativamente sencillo modificarla o dibujarla utilizando algoritmos de visualización clásicos como el scan-line [FvDFH92], el Z-buffer [FvDFH92] o el trazado de rayos [Gla93]. Los mayores inconvenientes de esta representación vienen del lado de la complejidad que pueden llegar a alcanzar las estructuras de datos necesarias para almacenar las relaciones topológicas, y los problemas derivados de la aplicación de operaciones booleanas sobre los modelos, puesto que el conjunto de modelos B-rep no es cerrado bajo este tipo de operaciones.

Aplicabilidad a sólidos de forma libre

La representación B-rep basada en poliedros no es adecuada para sólidos de forma libre, ya que presenta problemas de precisión. Sin embargo, si en lugar de poliedros se utilizan parches curvos, se pueden modelar sólidos de forma libre más o menos fácilmente [KKM97, KGMM97]. La complejidad de uso y coste computacional de estas últimas representaciones depende fundamentalmente del tipo de parches que se utilicen. Los problemas de este esquema están asociados a los cálculos de propiedades relacionadas con el volumen, fundamentalmente, puesto que esta representación se basa únicamente en la frontera, y ese tipo de cálculos no se suelen adaptar bien a ella.

2.4.6 Hiperparches

Los hiperparches [Mor97] son una extensión de las curvas y superficies paramétricas, ampliando el número de dimensiones parámetricas a tres. Un sólido se define como un conjunto de puntos (x, y, z) cuyas coordenadas están determinadas por funciones univaluadas continuas de tres parámetros de la forma:

$$x = x(u, v, w); \qquad y = y(u, v, w); \qquad z = z(u, v, w);$$
$$u, v, w \in [0, 1]$$

La relación con los elementos de menor dimensión paramétrica es simple, ya que fijando uno o dos de los parámetros a un valor y haciendo variar libremente los restantes, se obtiene una superficie o una curva (denominadas isoparamétricas). En la figura 2.11 se muestra un hiperparche con las coordenadas paramétricas de sus esquinas. Se puede comprobar que los lados curvos del hiperparche se correponden a las curvas isoparamétricas resultantes de fijar dos parámetros a cero o a uno, y las superficies que limitan el parche son superficies isoparamétricas obtenidas fijando el valor de uno de los parámetros a cero o uno.

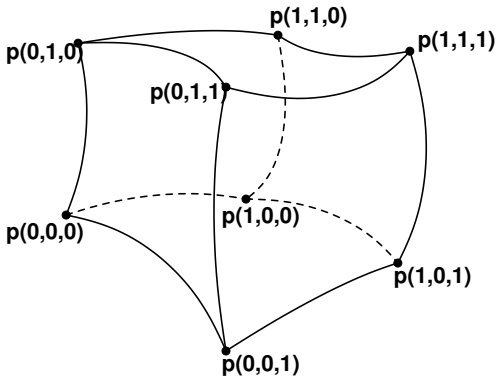


Figura 2.11 Hiperparche con las coordenadas paramétricas de sus esquinas

El desarrollo matemático de las funciones que definen un hiperparche es similar a los desarrollos para las curvas y las superficies, con la diferencia de que al añadir dimensiones paramétricas aumenta la complejidad de las funciones y también es necesario aumentar el número de puntos de control. Así, si una curva paramétrica cúbica está determinada por cuatro puntos de control (4^1), y una superficie bicúbica por dieciséis (4^2), un hiperparche tricúbico está determinado por sesenta y cuatro puntos de control (4^3). Del mismo modo que se pueden unir curvas o superficies con distintos niveles de continuidad, se pueden también unir hiperparches, formando un sólido polinómico a trozos.

Los hiperparches se prestan muy bien para la representación de sólidos heterogéneos, cuyas propiedades, tales como la densidad o la temperatura, varían en el interior del sólido, ya que se pueden asociar con cada punto de control valores de propiedades que luego pueden ser interpolados. Por otra parte, el diseño interactivo de hiperparches no es fácil, puesto que el número

de puntos de control es muy grande, y puede llegar a ser una labor muy tediosa editarlos uno por uno.

Aplicabilidad a sólidos de forma libre

Los hiperparches son una opción muy válida para representar este tipo de sólidos, siendo su principal inconveniente la creación y edición de modelos a partir de los puntos de control, ya que, dependiendo del tipo de formulación utilizada (basada en Bézier, B-splines, ...), un punto de control puede afectar a una región importante del sólido, o bien puede ser difícil ajustar alguna característica del sólido a algún tipo de restricción (colocar una esquina en un punto del espacio o ajustar las dimensiones del sólido a unas medidas dadas, por ejemplo). Las operaciones booleanas con hiperparches suelen ser también complejas de realizar.

2.4.7 Deformaciones controladas

Las deformaciones controladas [SP86, Bec98] se basan en una transformación del espacio de referencia en el que se define el sólido, trasladando luego el efecto de esa transformación al sólido en cuestión. Matemáticamente, se define un espacio de referencia normalizado $\mathbf{p}$(u,v,w) que contiene al sólido, y una transformación sobre él $\mathbf{t}(\mathbf{p})$, tal que a cada punto de $\mathbf{p}$ le corresponde un nuevo punto $\mathbf{p_t}$, resultado de aplicarle la transformación $\mathbf{t}$. En la figura 2.12 se puede ver un ejemplo de este método.

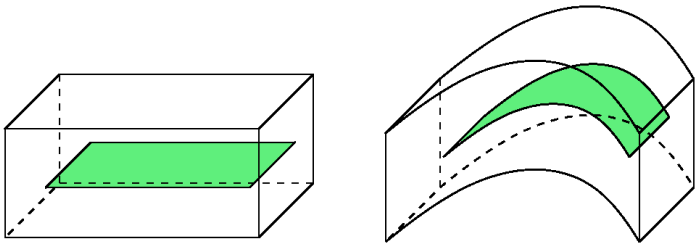


Figura 2.12 Un ejemplo de deformación controlada aplicada a un polígono

Las deformaciones controladas son consideradas en general como un método de edición, puesto que realmente no proporcionan por sí mismas un medio de construcción y almacenamiento de modelos, sino que es precisa una representación previa de los objetos que permita evaluar los puntos del sólido que luego se harán correspoder con los puntos del espacio de referencia transformado.

Aplicabilidad a sólidos de forma libre

Los modelos basados en deformaciones controladas pueden representar sólidos de forma libre de un modo apropiado [RSB96, HML00], aunque presentan problemas similares a los hiperparches, ya que el espacio de referencia que se utiliza para aplicar las deformaciones es en cierto modo similar a un hiperparche.

2.4.8 Métodos híbridos

Hasta el momento, no existe el esquema de modelado ideal. Todos tienen sus puntos fuertes, pero también tienen puntos débiles, y ninguno se ajusta perfectamente a todas y cada una de las necesidades de cualquier campo de aplicación del modelado de sólidos. Por este motivo, una de las soluciones habituales en los sistemas de modelado es recurrir a varios esquemas conjuntamente: utilizar un esquema que facilite la interacción con el usuario, para que la creación y edición de modelos sea lo más cómoda y eficiente posible (CSG, por ejemplo), y una representación en memoria que permita dar rápida respuesta a operaciones como consultas topológicas o cálculo de operaciones booleanas (octrees, B-rep, ...). Son los denominados sistemas o métodos híbridos [Män88, FvDFH92, Sha02].

Aplicabilidad a sólidos de forma libre

Un ingrediente fundamental para que estos sistemas sean factibles son los algoritmos de conversión, que permiten mantener la consistencia entre las distintas representaciones del mismo objeto. No todas las combinaciones de esquemas de representación permiten la conversión entre ellos, y en algunos casos este paso puede llegar a ser muy costoso. Dependiendo de la complejidad de este paso, y del ámbito de aplicación del sistema, algunos modeladores actualizan continuamente todas las representaciones, y otros sólo actualizan las representaciones auxiliares cuando es necesario utilizarlas para realizar alguna operación.

Otra posibilidad es construir representaciones mixtas. Así, el sistema de octrees extendidos mencionado en la sección 2.4.2.2 puede entenderse como un esquema híbrido, en el que la estructura de octree llega hasta un cierto nivel de profundidad, y a partir de ahí se combina con una estructura B-rep.

2.5 Sistemas de modelado de sólidos de forma libre

La representación y manipulación de sólidos de forma libre sigue siendo hoy en día un campo de investigación en el ámbito del modelado de sólidos. Prueba de ello es que siguen apareciendo propuestas centradas en este tipo de sólidos, como las de Allègre y otros [AGCA06], que proponen una estructura de datos mixta, basada en un árbol CSG, y en la que se combinan nubes de puntos, mallas de triángulos y superficies esqueléticas implícitas [BGA04], de tal forma que se selecciona una u otra representación en base a la operación que se va a realizar con los modelos.

Otras aproximaciones, como la de Linsen [Lin00], que utiliza parches paramétricos para representar la superficie de los sólidos, o la de Biermann y otros [BKZ01], que trabajan con superficies de subdivisión, penalizan la exactitud de los resultados en las operaciones de modelado en favor de la eficiencia, ya que trabajan con las mallas de control de las superficies para calcular los resultados en lugar de con las superficies directamente. En contraste con estos trabajos, la labor de Keyser y otros [KCF+04] se centra en la exactitud del resultado en la evaluación de la superficie obtenida en las operaciones, con lo que el tiempo de ejecución de los algoritmos se ve notoriamente afectado.

Una opción intermedia que pondera la eficiencia de los algoritmos y la exactitud del resultado está reflejada en el trabajo de Krishnan y Manocha [KM96], que consigue resultados exactos de manera eficiente en operaciones con sólidos limitados por parches paramétricos, aunque con problemas de inestabilidad numérica debidos a la complejidad del aparato matemático desarrollado.

Un enfoque distinto es el que presentan Ruiz de Miras y Feito en el modelo de Cadenas Simpliciales Extendidas, también denotado modelo ESC [RF97, RF99a]. Este modelo desarrolla un esquema de representación de sólidos basado en la descomposición de los mismos en celdas no necesariamente disjuntas con signo, de forma que los volúmenes de las celdas que se superponen se compensan en función de su signo. Este modelo fue inicialmente desarrollado para sólidos bidimensionales limitados por cónicas y curvas de Bézier; posteriormente, se adaptó de forma satisfactoria a sólidos tridimensionales limitados por parches algebraicos, obteniéndose tanto en dos como en tres dimensiones algoritmos robustos y eficientes para resolver problemas como el test de inclusión de puntos, la visualización de operaciones booleanas no evaluadas o la conversión a otras representaciones [RF98, RF02]. Esta versatilidad hace

de este modelo una opción muy válida para el modelado de sólidos con caras curvadas, y por tal motivo ha sido adoptado como la base de este trabajo.

En los siguientes capítulos se expondrán con detalle los fundamentos del modelo ESC, así como la particularización del mismo a sólidos delimitados por superficies paramétricas triangulares, junto con los algoritmos implementados y los resultados obtenidos.

3 El modelo de Cadenas Simpliciales Extendidas

El punto de partida de este trabajo es el modelo de Cadenas Simpliciales Extendidas. En este capítulo se resume este modelo, sin hacer referencia al tipo de parches utilizados para la representación de la superfice de los sólidos. Así mismo, se recoge la representación algebraica de las operaciones booleanas entre sólidos representados por cadenas y el algoritmo para el cálculo del test de inclusión de un punto respecto de un sólido.

3.1 Fundamento

El modelo de Cadenas Simpliciales Extendidas o modelo ESC[1] fue propuesto por Ruiz y Feito como un esquema formal para la representación y manipulación de sólidos de forma libre [RF99a]. Este modelo a su vez extiende el modelo de Cadenas Simpliciales o modelo SC[2], que fue propuesto anteriormente por Feito y Rivero para la representación y manipulación de sólidos poliédricos [FR98].

La idea fundamental subyacente en estos modelos es aplicar el principio *divide y vencerás* [BB97] al modelado, descomponiendo los sólidos en elementos más simples, de forma que el resultado de las operaciones de modelado se determina aplicando las operaciones a los elementos individuales y componiendo los resultados parciales en el resultado final. Este principio se asemeja a la

[1] La traducción a la lengua inglesa es *Extended Simplicial Chains*. Para abreviar el nombre del modelo utilizaremos las siglas de la traducción inglesa, al estilo del modelo CSG

[2] *Simplicial Chains*

descomposición espacial expuesta en la sección 2.4.2.3, pero se diferencia de ella en el hecho de que no se hace necesario que las celdas sean disjuntas, lo que facilita notoriamente el proceso de construcción de las mismas.

3.1.1 El modelo de Cadenas Simpliciales

El modelo de Cadenas Simpliciales para la representación de sólidos de caras planas puede estudiarse como un caso particular de las Cadenas Poliédricas definidas por Whitney [Whi57]. Los sólidos se representan bajo este modelo como conjuntos de celdas poliédricas elementales no disjuntas, de forma que se permite el solapamiento de celdas. Estas celdas poliédricas elementales se denominan *símplices*.

Formalmente, se define un ***símplice d-dimensional*** como la envolvente convexa de $d+1$ puntos linealmente independientes [Kur73], de tal modo que, por ejemplo, un símplice bidimensional se corresponde con un triángulo, y un símplice tridimensional se corresponde con un tetraedro.

Una cadena simplicial que representa a un sólido está formada por un conjunto de símplices tales que todos ellos tienen un vértice en común. Este vértice es un punto arbitrario del espacio que puede ser tanto interior como exterior al sólido a representar, y que se etiqueta como ***origen*** de la cadena. El resto de los vértices de cada símplice se corresponden con vértices del sólido. Los símplices construidos de esta forma se denominan ***símplices originales***, y los elementos geométricos que unen el origen del símplice con el resto de vértices también se califican como originales; de este modo, se dispone de ***aristas originales***, ***caras originales***, etcétera.

Formalmente, una ***cadena simplicial d-dimensional*** se define como:

$$\chi = \sum_{i=1}^{n} \alpha_i \cdot S_i \tag{3.1}$$

donde los coeficientes α_i son números enteros, y las celdas S_i son símplices d-dimensionales (tetraedros si nos ceñimos a sólidos 3D). El signo de cada coeficiente indica si el volumen de la celda con la que se corresponde se añade o se sustrae en la definición del sólido, dependiendo de si α_i es positivo o negativo, respectivamente.

En la figura 3.1 se puede observar el proceso de construcción de un sólido bidimensional a partir de una cadena simplicial que lo representa, formada por los símplices positivos **OV1V2** y **OV3V1**, y por el símplice negativo **OV2V3**. El origen de la cadena es en este caso el origen de coordenadas. En azul se muestran las áreas con signo positivo, y en rojo las que tienen signo negativo. Conforme se van sumando símplices, se puede observar que en las regiones del espacio en que los símplices positivos y negativos se superponen se contrarrestan los signos, de forma que el volumen del sólido resultante coincide con el del sólido de partida; esto permite que se pueda escoger cualquier punto interior o exterior al sólido como origen de la cadena, sin que ello modifique el resultado [Fei95].

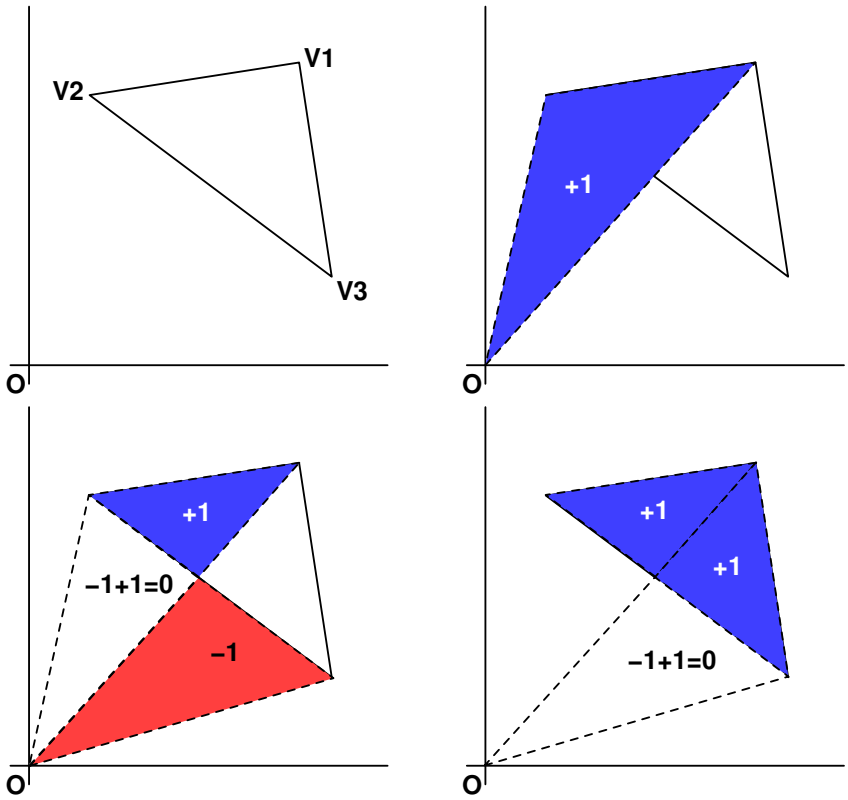


Figura 3.1 Proceso de construcción de un sólido bidimensional a partir de una cadena simplicial que lo representa

En [FR98] se describe el fundamento teórico del modelado de sólidos poliédricos con cadenas simpliciales. Este esquema de representación es válido para sólidos variedad y no variedad, con o sin agujeros, y proporciona soluciones válidas

a problemas como el cómputo de operaciones booleanas [RF00, Riv02] o el cálculo del test de inclusión de un punto en un sólido [FT97], reduciéndolos al cálculo local de las soluciones en cada símplice y la combinación posterior de los resultados en una suma. Esta forma de trabajar hace a los algoritmos sencillos, robustos y fácilmente paralelizables, puesto que la información de una celda es independiente de la del resto.

El modelo ESC, al ser una extensión del modelo SC, mantiene la misma base teórica, por lo que dicha base se revisará con más detalle en los siguientes apartados de este capítulo.

3.2 Celdas extendidas

El modelo ESC se basa, al igual que el modelo SC del que deriva, en la descomposición del volumen del sólido en celdas *no necesariamente disjuntas*. Estas celdas se denominan ***celdas extendidas***, y se clasifican en dos tipos: símplices y celdas de forma libre. A continuación se detallan las características más importantes de ambos tipos en relación a su uso en el modelo ESC.

3.2.1 Símplices

Tal y como se ha definido en la sección 3.1.1, un símplice d-dimensional es la envolvente convexa de d+1 puntos linealmente independientes. Particularizando al caso 3D, que es el habitual en el Modelado de Sólidos, un símplice es un tetraedro, como se puede apreciar en la figura 3.2.

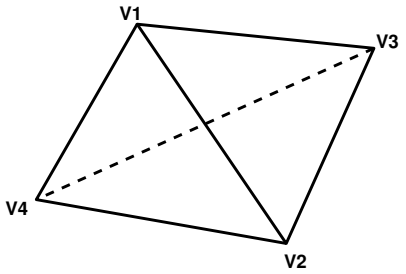


Figura 3.2 Símplice tridimensional

Los símplices cumplen la propiedad de que cualquier punto del espacio puede ser representado mediante una combinación lineal de sus vértices. Los coeficientes de esta combinación lineal se denominan ***coordenadas baricéntricas***,

y son utilizadas de manera intensiva tanto en el modelo SC como en el ESC, ya que a través del estudio de estas coordenadas se puede saber la posición relativa de un punto respecto de un símplice concreto, como se verá más adelante.

En el caso de símplices bidimensionales, las coordenadas baricéntricas se obtienen de la siguiente forma [Bus05]:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} \tag{3.2}$$

donde u, v y w son las coordenadas baricéntricas y el subíndice p se aplica a las coordenadas del punto que se está estudiando. La matriz 3×3 está formada por las coordenadas homogéneas de los vértices del símplice.

De manera análoga se calculan las coordenadas baricéntricas de un punto respecto de un símplice tridimensional [Men92]:

$$\begin{pmatrix} t \\ u \\ v \\ w \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} \tag{3.3}$$

Tal y como se ha mencionado anteriormente, el signo de las celdas indica si su volumen se añade o se sustrae en la construcción del volumen del sólido representado. En el caso de los símplices, este signo viene determinado por la orientación de sus vértices, tomándose la orientación antihoraria como positiva.

Una forma rápida de calcular el signo de un símplice es consultar el signo de su área signada, en el caso de símplices bidimensionales, o el de su volumen signado, en el caso de símplices tridimensionales. En el caso bidimensional, el ***área signada de un triángulo*** se define como [O'R98]:

$$\Delta = \frac{1}{2} \cdot \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \tag{3.4}$$

En el caso tridimensional, el ***volumen signado de un tetraedro*** se define como [O'R98]:

$$\Delta = \frac{1}{6} \cdot \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \tag{3.5}$$

El signo de los símplices es fundamental en el desarrollo de los algoritmos basados en los modelos SC y ESC, como veremos posteriormente.

3.2.2 Celdas de forma libre

El modelo ESC nace con la idea de cubrir las carencias del modelo SC en cuanto a la representación exacta de sólidos de forma libre, ya que al estar este último basado en la descomposición de los sólidos en símplices, sólo puede aproximar las superficies curvas de este tipo de sólidos. Para conseguir este objetivo, el modelo ESC incluye un nuevo tipo de celda, denominado celda de forma libre o *ffc*[3].

Semánticamente, las ffc son equivalentes a los símplices, con la diferencia de que mientras que éstos están delimitados exclusivamente por elementos planos, aquéllas incluyen un elemento delimitador de forma libre, que las habilita para ajustarse a la superficie de los sólidos que se desea representar.

Formalmente, se define una ***celda de forma libre d-dimensional*** como un conjunto de puntos de $\mathbb{R}^d$, resultado de la intersección de los semiespacios definidos por un elemento de forma libre de dimensión $d-1$, y uno o varios elementos planos de dimensión $d-1$ tal que cumple [RF99a, Rui01]:

- Es un conjunto cerrado y conexo en $\mathbb{R}^d$.
- Todos los puntos del conjunto, menos los de su frontera, pertenecen a la misma componente conexa respecto del elemento de forma libre.
- Los puntos del conjunto comunes a un número de elementos delimitadores mayor o igual a d también pertenecen al elemento delimitador de forma libre.

La primera propiedad permite tratar casos como el de la figura 3.3, en los que el elemento de forma libre intersecta con los elementos planos, dando

[3] *Free-form cell*

lugar a más de una componente conexa. Estos casos se resuelven dividiendo el conjunto de puntos en dos ffc distintas.

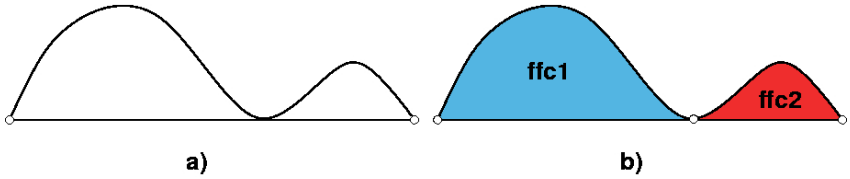


Figura 3.3 Presentación en 2D de la primera propiedad de la definición de ffc

La segunda propiedad se justifica para evitar situaciones como la que se muestra en la figura 3.4, en las que determinar la posición de un punto respecto a una ffc es complejo. La solución que se aplica en estos casos es dividir el conjunto de puntos para que verifique la segunda propiedad antes expuesta, lo cual puede no ser trivial, ya que hay que encontrar las zonas de autointersección del elemento de forma libre.

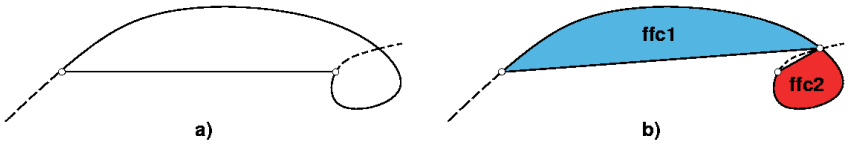


Figura 3.4 Presentación en 2D de la segunda propiedad de la definición de ffc

Por último, la tercera propiedad permite discriminar correctamente los vértices de las ffc, así como los conjuntos de puntos que sí que son ffc frente a los que no lo son (figura 3.5a). Esta propiedad también da pie a la simplificación de disposiciones más complejas de lo aconsejable (figuras 3.5b y 3.5c).

Toda ffc, al igual que los símplices, tiene un signo asociado que indica si el volumen de la celda se añade o se sustrae al volumen del sólido representado, lo que permitirá añadir a la representación basada en símplices los detalles de forma libre que se deseen.

La definición de ffc menciona la existencia de un "elemento delimitador de forma libre", sin indicar condiciones de continuidad, formulación (paramétrica, algebraica, implícita, ...), etcétera. Al ser el concepto de ffc independiente del tipo y forma de construcción de la frontera de los sólidos, es posible adaptarlo

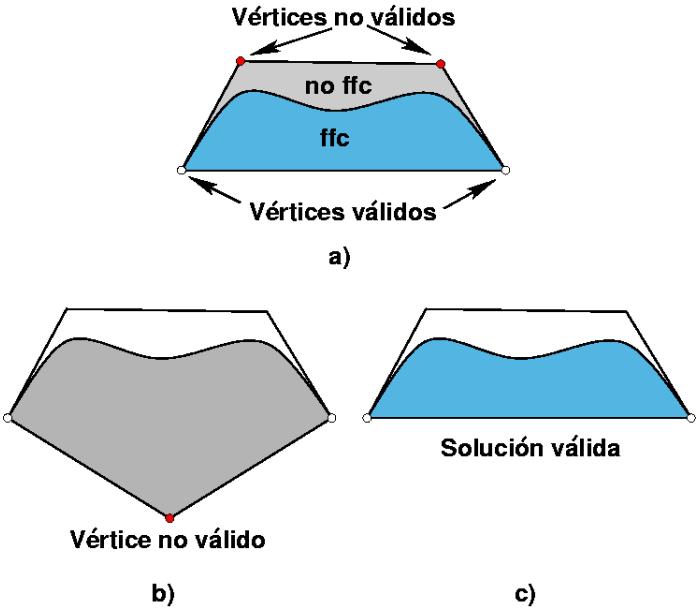


Figura 3.5 Presentación en 2D de la tercera propiedad de la definición de ffc

a cualquier esquema de construcción de sólidos n-dimensionales basado en elementos curvos. Esta adaptación consiste en la determinación de las condiciones bajo las cuales se verifica la definición de ffc, y la definición de un algoritmo para conocer la posición relativa de un punto respecto de una celda de este tipo.

La figura 3.6 muestra un ejemplo de ffc tridimensional delimitada por parches triangulares de Bézier y un conjunto de planos, representados como polígonos. En el capítulo 4 se explicará con detalle la particularización de la definición general de ffc al caso de parches triangulares de Bézier.

3.3 Definición de Cadena Simplicial Extendida

Como ya se ha mencionado anteriormente, el modelo ESC es una extensión del modelo de cadenas simpliciales, o modelo SC descrito en el apartado 3.1.1. La diferencia estriba en la aparición de las celdas de forma libre o ffc explicadas anteriormente como elementos de las cadenas.

Se define una ***cadena simplicial extendida*** o ***ESC*** como [RF99a]:

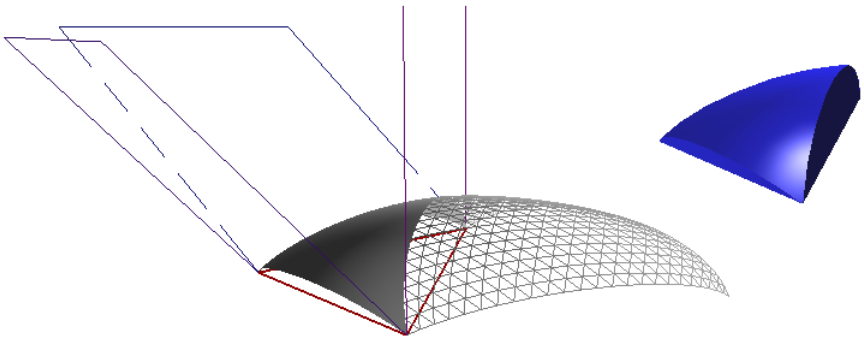


Figura 3.6 Ffc tridimensional. Izquierda: elementos delimitadores. Derecha: volumen

$$\delta = \sum_{i=1}^{n} \alpha_i \cdot E_i \tag{3.6}$$

en la que los α_i representan coeficientes de tipo entero, y donde los E_i se corresponden con celdas extendidas d-dimensionales, ya sean símplices o ffc. Al igual que en el modelo SC, los signos de los coeficientes asociados a las celdas indican si éstas se suman o se restan al sólido representado por la ESC.

La presencia de las ffc en el modelo permite aumentar el dominio de la representación, incluyendo ahora no sólo sólidos poliédricos, sino también sólidos de forma libre. Además, como el tratamiento a nivel de cadena tanto de símplices como de ffc es el mismo, se favorece la abstracción en el diseño de algoritmos que trabajen con ESC.

Un concepto fundamental que comparten tanto el modelo SC como el ESC es el de ***función asociada a una cadena***, que se define de la siguiente forma, dada una ESC δ:

$$\begin{aligned} f_\delta &: \mathbb{R}^d \longrightarrow \mathbb{Z} \\ f_\delta(\mathbf{Q}) &= \sum_{\forall E_i / \mathbf{Q} \in E_i} \alpha_i \end{aligned} \tag{3.7}$$

La función asociada a una cadena devuelve para cada punto del espacio euclídeo d-dimensional la suma de los coeficientes asociados a las celdas en las que dicho punto está incluido. Basado en el concepto de función asociada a una cadena, se define el ***sólido asociado a una cadena*** δ como el conjunto

de puntos que verifican que el valor de la función asociada a δ para esos puntos es distinta de cero. Matemáticamente:

$$FF_\delta = \left\{ \mathbf{Q} \in \mathbb{R}^d / f_\delta(\mathbf{Q}) \neq 0 \right\} \tag{3.8}$$

En la figura 3.7 se puede observar un sólido bidimensional, así como las celdas extendidas que forman parte de una ESC que lo representa. En este ejemplo se ha tomado como origen de los símplices un punto exterior al sólido, y las celdas positivas se colorean con un tono azulado, mientras que las negativas aparecen en un tono rojizo. Por último, se muestra la superposición de las celdas, mostrando cómo las áreas positivas y negativas se complementan, dando como resultado el área del sólido a representar.

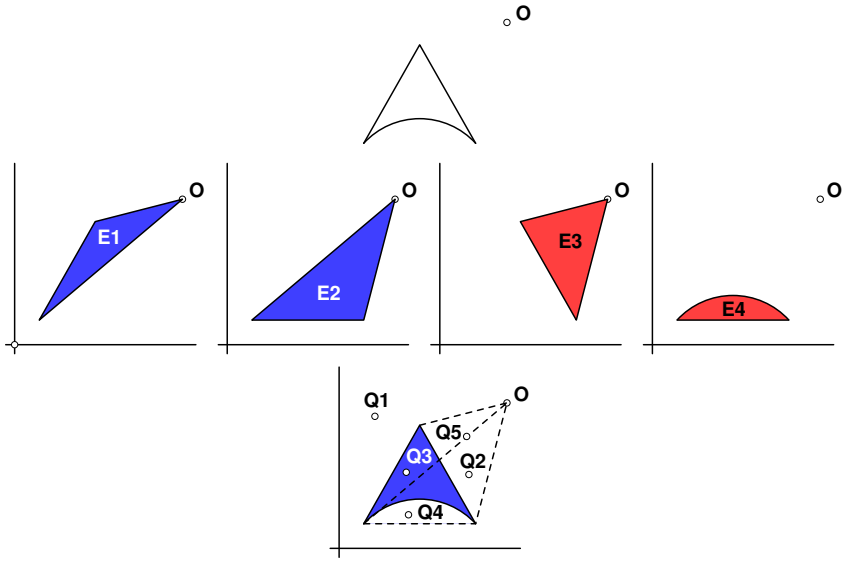


Figura 3.7 Ejemplo de construcción de una ESC para un sólido bidimensional. De arriba a abajo: sólido y origen de la ESC, celdas extendidas y composición de las celdas.

La ESC δ representada en la figura 3.7 está formada por cuatro celdas extendidas, dos positivas y dos negativas. Formalmente:

$$\delta = E_1 + E_2 - E_3 - E_4 \tag{3.9}$$

Los valores de la función asociada a δ para los puntos $\mathbf{Q_i}$, $1 \leq i \leq 4$, marcados en la figura son los siguientes:

$$\begin{aligned} &f_\delta(\mathbf{Q1}) = 0; \\ &f_\delta(\mathbf{Q2}) = \alpha_2 + \alpha_3 = 1 - 1 = 0; \\ &f_\delta(\mathbf{Q3}) = \alpha_1 = 1; \\ &f_\delta(\mathbf{Q4}) = \alpha_2 + \alpha_4 = 1 - 1 = 0; \end{aligned} \tag{3.10}$$

En el capítulo 5 se describirá un algoritmo basado en la función asociada a una ESC para el cálculo del test de inclusión de un punto respecto de un sólido de forma libre. Sin embargo, antes es necesario resolver el caso particular de aquellos puntos situados en la frontera entre dos celdas; un ejemplo claro es el del punto $\mathbf{Q_5}$, que está situado en la frontera entre E_1 y E_2, y que a su vez está contenido dentro de E_3. Dicho punto está situado en el exterior del sólido; sin embargo, el valor de la función asociada para ese punto es:

$$f_\delta(\mathbf{Q_5}) = \alpha_1 + \alpha_2 + \alpha_3 = 1 + 1 - 1 = 1 \tag{3.11}$$

Para resolver este tipo de situaciones se recurre al concepto de regularización descrito por Requicha [RV77] y reseñado en la sección 2.4.3:

Sea f_δ la función asociada a la cadena δ en $\mathbb{R}^d$, $\mathbf{Q}$ un punto arbitrario de $\mathbb{R}^d$ y $N_\varepsilon(\mathbf{Q})$ un entorno centrado en $\mathbf{Q}$ arbitrariamente pequeño; se definen:

$$\begin{aligned} p &= max\left\{ f_\delta\left(\mathbf{Q}'\right) \mid \forall\ \mathbf{Q}' \in N_\varepsilon(\mathbf{Q}) \right\} \\ n &= min\left\{ f_\delta\left(\mathbf{Q}'\right) \mid \forall\ \mathbf{Q}' \in N_\varepsilon(\mathbf{Q}) \right\} \end{aligned} \tag{3.12}$$

Se define entonces la ***función regular asociada a la cadena*** δ como:

$$\begin{aligned} f_\delta^* : \mathbb{R}^d &\longrightarrow \mathbb{Z} \\ f_\delta^*(\mathbf{Q}) &= 0 \quad si\ \mathbf{Q} \notin clausura\left(interior\left(FF_\delta\right)\right) \\ f_\delta^*(\mathbf{Q}) &= p \quad si\ p > 0 \\ f_\delta^*(\mathbf{Q}) &= n \quad en\ otro\ caso \end{aligned} \tag{3.13}$$

De esta forma se resuelve el problema de puntos como el $\mathbf{Q_5}$, puesto que ahora el valor de la función regular es 0, tal y como se esperaba inicialmente.

Se define una ***cadena simplicial extendida normal*** [Rui01] como aquella que verifica que su función regular asociada toma valor uno para todos los puntos del sólido que representa. Matemáticamente:

$$\delta \; es \; normal \; sii \; f_{\delta}^{*}(\mathbf{Q}) = 1 \quad \forall \; \mathbf{Q} \in FF_{\delta} \tag{3.14}$$

Se seleccionará siempre una ESC normal para representar un sólido de forma libre. El método de construcción de dicha cadena dependerá de cómo esté representada la frontera del sólido a modelar.

3.4 Operaciones con cadenas

Whitney [Whi57] definió las cadenas poliédricas, y sobre ellas las operaciones de suma y producto por un escalar. Demostró que si se reduce el conjunto de números enteros al intervalo $[-(p-1)/2, (p-1)/2]$ con $p > 2$, siendo p un número primo, y se restringen las operaciones de suma y producto de enteros a dicho intervalo, las cadenas poliédricas tienen una estructura de espacio vectorial sobre el cuerpo $\mathbb{Z}/p$.

Las cadenas simpliciales definidas en el modelo SC [FR98], al ser una particularización de las cadenas poliédricas, mantienen estas mismas operaciones y propiedades. Se definen a continuación las operaciones con ESC, de forma que se mantenga la estructura de espacio vectorial.

3.4.1 Suma de dos ESC. Producto por un escalar

Dadas dos cadenas simpliciales extendidas, δ y δ':

$$\delta = \sum_{i=1}^{n} \alpha_i \cdot E_i \tag{3.15a}$$

$$\delta' = \sum_{j=1}^{m} \alpha'_j \cdot E'_j \tag{3.15b}$$

se define la suma de δ y δ' como la cadena [Rui01]:

$$\delta + \delta' = \sum_{i=1}^{n} \alpha_i \cdot E_i + \sum_{j=1}^{m} \alpha'_j \cdot E'_j \tag{3.16}$$

Se define la operación de producto de una ESC δ por un escalar λ como otra cadena formada por las mismas celdas extendidas que la original, y en la

que el coeficiente asociado a cada una de ellas está multiplicado por el escalar [Rui01]:

$$\lambda \cdot \delta = \sum_{i=1}^{n} (\lambda \cdot \alpha_i) \cdot E_i \tag{3.17}$$

Estas operaciones son fundamentales tanto en el modelo SC como en el ESC, ya que todo el trabajo con ESC se basa en sumas de cadenas y productos de escalares por cadenas.

3.4.2 Regularización

Como se ha visto en el apartado anterior, el resultado de sumar dos cadenas, así como el del producto de una cadena por un escalar, es otra cadena simplicial extendida. Por tanto, la cadena resultado tiene una función asociada que tiene que ser regularizada de la misma forma que las cadenas operando para evitar situaciones anómalas. Es por esto por lo que se definen las funciones regulares asociadas a la suma de cadenas y al producto de una cadena por un escalar de la siguiente forma [Rui01]:

$$\begin{aligned} f_\delta +^* f_{\delta'} &= f^*_{(\delta+\delta')} \\ \lambda \cdot^* f_\delta &= f^*_{(\lambda\cdot\delta)} \end{aligned} \tag{3.18}$$

Las ESC tienen estructura de espacio vectorial respecto de las operaciones regularizadas descritas [FR98]. Desde este punto en adelante, y mientras no se indique lo contrario, se considerarán todas las operaciones como regularizadas.

3.4.3 Operaciones booleanas

Las operaciones booleanas de unión, intersección, diferencia y complemento entre sólidos representados mediante ESC se pueden resolver aplicando nuevamente la técnica *divide y vencerás* [BB97], reduciéndolas al cálculo de las operaciones entre las celdas extendidas que forman las cadenas y combinando los resultados en una nueva ESC. Para ello es necesario estudiar la intersección entre dos celdas extendidas, que en general no es otra celda extendida; sin embargo, es un conjunto de puntos que puede ser descompuesto en celdas extendidas, y por tanto se puede representar mediante una ESC normal [RF99a]. La descomposición en celdas extendidas de la intersección entre las

celdas extendidas E y E' se va a representar a partir de ahora como la función $\boldsymbol{ExCell}\,(\boldsymbol{E} \cap \boldsymbol{E'})$ [Rui01].

Dada la representación para la intersección de celdas extendidas, y dadas las cadenas definidas en las fórmulas 3.15a y 3.15b y sus respectivos sólidos asociados FF_δ y $FF_{\delta'}$, las distintas operaciones booleanas se calculan de la siguiente forma:

El sólido intersección $FF_\delta \cap FF_{\delta'}$ se calcula como el sólido asociado a la cadena (véase demostración en [Rui01]):

$$\delta_{FF_\delta \cap FF_{\delta'}} = \sum_{i=1}^{n}\sum_{j=1}^{m} \left(\alpha_i \cdot \alpha'_j\right) \cdot ExCell\ \left(E_i \cap E'_j\right) \tag{3.19}$$

En la figura 3.8 se puede ver el esquema del cálculo de la ESC asociada a la intersección de dos sólidos de forma libre bidimensionales. En la primera fila se puede observar la disposición inicial de los sólidos, así como las ESC que los representan. La segunda y tercera fila muestran los resultados de las intersecciones que no son vacías entre las celdas de las cadenas de los sólidos, así como el cálculo de sus coeficientes asociados, y en la última fila se puede observar el resultado de la composición de dichos resultados en la ESC resultante. La ESC δ que representa el resultado de este ejemplo es la siguiente:

$$\begin{aligned} \delta = {} & ExCell\ (S3 \cap s3) - ExCell\ (S2 \cap s3) \\ & + ExCell\ (ffc1 \cap s3) - ExCell\ (S3 \cap s1) \\ & + ExCell\ (S2 \cap s1) - ExCell\ (ffc1 \cap s1) \end{aligned} \tag{3.20}$$

La función asociada a δ, f_δ se calcula tal y como se indica en la ecuación 3.13. Los valores que adopta dicha función para los puntos indicados en la última fila de la figura 3.8 es como sigue:

$$\begin{aligned} & f_\delta\,(\mathbf{Q1}) = \alpha_1 + \alpha_2 + \alpha_4 + \alpha_5 = 1 - 1 - 1 + 1 = 0 \\ & f_\delta\,(\mathbf{Q2}) = \alpha_2 + \alpha_5 = -1 + 1 = 0 \\ & f_\delta\,(\mathbf{Q3}) = \alpha_4 + \alpha_5 = -1 + 1 = 0 \\ & f_\delta\,(\mathbf{Q4}) = \alpha_5 = 1 \\ & f_\delta\,(\mathbf{Q5}) = \alpha_2 + \alpha_3 + \alpha_5 + \alpha_6 = -1 + 1 + 1 - 1 = 0 \\ & f_\delta\,(\mathbf{Q6}) = \alpha_5 + \alpha_6 = 1 - 1 = 0 \end{aligned} \tag{3.21}$$

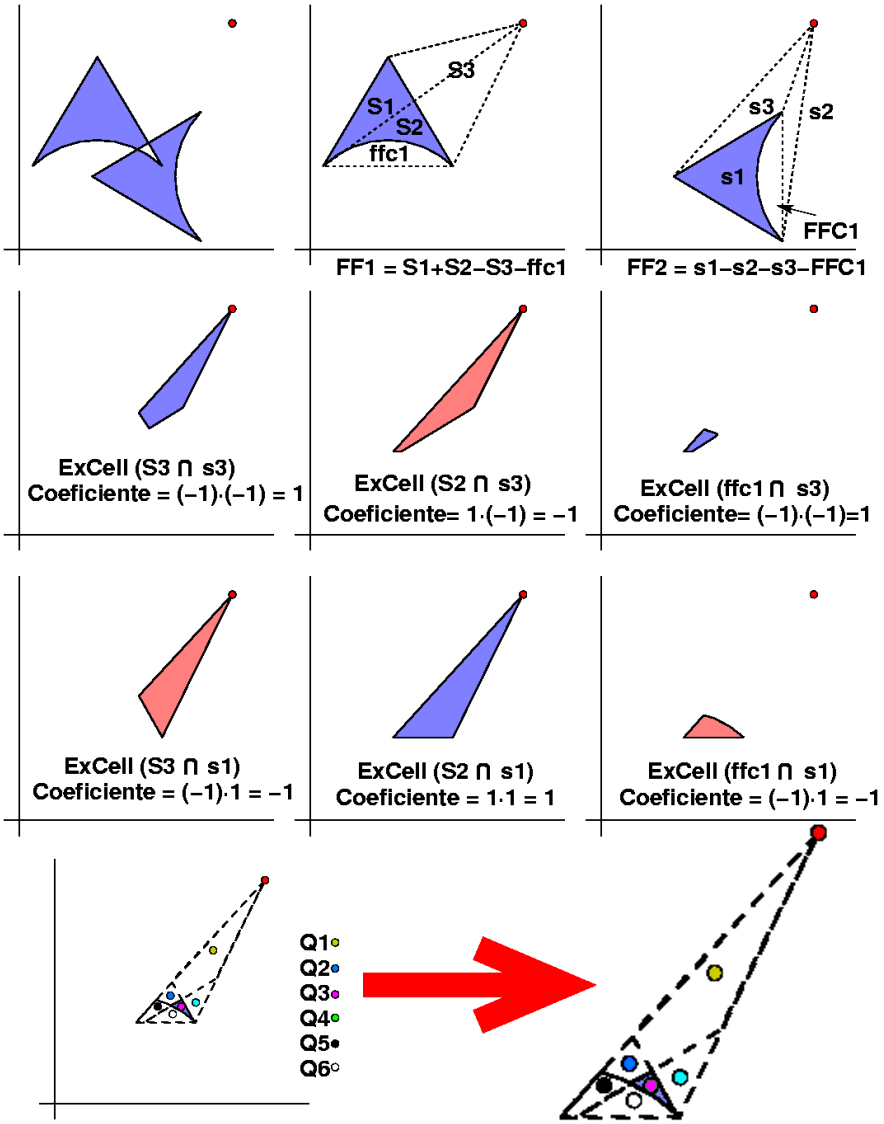


Figura 3.8 Fila superior: dos sólidos de forma libre bidimensionales, y las ESC que los representan. Filas intermedias: términos de la ESC que representa el sólido intersección. Fila inferior: sólido resultado y ampliación

Como se puede observar en los ejemplos y se demuestra en [Rui01], la evaluación de la función asociada a la cadena sólo vale uno para aquellos puntos que pertenecen al sólido resultado de la intersección, con lo que se mantiene la validez del modelo presentado.

Las demás operaciones booleanas se pueden expresar en base a la intersección más las operaciones con cadenas antes descritas [Rui01]. Así, se puede demostrar que la ESC que representa a la unión de dos sólidos se puede calcular sumando las cadenas asociadas a los sólidos operando y restando a este resultado la cadena que representa la intersección [Rui01]. Formalmente se expresa así:

$$\delta_{FF_\delta \cup FF_{\delta'}} = \delta + \delta' - \delta_{FF_\delta \cap FF_{\delta'}} \tag{3.22}$$

Las operaciones entre cadenas también se utilizan para calcular la diferencia de sólidos de manera sencilla, puesto que basta restar a la ESC que representa al minuendo la cadena que representa al sustraendo [Rui01]:

$$\delta_{FF_\delta - FF_{\delta'}} = \delta - \delta' \tag{3.23}$$

Sin embargo, de esta forma la función asociada a la cadena diferencia tomaría valor negativo para un punto que perteneciera a $FF_{\delta'}$ y no perteneciera a FF_δ. Para evitar esta situación, que supondría que la ESC resultado no fuera normal, es preferible calcular la cadena del objeto diferencia como la diferencia entre las ESC que representan al minuendo y a la intersección de los dos operandos [Rui01]:

$$\delta_{FF_\delta - FF_{\delta'}} = \delta - \delta_{FF_\delta \cap FF_{\delta'}} \tag{3.24}$$

La última operación que se presenta en esta sección es la de complemento. El sólido complementario a FF_δ se representa FF_δ^C, y se puede calcular una ESC que lo represente restando la cadena del sólido original de otra que represente el conjunto universal [Rui01]; a esta cadena se la denominará ***cadena universal***, y se la representará como $\delta_{\mathbb{R}}$:

$$\delta_{FF_\delta^C} = \delta_{\mathbb{R}} - \delta \tag{3.25}$$

A efectos prácticos, para representar la cadena universal $\delta_{\mathbb{R}}$ se puede utilizar la ESC de un cubo o una esfera que abarque todo el espacio de trabajo.

En [Rui01] se demuestra que el modelo ESC permite homogeneizar el tratamiento de las operaciones booleanas, reduciéndolas a las operaciones de suma de cadenas y producto por un escalar que fueron presentadas en la sección 3.4.1.

Esto facilita en gran medida la implementación de las operaciones y tratamiento de los resultados, y asegura que dichos resultados son sólidos válidos según el modelo.

3.5 Conclusiones

En este capítulo se ha resumido la base teórica del modelo de Cadenas Simpliciales Extendidas o modelo ESC para el modelado de sólidos de forma libre. Este modelo presenta ventajosas propiedades frente a otros modelos, derivadas de sus especiales características: posibilidad de que las celdas se superpongan, utilización de sólo dos tipos de celdas, de los cuales uno es extremadamente sencillo, y reducción de las operaciones entre sólidos a operaciones algebraicas entre los elementos de las cadenas que los representan.

La eficacia del modelo ESC ya ha sido demostrada para sólidos bidimensionales delimitados por cónicas y curvas de Bézier [RF97], así como para sólidos tridimensionales delimitados por parches algebraicos de formulación implícita [RF99a]. Se han podido desarrollar algoritmos para el cálculo del test de inclusión de un punto en un sólido en dos y tres dimensiones [RF97, RF99b], así como para la conversión a otros modelos de representación [RF98, Rui01, RF02]. La adaptación del modelo ESC a sólidos de forma libre delimitados por distintos tipos de elementos curvos implica concretar la definición de celda de forma libre utilizando dichos elementos como delimitadores, así como definir un algoritmo para el cálculo del test de inclusión de puntos dentro de las celdas definidas de esa manera.

Todo lo anteriormente expuesto invita a pensar que la adaptación del modelo ESC a sólidos delimitados por superficies paramétricas proporcionará similares resultados, con la ventaja añadida de que este tipo de superficies es el estándar utilizado en el mundo del diseño y fabricación asistidos por computador. Así pues, en el siguiente capítulo se estudiará la adaptación de este modelo a parches paramétricos triangulares.

76

76
76

4 Aplicación del modelo ESC a parches paramétricos triangulares

Este capítulo se dedica a la definición de la aplicación del modelo ESC a sólidos de forma libre delimitados por parches paramétricos triangulares. Para este fin se han escogido los parches triangulares de Bézier como superficie de referencia debido a la simplicidad de su formulación y su adaptabilidad al objetivo perseguido. En primer lugar se describe con detalle este tipo de parches y sus propiedades. Posteriormente se define la nueva ffc y se desarrolla un algoritmo para la construcción de una ESC que representa un sólido de forma libre delimitado por parches triangulares de Bézier.

4.1 Parches triangulares de Bézier

En este apartado se van a repasar los fundamentos y las propiedades de los parches triangulares de Bézier.

4.1.1 Definición

La formulación paramétrica definida por Pierre Bézier para curvas se extiende de forma natural a superficies en los parches triangulares de Bézier [Far86, Far93, PBP02]. Estos parches se definen sobre un dominio triangular con tres parámetros denominados u, v y w, tales que:

$$u + v + w = 1$$
$$0 \leq u \leq 1, \quad 0 \leq v \leq 1, \quad 0 \leq w \leq 1$$

y su malla de control está formada por $\frac{1}{2} \cdot (n+1) \cdot (n+2)$ puntos para un parche de grado n. La figura 4.1 ilustra el caso $n = 3$.

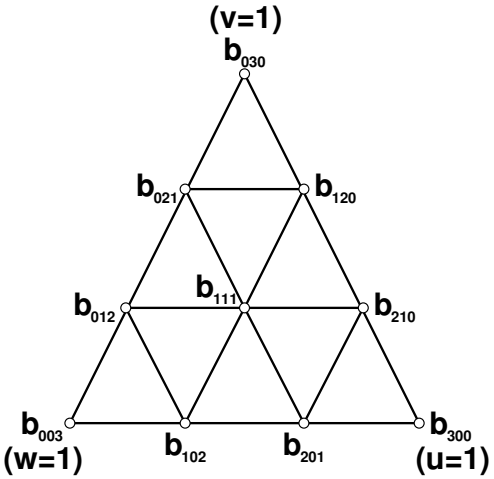


Figura 4.1 Distribución de los puntos de control en un parche triangular de Bézier de grado 3 y su dominio paramétrico

Cada punto de control se identifica mediante tres índices, i, j y k, que verifican la siguiente propiedad para un parche de grado n:

$$i + j + k = n$$
$$0 \leq i \leq n, \quad 0 \leq j \leq n, \quad 0 \leq k \leq n$$

Los puntos de la superficie se definen mediante los polinomios de Bernstein, que para este caso se expresan de la siguiente forma [Far86]:

$$B^n_{i,j,k}(u,v,w) = \frac{n!}{i! \cdot j! \cdot k!} \cdot u^i \cdot v^j \cdot w^k \tag{4.1}$$

notando la tupla (i,j,k) como λ, y la tupla (u,v,w) como τ, la expresión para evaluar un punto perteneciente a un parche queda como sigue [Far86]:

$$\mathbf{b}^n(\tau) = \sum_{|\lambda|=n} \mathbf{b}_\lambda \cdot B^n_\lambda(\tau) \tag{4.2}$$

donde $|\lambda| = n$ representa todas las tuplas (i,j,k) que verifican que $i+j+k = n$.

Otra posible forma de describir los puntos en la superficie utiliza el algoritmo de De Casteljau [dC63], que partiendo de la interpolación de los puntos de control, calcula de forma recursiva nuevas interpolaciones entre los valores calculados en los pasos anteriores para obtener el punto en cuestión. La expresión general de este algoritmo es la siguiente [PBP02]:

$$\mathbf{b}_{\mathbf{i}}^{r}(\tau) = u \cdot \mathbf{b}_{\mathbf{i+e1}}^{r-1}(\tau) + v \cdot \mathbf{b}_{\mathbf{i+e2}}^{r-1}(\tau) + w \cdot \mathbf{b}_{\mathbf{i+e3}}^{r-1}(\tau) \tag{4.3}$$

El algoritmo se desarrolla en n pasos para un parche de grado n. El valor de r va desde 1 a n, incrementándose en 1 en cada iteración. $\mathbf{i}$ representa las tuplas (i, j, k), con la particularidad de que en cada paso del algoritmo $|\mathbf{i}| = n - r$, y los vectores **e1**, **e2** y **e3** representan las tuplas $(1, 0, 0)$, $(0, 1, 0)$ y $(0, 0, 1)$, respectivamente. En el caso base de la recursión, $\mathbf{b}_{\mathbf{i}}^{0}(\tau) = \mathbf{b}_{\mathbf{i}}$.

En la figura 4.2 se puede ver un esquema de los puntos que se calculan en cada paso de un ejemplo de ejecución del algoritmo de De Casteljau para un parche de grado tres. Los puntos calculados en el primer paso se han unido con líneas azules, y los del segundo paso con líneas verdes. El punto resultante se ha marcado en rojo. Se puede deducir de la fórmula 4.3, y se aprecia en la mencionada figura 4.2, que el cálculo de los puntos del parche no es más que un proceso de continuas interpolaciones de los puntos obtenidos en el paso anterior, en el que los valores de la tupla (u, v, w) son los factores de ponderación [Far86]. Una aplicación interesante de este proceso es que el conjunto de puntos obtenido en la penúltima iteración del algoritmo, unidos por líneas verdes en la figura 4.2, determinan el plano tangente al parche en el punto evaluado en el último paso. Así mismo, este algoritmo permite calcular los puntos de control que determinan un subparche definido por un subdominio paramétrico, tal y como se verá en la sección 4.1.4.

En la figura 4.3 se puede ver un parche triangular de Bézier de tercer grado junto con su malla de puntos de control. Al triángulo definido por los puntos b_{003}, b_{030} y b_{300} se le denominará ***triángulo base*** del parche (en rojo en la figura).

Debido a su simplicidad, estos parches se están utilizando en implementaciones hardware; por ejemplo: la tecnología TRUFORM© [ATI01, VPBM01] implementada en las últimas tarjetas gráficas de gama alta fabricadas por ATI está basada en ellos. También se está utilizando este tipo de parches en el desarrollo de esquemas de interpolación de superficies de forma libre a partir

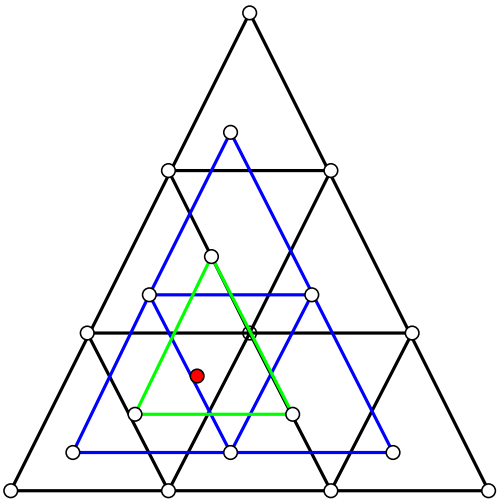


Figura 4.2 Distribución en el dominio paramétrico de los puntos calculados en una ejecución del algoritmo de De Casteljau para un parche de grado tres

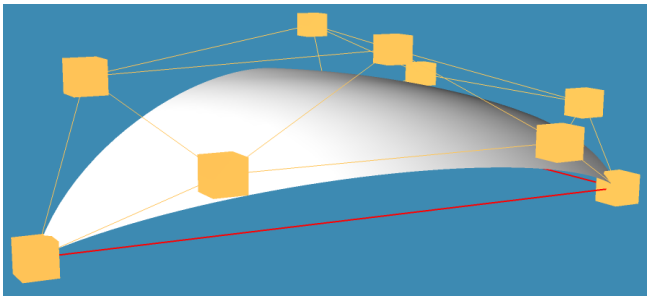


Figura 4.3 Parche triangular de Bézier junto con sus puntos de control y su triángulo base

de muestras de la misma, ya que en general suele ser más fácil modelar superficies complejas utilizando elementos de forma triangular, más simples que los parches rectangulares [Far86]. Algunos de estos esquemas se revisarán con más profundidad posteriormente.

4.1.2 Propiedades

Los parches triangulares de Bézier cumplen una serie de propiedades interesantes, que pueden ser explotadas para optimizar los algoritmos que trabajan

con ellos [Far86, Far93, PBP02]. Estas son las más interesantes para los fines perseguidos en este trabajo:

- **Invarianza ante transformaciones afines**: debido a que el algoritmo de De Casteljau sólo utiliza interpolaciones lineales, y que éstas son correspondencias afines, se deduce que la forma de los parches no se ve afectada ante transformaciones que mantengan la colinealidad de puntos y las proporciones de los objetos.

- **Invarianza ante transformaciones afines de los parámetros**: este tipo de reparametrizaciones sólo implica cambiar el dominio paramétrico, pero no afecta a los cálculos mostrados anteriormente.

- **Inclusión en la envolvente convexa**: todo parche está contenido en la envolvente convexa de sus puntos de control, ya que los factores de ponderación que se utilizan (u, v, w) toman valores entre 0 y 1 siempre.

- **Curvas frontera bien conocidas**: las fronteras de todo parche triangular de Bézier son curvas de Bézier del mismo grado que el parche, y cuyos puntos de control son los puntos de la malla de control correspondientes a los respectivos lados.

- **Convexidad**: un parche triangular de Bézier es convexo si su malla de puntos de control también lo es. Sin embargo, si un parche es convexo, no se puede garantizar que su malla de control también lo sea. Para un estudio más detallado de las limitaciones de esta propiedad, se puede consultar [PBP02].

- **Interpolación de los extremos**: dado un parche triangular de Bézier de grado n, los puntos de control b_{00n}, b_{0n0} y b_{n00} pertenecen al parche.

- **Precisión lineal**: si todos los puntos de la malla de control están contenidos en el mismo plano, el parche es plano.

4.1.3 Continuidad entre parches triangulares de Bézier

Un elemento fundamental a estudiar cuando se construyen superficies a partir de parches de cualquier tipo lo constituyen las condiciones que se han de cumplir para obtener un grado de continuidad entre parches aceptable. Tradicio-

nalmente, se definen dos tipos básicos de continuidad [FCS95]: la continuidad paramétrica (C^r) y la continuidad geométrica (G^r). La unión entre dos curvas tiene continuidad paramétrica de grado r (C^r) si en el punto de unión coinciden, además de los extremos de dichas curvas, las r primeras derivadas de una y otra curva. Si las derivadas sólo son proporcionales, entonces la continuidad se denomina geométrica.

En el caso de las superficies, la unión no es un punto, sino una curva. La continuidad paramétrica se caracteriza entonces de la siguiente forma [Far93]:

> *Dados dos parches cuyos dominios paramétricos son adyacentes, y dada una recta en el espacio paramétrico que atraviesa la frontera entre esos dominios, a la cual le corresponde una curva compuesta por un segmento de curva contenido en cada parche, se dice que la unión entre los parches tiene continuidad C^r si cualquier curva obtenida de este modo tiene continuidad C^r en todos sus puntos.*

Para que dos parches triangulares de Bézier $\mathbf{b}^n$ y $\mathbf{c}^n$, definidos por sus puntos de control $\mathbf{b}_\lambda$ y $\mathbf{c}_\lambda$ que comparten un lado tengan una continuidad de grado C^r en su curva de unión, han de verificar que [Far93]:

$$\mathbf{c}_{\lambda_1} = \mathbf{b}^s_{\lambda_2}(\sigma)\,; \qquad s = 0, 1, \ldots, r \tag{4.4}$$

donde λ_1 representa todas las tuplas de índices (i, j, k) tales que el índice que se corresponde con el vértice de $\mathbf{c}^n$ no compartido por el parche $\mathbf{b}^n$ es siempre igual a s, λ_2 representa todas las tuplas de índices (i', j', k') tales que el índice que se corresponde con el vértice de $\mathbf{b}^n$ no compartido por el parche $\mathbf{c}^n$ es siempre igual a 0, y σ es igual a las coordenadas baricéntricas respecto al dominio base de $\mathbf{b}^n$ del vértice del dominio base de $\mathbf{c}^n$ que no es compartido.

En la figura 4.4 se pueden ver los dominios paramétricos de dos parches, determinados por los triángulos $T_1 = (\mathbf{PQR})$ y $T_2 = (\mathbf{P'QR})$ que comparten el lado entre los puntos $\mathbf{Q}$ y $\mathbf{R}$. Respecto a la expresión 4.4, y tomando T_1 como base de $\mathbf{b}^n$ y T_2 como base de $\mathbf{c}^n$, con $n = 3$ en este ejemplo, σ representa las coordenadas baricéntricas de **P'** respecto de T_1, λ_1 toma valores del tipo (s, j, k), y λ_2 toma valores del tipo $(0, j', k')$. Para que se verifique la continuidad, los puntos de control del parche $\mathbf{c}^n$ se obtienen aplicando r pasos del algoritmo de De Casteljau cumpliendo las restricciones indicadas.

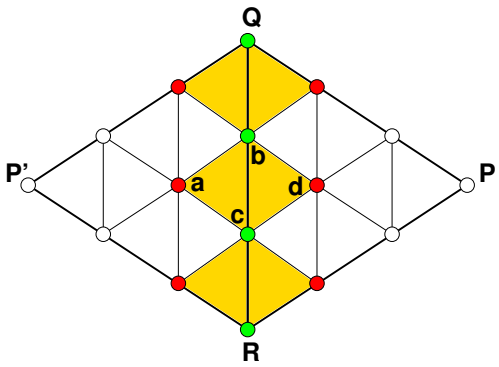


Figura 4.4 Representación en el espacio paramétrico de las condiciones de continuidad C^0 y C^1

También se puede observar en la figura 4.4 la interpretación geométrica de esta condición para los grados de continuidad C^0 y C^1: para que exista continuidad C^0, basta con que los puntos de la frontera de las mallas de control que es compartida, de color verde en la figura 4.4, coincidan. Para que la continuidad sea C^1, es necesario que los triángulos determinados por las dos primeras filas de puntos de control, que aparecen en verde y rojo en la figura 4.4, sean coplanarios, y además sean imágenes afines de sus respectivos dominios triangulares. En el caso más habitual, en que los parches están definidos sobre dominios triangulares equiláteros, estos puntos de control formarán paralelogramos, y verificarán por tanto la condición $\mathbf{a} - \mathbf{b} - \mathbf{c} + \mathbf{d} = \mathbf{0}$, siendo $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ y $\mathbf{d}$ los puntos indicados en la figura.

En general, las condiciones de continuidad paramétrica son muy restrictivas a la hora de desarrollar un esquema de interpolación con cualquier tipo de parches. Restringiéndonos al caso de los parches triangulares de Bézier, para parches de tercer grado el obtener continuidad C^1 por un solo lado ya determina la posición de siete de los diez puntos de control de los parches, de los cuales cuatro estarían implicados en caso de querer unir un nuevo parche por alguno de los otros dos lados. Todo esto lleva a la conclusión de que para construir una superficie compleja con continuidad C^1 sería necesario utilizar parches de un grado mayor, que aumenten el número de grados de libertad. El inconveniente de utilizar parches de grado superior a tres viene dado por varios factores, entre los que se incluyen la inestabilidad numérica, las excesivas oscilaciones de la superfices o el coste computacional que supone el operar con ellos para cálculos como evaluación, intersección, etcétera [Far86, Far93, PBP02], además de la necesidad de desarrollar métodos heurísticos o basados

en complejos métodos matemáticos para determinar los puntos de control no condicionados por el grado de continuidad [Loo94, KPS95, HB03].

Por todo esto, lo más habitual es recurrir a la continuidad geométrica. La continuidad G^1 se denomina también continuidad en el plano tangente o continuidad visual, y se expresa de la siguiente forma:

> *Dados dos parches cuyos dominios paramétricos son adyacentes, se dice que su unión tiene continuidad en el plano tangente si para todos los puntos de la curva de unión entre ellos coinciden los planos tangentes respecto a cada uno de los parches.*

En [Far93] se desarrolla una condición suficiente para conseguir continuidad G^1 en la unión entre dos parches triangulares, basada en el hecho de que los puntos obtenidos en el penúltimo paso del algoritmo de De Casteljau definen el plano tangente al parche en el punto calculado. Por tanto, lo único que se necesita para obtener continuidad G^1 es que estos puntos sean coplanares, y Farin [Far93] obtiene una expresión en función de los puntos de control de los parches a partir de esta restricción.

4.1.4 Subdivisión

La posibilidad de obtener subparches a partir de un parche inicial es una propiedad muy deseable para algoritmos como el cálculo de intersecciones entre un parche y distintos elementos geométricos, como rayos, polígonos u otros parches [AB90, HEFS85, LJCW04, Stü88, RDG01]. Las superfices basadas en la formulación de Bézier, y por tanto los parches triangulares de Bézier, se pueden subdividir fácilmente aplicando el algoritmo de De Casteljau, lo que las hace especialmente interesantes a la hora del modelado [Sei89, PBP02, Gol03].

Dado un parche $\mathbf{b}^n$ de grado n, definido por sus puntos de control $\mathbf{b}_\lambda$, se pueden calcular los puntos de control de un subparche de $\mathbf{b}^n$ de su mismo grado, y que se notará como $\mathbf{c}^n$, de la siguiente forma (véase la figura 4.5):

Suponiendo que las esquinas en el dominio paramétrico del subparche que se quiere obtener poseen las coordenadas baricéntricas $\mathbf{p}$, $\mathbf{q}$ y $\mathbf{r}$, las coordenadas de cada punto de control del subparche c_{ijk} se obtienen aplicando i pasos del algoritmo de De Casteljau (fórmula 4.3) utilizando las coordenadas $\mathbf{p}$ como los

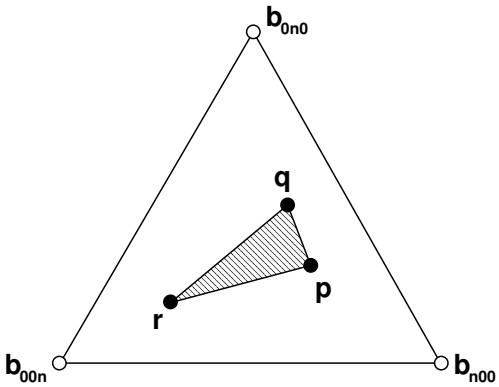


Figura 4.5 Representación de un subparche en el dominio paramétrico

elementos de la tupla τ, luego j pasos con las coordenadas $\mathbf{q}$, y por último k pasos con las coordenadas $\mathbf{r}$.

En general, todo parche triangular de Bézier está separado de su triángulo base una cierta distancia que varía según la curvatura del parche. Al subdividirlo, se obtienen subparches cuyos triángulos base están formados por tres puntos en la superficie del parche, con lo que la distancia entre el subparche y su triángulo base es menor que la distancia entre el parche original y su base. En el límite, para un número suficiente de subdivisiones, no existirá una distancia apreciable entre los subparches y sus bases, y se podrá aproximar el parche por la triangulación formada por los triángulos base de sus subparches. La figura 4.6 ilustra esta característica.

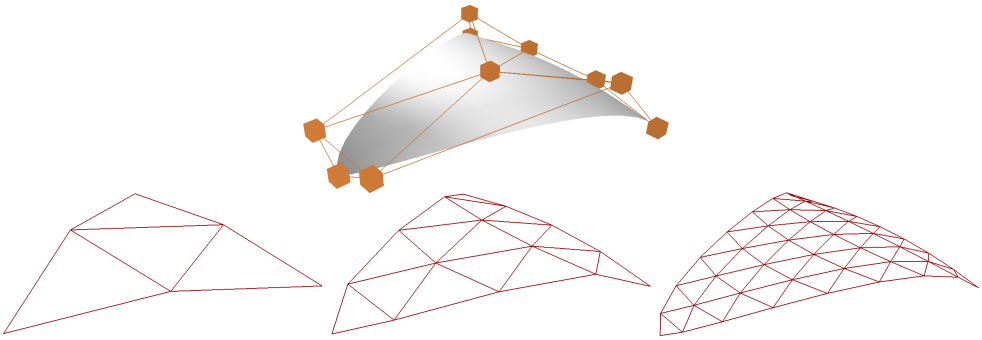


Figura 4.6 Aproximaciones de un parche triangular de Bézier por los triángulos base de sus subparches

4.1.5 Cálculo de intersecciones

La intersección entre superficies es un problema a resolver en todo sistema de modelado de sólidos, puesto que operaciones como la adición y eliminación de material o las operaciones booleanas entre sólidos actúan directamente sobre la frontera de los objetos modelados, y es necesario calcular curvas de intersección en ellas para delimitar las nuevas fronteras del sólido resultante.

Existe una vasta bibliografía tratando el tema de la intersección de superficies, bien como tal o bien en relación al problema del recorte de parches. En general, en el cálculo de las curvas de intersección se pueden distinguir dos tipos de métodos: analíticos y numéricos.

Los **métodos analíticos** buscan una expresión matemática exacta de la solución, pero sólo son aplicables en aquellos casos en los que las superficies son especialmente sencillas [HKE99]. En el caso de las superficies paramétricas, en el que se engloban los parches triangulares de Bézier, no es factible calcular una solución analítica debido a la complejidad y el grado del sistema de ecuaciones resultante.

Los **métodos numéricos** son los más utilizados en el cálculo de intersecciones, y utilizan distintas estrategias para aproximar con el mayor detalle posible la curva de intersección. Estos métodos se pueden clasificar a su vez en [PM02]:

- *Métodos de trazado* [AMY96, LCK02, PMKM04], que partiendo de un punto de la intersección, normalmente situado en una frontera de uno de los parches, hacen un seguimiento de la misma basándose en la geometría local de las superficies.

- *Métodos de subdivisión* [HEFS85, Hoh92, HMPY97], que aplican la técnica *divide y vencerás* al cálculo, aplicando sucesivas divisiones recursivas de los parches hasta que éstos se pueden aproximar mediante elementos planos.

- *Métodos de rejilla* [Pat93], que intersectan isocurvas procedentes de los dos parches para luego interpolar una curva que pase por los puntos de intersección.

También existen trabajos que mezclan dos de las estrategias anteriores, como el de Li y otros [LJCW04], que combina el trazado con la subdivisión, y otros trabajos que no entran en ninguno de los grupos anteriores, como el de Cho y otros [CKKK97], que utiliza algoritmos genéticos.

En el caso concreto de los parches triangulares de Bézier no se han encontrado referencias sobre la intersección o el recorte de los mismos, por lo que como parte de este trabajo se ha implementado un algoritmo numérico basado en subdivisiones para el cálculo de la intersección y el recorte de parches triangulares de Bézier, que hace uso de una estructura de datos basada en árboles de detalle para representar dichos parches [GRF06] y será desarrollado con más profundidad en el capítulo 6. Dicha estructura también es utilizada para otros usos, como la visualización de los parches o el cálculo del test de inclusión de puntos en sólidos de forma libre, por lo que dada su importancia será descrita con más detalle en la sección 4.1.7.

4.1.6 Esquemas de interpolación

Existen numerosos trabajos que utilizan los parches triangulares de Bézier como base para esquemas de interpolación que permiten la reconstrucción de superficies de forma libre a partir de mallas de triángulos que aproximan la geometría a modelar. La mayoría de ellos construyen más de un parche por triángulo, con el objetivo de que la superficie obtenida sea suficientemente continua sin aumentar excesivamente el grado de los parches, ya que para conseguir una continuidad C^1 con un solo parche por triángulo se necesitan al menos parches de quinto grado [PBP02]. Los esquemas que hacen uso de esta división denominan a la superficie formada por la unión de los parches correspondientes a un único triángulo de la malla ***macroparche***, mientras que cada parche individual se denomina ***miniparche***.

En todo esquema de interpolación hay una serie de puntos clave: la continuidad global, que es la que se alcanza entre las superficies asociadas a cada triángulo (macroparches o en su caso, parches individuales); en aquellos casos que se utilizan macroparches, la continuidad local (la que se consigue entre los miniparches), y el conjunto de datos que se necesitan para construir los parches, así como la complejidad asociada al cálculo de la superficie. A continuación se resumen brevemente algunos de los esquemas más representativos.

- El esquema de interpolación de Clough-Tocher [Far86, Far93, PBP02] utiliza macroparches compuestos de tres (mini)parches de grado tres.

En la figura 4.7 se puede ver la subdivisión del dominio paramétrico y los puntos de control que se calculan. Típicamente, las divisiones se hacen uniendo los vértices del dominio con el centroide del mismo por motivos de simetría, aunque es factible utilizar cualquier otro punto del interior del triángulo. Los datos necesarios para construir cada macroparche son: vértices del triángulo base, valores de gradiente en los mismos, y derivadas a través del punto medio en el dominio paramétrico de cada frontera del macroparche.

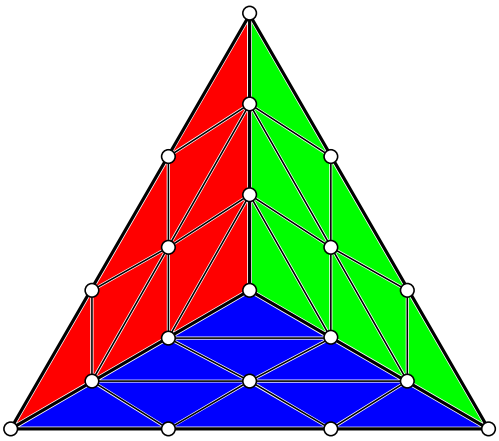


Figura 4.7 Esquema de la división del dominio paramétrico según el esquema de interpolación de Clough-Tocher

Las derivadas a través de la frontera se definen como derivadas direccionales evaluadas en la curva frontera del parche. La dirección respecto a la que se deriva está expresada respecto al dominio paramétrico, y no puede ser paralela al lado del dominio en el que se evalúa. En este esquema de interpolación se suele tomar la dirección perpendicular al lado que se trate, puesto que se va a utilizar la misma información para los dos macroparches que comparten esa curva.

Los puntos de control de las fronteras exteriores del macroparche se obtienen directamente de los datos proporcionados sobre los vértices del triángulo. Los interiores se calculan aplicando tanto la información proporcionada como los puntos calculados previamente, aplicando las

condiciones de continuidad C^1. La superficie obtenida de este modo tiene continuidad C^1 global y local, y C^2 en el centroide del dominio.

- El esquema de interpolación de Powell-Sabin [Far86, Far93, PBP02] realiza una división más compleja. Considera un punto del interior del dominio y un punto en cada lado del mismo para realizar una división en seis miniparches de segundo grado (figura 4.8). Establece como condición que la línea que une los puntos de división interiores de triángulos vecinos ha de intersectar el lado común por el punto de división correspondiente a ese lado, tal y como se aprecia en la figura 4.8. Esta condición se verifica si los puntos de división interiores son los incentros de los triángulos.

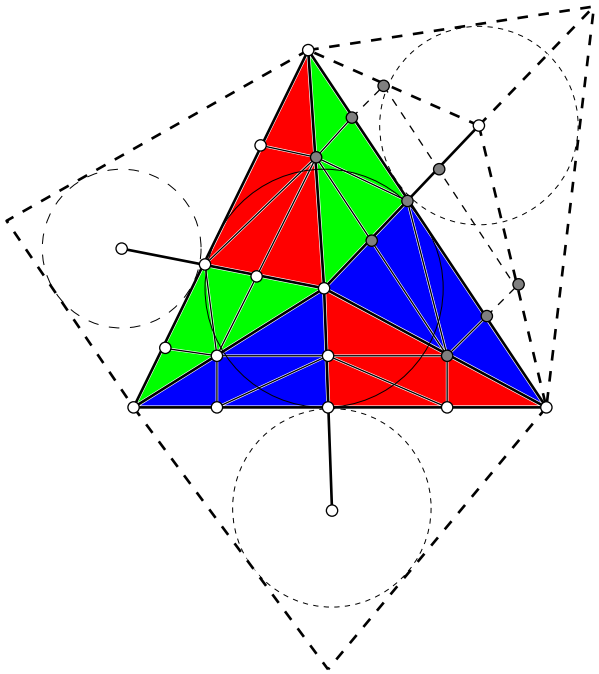


Figura 4.8 Esquema de división de Powell-Sabin

Este esquema proporciona una superficie con continuidad C^1 tanto local como global. Los tres puntos de control vecinos de los vértices del macroparche se obtienen a partir de la información que se le pasa al algoritmo, que es la posición y los gradientes de la superficie en los

vértices. Los demás puntos de control vienen determinados por las condiciones de continuidad C^1. Es de destacar que los nueve puntos de control marcados en gris en la figura 4.8 resultan dispuestos en una superficie bilineal, lo que es característico de la continuidad C^1.

- Los esquemas de Clough-Tocher y Powell-Sabin presentan problemas a la hora de modelar superficies complejas debido a la escasez de grados de libertad disponibles, porque todos los puntos de control están determinados por las condiciones de continuidad. Piper [Pip87] estudia las condiciones de continuidad visual para los parches triangulares de Bézier, y llega a la conclusión de que para conseguir este tipo de continuidad en cualquier circunstancia se necesitan parches de cuarto grado como mínimo. Basándose en estas conclusiones, propone un esquema de interpolación similar al de Clough-Tocher reseñado anteriormente, pero utilizando miniparches de grado cuatro en lugar de cúbicos.

  En primer lugar, y basándose en los datos sobre la posición de los vértices de los triángulos y los vectores normales a la superficie en los mismos, construye un único parche triangular de grado cúbico por triángulo de la malla, de forma que los parches se unen con continuidad C^0. Estos parches son posteriormente subdivididos en tres miniparches también de grado cúbico, cuyos puntos de control son ajustados heurísticamente para mejorar la continuidad obtenida hasta el momento. A continuación, realiza una elevación de grado [Far86, Far93, PBP02] para obtener una malla de puntos de control de cuarto grado para cada uno de los miniparches. Sobre estos puntos de control aplica un ajuste de mínimos cuadrados para conseguir continuidad visual global entre los macroparches, y un ajuste forzado por las condiciones de continuidad C^1 local entre los miniparches.

- Shirman y Séquin proponen un esquema de interpolación basado en macroparches, pero utilizando un método de construcción distinto de los vistos hasta el momento [SS87, SS91]. En su sistema, construyen una red de curvas de Bézier cúbicas entre los vértices de la malla de triángulos, utilizando para ello los datos sobre la posición de los vértices y los vectores normales en los mismos. Las curvas así construidas se unen con continuidad G^1 y forman las fronteras de los macroparches.

  Estas curvas son posteriormente elevadas a cuarto grado, constituyendo así los puntos de control exteriores de cada macroparche. En el

interior de cada macroparche aplica una subdivisión como la de Clough-Tocher o Piper, obteniendo los puntos de control restantes a partir de métodos heurísticos o bien basados en las condiciones de continuidad. Su método asegura un grado de continuidad G^1 tanto local como globalmente.

- El esquema de Loop [Loo94] construye un solo parche de sexto grado por cada triángulo de la malla. Partiendo de la información sobre la posición de los vértices, construye en primer lugar una red de curvas de Bézier de cuarto grado que se corresponderán, previa elevación de grado, con las fronteras de los parches finales. Posteriormente, calcula la capa de puntos de control vecinos de los de las fronteras basándose en las restricciones que impone la continuidad G^1 sobre las derivadas a través de la frontera de los parches. Finalmente, el punto de control central de cada parche se obtiene como una combinación lineal de los ya calculados.

  Una característica importante de este esquema es que no es estrictamente una interpolación de los vértices de la malla de triángulos, sino sólo una aproximación, puesto que utiliza dos parámetros de forma que proporcionan cierto grado de control sobre la forma de los parches. Sin embargo, si a estos parámetros se les asigna los valores adecuados, la superficie interpola los vértices de los triángulos.

  La continuidad de la superficie obtenida es G^1, aunque según las circunstancias, puede aumentar en casos particulares dependiendo del grado de los vértices de la malla de triángulos (número de vértices tales que están conectados con el vértice en cuestión por un lado de un triángulo), o se puede reducir el grado de los parches hasta incluso un valor de cuatro sin perder la continuidad.

- Otro esquema que sigue la metodología de Shirman y Séquin es el de Kolb, Pottmann y Seidel [KPS95]. En este esquema se recurre también a la construcción de una malla de curvas de Bézier de tercer grado que se unen con continuidad G^1 y constituyen las fronteras de los macroparches, y luego construye éstos aplicando el mismo esquema de división que Clough y Tocher o Shirman y Séquin; esto es, dividiendo cada macroparche en tres miniparches triangulares de Bézier que en esta ocasión son de cuarto grado. La continuidad obtenida de esta forma es G^1 tanto local como global.

A diferencia del esquema de Shirman-Séquin, que utiliza métodos heurísticos para establecer los puntos de control que no están directamente fijados por las condiciones de continuidad, Kolb, Pottmann y Seidel utilizan métodos de optimización numérica para minimizar funcionales que caracterizan los parches en función de los puntos de control no determinados por las condiciones de continuidad. De esta forma pueden obtener los valores óptimos para esos puntos.

Una variante del mismo esquema utiliza una malla de curvas de quinto grado que se unen con continuidad G^2 en lugar de G^1, y en lugar de utilizar macroparches, utilizan un único parche de grado siete por cada triángulo de la malla. La superficie final tiene entonces continuidad G^2.

Los datos de entrada en estos algoritmos están constituidos únicamente por la malla de triángulos.

- Un esquema de interpolación rápido y eficiente, aunque de baja continuidad, es el desarrollado por ATI bajo el nombre de TRUFORM© o ***curved PN-Triangles*** [ATI01, VPBM01]. El objetivo de esta técnica es conseguir que la visualización de modelos poligonales tales como personajes de videojuegos sea lo más realista posible, sin necesidad de transferir un alto número de triángulos a la tarjeta gráfica. Con este objetivo, se genera mediante el hardware gráfico un parche triangular de Bézier de tercer grado por cada triángulo que se le envía, definido por la posición de sus vértices y los vectores normales en los mismos, y posteriormente se tesela en un número suficientemente importante de triángulos, que le dé un aspecto más suave al modelo. Incluye además un esquema cuadrático de interpolación de los vectores normales, de forma que se mejora aún más el aspecto visual del resultado. En la figura 4.9 se puede ver un modelo poligonal junto con el resultado de aplicar esta técnica, aproximando cada parche con 256 triángulos.

  Este esquema, aunque muy rápido y sencillo de implementar, sólo garantiza continuidad de tipo C^0 entre los parches, ya que está orientado a la visualización.

  El trabajo de Boubekeur, Reuter y Schlick [BS05, BRS05] está basado en este esquema de interpolación. Básicamente, añaden tres coeficientes de forma a cada vértice de la triangulación inicial, de tal manera

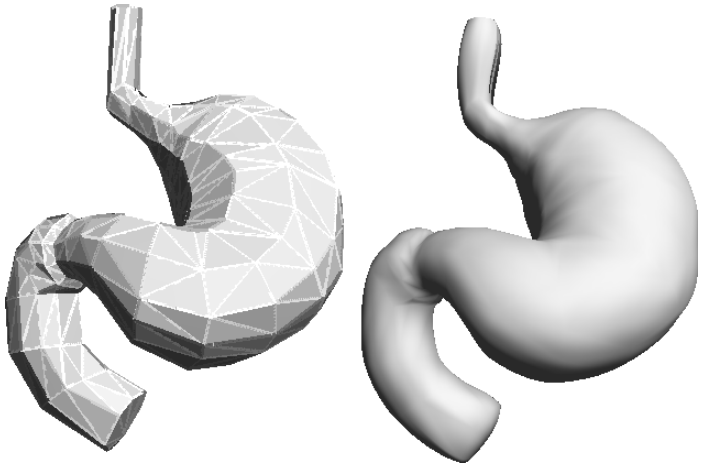


Figura 4.9 Izquierda: malla de 533 triángulos. Derecha: resultado de la visualización aplicando TRUFORM©

que es posible modificar localmente la malla de puntos de control de los parches generados, dando al usuario mayor capacidad expresiva. Los cálculos que requiere esta variante no suponen excesivo coste computacional, y son lo bastante sencillos como para poder implementarse directamente en la GPU de las tarjetas gráficas actuales.

- Los investigadores franceses Hahmann y Bonneau han desarrollado un esquema de interpolación [HB03] que utiliza los macroparches con un esquema de división distinto de los vistos hasta el momento (figura 4.10). Su superficie está formada por macroparches compuestos de cuatro (mini)parches triangulares de Bézier de quinto grado, que se unen con continuidad G^1 global, y C^1 local. En la figura 4.10 se puede apreciar la división que hacen del dominio paramétrico.

  Al igual que algunos de los esquemas expuestos anteriormente, las curvas frontera del macroparche son lo primero que se calcula. Para cada lado se construyen dos curvas de quinto grado, cada una partiendo de uno de los vértices del macroparche, que se unen con continuidad C^1 en el centro de cada lado del dominio paramétrico, y con continuidad G^1 en las uniones con las curvas de otros macroparches. Al mismo tiempo que se calculan los puntos de control de esas curvas, se obtiene la segunda fila de puntos de control del macroparche, asegurando que la unión entre macroparches es continua. De esta forma, se obtienen los puntos de control marcados en blanco en la figura.

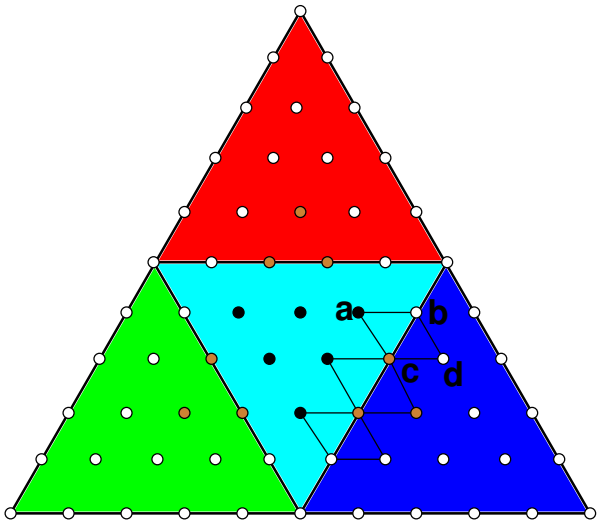


Figura 4.10 Esquema de división paramétrica de Hahmann-Bonneau

Los restantes puntos de control, que aparecen marcados en negro y naranja en la figura 4.10, han de ser tales que se verifiquen las condiciones de continuidad entre miniparches, que dada la forma en que se construye el macroparche se reducen a la expresión $\mathbf{a}-\mathbf{b}-\mathbf{c}+\mathbf{d}=\mathbf{0}$ para cada uno de los paralelogramos formados por los puntos de control en las fronteras de los miniparches, como se puede ver en la figura 4.10. De este modo, los puntos marcados en naranja quedan fijados a expensas de los valores de los puntos marcados en negro, que quedan como grados de libertad para ajustar la forma de la superficie, y que se calculan mediante métodos numéricos de minimización de funciones.

Los cálculos que realizan están condicionados por la posición de los vértices y los vectores normales en los mismos. También realizan estimaciones mediante métodos numéricos de los vectores twist [FCS95] en los vértices del macroparche para calcular los puntos de control.

- Por último, el esquema de Kohlmüller, Nürnberger y Zeilfelder [KNZ03] utiliza los parches triangulares de Bézier para realizar una interpolación de Lagrange sobre una malla de triángulos de mucho nivel de detalle. La propuesta de estos investigadores está orientada a la utilización de los parches como una representación más compacta del modelo, pero sin pérdida de precisión. Su algoritmo tiene los siguientes pasos:

En primer lugar, obtienen una malla de triángulos a partir de una simplificación de la original, y toman cada triángulo de esta nueva malla como base para un parche triangular de Bézier de tercer grado que pasa por diez puntos de la triangulación original. Estos parches inicialmente se unen con continuidad C^0.

A continuación, aplica un algoritmo de coloreado de grafos a los parches con dos colores, blanco y negro, de forma que cada parche tiene como máximo un vecino de su mismo color.

Finalmente, subdivide los parches marcados como blancos aplicando un esquema similar al de Clough-Tocher, y aplica una serie de ajustes a las mallas de puntos de control de todos los parches, de forma que la continuidad global de la superficie es similar a C^1.

A partir de ahora se tomará como esquema de interpolación de referencia el de los *curved PN-Triangles* desarrollado por Vlachos y otros para el fabricante de hardware gráfico ATI [ATI01, VPBM01]. Esta decisión se basa fundamentalmente en la facilidad de implementación, el bajo grado de los parches resultantes y en el hecho de que este esquema está siendo implementado en varios modelos de tarjetas del fabricante antes mencionado, lo que abre las posibilidades de explotación del hardware gráfico en futuros desarrollos. No obstante, cualquier otra elección no hubiera supuesto grandes cambios en los conceptos y algoritmos que en los sucesivos capítulos se irán desarrollando, ya que se aplican a nivel de parche triangular de Bézier; bastaría por tanto trabajar a nivel de los (mini)parches que componen cada macroparche en caso de utilizar algún otro esquema de interpolación.

4.1.7 Jerarquía de niveles de detalle para un parche triangular de Bézier

Los parches triangulares de Bézier son el elemento curvo utilizado en este trabajo para definir las fronteras de los sólidos de forma libre. Las características y propiedades de estos parches ya se indicaron en apartados anteriores, de forma que aquí se tratará la forma en la que se ha gestionado la representación de dichas superficies en la implementación de los algoritmos que se describirán en capítulos sucesivos.

Es necesaria una representación que permita operar de distintas formas con los parches para visualizarlos mediante aproximaciones poliédricas o trazado de rayos, realizar el test de inclusión de puntos, calcular intersecciones con otros elementos geométricos y practicar operaciones de recorte.

La estructura elegida está basada en niveles de detalle, pues se consideró la más apropiada para cubrir de manera correcta todas las necesidades expuestas, basándose en el trabajo existente sobre los PN-Triangles [VPBM01, BRS05] pero ampliando la semántica de la representación. Así, para cada parche se almacena no sólo información sobre sus características significativas (sus puntos de control, normales de referencia, caja englobante de sus puntos de control, etc.), sino que también se almacena una jerarquía de niveles de detalle formados por subparches obtenidos mediante subdivisión [Sei89]. La figura 4.11 muestra un esquema de esta representación.

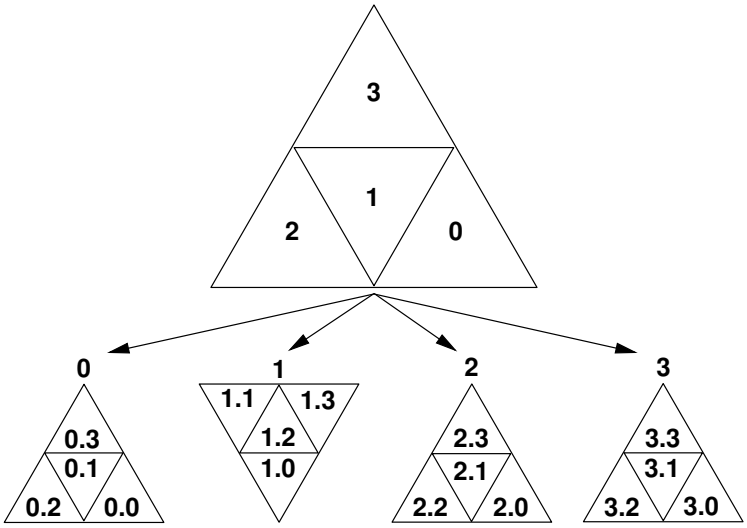


Figura 4.11 Esquema de la subdivisión utilizada en la jerarquía de niveles de detalle. $lod = 1$

Una jerarquía está definida por dos valores:

- El número de niveles del árbol, que se notará de aquí en adelante como $\boldsymbol{n}$.

- El número de puntos que se evalúan en cada lado del dominio paramétrico de cada subparche, sin contar con los extremos del lado, tal y

como definen Vlachos y otros [VPBM01]. A este valor se le denominará de aquí en adelante ***nivel de detalle*** o abreviadamente, ***lod***.

La división del dominio paramétrico es regular, tal y como se ilustra en la figura 4.11, y el número de subparches que se almacenan en el último nivel de la jerarquía viene dado por:

$$n_{SP} = (lod + 1)^{2 \cdot n} \tag{4.5}$$

Por ejemplo: una jerarquía que almacena tres niveles, $n = 3$ con $lod = 1$ mantiene 64 subparches en sus nodos hoja. La figura 4.12 muestra un ejemplo de los niveles para un parche concreto.

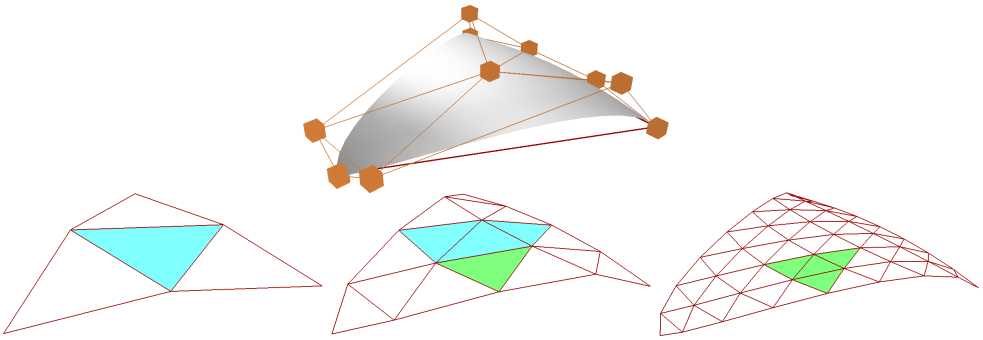


Figura 4.12 Ejemplo de la jerarquía para un parche. $n = 3; lod = 1$

Al almacenar en cada nodo un subparche con las características antes indicadas, es posible evaluar puntos en la superficie del parche en cualquier nivel de la jerarquía, y se puede refinar tanto como se desee la representación añadiendo nuevos niveles en cualquier momento. Por otra parte, se puede obtener una aproximación poliédrica del parche utilizando los triángulos base de los subparches, y tal y como se verá en los capítulos 5 y 6, los algoritmos para el cálculo del test de inclusión de puntos e intersección y recorte de parches utilizan la jerarquía de manera intensiva.

Li y otros [LKL+02] desarrollaron un criterio que permite determinar con antelación el número de subdivisiones a aplicar para obtener una aproximación poliédrica de un parche triangular de Bézier dependiendo de la curvatura del mismo y la precisión que se desee alcanzar. Este criterio podría ser aplicado a la hora de determinar cuántos niveles y subdivisiones por nivel va a tener una jerarquía, pero en las pruebas realizadas tras implementarlo demostró tener

cierta inestabilidad numérica, que provocaba que el número de subdivisiones necesarias según este criterio para algunos parches sencillos fuera anormalmente grande, hasta dos órdenes de magnitud más grande que otros parches similares. Este hecho motivó que se descartara su uso para la construcción de las jerarquías de detalle.

A la hora de fijar el número de niveles y subdivisiones de una jerarquía, es recomendable establecer valores medios para ambos, puesto que pocos niveles con muchas divisiones pueden reducir la eficiencia de los algoritmos, y muchos niveles con pocas divisiones pueden afectar a la precisión. La mayoría de las pruebas se han hecho con jerarquías de entre uno y cuatro niveles, y el número de subdivisiones por nivel ha oscilado entre uno y ocho. Esta decisión ha estado condicionada por el hecho de que el esquema de interpolación utilizado para obtener los parches a partir de una triangulación del sólido [VPBM01] produce parches con poca curvatura, que se aproximan con mucha precisión utilizando pocos niveles.

4.2 Definición de ffc basada en parches triangulares de Bézier

En la sección 3.2.2 se definió una celda de forma libre d-dimensional o ffc como un conjunto de puntos delimitado por un elemento de forma libre de dimensión $d-1$ y uno o varios elementos planos de dimensión $d-1$. Nos centraremos ahora en el caso tridimensional, recurriendo a los parches triangulares de Bézier como elementos delimitadores de forma libre [GRF03].

El punto de partida para la construcción de las ffc es una malla de triángulos que aproxima la geometría de la frontera de un sólido de forma libre, sobre la que se ha aplicado un esquema de interpolación basado en parches triangulares de Bézier como los descritos en la sección 4.1.6. A esta malla se la denominará ***triangulación inicial***. Para que las ffc queden completamente definidas, es necesario determinar el número, características y disposición de los elementos delimitadores de las mismas. Para ello, se define el concepto de plano asociado a dos triángulos de la siguiente forma:

> *Dados dos triángulos t_1 y t_2 pertenecientes a una triangulación de la frontera de un sólido que comparten un lado* **e**, *el* ***plano asociado a t_1 y t_2***, *representado $\pi(t_1, t_2)$, es el plano que contiene al lado* **e**, *y cuyo*

vector director es perpendicular a la media ponderada de los vectores normales a t_1 y t_2.

La estrategia para determinar el factor de ponderación para los vectores normales a t_1 y t_2 determinará la inclinación del plano asociado. Posibles estrategias son: utilizar factores de ponderación iguales a $\frac{1}{2}$, con lo que se obtiene la media aritmética de las normales; factores de ponderación iguales a las áreas de los triángulos, con lo que el plano asociado estará más inclinado hacia el triángulo de mayor superficie; factores iguales a la suma de los dos ángulos que comparten ese lado, etcétera.

Una ffc utilizando parches triangulares de Bézier se define de la siguiente forma:

Sea $\mathbf{S}$ *un sólido de forma libre cuya frontera está representada por un conjunto* $\mathbf{P}$ *de parches triangulares de Bézier, y* $\mathbf{b}^n$ *un parche de dicho conjunto, cuyo triángulo base es* t_0*. Sean* $\mathbf{b}_1^n$, $\mathbf{b}_2^n$ *y* $\mathbf{b}_3^n$ *los parches triangulares de Bézier del conjunto* $\mathbf{P}$ *que comparten un lado con* $\mathbf{b}^n$*, y* t_1, t_2 *y* t_3 *sus respectivos triángulos base. Se define entonces una* ***celda de forma libre*** *de la ESC que representa al sólido* $\mathbf{S}$ *como la intersección de los semiespacios determinados por el plano que contiene a* t_0*, los planos asociados* $\pi(t_0, t_1)$, $\pi(t_0, t_2)$ *y* $\pi(t_0, t_3)$*, y los parches* $\mathbf{b}^n$ *y todos sus parches vecinos que están total o parcialmente contenidos en los semiespacios definidos por dichos planos.*

A efectos de la definición de ffc, se considera que un parche es vecino de otro si comparte al menos un vértice con él. Se denominará ***triángulo base de una ffc*** al triángulo t_0, base del parche $\mathbf{b}^n$ en la definición anterior.

En la figura 4.13 se muestra un ejemplo de celda de forma libre. En ella, los parches vecinos se han pintado en modelo de alambre para distinguirlos del parche principal $\mathbf{b}^n$. Los planos asociados se han representado como cuadriláteros y el triángulo base de la ffc se ha dibujado en rojo.

La estrategia elegida para el cálculo de los planos asociados influye en el número de parches vecinos del principal que entran a formar parte de los elementos delimitadores de la celda. La figura 4.13 ilustra el caso en el que sólo es necesario incluir en los elementos delimitadores de la ffc uno de los parches vecinos, ya que el parche principal y los planos cierran completamente los otros dos laterales de la ffc. En el software diseñado se ha utilizado como factor de

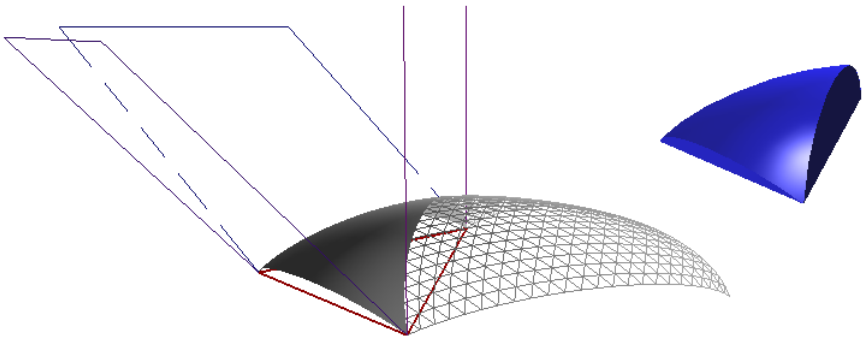


Figura 4.13 Ffc delimitada por parches triangulares de Bézier. Izquierda: elementos delimitadores. Derecha: volumen encerrado

ponderación para calcular los planos asociados el área de los triángulos, ya que se ha comprobado empíricamente que se reduce ligeramente el número de parches vecinos que son necesarios para delimitar las ffc.

El signo asociado a cada ffc vendrá dado por la posición del volumen encerrado con respecto al plano que contiene al triángulo base de la celda. Si el volumen está contenido en el semiespacio positivo, su signo será +1, y si está contenido en el semiespacio negativo, tendrá signo -1, tomando el triángulo base con orientación antihoraria. Así mismo, si los elementos de forma libre atraviesan el triángulo base, se consideran tantas celdas positivas o negativas como componentes conexas haya. En la figura 4.14 se muestran sólo el parche principal y el triángulo base de dos ffc que comparten elementos delimitadores. La región positiva se indica con color verde, y la negativa se indica con un color azulado. También se indica el vector normal del plano que contiene al triángulo base de las celdas, que indica el semiespacio positivo.

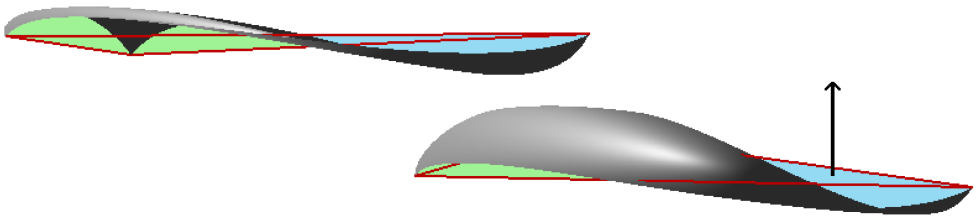


Figura 4.14 Dos vistas desde distintos puntos de dos ffc que comparten los elementos delimitadores

En la sección 5.4 se tratará el algoritmo para calcular el test de inclusión de puntos en las celdas de forma libre así definidas. Este algoritmo, junto con la definición de ffc que se acaba de hacer, posibilitan la particularización del modelo ESC a sólidos delimitados por parches triangulares de Bézier.

4.3 Obtención de la ESC asociada a un sólido de forma libre

Una vez definidas las ffc utilizando los parches triangulares de Bézier, la cadena simplicial extendida asociada a un sólido de forma libre delimitado por este tipo de parches se construye tal y como se indica a continuación.

Partiendo de un sólido de forma libre cuya frontera ha sido aproximada mediante una malla de triángulos, sobre la cual se aplica un esquema de interpolación basado en parches triangulares de Bézier para construir la superficie de forma libre, se crean las celdas extendidas:

- Cada triángulo de la malla forma un único símplice original, formado por sus tres vértices y un punto origen **O**, compartido por todos los símplices.

- Cada triángulo de la malla determina un único (macro)parche, y una o dos celdas de forma libre asociadas a él y sus parches vecinos, tal y como se indica en la sección 4.2.

Es necesario también asignar a cada celda extendida un coeficiente, tal y como se indicó cuando se definió la ESC en la sección 3.3. Para cada símplice, su coeficiente se hace igual al signo de su volumen signado [FT97], mientras que para cada ffc, su coeficiente se hace igual a su signo, calculado como se ha indicado en la sección 4.2. La ESC definida de esta manera es una ESC normal. En la figura 4.15 se pueden ver el símplice y la ffc asociadas a un triángulo.

El punto origen para la construcción de los símplices **O** puede ser cualquier punto del espacio euclídeo $\mathbb{R}^3$. En el capítulo 6 se volverá a tratar la elección del origen de una ESC.

La figura 4.16 muestra la composición de los volúmenes de las celdas de la ESC normal que representa un sólido de forma libre, obtenido a partir de una operación booleana. Visualmente, se han pintado de rojo los píxeles que

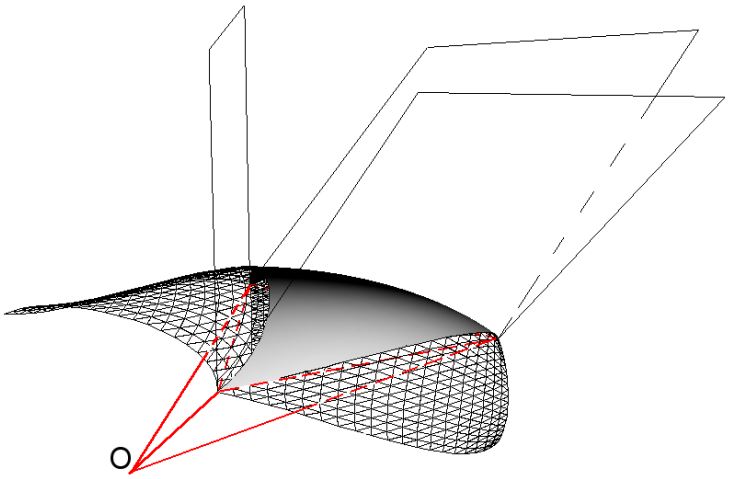


Figura 4.15 Símplice original y ffc

se corresponden con volumen negativo, y de azul los que se corresponden con volumen positivo; de este modo, mientras que en los pasos intermedios del proceso aparecen porciones pintadas de rojo, al final del proceso sólo se pinta de color azul el volumen correspondiente al sólido representado.

4.4 Conclusiones

Los parches triangulares de Bézier son un tipo de superficie paramétrica que se ajusta perfectamente al modelo ESC para la representación de sólidos de forma libre. En este capítulo se ha descrito este tipo de parches, y se ha definido formalmente el concepto de celda de forma libre utilizando parches triangulares de Bézier, con lo que se concreta el primer paso de la adaptación del modelo general descrito en el capítulo 3. El segundo paso de dicha adaptación es el diseño de un algoritmo para el cálculo del test de inclusión de puntos en las celdas de forma libre así definidas, que será tratado en el siguiente capítulo.

Cabe destacar que la definición de celda de forma libre que se ha hecho no depende del esquema de interpolación utilizado para la generación del conjunto de parches que definen la frontera del sólido, con lo que se da total libertad en la elección del mecanismo a utilizar para obtener dichos parches.

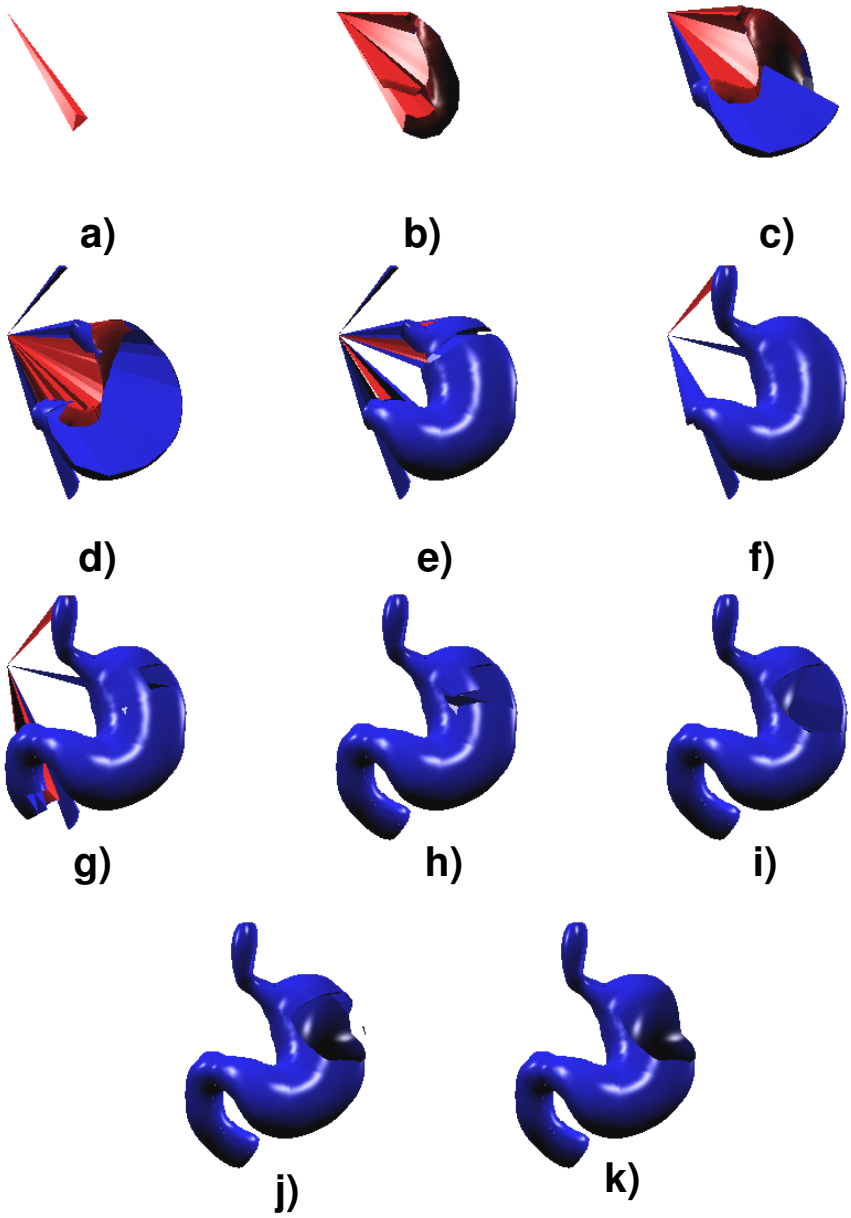


Figura 4.16 Etapas de la construcción de una ESC

104

104
104

5 Test de inclusión de puntos

Una operación fundamental en todo esquema de modelado de sólidos es el test de inclusión de un punto en un modelo de un sólido creado utilizando dicho esquema. Operaciones como la visualización, la conversión entre representaciones, la clasificación de lados o la evaluación de operaciones booleanas dependen de la disponibilidad de un algoritmo robusto y eficiente que resuelva este test.

En este capítulo se detallará el algoritmo desarrollado para el cálculo del test de inclusión de un punto en un sólido de forma libre tridimensional que proporciona el modelo ESC de forma general, para luego profundizar en la adaptación del mismo a sólidos delimitados por parches triangulares de Bézier.

5.1 Definición del problema

El test de inclusión de un punto en un sólido de forma libre se puede enunciar de la siguiente forma:

> *Dado un punto y un sólido de forma libre, determínese si el punto está dentro, fuera o en la frontera del sólido.*

Los algoritmos clásicos para resolver este problema se basan en el teorema de la curva de Jordan [Jor87, Tve80], que establece que una curva cerrada y homeomorfa a una circunferencia en $\mathbb{R}^2$ divide al plano en dos subconjuntos disjuntos, uno interior y otro exterior, delimitados por dicha curva. Como consecuencia, una semirrecta que parte de un punto en cualquier dirección del plano tiene un número impar de intersecciones con la curva si el punto se encuentra en el subconjunto interior, y un número par o cero en caso contrario

(figura 5.1). Este principio se extiende directamente a tres dimensiones, de forma que una superficie homeomorfa a una esfera en $\mathbb{R}^3$ divide el espacio en dos subconjuntos disjuntos, uno interior y otro exterior, delimitados por dicha superficie, y una semirrecta que parte de un punto cualquiera de $\mathbb{R}^3$ intersecta dicha superficie un número impar de veces si el punto está en el subconjunto interior, y un número par de veces o cero en caso contrario.

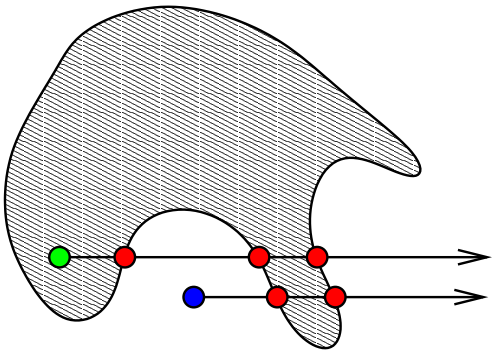


Figura 5.1 Aplicación del teorema de Jordan en el caso de un sólido de forma libre bidimensional

El cálculo de la inclusión de un punto en un sólido de forma libre aplicando el teorema de Jordan implica calcular la solución de complejos sistemas de ecuaciones [HPY96, PS85, YS91], lo que puede llevar a situaciones de inestabilidad numérica [KGMM97, KM96] y con muchos casos especiales [Hui97], como cuando la semirrecta es tangente a la superficie del sólido.

El modelo ESC permite implementar un algoritmo robusto y eficiente para el cálculo del test de inclusión de puntos en sólidos de forma libre [RF97, RF99b, GRF04a], basado en la descomposición del problema en tests más simples realizados sobre las celdas extendidas que forman la ESC que representa al sólido en cuestión. A continuación se describe de forma general este algoritmo, y luego se detallan los elementos directamente relacionados con la representación descrita en el capítulo 4.

5.2 Algoritmo general para el cálculo del test de inclusión

El algoritmo general para el cálculo del test de inclusión de un punto en un sólido modelado con una cadena simplicial extendida se basa en los conceptos de función y sólido asociados a una cadena simplicial extendida normal,

descritos en el apartado 3.3. Teniendo en cuenta las expresiones 3.6, que define una ESC, 3.7, que define la función asociada a una ESC, 3.8, que define el sólido asociado a una ESC y 3.14, que define la condición que verifica el sólido asociado a una ESC normal, el cálculo de la inclusión de un punto en un sólido de forma libre tridimensional modelado mediante una cadena simplicial extendida normal se lleva a cabo mediante la comprobación de la inclusión del punto dentro de cada una de las celdas extendidas de la cadena, y la suma de los coeficientes asociados a aquellas celdas que contengan al punto en su interior. Si dicha suma es igual a uno, entonces el punto es interior al sólido. Si *escSolido* es una ESC que representa un sólido de forma libre y P es un punto, el algoritmo 5.1 detalla este proceso.

```
Algoritmo inclusionESC ( escSolido, P )

  acumulador ← 0

  Para cada celda extendida E de escSolido hacer
    Si P ∈ E entonces
      acumulador ← acumulador + signo ( E )

  Si acumulador = 1 entonces
    Devolver dentro del sólido

  Si no
    Devolver fuera del sólido
```

Algoritmo 5.1 Algoritmo general para el cálculo del test de inclusión

El algoritmo 5.1 no tiene en cuenta casos particulares que dependen de la implementación concreta que se haya realizado de las estructuras de datos para las cadenas y las celdas extendidas. Además, deja pendiente la labor de calcular la posición de un punto respecto de una celda extendida. A continuación se estudiará la forma de determinar de forma eficiente y robusta la posición de un punto respecto de los dos tipos de celdas extendidas que se utilizan: símplices y celdas de forma libre delimitadas por parches triangulares de Bézier. Posteriormente se describirá el algoritmo para el cálculo de la inclusión en un sólido de forma libre implementado y algunos resultados empíricos.

5.3 Cálculo del test de inclusión en un símplice

En la sección 3.2.1 se describieron las características de los símplices tridimensionales. Una característica fundamental para el cálculo de la inclusión de un punto en un símplice es la posibilidad de representar cualquier punto del espacio en función de sus vértices utilizando coordenadas baricéntricas, obtenidas tal y como se indica en la expresión 3.3.

Las coordenadas baricéntricas representan los coeficientes de una interpolación lineal de forma que, si un símplice está definido por sus vértices $\{\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3, \mathbf{T}_4\}$, se verifica que:

$$\mathbf{P} = t \cdot \mathbf{T}_1 + u \cdot \mathbf{T}_2 + v \cdot \mathbf{T}_3 + w \cdot \mathbf{T}_4 \tag{5.1}$$

por tanto, se puede determinar fácilmente la posición de un punto respecto de un símplice con sólo estudiar sus coordenadas baricéntricas:

- Si todas las coordenadas baricéntricas son mayores que cero, entonces el punto está contenido dentro del símplice.

- Si al menos una de las coordenadas baricéntricas del punto es menor que cero, el punto está fuera del símplice.

- Si una de las coordenadas baricéntricas es cero y las otras tres son positivas, el punto está contenido en la cara definida por los tres vértices que se corresponden con las coordenadas baricéntricas positivas.

- Si dos coordenadas baricéntricas son cero y las otras dos son positivas, el punto está contenido en el lado definido entre los vértices que se corresponden con las coordenadas baricéntricas positivas.

- Si tres de las coordenadas baricéntricas son cero, el punto es igual al único vértice que no se corresponde con una coordenada nula.

En el algoritmo para el cálculo de la inclusión en un sólido completo será necesario conocer, en caso de que el punto en cuestión esté en un elemento frontera del símplice, si se trata de un elemento original o no. Esto también es fácilmente verificable, comprobando si la coordenada baricéntrica correspondiente al origen del símplice es distinta de cero.

5.4 Cálculo del test de inclusión en una celda de forma libre

Ya se ha mencionado anteriormente que para la adaptación del modelo ESC a un tipo concreto de superficie son necesarias dos cosas: la definición de la nueva ffc, y un algoritmo para el cálculo del test de inclusión en la ffc así definida. En la sección 4.2 se definió la celda de forma libre delimitada por parches triangulares de Bézier, y ahora se va a proceder a describir el algoritmo para el cálculo del test de inclusión.

El algoritmo propuesto calcula las intersecciones de los elementos delimitadores planos y no planos de la ffc con un rayo que pasa por el punto a estudiar, y que tiene como vector director el vector normal al triángulo base de la ffc (figura 5.2).

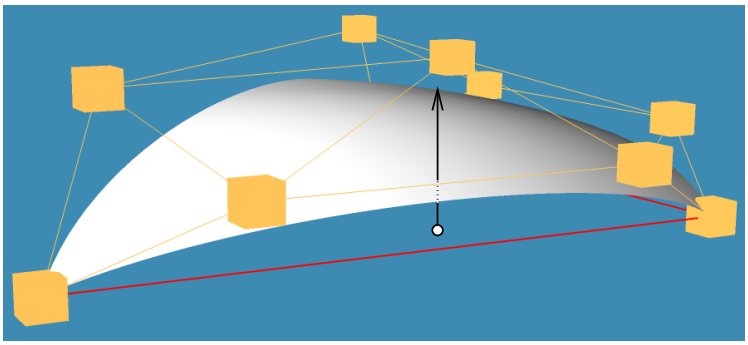


Figura 5.2 Rayo lanzado desde el punto intersectando el parche principal de una ffc

El rayo intersecta al triángulo base y los planos delimitadores de la ffc una vez como máximo. En el caso de los parches, puede haber más de una intersección, con una probabilidad mayor cuanto mayor sea el grado de los mismos, por lo que esta circunstancia es tenida en cuenta en el algoritmo propuesto.

Para calcular la inclusión de un punto en una ffc no es necesario conocer la posición exacta de los puntos de intersección del rayo con los parches, sino que basta saber si son puntos de entrada o de salida del rayo respecto de la superficie del sólido y la posición del punto a estudiar respecto a estas intersecciones. Por este motivo, basta con intersectar el rayo con una triangulación que aproxima con suficiente detalle cada parche. Más concretamente, se ha utilizado la

jerarquía de niveles de detalle descrita en la sección 4.1.7, de forma que sólo se ha recurrido a las triangulaciones de mayor detalle en aquellas zonas en las que realmente podía existir una intersección. El algoritmo 5.2 describe este proceso con mayor detalle.

```
Algoritmo interseccionRayoPTB ( rayo, parche )

  nivelTriangulacion ← 1
  dominio ← parche.dominio

  Repetir
    t ← triangulacion ( parche, nivelTriangulacion, dominio )
    interseccionRayoTriangulos ( rayo, t, p, nuevoD )
    dominio ← nuevoD

    Si p ≠ ∅ entonces
      nivelTriangulacion ← nivelTriangulacion + 1
  Hasta que ( p = ∅ ) o ( nivelTriangulacion = maxNivel )

  Devolver p
```

Algoritmo 5.2 Algoritmo para el cálculo de la intersección rayo-parche triangular de Bézier

La función *triangulacion* que aparece en el algoritmo 5.2 devuelve la triangulación que aproxima el subparche definido por el subdominio paramétrico que se le pasa como tercer argumento según el nivel de detalle que se le indique. El procedimiento *interseccionRayoTriangulos*, por su parte, almacena en sus dos últimos parámetros las intersecciones del rayo con la triangulación que se le pasa como argumento, así como los subdominios paramétricos en los que se han hallado dichas intersecciones; esta información es la que permite incrementar sólo de manera local el refinamiento de la malla y reducir el número de tests rayo-triángulo necesarios.

Las intersecciones del rayo con los parches se etiquetan como *intersecciones de entrada* o *intersecciones de salida* dependiendo de la orientación del triángulo que se intersecta y del signo de la ffc. Para ello, se estudia el signo del producto escalar entre el vector normal al triángulo base de la ffc y el

vector normal al triángulo intersectado. Los casos posibles se muestran en la figura 5.3:

- Si el producto escalar es positivo, y la ffc es positiva, el punto de intersección se etiqueta como de salida. Punto **A** en la figura 5.3.

- Si el producto escalar es negativo, y la ffc es positiva, el punto de intersección se etiqueta como de entrada. Punto **B** en la figura 5.3.

- Si el producto escalar es positivo, y la ffc es negativa, el punto de intersección se etiqueta como de entrada. Punto **C** en la figura 5.3.

- Si el producto escalar es negativo, y la ffc es negativa, el punto de intersección se etiqueta como de salida. Punto **D** en la figura 5.3.

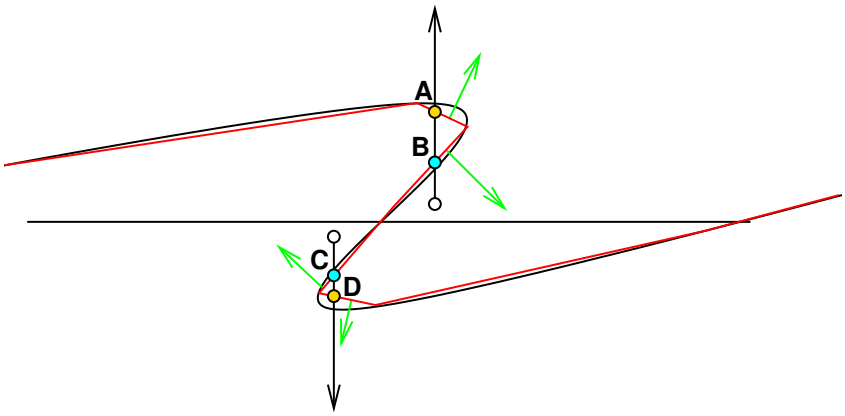


Figura 5.3 Detección de intersecciones de entrada (color cian) y salida (color dorado). Los puntos a estudiar aparecen en blanco

La intersección del rayo con el plano que contiene al triángulo base de la ffc se etiqueta como de entrada siempre, salvo el caso en que la primera intersección con los parches que se encuentre por encima de ella también sea de entrada; en dicho caso, se etiquetará como de salida.

Una vez que se dispone de las intersecciones del rayo con los parches y los planos que delimitan la ffc, se ordenan y se descartan aquellas que no se encuentran en los semiespacios determinados por los elementos delimitadores que participan en la definición de la celda, tal y como se ilustra en los casos de los puntos **A**, **B**, **D** y **E** de la figura 5.4, en la que las intersecciones descartadas

aparecen en rojo. Cuando una intersección es descartada, su etiqueta pasa a la siguiente más próxima al plano que contiene al triángulo base de la ffc, tal y como se observa en la figura 5.4.

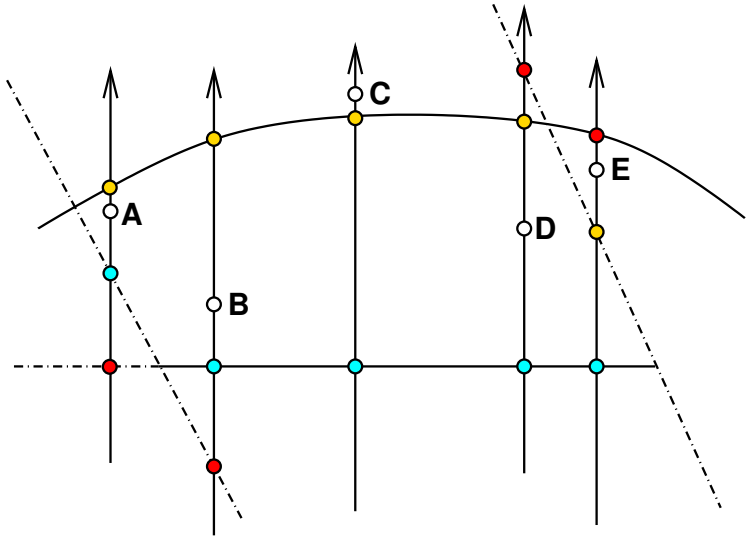


Figura 5.4 Esquema 2D mostrando la selección de puntos de intersección para el estudio de la inclusión en una ffc. Los puntos a estudiar aparecen en blanco, en azul las intersecciones de entrada, y en amarillo las intersecciones de salida

Una vez se ha hecho el descarte de las intersecciones, para calcular la inclusión de un punto en la celda basta comprobar si se encuentra situado entre una intersección de entrada y otra de salida. Tal es el caso de los puntos **A**, **B** y **D** de la figura 5.4, que son clasificados como dentro de la ffc. En cualquier otra situación, como las de los puntos **C** y **E** de la figura 5.4, los puntos son clasificados como fuera de la ffc.

Este algoritmo se puede optimizar en varios aspectos que se detallan a continuación:

- Si sólo hay una intersección de salida con los parches, y el punto está entre la base de la celda y dicha intersección, no es necesario refinar

hasta el máximo nivel de detalle, puesto que tal y como se ha mencionado antes, sólo es necesario conocer la posición del punto respecto de las intersecciones, no las intersecciones exactas.

- Se puede hacer un test eliminatorio preliminar, consistente en comprobar si el punto se encuentra dentro de los semiespacios determinados por los planos laterales, el plano que contiene a la base, y un quinto plano, determinado por los puntos de control de los parches que delimitan la celda más alejados del plano que contiene a la base (figura 5.5). Estos planos delimitan un volumen englobante de la ffc más ajustado que una caja englobante clásica, de tal manera que si el punto se encuentra fuera, no se hace necesario el calcular las intersecciones del rayo, que son la parte más costosa del algoritmo.

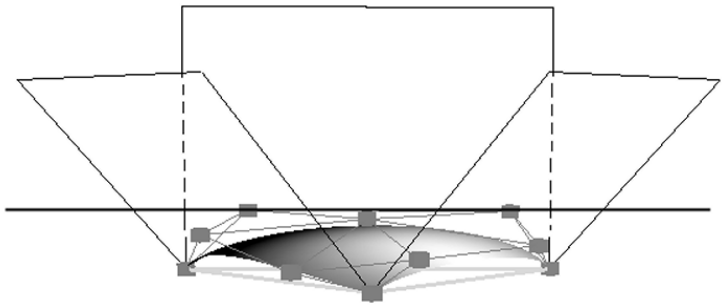


Figura 5.5 Planos utilizados en el test preliminar

- Durante el cálculo de las intersecciones con los parches, si se da el caso de que el rayo intersecta a un triángulo en uno de sus vértices, no es necesario continuar refinando, puesto que los vértices de los triángulos siempre son puntos pertenecientes a los parches.

- Si el punto a estudiar coincide con una de las intersecciones no descartadas, y ésta coincide con un vértice de la triangulación, el punto está en la frontera del sólido. Si el punto coincide con una intersección con los planos delimitadores, está en la frontera de la ffc.

El algoritmo 5.3 resume el algoritmo descrito para calcular la inclusión de un punto en una ffc. El parámetro P se corresponde con un punto, y el parámetro *celda* con una ffc.

```
Algoritmo inclusionPuntoFfc ( P, celda )

  Si ( P está fuera de celda.planos ) entonces
    Devolver FUERA

  r ← rayo ( P, celda.base.normal )

  Para cada ptb ∈ celda.parches hacer
    interseccionRayoPTB ( r, ptb, intsParche )
    etiquetaIntersecciones ( intsParche )
    ints ← ints ∪ intsParche

  DescartaIntersecciones ( ints, celda.planos, r )

  pos ← posicionPunto ( P, ints )

  Si ( pos = entre entrada y salida ) entonces
    Devolver DENTRO

  Si no
    Devolver FUERA
```

Algoritmo 5.3 Algoritmo para el cálculo de la inclusión de un punto en una celda de forma libre

5.5 Test de inclusión para un sólido delimitado por parches triangulares de Bézier

Una vez presentados en las secciones anteriores los algoritmos para determinar la inclusión de un punto en los dos tipos de celdas extendidas utilizadas en la adaptación del modelo ESC que se presenta en esta tesis, se detalla ahora el algoritmo que adapta el algoritmo genérico descrito en la sección 5.2 para el cálculo del test de inclusión de un punto en un sólido de forma libre delimitado por parches triangulares de Bézier y modelado utilizando una cadena simplicial extendida.

Este algoritmo tiene muy pocos casos especiales a tratar, lo que lo hace simple y estable. Como ya se mencionó anteriormente, la idea es reducir el test a la solución del mismo en las celdas extendidas y la combinación posterior

de los resultados en una suma, ya que las celdas se pueden solapar total o parcialmente [RF97, RF99b, GRF04a]. En el caso de los símplices, la inclusión de un punto se calcula mediante el estudio de las coordenadas baricéntricas, tal y como se indicó en la sección 5.3, y en el caso de las ffc, se utiliza el algoritmo de inclusión descrito en la sección 5.4.

La figura 5.6 muestra parte de una malla de triángulos que aproxima un sólido de forma libre, obtenida separando las caras de un modelo, tal y como se indica en la misma figura, y el mismo trozo de malla sobre el que se han construido los parches y marcado los símplices originales y las celdas de forma libre. Así mismo, se han marcado cinco puntos con los cinco posibles casos particulares que aparecen en el algoritmo de inclusión.

Los casos particulares mostrados en la figura 5.6 se tratan de la siguiente forma:

- Si un punto está en una cara original de un símplice, es compartido por éste y su vecino. Es el caso del punto **Q1**, que es compartido por los símplices asociados a los triángulos **T5** y **T6**. En este caso, cada símplice aporta a la suma sólo $+\frac{1}{2}$ ó $-\frac{1}{2}$, dependiendo de su signo.

- Si un punto está en una arista original de un símplice, puede ser compartido por muchos tetraedros. El punto **Q2** ilustra este caso, en el que está siendo compartido por todos los símplices dibujados. La solución es sumar $+1$ una sola vez para todos los símplices positivos, y -1 también una sola vez para todos los símplices negativos.

- Si un punto está en uno de los triángulos de la malla inicial, pertenece a la ffc y al símplice que se construyen a partir de dicho triángulo. El punto **Q3** se encuentra en esta situación al encontrarse en el triángulo **T6**; es por tanto compartido por la celda ffc6 y el símplice asociado a dicho triángulo. En estos casos, ambas celdas extendidas contribuyen a la suma final con $+\frac{1}{2}$ ó $-\frac{1}{2}$, dependiendo de su signo.

- Si un punto está en un lado de un triángulo de la malla inicial es compartido por dos símplices y dos ffc, como se puede ver en el punto **Q4**, que es compartido por las ffc **ffc1** y **ffc6**, y por los símplices asociados a **T1** y **T6**. Este caso se resuelve sumando $+1$ ó -1 una sola vez y sólo en el caso de que el signo del símplice y la ffc asociados al mismo triángulo

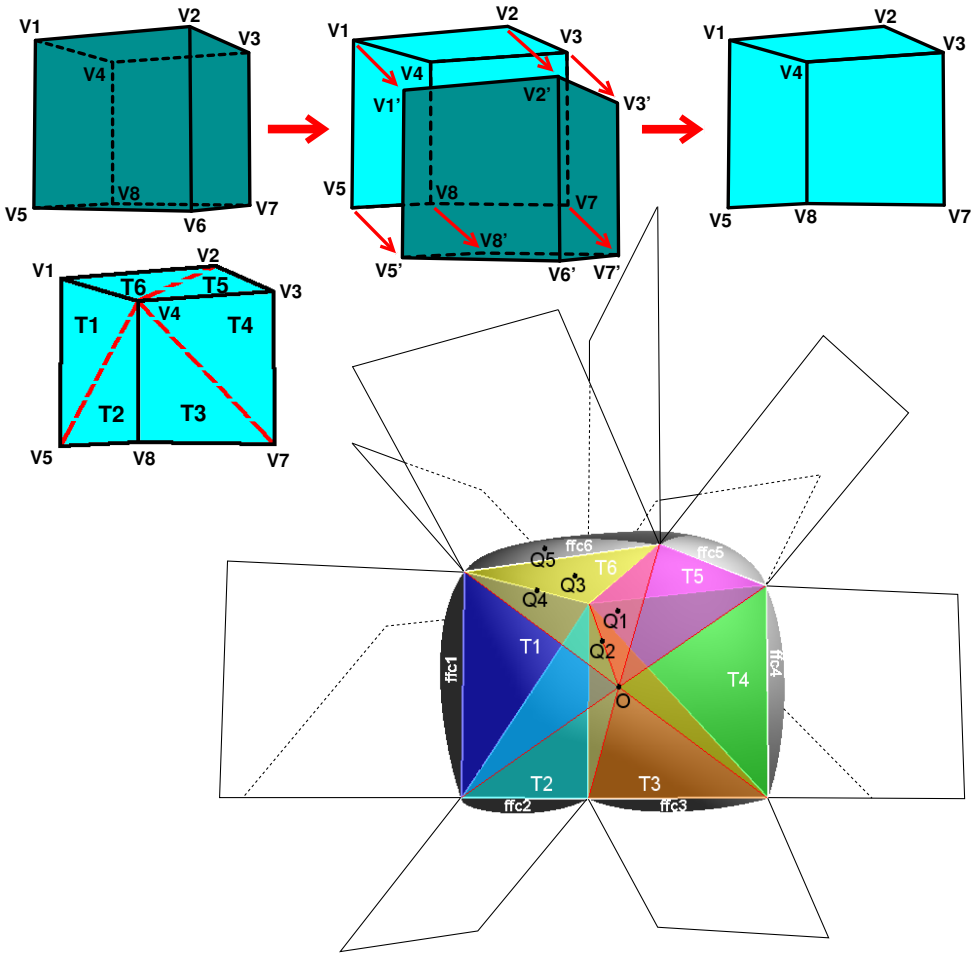


Figura 5.6 Arriba: proceso de obtención de los triángulos, separando las caras de un modelo. Abajo: a la izquierda, triangulación inicial; a la derecha: símplices originales, celdas de forma libre y puntos marcando casos especiales. El origen de los símplices es un punto en la línea de visión entre el observador y la malla

coinciden. El signo del sumando, una vez más, viene determinado por el signo de las celdas extendidas.

- Si el punto está en uno de los planos laterales que delimitan una ffc, el punto pertenece a las dos celdas que comparten dicho plano. **Q5** se encuentra en esta situación. El tratamiento para este caso consiste en sumar $+\frac{1}{2}$ ó $-\frac{1}{2}$, por cada ffc, dependiendo de su signo.

La detección de estos casos particulares en el caso de los símplices se basa en el estudio de las coordenadas baricéntricas; más concretamente, en la comprobación de cuántas y cuáles son iguales a cero, tal y como se explica en la sección 5.3. En el caso de las ffc, la detección de casos particulares se basa en la comprobación de si el punto a estudiar es igual a alguna de las intersecciones del rayo con los planos laterales, o si el punto está en el triángulo base de una celda.

En el algoritmo 5.4 se incluye el pseudocódigo que describe el algoritmo para el cálculo del test de inclusión de un punto en un sólido de forma libre definido con parches triangulares de Bézier.

```
Algoritmo inclusionPuntoSolido ( P, S )

  inclusion ← 0.0

  Para cada tr ∈ S.triángulos hacer
    tet ← símplice asociado a tr
    ffcs ← celdas de forma libre asociadas a tr

    sT ← signo ( tet )

    Para cada ffc ∈ ffcs hacer
      sFfc ← signo ( ffc )
      resFfc ← inclusionPuntoFfc ( P, ffc )
      inclusion ← inclusion + procesaResFfc ( resFfc,
                                              sFfc, sT )

    resTet ← inclusionPuntoSimplice ( P, tet )
    inclusion ← inclusion + procesaResTet ( resTet, sT )

  Si ( inclusion = 1 ) entonces
    Devolver DENTRO

  Si no
    Devolver FUERA
```

Algoritmo 5.4 Algoritmo para el cálculo de la inclusión de un punto en un sólido de forma libre

La función *procesaResFfc* del algoritmo 5.4 devuelve el valor que ha de ser añadido a la sumatoria sobre la variable *inclusion*, de acuerdo al resultado del algoritmo de inclusión en una celda de forma libre. El valor a devolver es el signo de la ffc si el punto pertenece al interior de la celda o está sobre los parches que la delimitan o en una arista todavía no procesada del triángulo base. También puede devolver $+\frac{1}{2}$ ó $-\frac{1}{2}$ si el punto está en un plano lateral de la celda o en el triángulo base de la misma, o cero si el punto está fuera de la ffc o en una arista del triángulo base procesada anteriormente.

La función *procesaResTet* en el algoritmo 5.4 devuelve el valor que ha de ser añadido a la sumatoria sobre la variable *inclusion*, de acuerdo al resultado del algoritmo de inclusión en un símplice. Los valores devueltos son el signo del símplice si el punto está dentro de él o en una arista original no procesada, cero si el punto está fuera o en una arista original procesada anteriormente o en una arista del triángulo asociado al símplice y $+\frac{1}{2}$ ó $-\frac{1}{2}$ si el punto está en una cara original del símplice.

Análisis del algoritmo

El algoritmo propuesto es robusto, ya que tiene muy pocos casos particulares, y éstos son fácilmente detectables. En cuanto a estabilidad numérica, el algoritmo puede presentar algunos problemas del mismo modo que otras soluciones propuestas, por lo que se trabaja con un valor umbral a la hora de realizar operaciones de comparación entre números en coma flotante.

Es importante el hecho de que en condiciones normales, la mayor parte del volumen de los sólidos es modelado sólo con símplices, por lo que el número de veces que es necesario recurrir al algoritmo para comprobar la inclusión de un punto en una ffc es muy pequeño. Así mismo, el uso que se hace de las triangulaciones de detalle está limitado a unos pocos parches en cada caso, con lo que se evita gran número de operaciones que podrían ralentizar el proceso.

En términos de complejidad computacional, hay que tener en cuenta que tanto el número de símplices como el número de celdas de forma libre depende linealmente del número de triángulos de la malla inicial. Como el algoritmo hace un recorrido por todas las celdas extendidas para calcular el test de inclusión local acumulando los resultados en una suma, la complejidad total del algoritmo es lineal respecto al número de triángulos de la malla inicial que aproxima la superficie del sólido.

Se han realizado una serie de pruebas con las cuatro mallas de triángulos que se pueden observar en la fila superior de la figura 5.7. La fila inferior de la figura muestra los sólidos de forma libre resultantes al construir los parches triangulares de Bézier sobre los triángulos de las mallas utilizando el esquema de interpolación de Vlachos y otros [VPBM01]. La tabla 5.1 muestra algunos datos sobre las mallas.

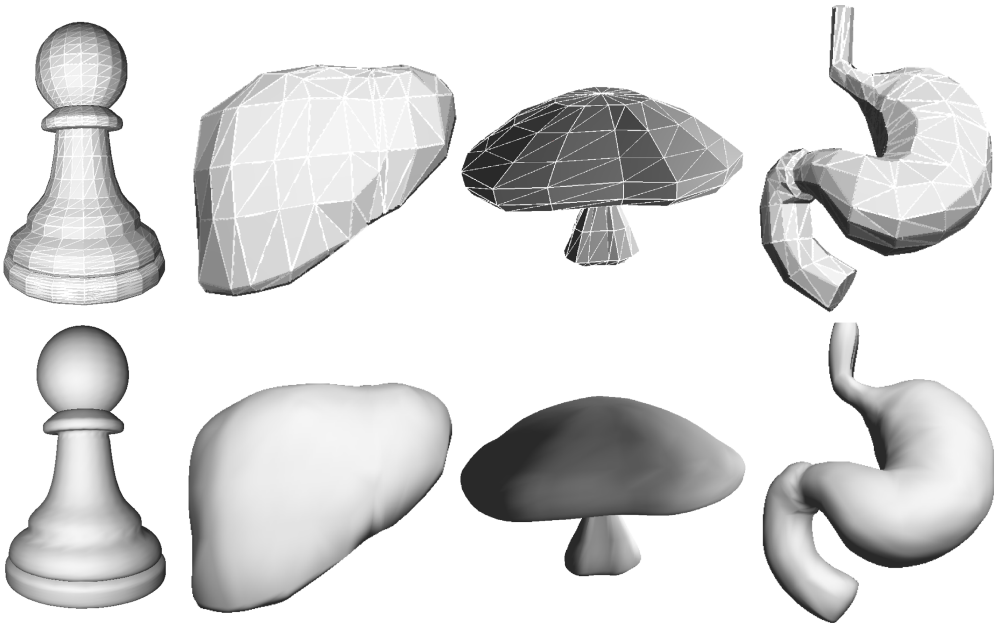


Figura 5.7 Arriba: mallas de triángulos originales. Abajo: sólidos de forma libre

Malla	Vértices	Triángulos
Peón	1202	2399
Hígado	139	274
Seta	226	448
Estómago	269	534

Tabla 5.1 Características de las mallas utilizadas en las pruebas del test de inclusión

Las tablas 5.2, 5.3 y 5.4 muestran algunos resultados obtenidos con el algoritmo para el cálculo de la inclusión propuesto. Para estas pruebas se ha aplicado el algoritmo a puntos distribuidos regularmente en el volumen de la caja englobante de los sólidos, y se ha comparado el número de intersecciones que es

necesario calcular para un rayo que pase por el punto a estudiar en el caso de aplicar un algoritmo basado directamente en el teorema de Jordan y en el caso de aplicar el nuevo algoritmo. La columna de la izquierda indica el tamaño del conjunto de puntos y su distribución en las tres dimensiones, mientras que en las otras columnas se muestra el número de intersecciones a calcular y el promedio de intersecciones por cada punto utilizando tanto el algoritmo basado en el teorema de Jordan como el algoritmo descrito aquí. Tal y como se ha comentado anteriormente, el número de intersecciones a calcular con el nuevo algoritmo se reduce drásticamente, puesto que la mayor parte del volumen del sólido está representado por símplices.

Número de puntos	Int. Jordan	Int./Pto. Jordan	Int. ESC	Int./Pto. ESC
1000 (10×10×10)	611	0.611	**58**	**0.058**
1728 (12×12×12)	1116	0.646	**229**	**0.133**
2744 (14×14×14)	1767	0.644	**378**	**0.138**
3375 (15×15×15)	2208	0.654	**399**	**0.118**
4096 (16×16×16)	2697	0.658	**609**	**0.149**
4913 (17×17×17)	3251	0.662	**610**	**0.124**

Tabla 5.2 Resultados de las pruebas del test de inclusión en el modelo del estómago

Número de puntos	Int. Jordan	Int./Pto. Jordan	Int. ESC	Int./Pto. ESC
1000 (10×10×10)	816	0.816	**147**	**0.147**
1728 (12×12×12)	1466	0.848	**303**	**0.175**
2744 (14×14×14)	2349	0.856	**566**	**0.206**
3375 (15×15×15)	2872	0.851	**681**	**0.202**
4096 (16×16×16)	3558	0.869	**741**	**0.181**
4913 (17×17×17)	4321	0.880	**940**	**0.191**

Tabla 5.3 Resultados de las pruebas del test de inclusión en el modelo del hígado

La gráfica de la figura 5.8 resume visualmente los resultados de las tablas. Se puede apreciar que la reducción del número de intersecciones a calcular es significativa. Este hecho, unido a la simplicidad y robustez del algoritmo lo hace ser objetivamente más ventajoso que otras soluciones anteriores.

Para evaluar la eficiencia del algoritmo, se ha recurrido a estudiar los tiempos empleados en la carga de los datos y en el cálculo de la inclusión de un

Número de puntos	Int. Jordan	Int./Pto. Jordan	Int. ESC	Int./Pto. ESC
1000 (10×10×10)	621	0.621	**30**	**0.030**
1728 (12×12×12)	1089	0.630	**130**	**0.075**
2744 (14×14×14)	1781	0.649	**138**	**0.050**
3375 (15×15×15)	2212	0.655	**136**	**0.040**
4096 (16×16×16)	2715	0.663	**166**	**0.041**
4913 (17×17×17)	3296	0.671	**136**	**0.028**

Tabla 5.4 Resultados de las pruebas del test de inclusión en el modelo del peón de ajedrez

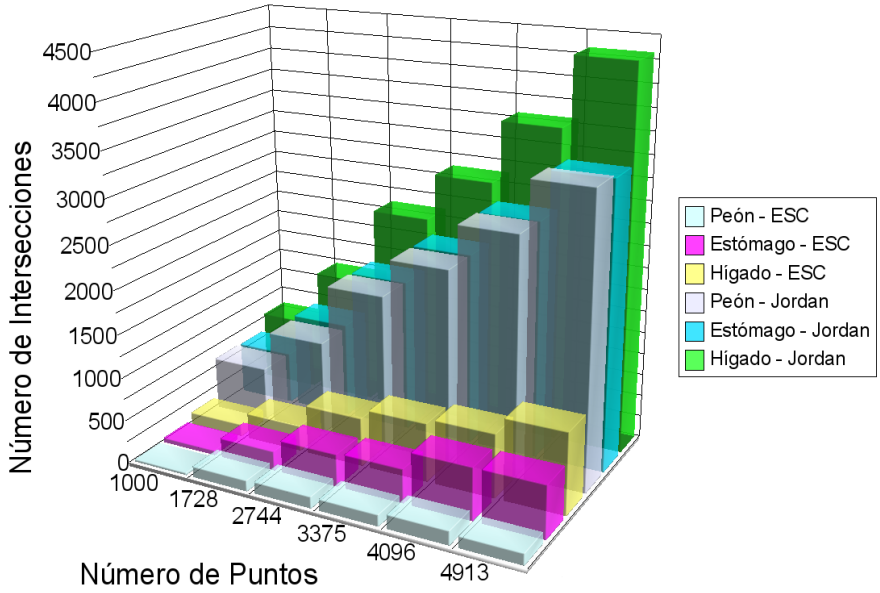


Figura 5.8 Gráfica resumen comparando el nuevo algoritmo con los basados en la aplicación del teorema de Jordan

conjunto de puntos de muestra en una computadora de gama media (Pentium IV a 2.4 GHz con 512 MB de RAM), mostrándose en la tabla 5.5 los datos obtenidos. La segunda columna de la tabla 5.5 indica el tiempo de carga de la malla en memoria desde un fichero que contiene las coordenadas de los vértices y los triángulos como tríadas de índices a los vértices, incluyendo el cálculo de las relaciones topológicas entre los triángulos; la tercera columna de la misma tabla 5.5 muestra a su vez el tiempo de construcción de la cadena simplicial extendida, que implica calcular la información relacionada con los símplices y las ffc. Por último, las columnas cuarta y quinta de la tabla 5.5 dan una referencia del tiempo que se tarda en construir las triangulaciones de detalle

que se utilizan en el algoritmo de inclusión: la primera de ellas indica el tiempo empleado en construir la triangulación del nivel inicial para todos los parches del sólido con $lod = 8$, mientras que la segunda indica el tiempo empleado en añadir dos niveles más a las jerarquías de detalle asociadas a los parches, utilizando el mismo valor de *lod*. Los tiempos resultantes denotan que el coste total en tiempo no es excesivamente grande en comparación con la mejora en la eficiencia que se consigue si se precalculan todas las triangulaciones necesarias una sola vez durante la carga de la malla para utilizarlas posteriormente.

Malla	Carga	Constr. ESC	T. nivel 1	T. niveles 2 y 3
Estómago	0.300	0.076	0.360	29.336
Hígado	0.083	0.040	0.180	14.696
Seta	0.213	0.063	0.300	24.420

Tabla 5.5 Desglose del tiempo de preparación de los datos (en segundos)

Es también importante conocer el número de ffc y cuántos parches delimitan cada ffc. En la tabla 5.6 se muestra esta información. La segunda columna muestra el número total de parches que definen la frontera de cada sólido; la tercera columna indica el número de ffc que resultan al final del proceso de construcción de la ESC, y la cuarta columna muestra cuántos parches delimitan en promedio cada ffc.

Malla	Parches	ffc	Parches/ffc
Estómago	534	818	2.587
Hígado	274	422	2.600
Seta	448	694	2.687

Tabla 5.6 Datos sobre la utilización de los parches en la definición de las ffc

La tabla 5.7 muestra la información de tiempos referida al cálculo de la inclusión para un conjunto de puntos de referencia. Para este test se dispuso un conjunto de $100\times100\times100$ puntos regularmente distribuidos en la caja englobante de los sólidos de prueba, y se calculó la inclusión de cada uno de ellos en los sólidos. La figura 5.9 muestra la visualización de los puntos clasificados dentro de los sólidos.

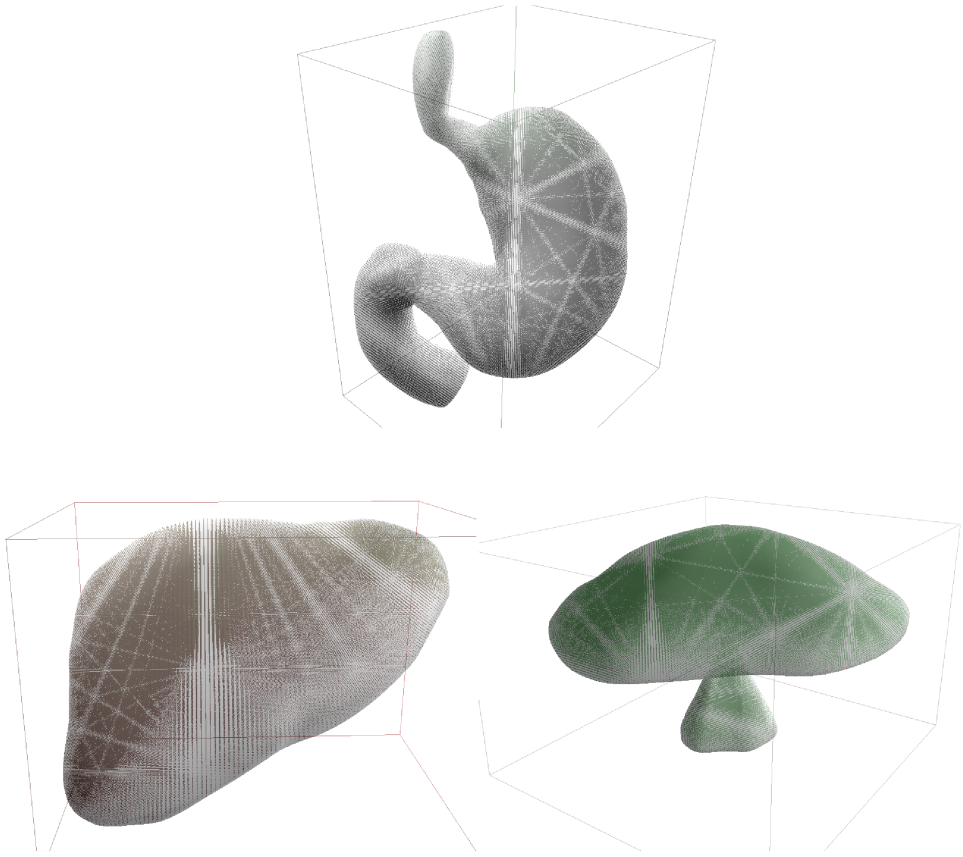


Figura 5.9 Resultados del test de inclusión con un conjunto regular de 100×100×100 puntos

En la tabla 5.7, la segunda columna indica el tiempo total para el conjunto de puntos de prueba; la tercera columna muestra el tiempo promedio por punto resultante; la cuarta columna muestra el número de rayos que ha sido necesario intersectar con los parches para calcular los tests; la sexta columna, por último, muestra el promedio de rayos que ha sido necesario intersectar para los tests. Como se puede comprobar, el número de rayos empleados es muy bajo, y el tiempo para un punto es de poco más de una milésima de segundo para la malla más compleja; esto se debe a que son muy pocos los casos en que un punto es susceptible de pertenecer a una ffc, y por tanto el algoritmo de inclusión en este tipo de celdas, que es la parte más costosa en tiempo, se ejecuta muy pocas veces.

Malla	Tiempo	Tiempo/Pto.	N. rayos	Rayos/Pto.
Estómago	19m 31s	0.001171s	64050	0.064050
Hígado	8m 32s	0.000512s	142886	0.142886
Seta	15m 19s	0.000919s	64156	0.064156

Tabla 5.7 Tiempos de las pruebas con un conjunto regular de 100×100×100 puntos

Por último, la tabla 5.8 muestra información acerca de cuántas veces un rayo ha intersectado más de una vez el mismo parche. La segunda columna indica el número de rayos que han presentado esta circunstancia, mientras que la tercera columna muestra el número total de rayos utilizados y la cuarta el tanto por ciento sobre la cifra total que suponen los rayos que presentan este comportamiento. Como se puede apreciar, gracias a la sencillez de los parches utilizados y al algoritmo utilizado para su generación, los casos más complejos se reducen a un número prácticamente testimonial.

Malla	N. múltiples int.	N. rayos	%
Estómago	162	64050	0.25%
Hígado	156	142886	0.11%
Seta	0	64156	0.00%

Tabla 5.8 Información sobre los rayos que intersectan más de una vez el mismo parche en los tests

5.6 Conclusiones

En este capítulo se ha descrito el algoritmo para el cálculo del test de inclusión de un punto en un sólido de forma libre cuya superficie ha sido modelada con parches triangulares de Bézier [GRF04a]. La importancia de este algoritmo es fundamental en tareas como la visualización utilizando trazado de rayos, la detección de colisiones o la evaluación de operaciones booleanas entre sólidos. El análisis del algoritmo denota que es una solución robusta y simple, a la par que eficiente. En el apéndice A se describen detalles concretos sobre la estructura de datos utilizada para el almacenamiento de la jerarquía de niveles de detalle que aproxima cada parche triangular de Bézier, y que tiene un papel fundamental en el algoritmo desarrollado.

En el siguiente capítulo se trata el problema de las operaciones booleanas y, como se ha mencionado antes, este algoritmo será masivamente utilizado en los procesos de visualización y evaluación de la frontera de los resultados.

126

126
126

127

6 Operaciones booleanas

Las operaciones booleanas juegan un papel clave en todo sistema de modelado de sólidos, ya que permiten a los usuarios diseñar y/o editar objetos de forma sencilla e intuitiva utilizando otros objetos como punto de partida.

En este capítulo se tratarán la representación y tratamiento de operaciones booleanas utilizando el modelo ESC presentado en el capítulo 3, particularizado a sólidos de forma libre delimitados por parches triangulares de Bézier, tal y como aparece en el capítulo 4. En todo ello juega un papel fundamental el algoritmo de inclusión de puntos descrito en el capítulo 5.

6.1 Representación de operaciones booleanas y sólidos CSG

En el modelo ESC los sólidos se representan mediante cadenas simpliciales extendidas, tal y como se expuso en el capítulo 3. En dicho capítulo también se hizo referencia a las operaciones de suma de cadenas y producto de una cadena por un escalar, que permiten establecer el conjunto de las cadenas simpliciales extendidas como un espacio vectorial sobre el cuerpo $\mathbb{Z}/p$ [Rui01]. Utilizando dichas operaciones, se puede representar de forma simple el resultado de la unión, intersección, diferencia y complemento entre sólidos de forma libre modelados con ESC. Para ello, es necesario prestar atención antes al resultado de la intersección entre los tipos de celdas extendidas que forman dichas cadenas:

127
127

- La intersección entre símplices puede ser vacía, un punto, un segmento, un polígono o un poliedro delimitado por caras planas.

- La intersección entre un símplice y una ffc puede ser vacía, o puede incluir uno o varios elementos de los siguientes tipos: punto, segmento, curva, polígono de lados planos y/o curvos y poliedro delimitado por caras planas y/o curvas.

- La intersección entre dos ffc puede ser vacía, o dar como resultado uno o varios elementos de los siguientes tipos: punto, segmento, curva, polígono de lados planos y/o curvos y poliedro delimitado por caras planas y/o curvas.

En el capítulo 3 se presentó el modelo ESC, así como las operaciones con cadenas y cómo se pueden expresar las operaciones booleanas en función de la suma de ESC y del producto de un escalar por una ESC. En la práctica, y dependiendo de la operación en concreto, pueden aparecer celdas de signos opuestos cuyos volúmenes se superponen, por lo que no contribuyen a la solución final al contrarrestarse sus volúmenes. Estas celdas pueden descartarse con antelación, reduciendo el volumen de información a procesar y mejorando la eficiencia del proceso. También ayuda a reducir el número de intersecciones de celdas el situar los orígenes de las cadenas simpliciales extendidas asociadas a cada operando alejados entre sí y del otro operando. La figura 6.1 ilustra esta circunstancia.

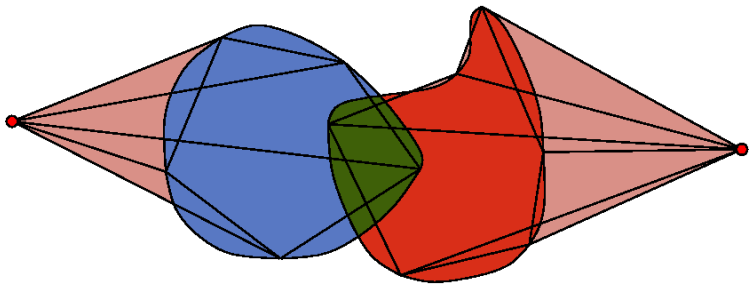


Figura 6.1 Disposición de dos ESC que reduce el número de celdas superpuestas a la hora de calcular una operación booleana

Al ser los resultados de las operaciones otras cadenas simpliciales extendidas, todos los algoritmos propuestos siguen siendo aplicables a éstos, por lo que pueden combinarse de nuevo en otras operaciones, formando modelos tan complejos como se requiera. De esta forma, se obtiene un sistema de modelado

consistente y cerrado, que permite calcular la cadena simplicial extendida que representa un sólido de forma libre modelado utilizando la estructura y operaciones del esquema de modelado CSG descrito en la sección 2.4.3. La figura 6.2 muestra un ejemplo de un árbol CSG que combina tres primitivas de forma libre con dos operaciones booleanas.

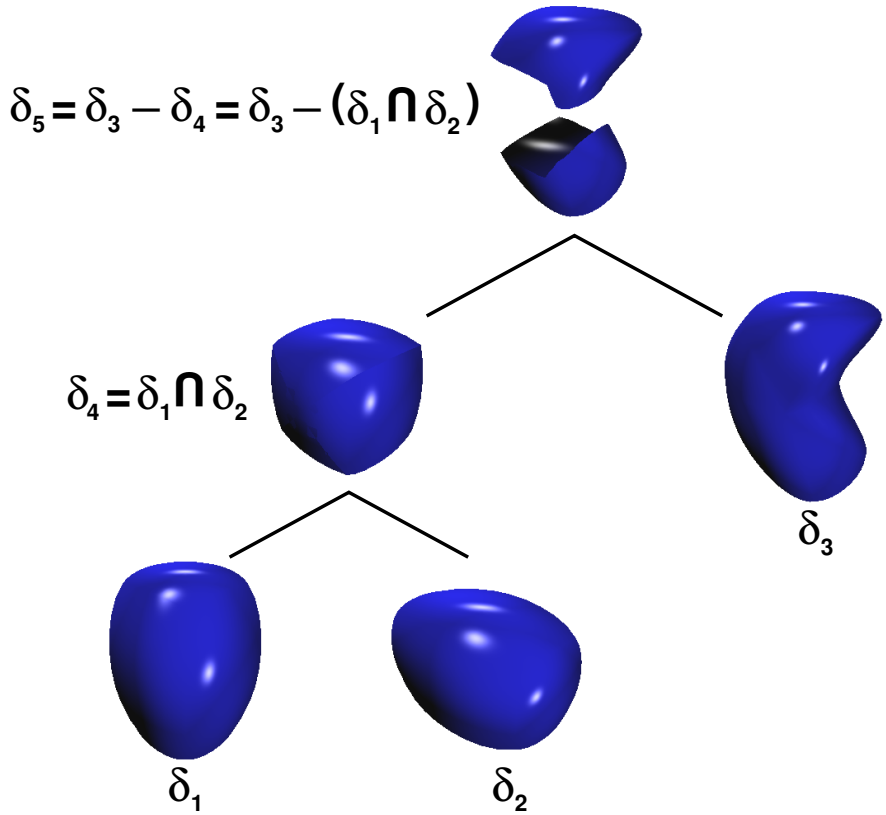


Figura 6.2 Ejemplo del cálculo de la ESC que representa un sólido CSG

6.2 Visualización de operaciones booleanas no evaluadas

La evaluación de la intersección de celdas extendidas $ExCell\,(E \cap E')$ es compleja cuando alguna de ellas es una celda de forma libre, pues el conjunto de puntos resultante puede tener componentes disjuntas e irregulares. Tal y como se ha explicado en el capítulo 3, este conjunto se puede modelar mediante una cadena simplicial extendida; sin embargo, el proceso de cálculo de esta cadena puede ser especialmente complejo, como se verá a lo largo de este capítulo,

por lo que es aconsejable demorar la evaluación de este tipo de intersecciones hasta que realmente sea necesario, ya que puede ocurrir que no formen parte del sólido final.

Teniendo en cuenta esto, cobra interés la posibilidad de visualizar el resultado de una operación booleana entre sólidos de forma libre modelados con cadenas simpliciales extendidas sin necesidad de hacer el cálculo, como una forma de obtener una vista preliminar del resultado final [GRF05].

La visualización de sólidos resultado de operaciones booleanas sin evaluarlas efectivamente es un campo de investigación bastante activo en la disciplina del Modelado de Sólidos [HG96, SL98, ET00, SLJ02, GKMV03, HR05]. Es un hecho que obtener una imagen previa del resultado final de una operación de modelado antes de ejecutarla es interesante tanto para los usuarios como para los desarrolladores de sistemas de modelado, pues permite agilizar la interacción de los usuarios con dichos sistemas.

En lo que se refiere al tratamiento de sólidos de forma libre, existen pocas referencias que traten la visualización del resultado de aplicar operaciones booleanas sobre primitivas de forma libre [HR05]. Es además común en estos casos recurrir a aproximaciones mediante mallas de triángulos.

El modelo ESC permite demorar en lo necesario el cálculo efectivo de las intersecciones entre celdas extendidas que forman parte de una cadena, sin por eso dejar de ser una representación plenamente utilizable del sólido que se trate. Por tanto, se ha diseñado un algoritmo para la visualización de cualquier sólido representado mediante una cadena simplicial extendida, que es también aplicable a sólidos CSG sin evaluar. Este algoritmo está basado en trazado de rayos [Gla93], y hace un uso intensivo del algoritmo para el cálculo del test de inclusión de puntos expuesto en el capítulo 5. Puede observarse su estructura en el algoritmo 6.1.

Las figuras 6.3 y 6.4 muestran ejemplos de los resultados de la visualización de las operaciones de unión, intersección y diferencia entre sólidos de forma libre sin evaluar de manera efectiva las fronteras del objeto resultado.

El trazado de rayos permite visualizar el sólido definido por cualquier cadena, e incluso es posible mostrar un fragmento de una cadena mediante una modificación del algoritmo 6.1, consistente en indicar los índices de las celdas inicial y final del fragmento que se quiere visualizar. De esta forma, se puede

```
Algoritmo visualizaESC ( cadena, imagen )

  Para cada pixel p ∈ imagen hacer
    rayo ← rayo que pasa por p
    ints ← ∅

    Para cada celda E ∈ cadena hacer
      ints.insercionOrdenada ( rayo.intersecciones ( E ) )

    Si ints ≠ ∅ entonces
      P ← ints.primero
      dentro ← inclusionESC ( cadena, P )

      Mientras ( No dentro ) y ( P ≠ ints.ultimo ) hacer
        P ← ints.siguiente
        dentro ← inclusionESC ( cadena, P )

      Si dentro entonces
        p.color ← calculaIluminacion ( P.material )

      Si no
        p.color ← color de fondo

    Si no
      p.color ← color de fondo
```

Algoritmo 6.1 Algoritmo para la visualización de un sólido representado mediante una cadena simplicial extendida

comprobar visualmente cómo los volúmenes de las celdas positivas y negativas se van compensando y dando como resultado el sólido final. La figura 4.16 fue elaborada de esta forma a partir de la ESC asociada al resultado de la diferencia de la figura 6.3.

6.3 Evaluación de la frontera

La evaluación de la frontera de un sólido modelado aplicando operaciones booleanas sobre primitivas sencillas es una tarea necesaria en entornos de diseño y fabricación asistidos por computador, que puede llegar a ser realmente compleja cuando se trata de sólidos y primitivas de forma libre. Los problemas de

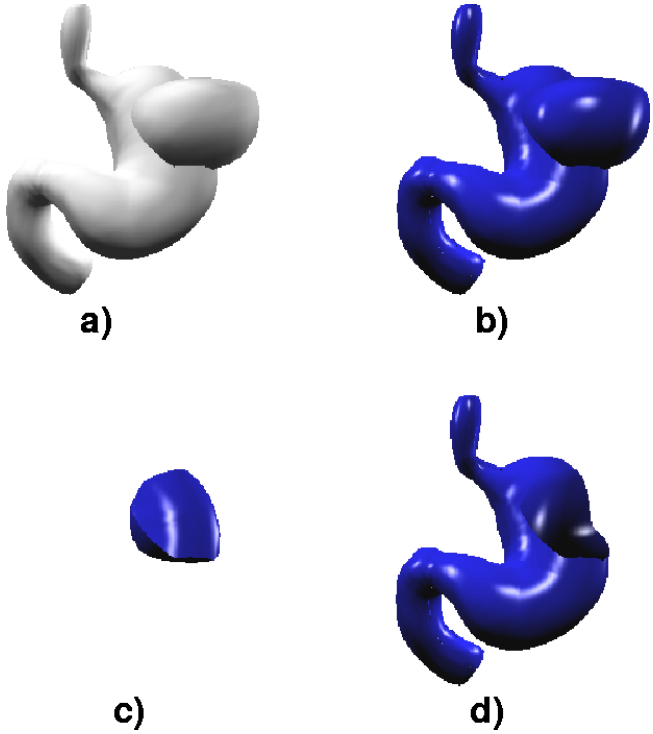


Figura 6.3 a) Dos sólidos de forma libre. b) Unión. c) Intersección. d) Diferencia

robustez y exactitud son habituales en los algoritmos propuestos para resolver este tipo de cálculos [For97, KGMM97, KCF+04], por lo que aún se considera un campo abierto en la investigación relacionada con el Modelado de Sólidos.

El proceso de evaluación de la frontera del sólido resultante de una operación booleana consta de tres pasos fundamentales:

1. Cálculo de las curvas de intersección entre las primitivas utilizadas.

2. Recorte de las superficies de acuerdo a estas curvas de intersección y al tipo de operación.

3. Composición de la frontera del sólido final uniendo los trozos resultantes del recorte anterior.

En las siguientes secciones se estudiarán estos problemas y se plantearán soluciones para el caso de sólidos limitados por parches triangulares de Bézier.

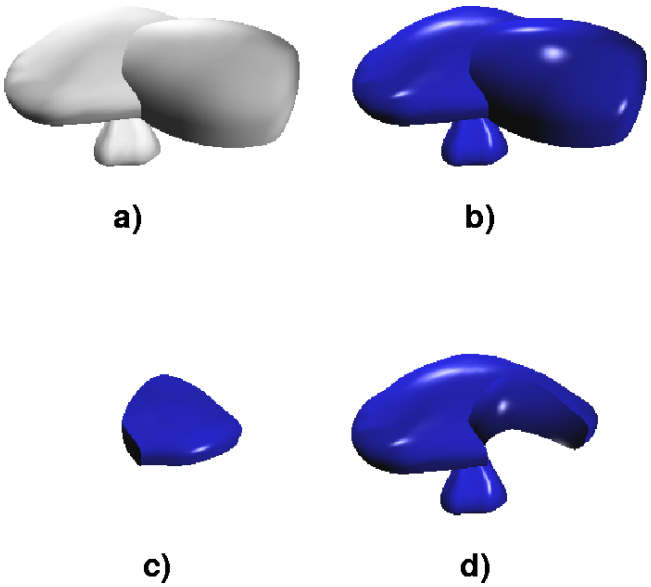


Figura 6.4 a) Dos sólidos de forma libre. b) Unión. c) Intersección. d) Diferencia

Un rasgo distintivo de la representación basada en cadenas simpliciales extendidas es el hecho de que éstas suponen una representación del volumen del sólido y no sólo de su frontera, por lo que se hace necesario determinar la forma de construir una nueva cadena que represente al volumen delimitado por las nuevas fronteras calculadas. A este proceso se dedica la sección 6.4.

6.3.1 Intersección de dos parches triangulares de Bézier

La intersección de superficies en general es un problema ampliamente tratado en la bibliografía [AMY96, GK97, LCK02, LJCW04, Muk05]. Sin embargo, poco o nada ha sido dedicado explícitamente al cálculo de la intersección de dos parches triangulares de Bézier, por lo que se describe aquí el desarrollo de un algoritmo para el cálculo de dicha intersección haciendo uso de las estructuras de datos y algoritmos diseñados para resolver el test de inclusión de puntos y la visualización de los parches descritos en los capítulos anteriores.

En general, la intersección de dos superficies puede ser vacía, o puede contener uno o más elementos de tipo punto, curva o superficie. Siguiendo la notación de Keyser [Key00], se considera que los parches están dispuestos en ***posición general***; esto quiere decir que una perturbación infinitesimal de los

datos de entrada no modificará las decisiones en las instrucciones de bifurcación de los algoritmos. En consecuencia, no se van a considerar las situaciones en las que los parches están superpuestos.

En general, los algoritmos propuestos para el cálculo de la intersección entre superficies utilizan enfoques analíticos para obtener una solución exacta, o técnicas numéricas para obtener aproximaciones con precisiones diversas del resultado. Existen otras propuestas de solución, como los algoritmos evolutivos [CKKK97], pero que no han experimentado un desarrollo notable.

Partiendo de las jerarquías de niveles de detalle que aproximan a cada parche, el algoritmo de intersección implementado [GRF06] realiza sucesivos refinamientos en cada una de ellas, demorando el cálculo efectivo de las intersecciones hasta alcanzar el nivel de detalle deseado [AB90]. En este punto, se intersectan los triángulos base de dicho nivel de cada jerarquía, y los segmentos resultantes de estas operaciones constituirán una aproximación de las curvas de intersección. El algoritmo 6.2 resume este proceso.

La intersección de triángulos puede implementarse utilizando cualquier método clásico existente en la bibliografía [Möl97]. Por otra parte, los segmentos que se calculan han de ser ordenados para posteriormente poder obtener los elementos que componen la intersección de los parches. Esa es la finalidad del algoritmo *añadirSegmentos* que se utiliza en el algoritmo 6.2. El algoritmo 6.3 detalla el funcionamiento de este procedimiento.

En el algoritmo 6.3 se almacenan aparte aquellos segmentos que pueden dar lugar a ciclos. Estos segmentos son los que comparten uno de sus extremos con más de un segmento, o bien aquellos que cierran un ciclo formado por otros segmentos añadidos anteriormente a la estructura (figura 6.5). Por otra parte, la reordenación de segmentos que se aplica en el citado algoritmo consiste en reorientar las secuencias de segmentos concatenados que se van construyendo, de forma que el final de un segmento coincida con el inicio del segmento siguiente en la estructura (figura 6.6). Así se facilita el proceso de aislamiento de componentes conexas que se describe a continuación.

El último paso en el cálculo de las curvas de intersección entre dos parches es la determinación de las componentes conexas que forman dichas curvas. Una ***componente*** puede ser abierta o cerrada, y está formada por una secuencia de segmentos tales que el final de un segmento coincide con el inicio

```
Algoritmo interseccionParches ( P1, P2, nMax )

  Si P1.BBox ∩ P2.BBox ≠ ∅ entonces
    Si ( P1.nivel = nMax ) Y ( P2.nivel = nMax ) entonces
      i ← interseccionTriangulos ( P1.base, P2.base )
      añadirSegmentos ( i )

    Si no
      Si ( P1.nivel ≠ nMax ) Y ( P2.nivel ≠ nMax ) entonces
        Pm ← parche con menor bounding box
        PM ← parche con mayor bounding box

        Para cada subparche SP ∈ PM.hijos hacer
          interseccionParches ( SP, Pm, nMax )

      Si no
        Si P1.nivel = nMax entonces
          Para cada subparche SP ∈ P2.hijos hacer
            interseccionParches ( P1, SP, nMax )

        Si no
          Para cada subparche SP ∈ P1.hijos hacer
            interseccionParches ( SP, P2, nMax )

  Si no
    Devolver ∅
```

Algoritmo 6.2 Algoritmo para el cálculo de la intersección de dos parches triangulares de Bézier

del siguiente y cada vértice de la componente está compartido por dos segmentos como máximo. Este proceso se lleva a cabo aplicando el algoritmo 6.4, que va construyendo secuencias de segmentos susceptibles de ser consideradas componentes. Al final de este algoritmo se llama al procedimiento *comprobarComponentes*, que comprueba si entre las secuencias de segmentos construidas existen dos que verifiquen que alguno de sus extremos coinciden. Si tal es el caso, se procede a unirlas en una única secuencia, y si dicha secuencia es cerrada, se almacena definitivamente como una componente cerrada. Si aún es abierta, se devuelve al proceso hasta que no haya ningún par de secuencias que

```
Algoritmo añadirSegmentos ( i )

  Si no hay segmentos almacenados entonces
    almacenar i

  Si no
    Si ∃ s ∈ segmentos / s ∩ i ≠ ∅ entonces
      saca s de segmentos
      p ← interseccion ( i, s )
      {i1, i2} ← division ( i, p )
      {s1, s2} ← division ( s, p )
      añadirSegmentos ( i1, i2, s1, s2 )

    Si no
      Si ∃ s ∈ segmentos / s enlaza con i entonces
        Si pueden aparecer ciclos entonces
          almacenar i aparte

        Si no
          almacenar i enlazado con s
          reordenar los segmentos

      Si no
        almacenar i
```

Algoritmo 6.3 Algoritmo para la inserción ordenada de segmentos de intersección

se puedan unir. El conjunto de secuencias de segmentos restantes se almacenan como componentes abiertas.

El resultado final de todo este proceso es un conjunto de componentes que o bien son cerradas, o bien son abiertas, verificándose que no existen dos componentes abiertas que puedan fusionarse.

6.3.2 Recorte de parches triangulares de Bézier

El recorte de superficies es una operación básica en los sistemas de modelado actuales, pues es la que permite determinar la frontera del resultado de una operación booleana. Consiste en la eliminación de parte del parche objeto del

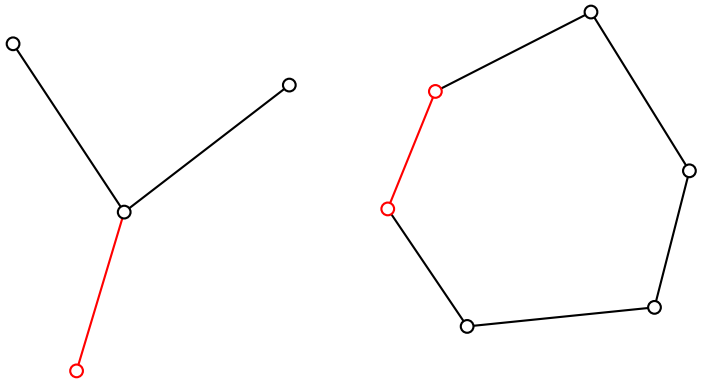


Figura 6.5 En rojo, ejemplos de segmentos almacenados por separado en el algoritmo 6.3

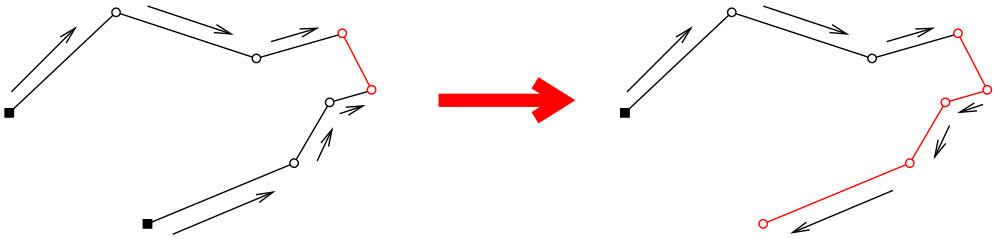


Figura 6.6 Reordenación de una secuencia de segmentos en función de un nuevo segmento introducido. Izquierda: situación provocada por el nuevo segmento (en rojo). Derecha: secuencia de segmentos reordenada

recorte, para lo cual es necesario establecer antes algún tipo de partición del espacio y un criterio de recorte que identifique las partes a eliminar.

La necesidad de establecer una partición del espacio hace que se consideren los casos de recorte de un parche contra un plano y recorte de un parche contra un objeto. Este segundo caso es el que se va a estudiar por ser el más complejo, ya que el tratamiento para el primer caso se puede deducir a partir del proceso que se describe a continuación.

Se considerará el recorte de un parche triangular de Bézier contra un sólido también delimitado por parches triangulares de Bézier. De este modo, el proceso de recorte incluye las siguientes fases (figura 6.7):

a) Cálculo de las curvas de intersección entre el parche a recortar y los parches que delimitan el sólido de recorte. Esto implica también obtener

```
Algoritmo construyeComponentes ( segmentos )

  c ← nuevaComponente ( segmentos.primero )

  Mientras ( segmentos ≠ ∅ ) hacer
    s ← segmentos.siguiente

    Si s enlaza con c entonces
      añadir s a c

    Si no
      guardaComponente ( c, componentes )
      c ← nuevaComponente ( s )

  s ← segmentosApartados.primero

  Mientras ( segmentosApartados ≠ ∅ ) hacer
    Para cada c ∈ componentes hacer
      Si s enlaza con c entonces
        añadir una copia de s a c

        Si aparece un ciclo entonces
          c2 ← cortaCiclo ( c )
          guardaComponente ( c2, componentes )

    s ← segmentosApartados.siguiente

  comprobarComponentes ( componentes )
```

Algoritmo 6.4 Algoritmo para la detección de componentes en la intersección de dos parches triangulares de Bézier

las componentes conexas que forman cada intersección entre parches, aplicando los algoritmos explicados en la sección 6.3.1.

b) Combinación de las componentes de todas las intersecciones entre parches que se han calculado. En esta fase se calculan los puntos de intersección entre las componentes si los hubiera, y se procede a trocear dichas componentes de forma que ningún trozo contiene puntos de intersección aparte de sus puntos inicial y final. En este proceso se incluyen

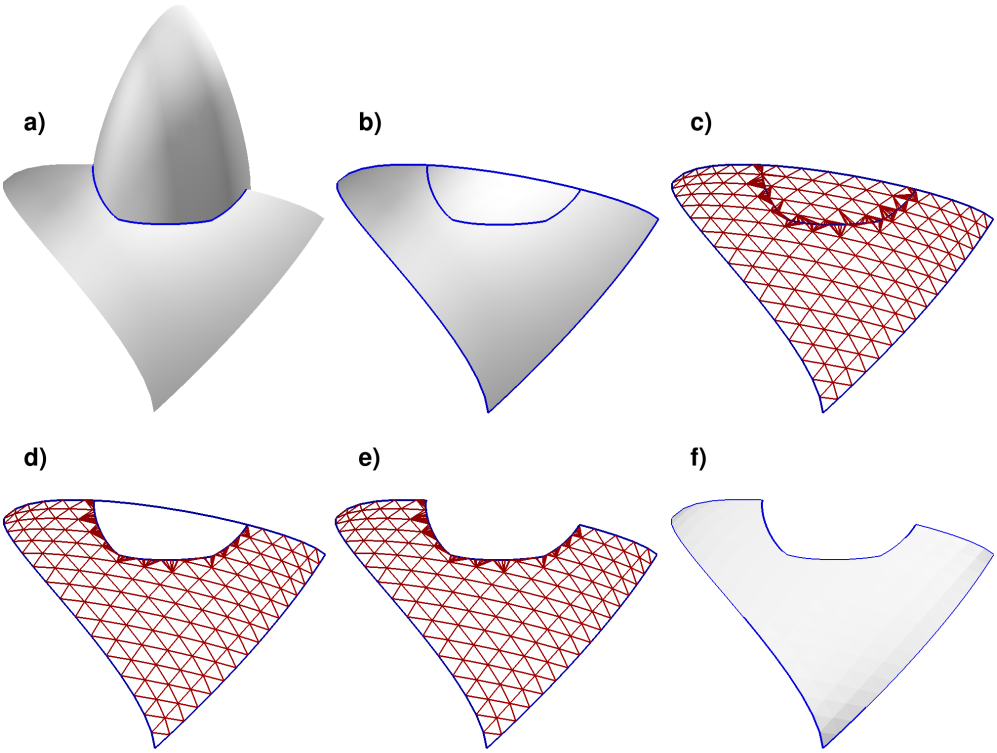


Figura 6.7 Fases del proceso de recorte de un parche triangular de Bézier

también las fronteras originales del parche a recortar, y se obtiene como resultado el conjunto de fronteras que delimitan las regiones interna y externa del parche a recortar.

Al estar las curvas de recorte y las fronteras del parche aproximadas mediante poligonales, el cálculo de intersecciones es rápido y simple.

c) Teselación del dominio paramétrico de los subparches almacenados en los nodos hoja de la jerarquía de niveles de detalle que son intersectados por los trozos de curva obtenidos en el paso anterior. Como resultado, todos los elementos almacenados en la jerarquía están ahora completamente dentro o completamente fuera del sólido de recorte.

   En este proceso, se calculan las coordenadas baricéntricas de los puntos de intersección de las curvas de intersección con el triángulo base de cada subparche, y se utilizan esos puntos como vértices de la teselación

junto con los vértices del subdominio que define el subparche. El algoritmo de teselación aplicado puede ser cualquiera de los habitualmente utilizados [O'R98, Las96].

d) Eliminación de los subparches almacenados en los nodos hoja de la jerarquía que están dentro o fuera del sólido de recorte, dependiendo del criterio de recorte a aplicar, y propagación del proceso de eliminación hacia arriba en la jerarquía, de forma que los nodos cuyos hijos han sido eliminados sean también borrados. Para determinar si un subparche está dentro o fuera del sólido de recorte se recurre al algoritmo para el cálculo del test de inclusión de puntos descrito en el capítulo 5, aplicándolo al baricentro de su triángulo base.

   En la propagación del proceso de borrado hacia los niveles superiores de la jerarquía, se etiquetan los nodos que han sufrido un recorte parcial de sus hijos, y se recalculan las cajas englobantes de los nodos superiores para que abarquen solamente las cajas de los subparches que no han sido eliminados. El algoritmo 6.5 resume este proceso.

e) Borrado de las fronteras que están dentro o fuera del sólido de recorte, dependiendo del criterio a aplicar.

f) Establecimiento de las nuevas fronteras del parche recortado.

El resultado final de todo el proceso es un parche recortado que mantiene la curvatura del original, ya que los subparches se obtienen aplicando subdivisión sobre el parche original [Sei89]. Este parche recortado puede ser recortado de nuevo o participar en otras operaciones de recorte como frontera del sólido de recorte sin ninguna diferencia en cuanto a tratamiento con respecto a los parches sin recortar.

Las figuras 6.8, 6.9 y 6.10 muestran ejemplos de parches recortados obtenidos con una jerarquía de dos niveles y tres subdivisiones en el dominio paramétrico en cada nivel. La tabla 6.1 muestra información sobre dichas figuras. La segunda columna muestra el número de parches que intersectaban entre sí; las columnas tercera y cuarta muestran el número de veces que se ha calculado la intersección entre triángulos y el número de segmentos que definen las fronteras del parche recortado, respectivamente. Por último, la quinta columna indica el tiempo empleado en el cálculo de los recortes en segundos.

```
Algoritmo aplicaRecorte ( parche, obj, criterio, maxN )

  Si parche.nivel ≠ maxN entonces
    Para cada subparche sp ∈ parche.hijos hacer
      aplicaRecorte ( sp, obj, criterio, maxN )

      Si sp.estado = paraBorrar entonces
        borrar sp
        parche.estado ← recortado

    Si parche.nHijos = 0 entonces
      parche.estado ← paraBorrar

  Si no
    vInc ← inclusionPuntoSolido ( parche.tBase.baric, obj )

    Si vInc = criterio entonces
      parche.estado ← paraBorrar
```

Algoritmo 6.5 Algoritmo para la eliminación de los subparches en el proceso de recorte

Figura	N. parches	N. int. triángulos	N. segmentos	Tiempo (segs.)
6.8	31	502624	303	0.44
6.9	13	821392	344	0.67
6.10	43	1305920	836	1.20

Tabla 6.1 Datos del cálculo de las figuras 6.8, 6.9 y 6.10

6.3.3 Obtención del B-Rep resultado de una operación booleana

Una operación booleana de modelado combina dos sólidos para obtener uno nuevo. La frontera de este nuevo sólido, que constituye su representación B-Rep (sección 2.4.5), está determinada tanto por la posición de los sólidos operando

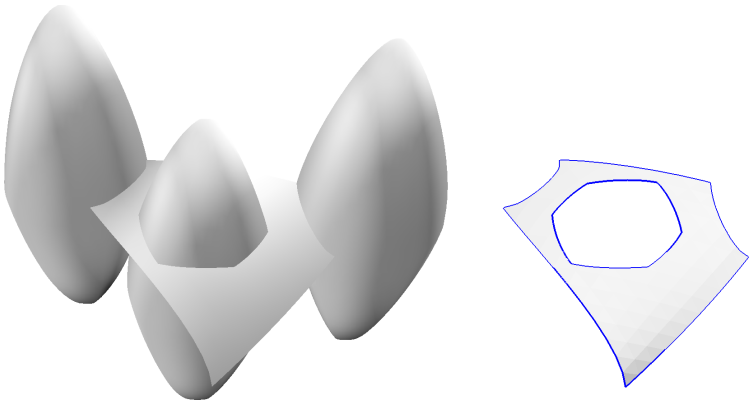


Figura 6.8 Ejemplos de recorte de parches (I).
Izquierda: configuración inicial. Derecha: parche recortado

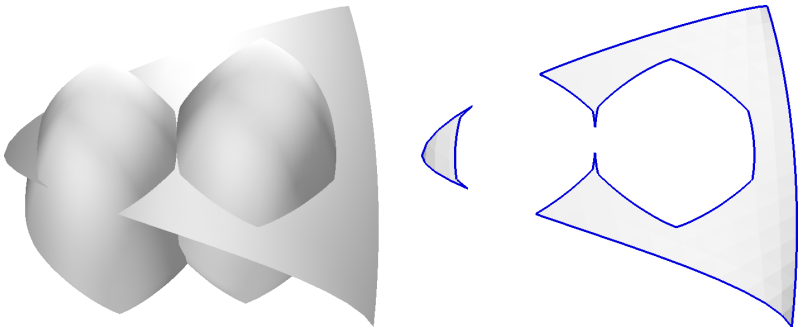


Figura 6.9 Ejemplos de recorte de parches (II).
Izquierda: configuración inicial. Derecha: parche recortado

como por el tipo de operación de que se trate. Se definen los sólidos resultado de las tres operaciones booleanas básicas y su frontera como sigue [RV77]:

- La intersección de dos sólidos comprende el volumen común a ambos. Este volumen estará limitado por las partes de la frontera de cada uno de ellos que están dentro del otro.

- El volumen del sólido resultante de la unión de otros dos comprende tanto el volumen de uno como el del otro. La frontera del nuevo sólido estará determinada por tanto por las porciones de la frontera de cada operando que están fuera del otro.

- La diferencia de dos sólidos es un nuevo sólido cuyo volumen comprende el volumen del minuendo que no está contenido en el del sustraendo. Por

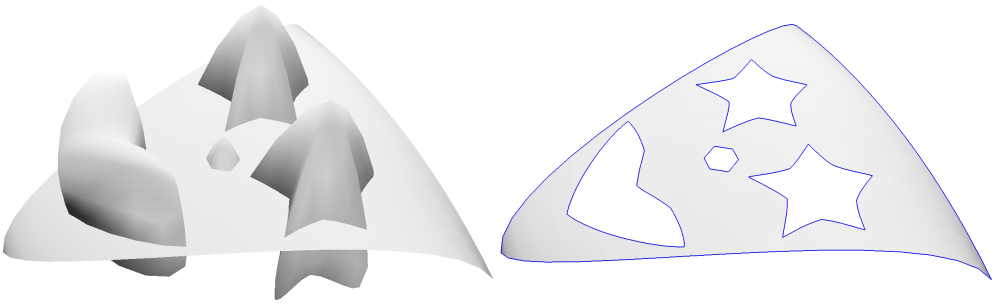


Figura 6.10 Ejemplos de recorte de parches (III). Izquierda: configuración inicial. Derecha: parche recortado

tanto, su frontera está formada por la parte de la frontera del minuendo que está fuera de la del sustraendo, junto con las partes de la frontera del sustraendo que están dentro del minuendo.

A efectos de homogeneizar la representación, es necesario invertir la orientación de las porciones de frontera procedentes de los sólidos sustraendo, para que así toda la frontera esté orientada de la misma forma respecto del interior del sólido resultado.

En el caso de sólidos de forma libre delimitados por parches triangulares de Bézier, la representación B-Rep del resultado de una operación booleana entre dos sólidos se obtiene recortando los parches que constituyen la frontera de cada operando contra el otro utilizando el criterio de recorte adecuado y aplicando un algoritmo como el propuesto en la sección 6.3.2, y seleccionando el conjunto de parches para la nueva frontera aplicando las reglas antes expuestas. De esta forma las fronteras del resultado mantienen las propiedades de las fronteras de los sólidos operando.

Debido a que los algoritmos para el cálculo de la intersección y el recorte de parches propuestos en las secciones 6.3.1 y 6.3.2 utilizan la estructura de jerarquías de detalle expuesta en la sección 4.1.7, el resultado obtenido es una aproximación del resultado real, con una precisión que depende del número de niveles de las jerarquías y del número de divisiones en el dominio paramétrico en cada nivel. Como consecuencia, los trozos de frontera procedentes de los sólidos operando no encajan correctamente, aunque sí lo hagan los triángulos a partir de los cuales se han calculado. La figura 6.11 muestra tres resultados obtenidos con un nivel de detalle y una división paramétrica por nivel (6.11.a), un nivel y tres divisiones (6.11.b) y dos niveles y tres divisiones (6.11.c). Como

se puede apreciar, no sólo pueden aparecen huecos, sino también intersecciones entre los parches recortados.

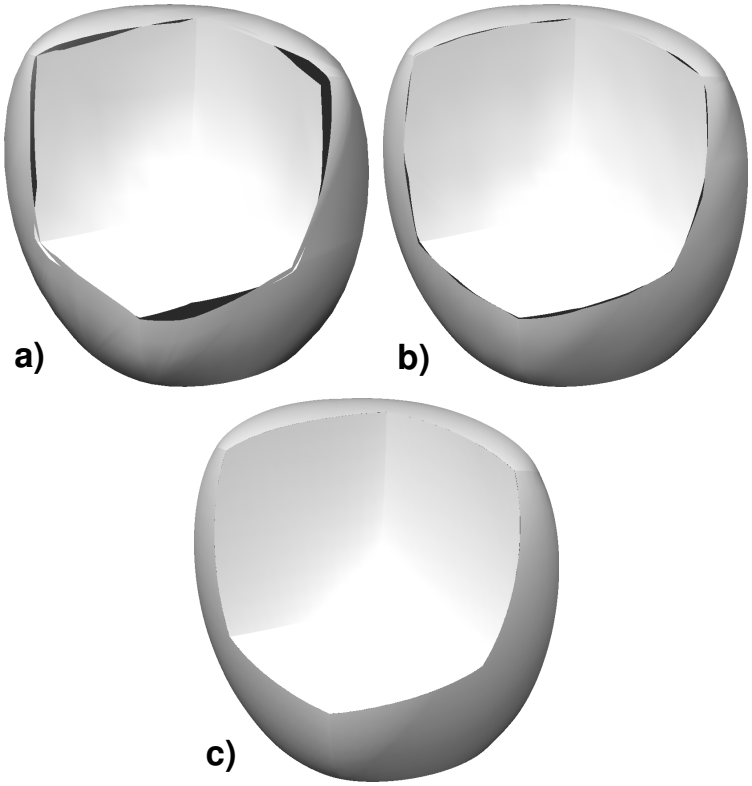


Figura 6.11 Resultados obtenidos con distintas configuraciones de los árboles de detalle. a) Un nivel y una división paramétrica. b) Un nivel y tres divisiones paramétricas. c) Dos niveles y tres divisiones paramétricas por nivel

Se hace necesario por tanto realizar un postproceso sobre el conjunto de parches resultado de una operación booleana para asegurar que la frontera obtenida es completamente cerrada, y determina por tanto el sólido final. Por simplicidad, se decidió trabajar a partir de los subparches triangulares resultantes del proceso de recorte, en lugar de intentar ajustar los parches recortados de manera global. Así, se han estudiado distintas posibilidades teniendo en cuenta

el algoritmo utilizado para la construcción de los parches [VPBM01], y se han implementado y probado las tres que se describen a continuación:

1. La primera solución de postprocesamiento estudiada es la más sencilla: puesto que las triangulaciones a partir de las cuales se han calculado los subparches encajan perfectamente, se toma esta triangulación como punto de partida para calcular un nuevo conjunto de parches que delimiten al sólido resultado. La bondad de esta solución depende en gran medida de la forma en que se calculen los parches, puesto que si se aplica el algoritmo de Vlachos [VPBM01], el resultado sufre un efecto de suavizado general no deseable, especialmente en las zonas de unión entre las porciones de frontera de los operandos. La figura 6.12 ilustra este hecho.

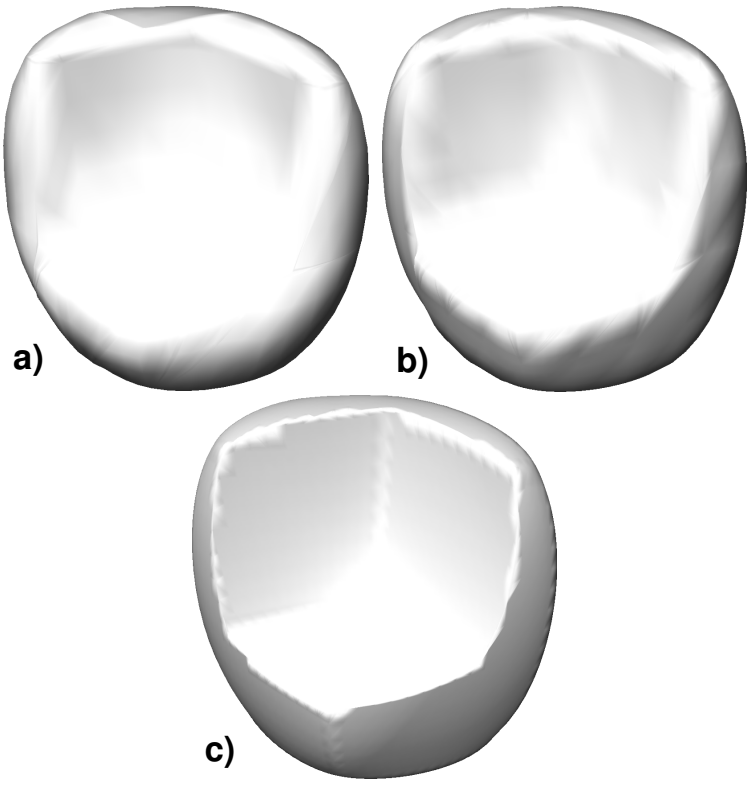


Figura 6.12 Resultados obtenidos aplicando la primera solución de postprocesamiento a los resultados de la figura 6.11

2. La segunda solución de postprocesamiento (figura 6.13) se ha diseñado con el propósito de evitar que el conjunto inicial de parches sufra modificaciones de suavizado no deseadas. El proceso de cálculo es el siguiente:

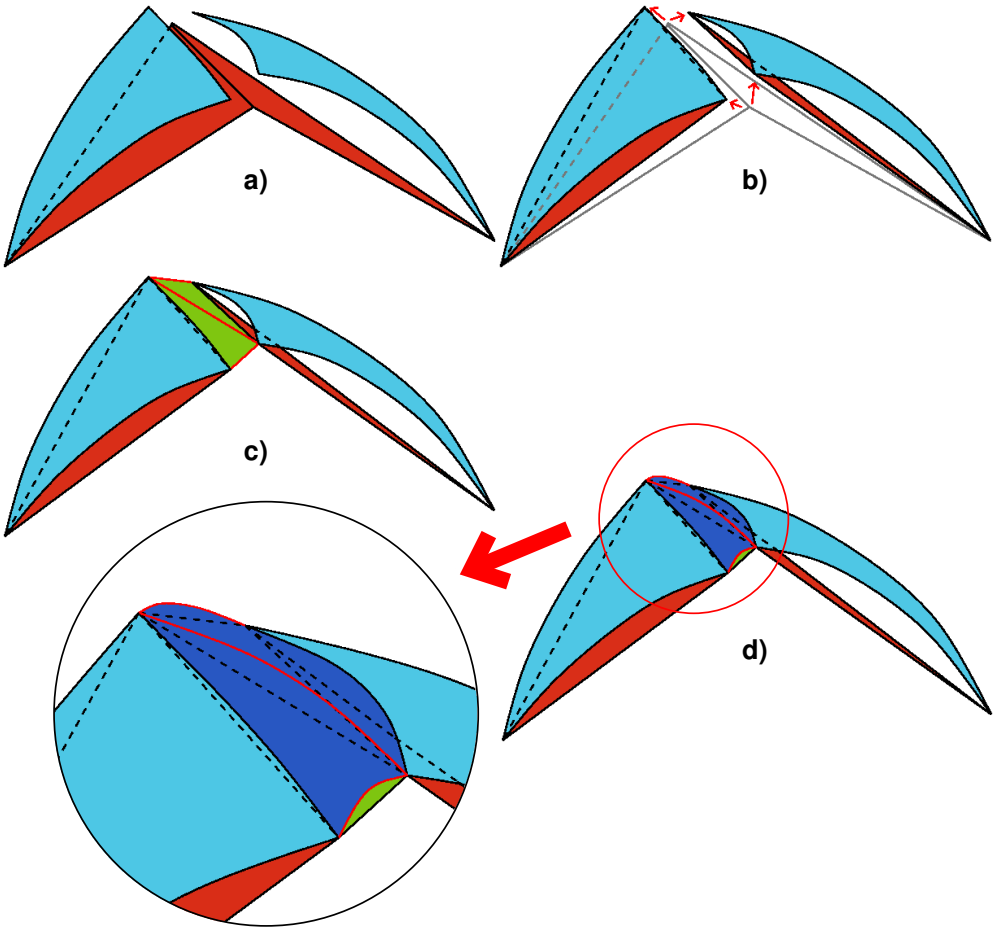


Figura 6.13 Fases de la segunda solución de postprocesamiento propuesta en el caso de huecos

   - En primer lugar, se modifica la triangulación a partir de la cual se han calculado los parches recortados, de forma que los vértices de los triángulos coincidan con los de los parches en aquellas zonas en las que dichos parches no coincidan (figura 6.13.b).

   - A continuación se construyen dos nuevos triángulos entre aquellos que antes tenían un lado en común, de forma que rellenen

el hueco que ha aparecido entre ellos al modificar sus vértices (figura 6.13.c).

– Por último, se calculan dos nuevos parches asociados a los nuevos triángulos que rellenen el hueco entre los parches (figura 6.13.d y ampliación).

Para aplicar esta técnica en los casos que los parches se intersectan, se ha implementado el intercambio de vértices entre los triángulos para posteriormente recalcular los parches implicados, tal y como se muestra en la figura 6.14.

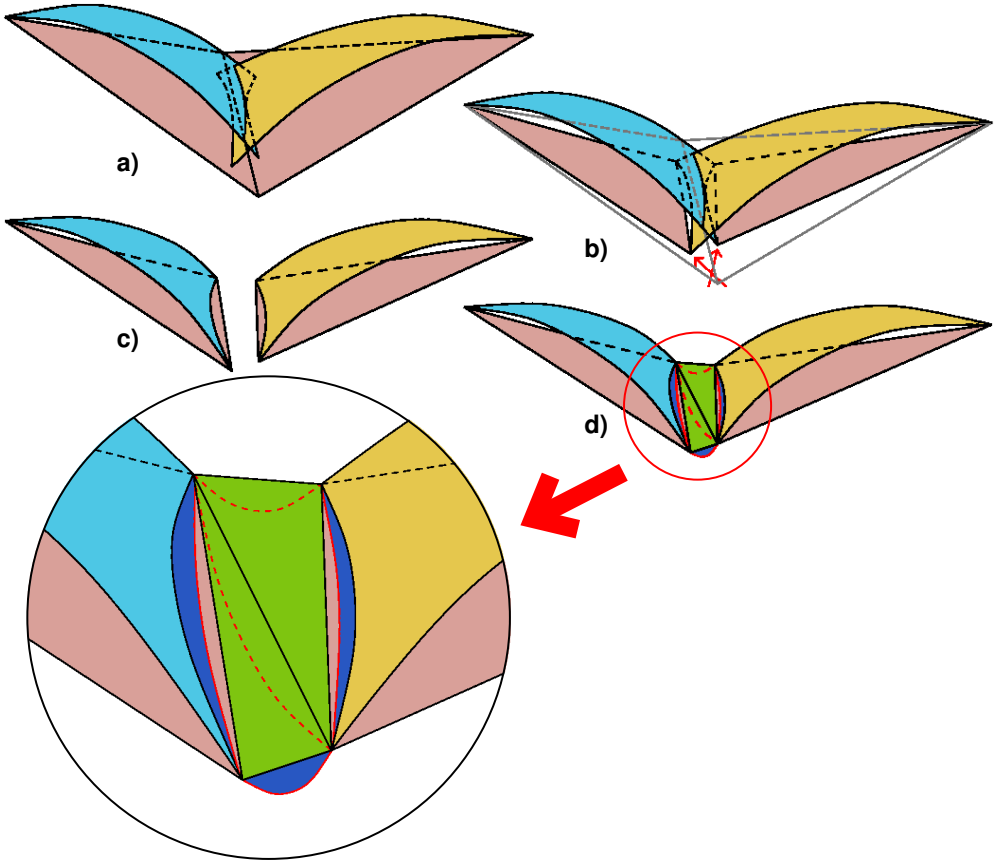


Figura 6.14 Fases de la segunda solución de postprocesamiento propuesta en el caso de huecos cuando los parches se intersectan

Finalmente esta solución, aunque visualmente produce mejores resultados que la primera, ilustrados en la figura 6.15, fue descartada debido a las frecuentes ocurrencias de casos que requieren tratamiento especial, como parches rotados respecto de su correspondiente vecino, que provocan que el rellenado de agujeros sea incorrecto. Estos casos son difíciles de caracterizar en general y añaden una complicación excesiva al proceso. En otras ocasiones, el intercambio de vértices entre dos parches provocaba la aparición de nuevos problemas de la misma índole en los parches que compartían alguno de los vértices modificados.

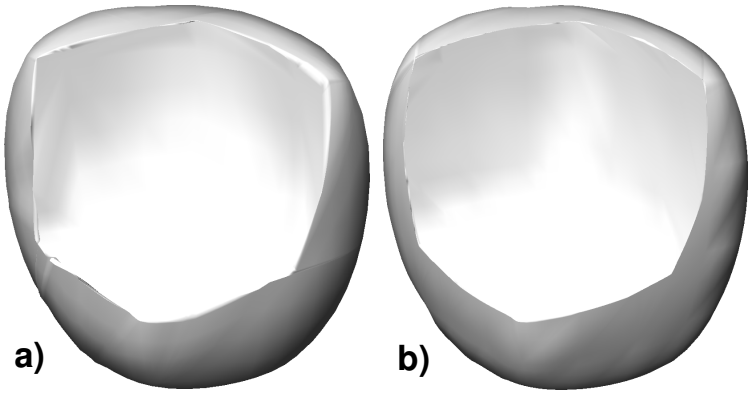


Figura 6.15 Resultados obtenidos aplicando la segunda solución de postprocesamiento a los resultados de la figura 6.11

En la figura 6.15 sólo se muestra el resultado para los dos primeros ejemplos de la figura 6.11, ya que en el tercero, al ser tan pequeños los agujeros a rellenar, no se aprecian visualmente los efectos del rellenado.

3. La tercera solución de postprocesamiento descarta el rellenado de agujeros. Actúa directamente sobre los puntos de control de los subparches, ajustándolos de forma que desaparezcan los huecos entre parches, tal y como se ilustra en la figura 6.16. El proceso de cálculo es el siguiente:

   – Los puntos de control situados en los vértices de los parches son sustituidos por el punto obtenido como el promedio de los puntos correspondientes de todos los parches que comparten o deberían compartir el vértice, según la topología de la triangulación a

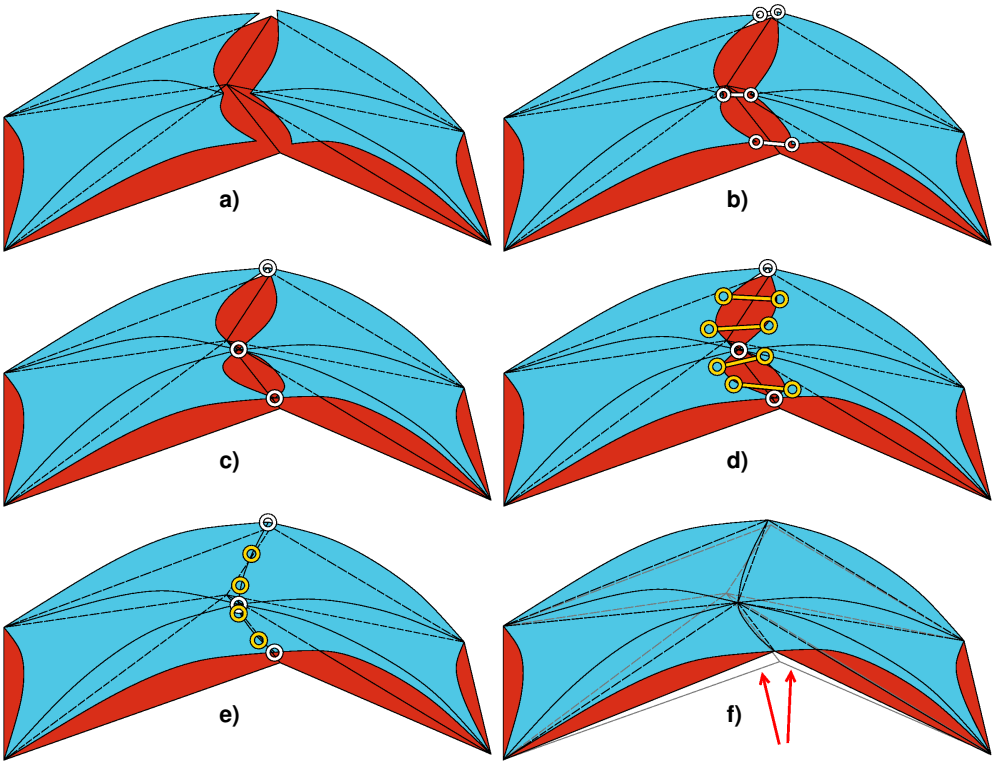


Figura 6.16 Fases de la tercera solución de post-procesamiento propuesta en el caso de huecos

partir de la cual se han calculado dichos parches (figuras 6.16.b y 6.16.c).

– Los dos puntos de control interiores que definen la curva correspondiente a cada lado son reemplazados por los puntos obtenidos como los promedios entre los puntos correspondientes de los dos parches que deberían compartir el lado, según la topología de la triangulación a partir de la cual se han calculado dichos parches (figuras 6.16.d y 6.16.e).

– Por último, se sustituyen los vértices de los triángulos por los nuevos puntos de control de las esquinas de los parches (figura 6.16.f).

Según el algoritmo de Vlachos [VPBM01], la normal en un vértice de la triangulación determina los tres puntos de control de la esquina de

cada parche que corresponda con ese vértice (figura 6.17). Esta tercera solución de postprocesamiento implica que en la gran mayoría de los casos esta relación se pierde, por lo que se ha decidido mantener las normales intactas en cada parche ante la imposibilidad de calcular una normal común para todos. De esta forma, la impresión visual de esta solución es la mejor de las tres que se han estudiado (figura 6.18).

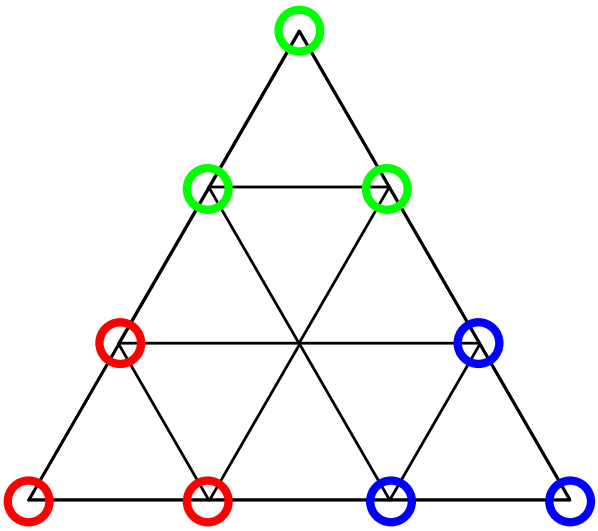


Figura 6.17 Puntos de control determinados por las normales en los vértices según el algoritmo de Vlachos [VPBM01].

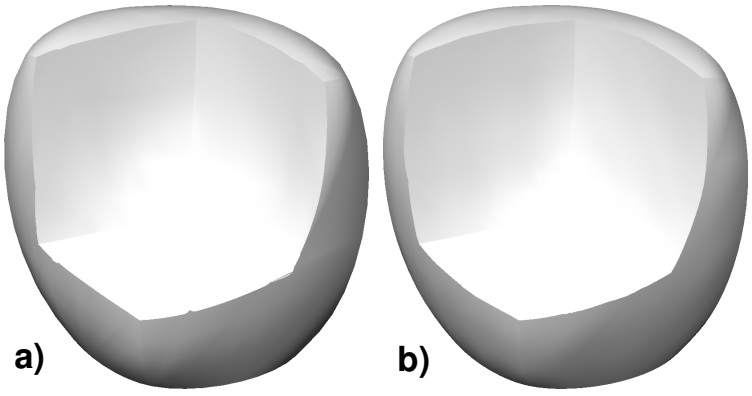


Figura 6.18 Resultados obtenidos aplicando la tercera solución de postprocesamiento a los resultados de la figura 6.11

Las figuras 6.20 a 6.32 muestran otros ejemplos de operaciones booleanas resueltas aplicando la tercera solución de postprocesamiento propuesta, utilizando distintos parámetros en las jerarquías de detalle. La figura 6.19 muestra las primitivas empleadas para estas operaciones.

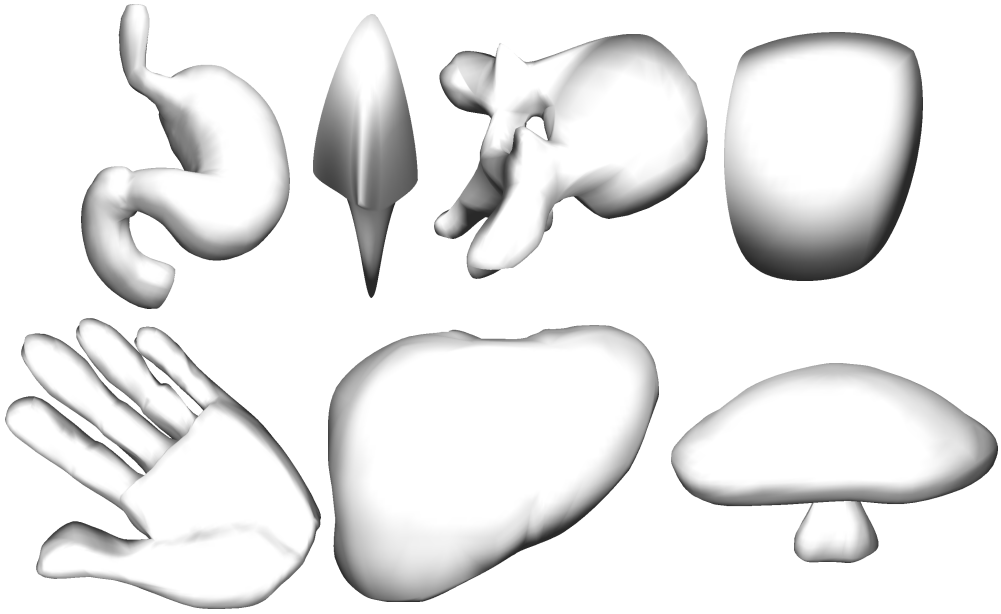


Figura 6.19 Modelos utilizados para las pruebas de evaluación de operaciones booleanas. De izquierda a derecha y de arriba a abajo: estómago (534 parches), punta de lanza (20 parches), vértebra (298 parches), prisma simple (12 parches), mano (768 parches), hígado (274 parches) y seta (448 parches)

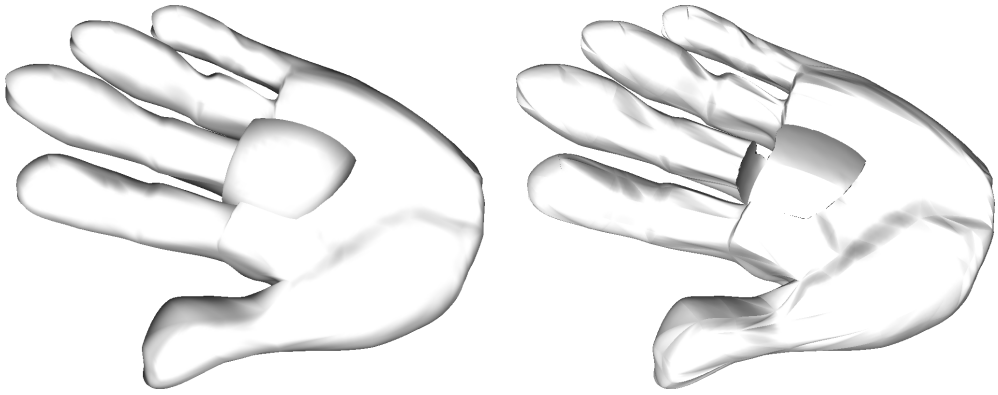


Figura 6.20 Diferencia entre el modelo de la mano y el prisma simple. Izquierda: posición de partida. Derecha: resultado de la operación

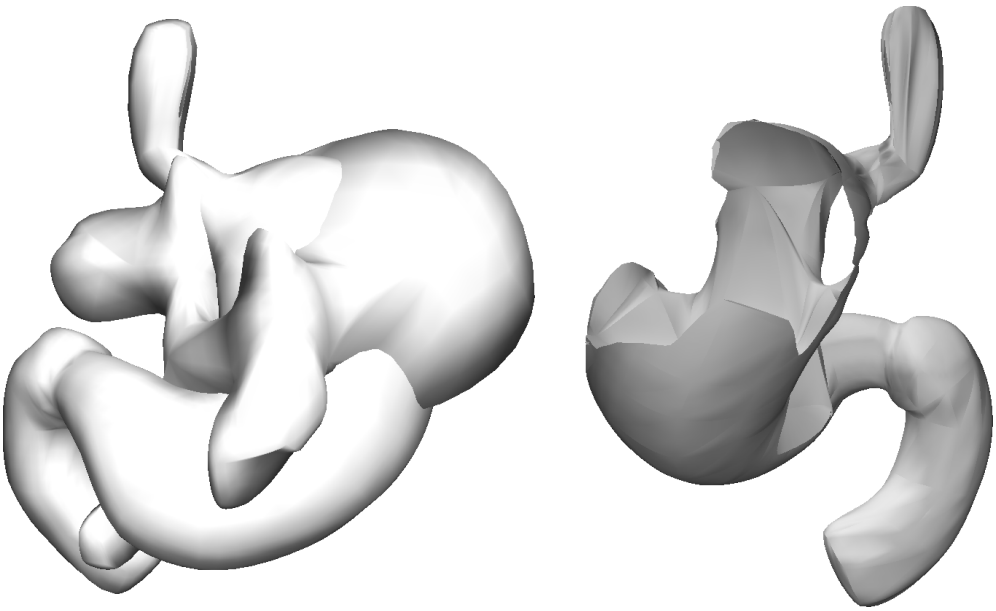


Figura 6.21 Diferencia entre el modelo del estómago y el de la vértebra. Izquierda: posición de partida. Derecha: resultado de la operación

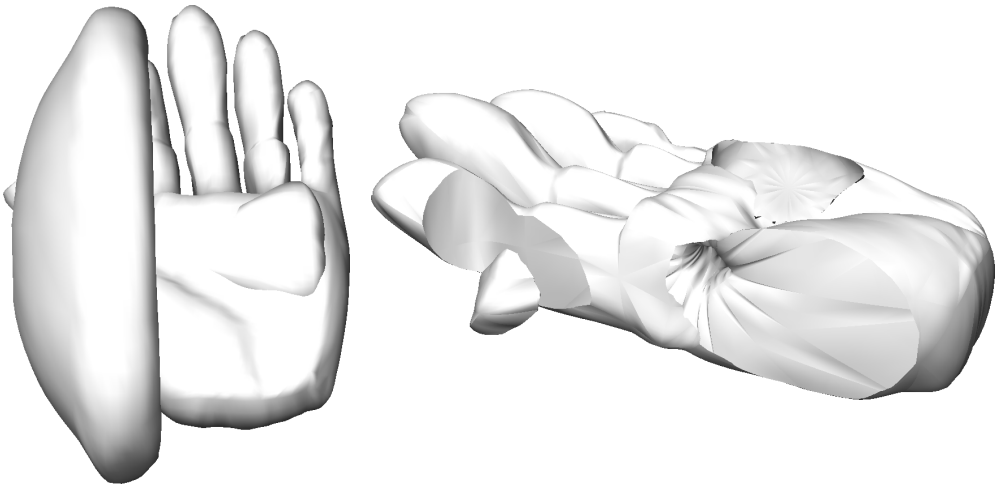


Figura 6.22 Diferencia entre el modelo de la mano y el de la seta. Izquierda: posición de partida. Derecha: resultado de la operación

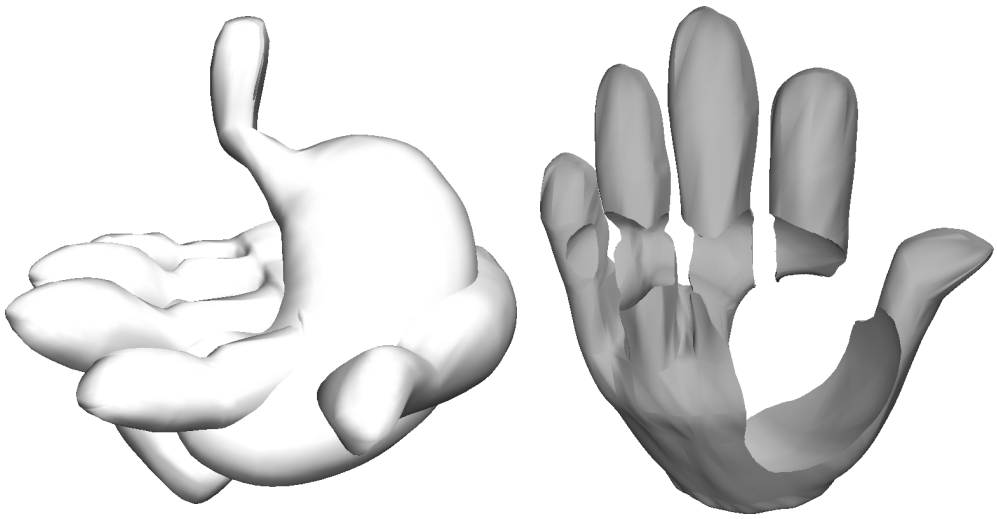


Figura 6.23 Diferencia entre el modelo de la mano y el del estómago. Izquierda: posición de partida. Derecha: resultado de la operación

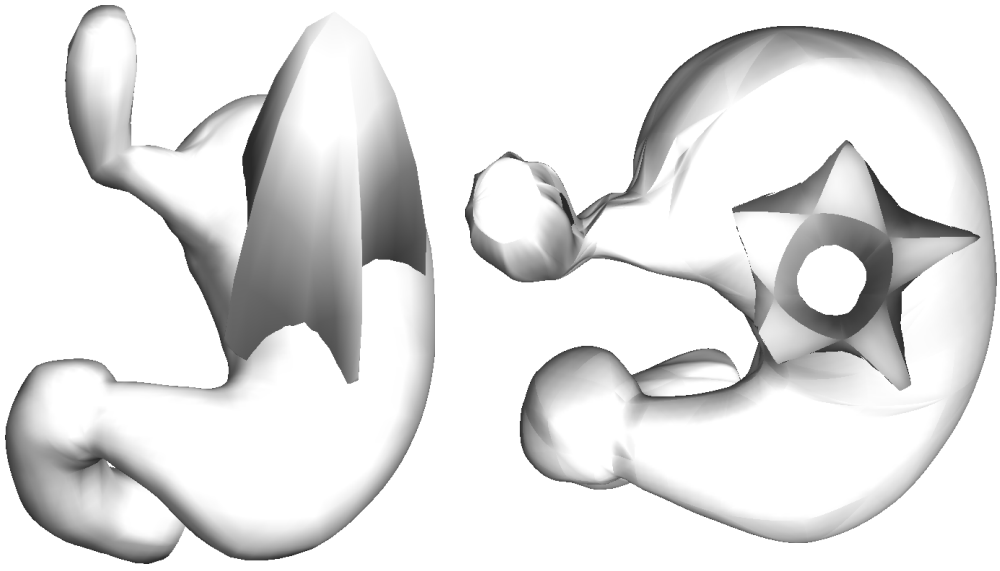


Figura 6.24 Diferencia entre el modelo del estómago y el de la punta de lanza. Izquierda: posición de partida. Derecha: resultado de la operación

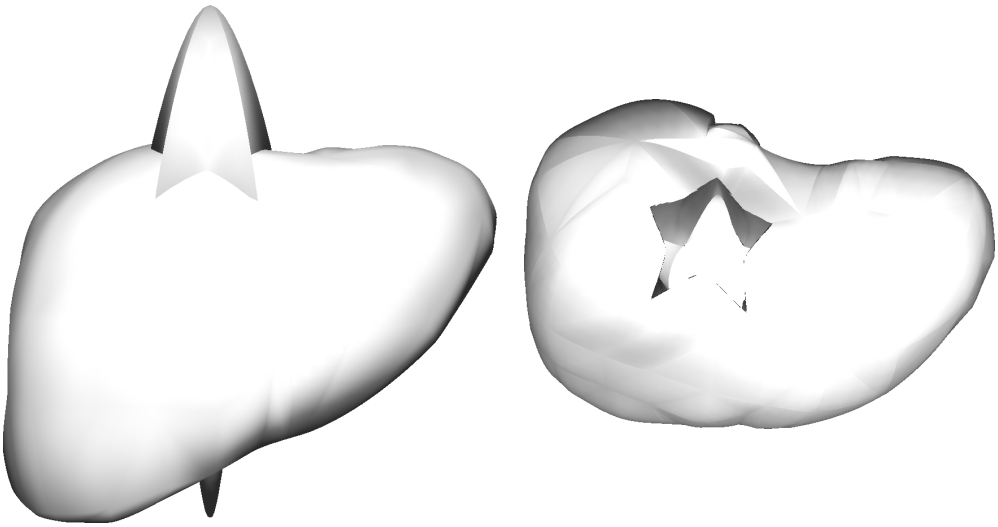


Figura 6.25 Diferencia entre el modelo del hígado y el de la punta de lanza. Izquierda: posición de partida. Derecha: resultado de la operación

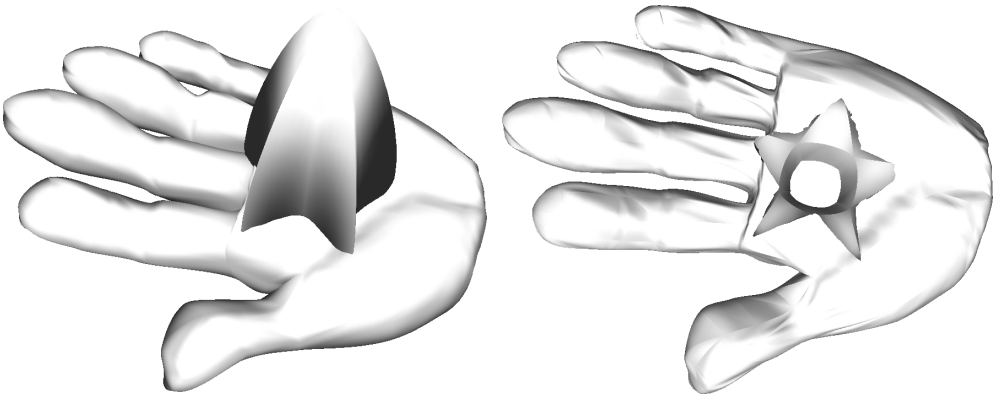


Figura 6.26 Diferencia entre el modelo de la mano y el de la punta de lanza. Izquierda: posición de partida. Derecha: resultado de la operación

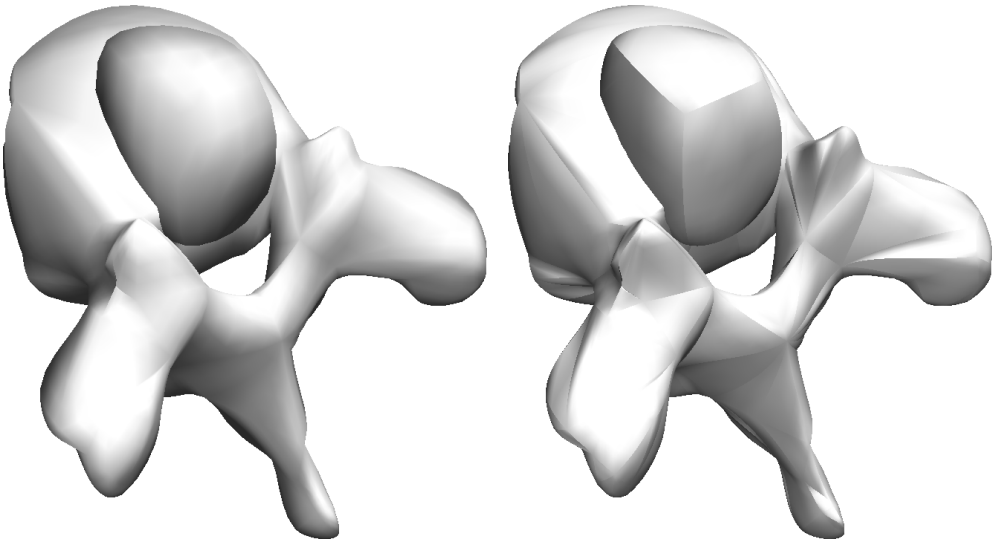


Figura 6.27 Unión entre el modelo de la vértebra y el prisma simple. Izquierda: posición de partida. Derecha: resultado de la operación

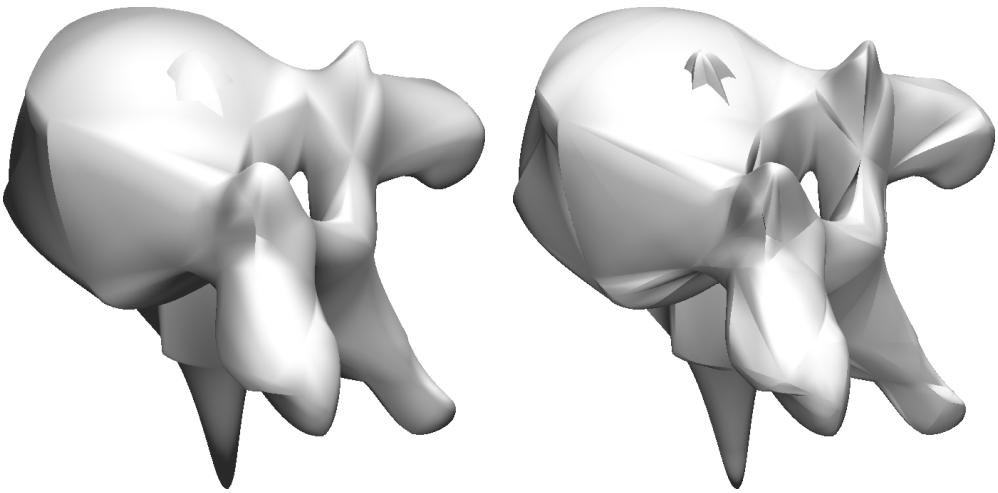


Figura 6.28 Unión entre el modelo de la vértebra y el de la punta de lanza. Izquierda: posición de partida. Derecha: resultado de la operación

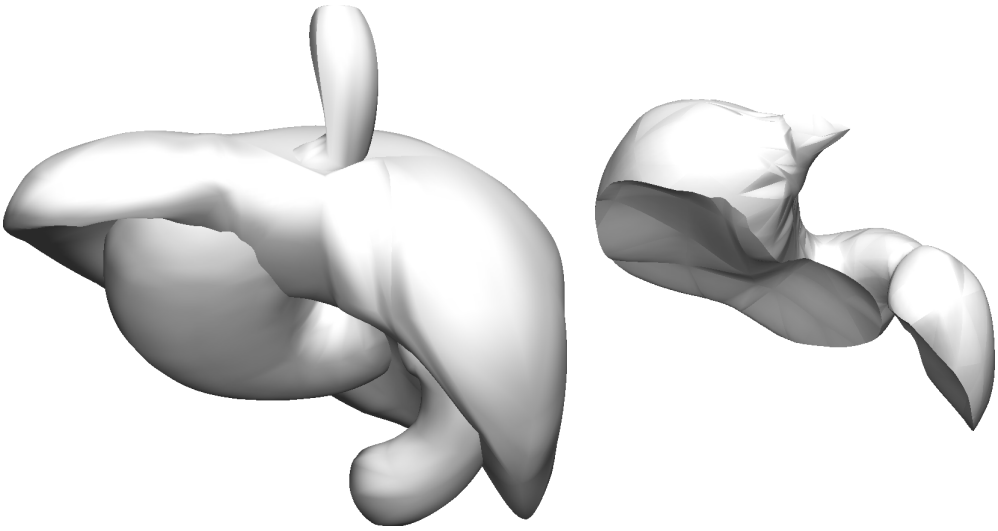


Figura 6.29 Intersección entre el modelo del estómago y el del hígado. Izquierda: posición de partida. Derecha: resultado de la operación

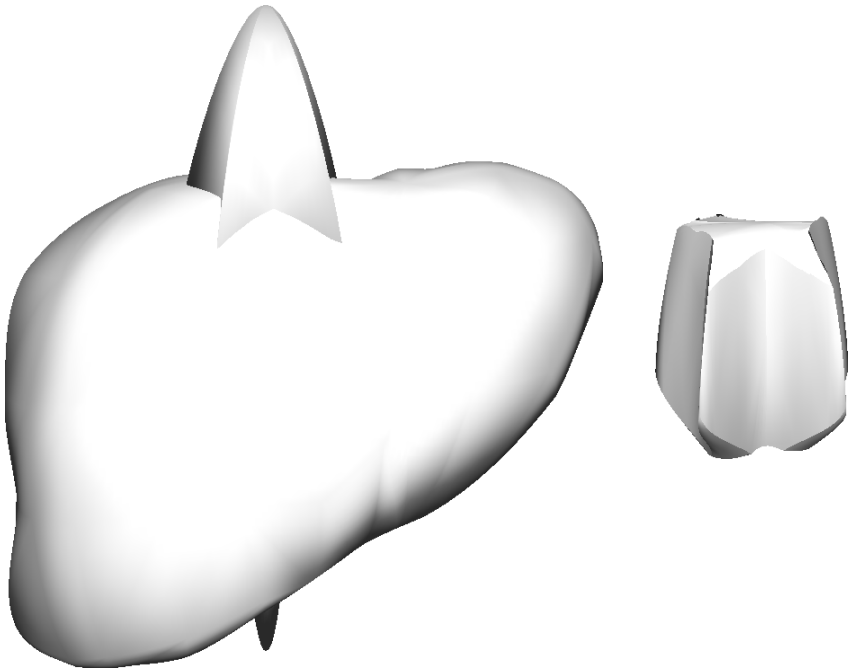


Figura 6.30 Intersección entre el modelo del hígado y el de la punta de lanza. Izquierda: posición de partida. Derecha: resultado de la operación

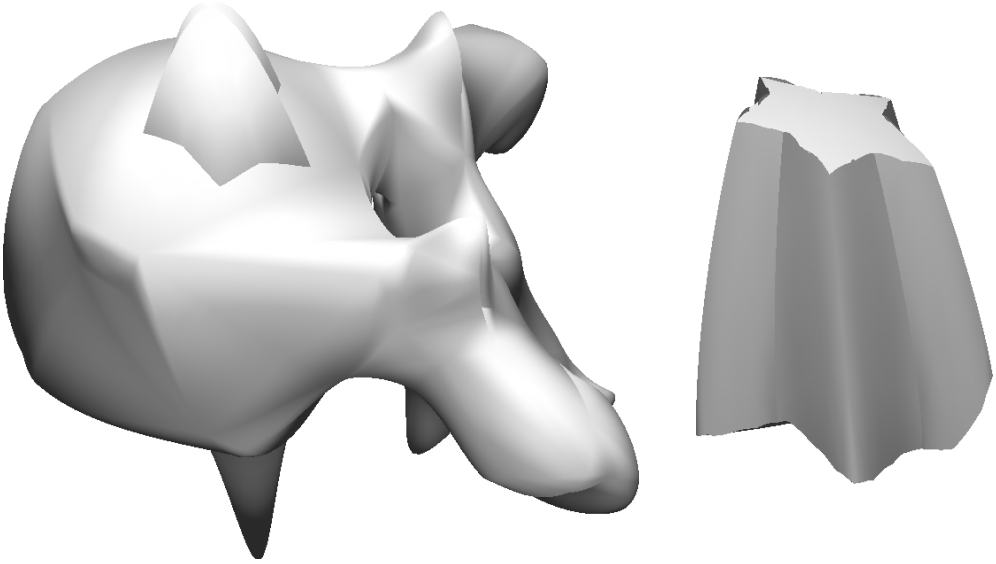


Figura 6.31 Intersección entre el modelo de la vértebra y el de la punta de lanza. Izquierda: posición de partida. Derecha: resultado de la operación

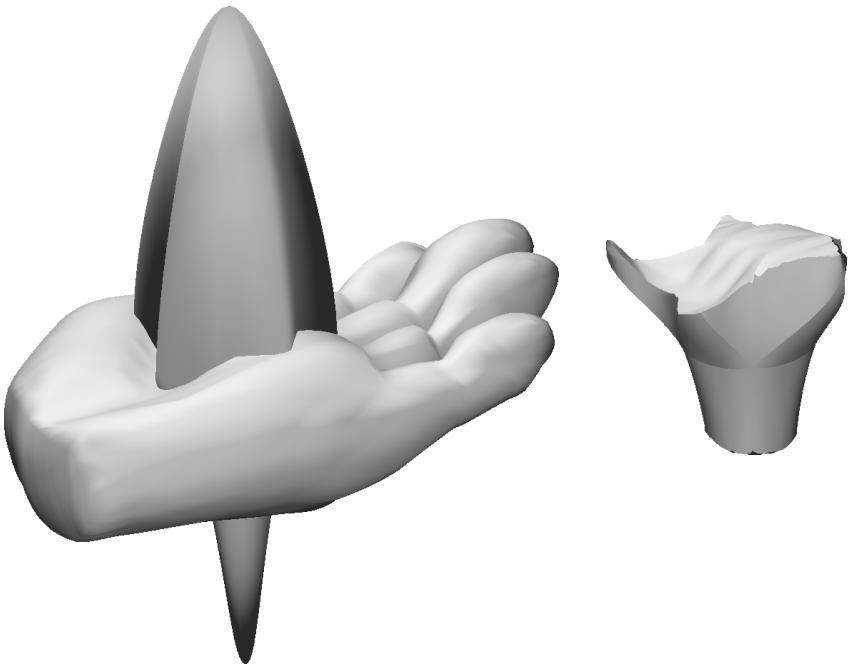


Figura 6.32 Intersección del modelo de la mano y el de la punta de lanza. Izquierda: posición de partida. Derecha: resultado de la operación

6.4 Construcción de la ESC resultado de una operación booleana

Un esquema de representación de sólidos es cerrado si los resultados de las operaciones con modelos de sólidos construidos según dicho esquema son a su vez representables de la misma forma. Para que el modelo ESC sea un esquema de representación cerrado es necesario por tanto definir el proceso que permita obtener una ESC a partir del resultado de una operación booleana como las descritas anteriormente.

6.4.1 Métodos directos. Simplificación de una ESC

La representación del resultado de una operación booleana mediante una cadena simplicial extendida propuesta en la sección 6.1 produce tras unas pocas operaciones un excesivo número de celdas en la cadena, debido a la necesidad de evaluar $ExCell\,(E \cap E')$ para cada pareja de celdas de los operandos. Sin embargo, muchas de estas intersecciones no formarán parte del sólido final, con lo que pueden ser simplificadas, reduciendo así el número total de cálculos a realizar, así como el tamaño de las ESC resultantes.

Para la simplificación se aplica el método propuesto por Ruiz [Rui01], consistente en evaluar en primer lugar la frontera del sólido resultado, para posteriormente obtener la cadena simplificada a partir de dicha frontera. Esta evaluación de la frontera puede hacerse de distintas formas; no obstante, se va a considerar la frontera evaluada tal y como se ha expuesto en la sección 6.3.

La solución más inmediata para la construcción de la ESC pasa por aprovechar los subparches generados para cada parche en el proceso de recorte, construyendo un símplice y una ffc por cada subparche situado en el nivel más bajo de las jerarquías de detalle (figura 6.33). Puesto que dichos subparches son a su vez parches triangulares de Bézier, el proceso de construcción de los símplices y las celdas de forma libre es exactamente el mismo que el descrito en la sección 4.3. Esta solución ha sido implementada satisfactoriamente para todas las operaciones booleanas definidas.

Una solución más elaborada consiste en construir un símplice y una ffc para cada (sub)parche que no haya sufrido cambios tras la aplicación del algoritmo 6.5 de recorte (figura 6.34). De este modo, los parches que no han sido recortados siguen proporcionando el mismo número de celdas que antes de la

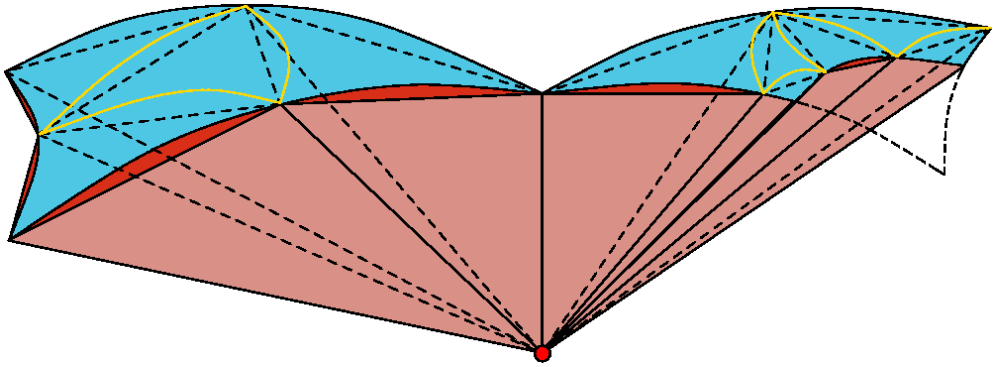


Figura 6.33 Primer método directo para la generación de la ESC para el resultado de una operación booleana

operación, y los que han sido recortados proporcionarán un número variable de celdas, dependiendo de la magnitud del recorte sufrido. Con el fin de evitar la aparición de discontinuidades entre los triángulos base de los símplices y las ffc, es necesario aplicar alguna técnica de subdivisión típica de las representaciones poliédricas basadas en niveles de detalle [LRC+03].

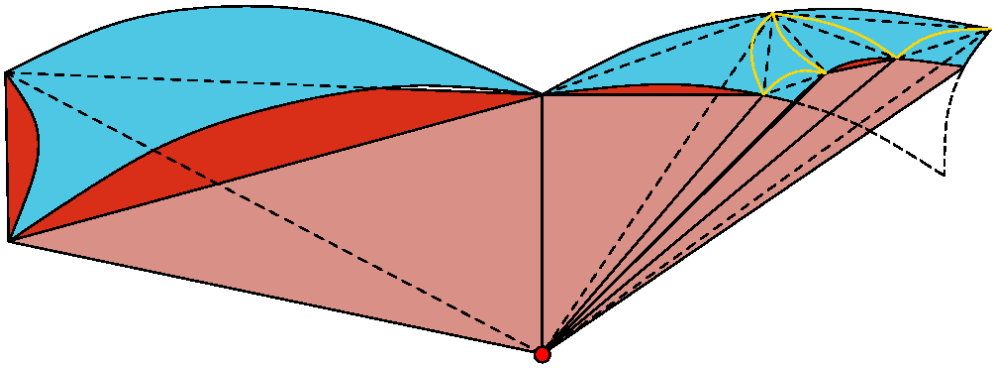


Figura 6.34 Segundo método directo para la generación de la ESC para el resultado de una operación booleana

Estas dos soluciones reducen significativamente el número de celdas respecto de la cadena que se obtendría aplicando la expresión descrita en la sección 3.4.3. La primera solución es más sencilla de implementar, mientras que la segunda consigue una reducción mayor en el número de celdas. Una característica importante de ambas soluciones es que facilitan la uniformidad en el tratamiento de las cadenas para el cálculo de los algoritmos propuestos anteriormente.

Otra posible solución pasa por la definición de un nuevo tipo de celda, que denominaremos *celda recortada*.

6.4.2 Uso de celdas recortadas

Se define una ***celda recortada*** como el resultado de la intersección de dos celdas extendidas, ya sean símplices, ffc u otras celdas recortadas. Como tal, puede estar delimitada por elementos planos o curvos, procedentes de las fronteras de las celdas intersectadas. La figura 6.35 muestra algunos ejemplos de celdas recortadas.

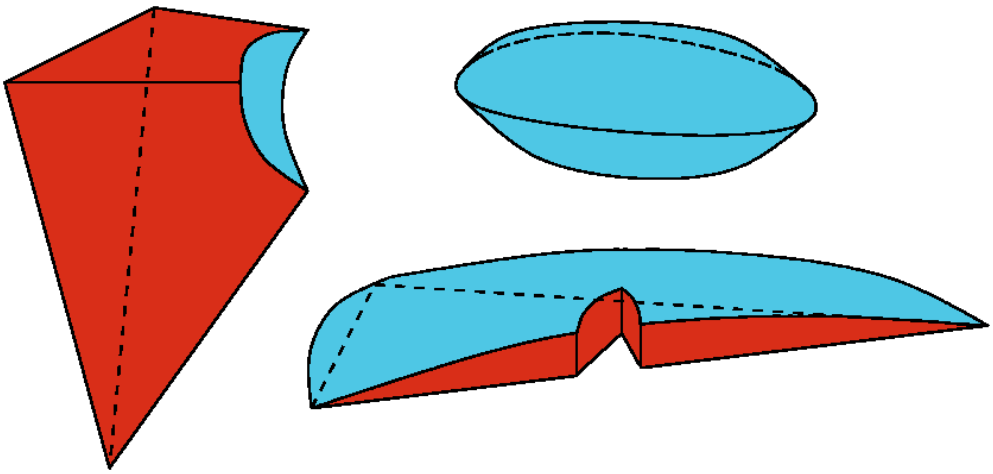


Figura 6.35 Ejemplos de celdas recortadas

La celda así definida tiene las mismas propiedades que las celdas de forma libre descritas en la sección 4.2, y puede formar parte de una cadena simplicial extendida. La mayor dificultad para la obtención de una celda recortada es la gran variedad de casos que se pueden dar al calcular la intersección entre celdas extendidas, por lo que determinar el número y tipo de los elementos delimitadores de su volumen puede llegar a ser una ardua tarea.

Una vez vista la definición de celda recortada, el proceso para la obtención de la cadena simplicial extendida que representa el resultado de una operación booleana entre dos sólidos de forma libre consiste en intersectar todas las celdas de las cadenas que los representan entre sí, y construir celdas recortadas en aquellos casos en que la intersección entre celdas no es vacía. Posteriormente,

se construiría la cadena resultado utilizando las celdas recortadas y las procedentes de las cadenas de los operandos, aplicando las expresiones de la sección 3.4.3.

La cadena simplicial extendida resultante no difiere en sus propiedades de las cadenas para sólidos originales, pudiendo utilizarse para otras operaciones del mismo modo que éstas. De esta forma, se mantiene la clausura de la representación.

6.5 Conclusiones

Este capítulo se ha centrado en el tratamiento de las operaciones booleanas entre sólidos representados utilizando el modelo ESC. Las especiales características de este modelo permiten tratar de forma uniforme dichas operaciones, permitiendo la representación de las mismas dentro de un sistema algebraico [Tor92, Fei95]. Esta representación permite demorar la evaluación de las operaciones booleanas, siendo al mismo tiempo plenamente utilizable, con lo que se posibilita la visualización del resultado sin evaluar dichas operaciones [GRF05].

Por otra parte, las características de los parches utilizados para la representación de la frontera de los sólidos permiten la evaluación de la frontera del resultado de una operación de forma robusta, primero calculando las intersecciones entre parches, y luego recortando los parches en función de las curvas de intersección obtenidas [GRF06]. Esta representación de la frontera puede ser a su vez considerada como punto de partida para la obtención de una nueva cadena simplicial extendida que representa al resultado de la operación, con lo que se mantiene la clausura de la representación.

162

162
162

163

7 Conclusiones y trabajo futuro

En este capítulo se resumen las principales aportaciones realizadas en el campo del Modelado de Sólidos, así como posibles líneas de trabajo para ampliar y complementar el trabajo realizado hasta el momento.

7.1 Resultados obtenidos

En el capítulo 1 se expusieron las motivaciones de este trabajo, así como los objetivos planteados y los resultados esperados. Se hace ahora una recapitulación de los mismos, así como de los resultados obtenidos en relación a ellos:

- En cuanto al primer objetivo planteado, relativo a la extensión del modelo ESC a superficies paramétricas triangulares, se presentó en el capítulo 4 una nueva definición del concepto de celda de forma libre compatible con ese tipo de superficies. Esta definición, junto con el algoritmo para el cálculo del test de inclusión de un punto en una ffc descrito en la sección 5.4, permite definir y operar con cadenas simpliciales extendidas que representan sólidos de forma libre delimitados por parches triangulares de Bézier [GRF03].

- El segundo objetivo planteado era el desarrollo de algoritmos para el cálculo del test de inclusión de un punto en un sólido de forma libre representado mediante una cadena simplicial extendida del nuevo tipo propuesto, así como para la visualización de los modelos. El primer algoritmo, que calcula el test de inclusión de un punto en un sólido, fue presentado en el capítulo 5 [GRF04a], mientras que la visualización de modelos se ha resuelto de dos formas: mediante el dibujado de los triángulos base de los subparches de las jerarquías (capítulo 4),

163
163

y mediante trazado de rayos, tal y como se describe en el capítulo 6 [GRF05].

- El tercer objetivo fundamental consistía en diseñar algoritmos para el cálculo de operaciones booleanas evaluadas y no evaluadas. Para la consecución de este objetivo se han implementado algoritmos robustos para el cálculo de las intersecciones entre dos parches triangulares de Bézier (sección 6.3.1), el recorte de parches utilizando las curvas de recorte anteriormente obtenidas [GRF06] (sección 6.3.2), y la combinación de los parches recortados para formar la frontera del sólido resultado de una operación aplicando distintos criterios según la operación que se trate (sección 6.3.3). Partiendo de esta representación de la frontera, se ha estudiado el método de construcción de una nueva cadena simplicial extendida que represente al resultado, de forma que se asegura la clausura de la representación con el modelo ESC (sección 6.4).

Todos los algoritmos propuestos se han implementado y probado en una aplicación experimental desarrollada utilizando C++ en un sistema GNU/Linux. Se han utilizado las librerías OpenGL para la visualización, y GLUI para la interfaz. Los datos e imágenes que se han mostrado a lo largo de este documento han sido obtenidos utilizando dicha aplicación de prueba.

7.2 Trabajo futuro

Aunque se ha conseguido una adaptación completa del modelo ESC a parches triangulares, cubriendo los objetivos planteados al inicio de la investigación, existen campos de trabajo abiertos que pueden ser interesantes para futuros desarrollos:

- Almacenamiento de propiedades de los materiales. El modelo ESC no sólo aporta una representación de la frontera de los sólidos, sino también del volumen de éstos. Por tanto, si se consigue una forma de asociar a ese volumen propiedades de material, se conseguiría ampliar la semántica del modelo para incluir sólidos heterogéneos.

- Conversión a otras representaciones estándar. El paso a otras representaciones tipo enumeración u octree parece factible, aplicando el algoritmo de inclusión de puntos [RF02].

- Evaluación eficiente de árboles CSG completos. Se ha presentado una solución para la evaluación de operaciones booleanas una a una, por lo que la evaluación de un árbol CSG paso a paso es inmediata. Sería interesante obtener un método para evaluar no sólo operaciones individuales, sino un árbol CSG completo en un solo paso y en un tiempo razonable.

- Simplificación de parches recortados. En la operación de recorte de parches se generan multitud de subparches que en ocasiones pueden reagruparse en parches más grandes, con lo que se simplifica la representación de la frontera, al tiempo que se mantiene la precisión en la representación.

- Aprovechando el trabajo ya existente con hardware gráfico para los parches triangulares utilizados [VPBM01, ATI01, BS05, BRS05], estudiar posibles implementaciones en hardware de algunos de los algoritmos propuestos.

Estos futuros desarrollos ampliarán y mejorarán las posibilidades del modelo.

7.3 Conclusiones

A lo largo de este documento se ha presentado el Modelo de Cadenas Simpliciales Extendidas para representación de sólidos de forma libre como un sistema cerrado que permite representar modelos y operar con ellos de la misma forma que se puede hacer con otros modelos de representación clásicos. Se ha presentado una adaptación de dicho modelo a parches paramétricos triangulares, justificada en que dicho tipo de superficies son ampliamente utilizadas en entornos CAD/CAM, y se han desarrollado algoritmos para el cálculo del test de inclusión de un punto en un sólido, así como la visualización de operaciones booleanas no evaluadas y como el cálculo de la frontera de los resultados de dichas operaciones.

Así mismo, se han presentado posibles líneas de continuación del trabajo realizado, con la idea de mejorar las prestaciones del modelo descrito y su aplicabilidad al trabajo con sólidos de forma libre en cualquier ámbito.

166

166
166

167

A Software

Este apéndice se dedica a la descripción de algunos detalles generales del software que se ha implementado para el desarrollo y prueba de los algoritmos descritos en este documento. No se pretende tratar en profundidad el diseño e implementación del mismo, sino dar una visión global que complemente el contenido de este documento.

A.1 Estructura de la aplicación

El software desarrollado ha sido diseñado utilizando un enfoque orientado a objetos, y ha sido implementado y probado en sistemas GNU/Linux, utilizando el lenguaje de programación C++ [Str97], así como las librerías OpenGL para la generación de imágenes [SWND05] y GLUI para el diseño de la interfaz [RSB]. En la figura A.1 se muestra una captura de pantalla de la aplicación en funcionamiento.

A continuación se describirán las clases más importantes y las relaciones entre ellas, de forma que quede clara la estructura de la aplicación.

A.1.1 Representación de árboles CSG

Tal y como se ha expuesto en capítulos anteriores, la representación de operaciones booleanas en un árbol CSG es muy utilizada en el mundo CAD/CAM. En la aplicación diseñada se ha recurrido a tres clases básicas para modelizar esta estructura: *Árbol_CSG*, que almacena la información general del árbol, *Nodo_CSG*, de la que heredan tres clases vinculadas con los tres tipos de nodos que puede contener un árbol CSG (nodos primitiva, nodos de transformación y nodos de operación), y *Objeto*, que es la clase destinada a la representación de sólidos. En la figura A.2 se pueden ver las relaciones entre estas clases.

167
167

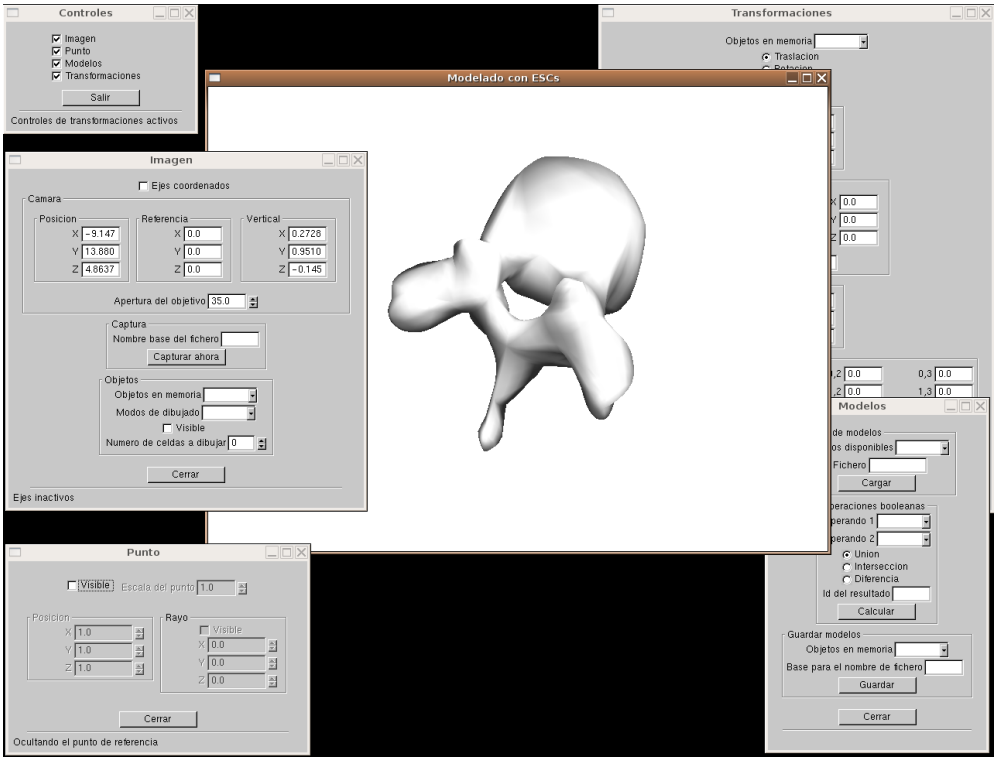


Figura A.1 Captura de pantalla del software desarrollado

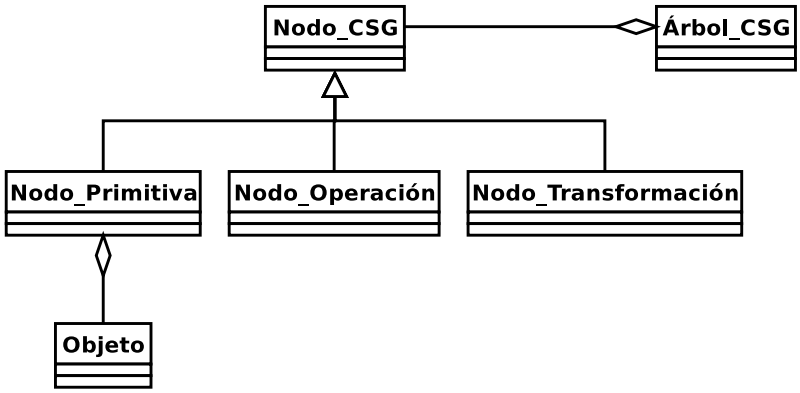


Figura A.2 Clases implicadas en la representación de árboles CSG

A.1.2 Representación de sólidos de forma libre

En el sistema de modelado implementado, un sólido de forma libre está definido

por los siguientes elementos (figura A.3):

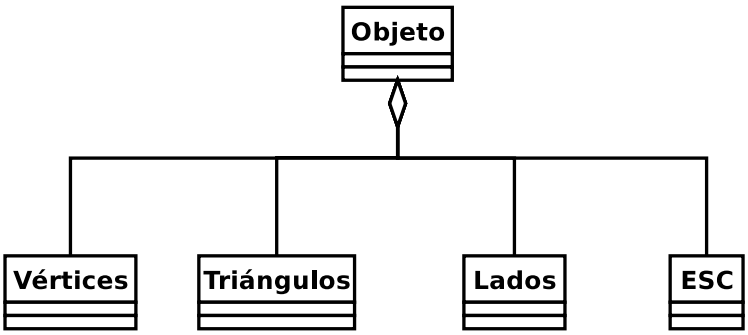


Figura A.3 Clases implicadas en la representación de sólidos de forma libre

- Una malla de triángulos que aproxima la superficie del sólido. Esta malla se almacena en forma de punteros a una lista de vértices, incluyendo información topológica de vecindad entre triángulos, necesaria para la definición de las celdas de forma libre (sección 4.2).

- La cadena simplicial extendida, formada por símplices y ffcs, cada uno de ellos definido tal y como se ha visto en capítulos anteriores.

- Otra información útil, como la caja englobante del sólido, definida como la caja englobante de los puntos de control de los parches triangulares de Bézier que determinan las celdas de forma libre de la cadena simplicial extendida que representa al sólido.

La representación no supone un gran incremento del espacio necesario respecto de una representación clásica de la malla de triángulos y sus parches asociados, puesto que cada símplice únicamente añade un vértice más a cada triángulo, y cada celda de forma libre sólo añade las ecuaciones de los planos delimitadores y la lista de parches que delimitan la celda.

A.1.3 Representación de cadenas simpliciales extendidas

Las cadenas simpliciales extendidas son el elemento básico alrededor del cual se articulan los algoritmos desarrollados. Para su representación en memoria se ha aplicado el concepto de *celda extendida* expuesto en el capítulo 3. Esta celda de propósito general es la clase abstracta de la que heredan las que se

dedican a la representación de *símplices* y *ffcs*, tal y como se muestra en la figura A.4.

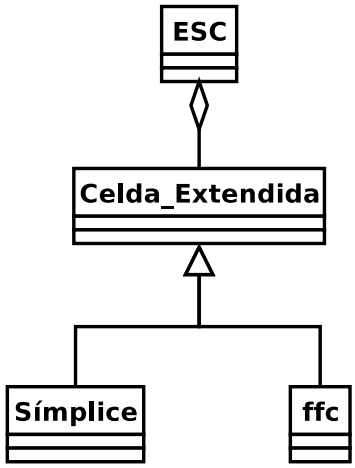


Figura A.4 Clases implicadas en la representación de cadenas simpliciales extendidas

La ejecución de los algoritmos se delega desde los objetos de la clase *ESC* a los objetos que representan las celdas extendidas que las forman. La composición de los resultados se hace posteriormente a nivel de cadena, tal y como se ha descrito en capítulos anteriores.

A.1.4 Representación de parches triangulares de Bézier

La representación de parches triangulares de Bézier ha de ser un reflejo de la estructura de detalle descrita en la sección 4.1.7, que además permita la representación de los parches recortados resultantes de la aplicación de los algoritmos descritos en el capítulo 6. Así, se ha definido una clase *Parche Triangular de Bézier* de la cual hereda la clase *Parche Triangular de Bézier Recortado* (figura A.5).

Dependiente de cada objeto de estas clases está su *jerarquía de detalle*, formada por tantos niveles como se hayan indicado a la hora de su construcción. Los algoritmos de cálculo de la inclusión de puntos, intersección y recorte de parches hacen uso intensivo de estas jerarquías.

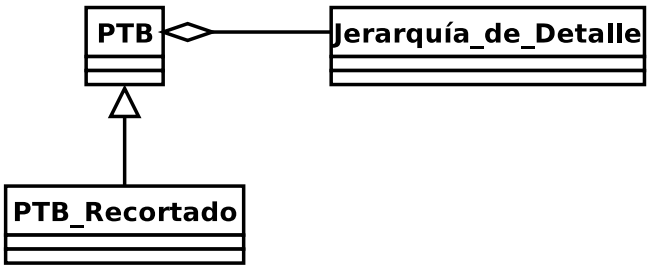


Figura A.5 Clases implicadas en la representación de parches triangulares de Bézier

A.2 Conclusiones

En este apéndice se ha hecho una breve descripción del software implementado durante el desarrollo y prueba de los algoritmos expuestos en este trabajo. Esta descripción no ha pretendido ser exhaustiva, sino simplemente orientativa de la distribución lógica de la aplicación.

La figura A.6 muestra un diagrama que agrupa las clases descritas y sus relaciones. En la descripción realizada se han eliminado algunas relaciones y clases simples muy utilizadas para favorecer la legibilidad de los diagramas, como pueden ser las clases que representan puntos y vectores 3D, o las cajas englobantes de las entidades geométricas utilizadas.

Una vez alcanzado un grado de madurez suficiente de la aplicación, se planteará la liberación del código bajo una licencia adecuada (GPL o Creative Commons).

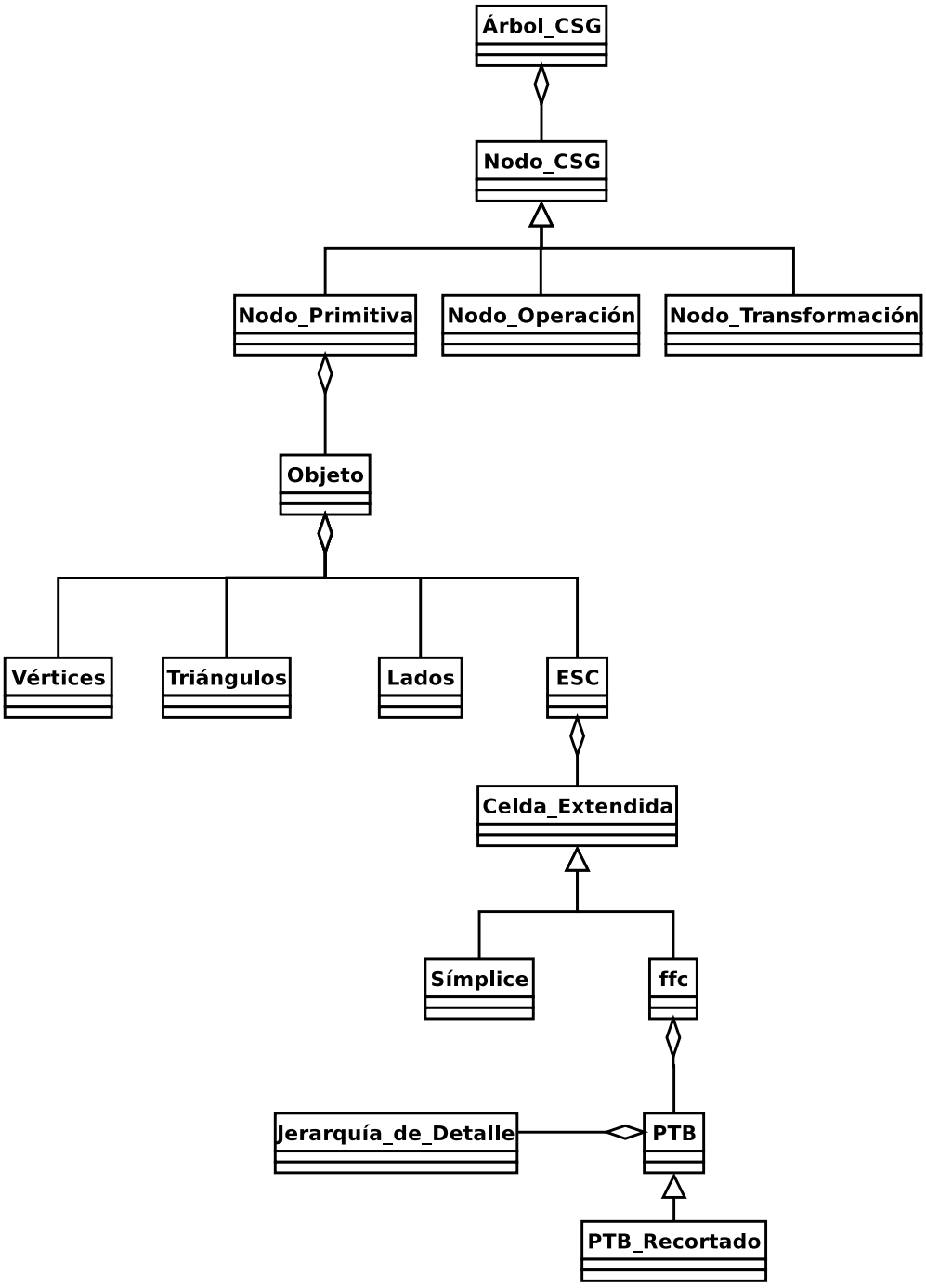


Figura A.6 Diagrama de clases simplificado

173

B English summary

A summary of this thesis is included here as partial fulfilment for the European Doctorate award.

B.1 Introduction

The main goal in Solid Modeling is to represent and handle data from existing or non-existing objects using computers, so that their characteristics and properties can be studied in a easy way. This is especially useful when it is impossible to work directly with the objects (because of their size, their value, or simply because they do not exist) [Män88, HR96, Sha02].

Working with free-form solids is an open issue in Solid Modeling. Free-form solids are bounded not only by planar surfaces, but also by curved surfaces; this makes especially difficult the representation and operation with this kind of solids.

This work proposes a solution for free-form solid modeling based on the concept of Extended Simplicial Chain [RF99a], using low-degree parametric patches to represent the boundary of the solids (triangular Bézier patches [Far86], to be precise).

The Extended Simplicial Chain model is specifically intended to represent the volume of free-form solids. It has been proved to be a valid model for two-dimensional solids bounded by conic and Bézier curves [RF97], as well as for three-dimensional solids bounded by algebraic patches [RF99a, RF99b, RF98, RF02]. Moreover, parametric surfaces are the *de facto* standard in CAD/CAM environments, because it is very easy to work with them. Therefore, this work presents the extension of the Extended Simplicial Chain model to parametric

173
173

surface patches, in order to combine the usability of these patches with the well-known advantages of the model for free-form solid modeling.

B.1.1 Goals

The main goal of this thesis is the extension of the Extended Simplicial Chain model to allow the representation of free-form solids bounded with triangular parametric patches. This will prove that the change on the surface type does not have influence on the properties of the model nor even on the algorithms developed.

The main goal is composed of the following minor goals:

- Extend the Extended Simplicial Chain model to free-form solids bounded with triangular parametric patches.
- Develop algorithms to compute the point in solid test and the visualization of solids.
- Develop algorithms to compute evaluated and non-evaluated boolean operations on free-form solids modeled using extended simplicial chains.

B.1.2 Expected results

The expected results of this work were the following:

- The formalization of the concept of *free-form cell* bounded with triangular parametric patches. This is a major element of the Extended Simplicial Chain model.
- A robust algorithm to compute the point in solid test for free-form solids modeled with extended simplicial chains.
- Application of the point in solid test algorithm to common issues in Solid Modeling, like the ray-casting based visualization.
- Algorithms to evaluate boolean operations on free-form solids modeled with extended simplicial chains.

B.2 Solid Modeling

Solid Modeling is a discipline closely related to CAD/CAM environments, as it is usually necessary to represent not only the surface of solids, but also their inside and their properties. Shapiro [Sha02] called this *the need for informational completeness*, and characterized it through these three conditions:

1. Any constructed representation should be *valid* in the sense that it should correspond to some real physical object.

2. Any constructed representation should represent *unambiguously* the corresponding physical object.

3. The representation should support (at least in principle) *any and all geometric queries* that may be asked of the corresponding physical object.

There are many definitions of Solid Modeling in the references. Some representative ones are:

> *"Solid modeling is a consistent set of principles for mathematical and computer modeling of three-dimensional solids"*
>
> *[Sha02]*

> *"Solid Modeling is a rapidly growing set of theories, algorithms and software techniques that deals with the representation, design, visualization and analysis of three-dimensional models"*
>
> *[RTA93]*

> *"The objective of solid modeling is to represent, manipulate, and reason about the three-dimensional shape of solid physical objects, by computer"*
>
> *[HR96]*

Model is a basic concept in Solid Modeling. It is an analytic and abstract representation that should contain enough information to solve the problem that has motivated its creation. Some examples of situations where models are used include simulation processes, volumetric property calculations or collision detection testing.

B.2.1 Conceptual framework for Solid Modeling

The classical conceptual framework for Solid Modeling was presented in 1980 by Requicha [Req80]. In this framework, phisycal objects are described as *abstract solids* using representation schemes; these descriptions are called *models*.

Abstract solids are subsets of the Euclidean space $\mathbb{E}^3$ that verify the following conditions:

- **Rigidity**: the shape of a solid does not depend on its position nor even its orientation.

- **Homogeneus three dimensionality**: a solid has to have an inside, and its boundary can not have isolated or dangling elements, like figure B.1.

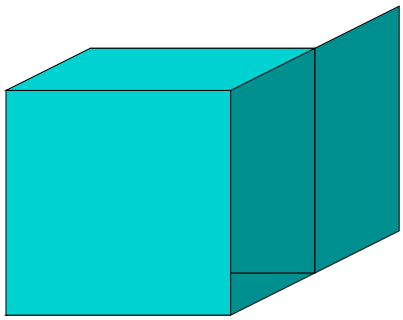


Figure B.1
Non-homogeneous set

- **Finiteness**: a solid has to occupy a finite portion of $\mathbb{E}^3$.

- **Closure under rigid motions and certain boolean operations**: the result of applying translations, rotations and boolean operations on a solid must be another solid.

- **Finite describability**: in order to obtain a representation that can be stored and manipulated in a computer, a solid has to be describable with a limited set of elements (polygons, surface patches, ...).

- **Boundary determinism**: the boundary of a solid has to determine clearly which are the inside and the outside regions of the solid.

Suitable models for solids are subsets of $\mathbb{E}^3$ that are bounded, closed, regular, and semianalitic. These subsets are called *r-sets*.

A *representation scheme* for solids [Req80] comprises a set of lexical elements and a set of syntactic rules to combine them in a valid description. It is formally defined as a relation **s** between two spaces: the *mathematical modeling space* **M**, composed by all the abstract solids, and the *representation space* **R**, which contains all the syntactically correct representations (figure B.2).

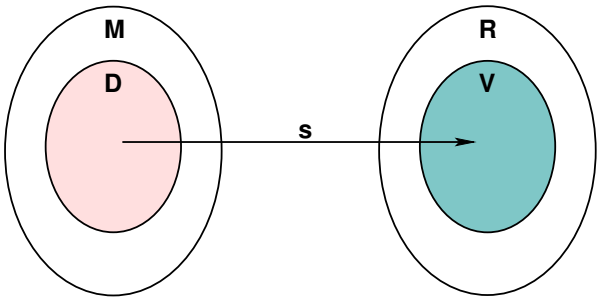


Figure B.2 Requicha's set diagram for a representation scheme

The domain of **s** is named **D**, and the range of **D** under **s** is named **V**. A representation in **V** is *valid* because it is syntactically and semantically correct, as it belongs to the set of correct representations **R**, and has corresponding elements in **M**.

A representation $r \in \mathbf{V}$ is *unambiguous* or *complete* if there is only one solid $\mathbf{s}^{-1}(r) \in \mathbf{D}$ that corresponds to r. Moreover, r is said to be *unique* if it is the only element in **V** that can represent that solid. A representation scheme is said to be unambiguous or complete if all of its valid representations are unambiguous; it is said to be unique if all of its valid representations are unique.

Requicha also defined the formal and informal properties of representation schemes for solids. These properties are the following:

- **Formal properties**:
  - *Domain*: set of solids that can be represented. The wider the domain, the higher the descriptive power of the scheme.
  - *Validity*: a representation scheme should produce only valid representations, otherwise it should provide an easy way to test the validity of the representations.
  - *Completeness*: a representation scheme is complete if all its representations are complete.
  - *Uniqueness*: if all of the representations are unique, it will be easier to compute the equality of solids, because it can be done just by comparing their representations syntactically.
- **Informal properties**:
  - *Conciseness*: it is desirable for any representation scheme that the size of its representations is small enough to allow them to be processed with a computer.
  - *Ease of creation*: it has to be easy to create a valid representation of a solid.
  - *Efficacy in the context of applications*: the scheme has to allow the development of correct, efficient and robust algorithms to work with the representations.

B.2.2 Representation of free-form solids

Free-form solids are bounded not only by planar faces, but also by curved surfaces. Some free-form solids can be represented analytically in an easy way, like spheres, ellipsoids or torii, but there also exist solids whose surface is completely irregular, and therefore it is necessary to use a representation scheme with a domain wide enough to provide a usable representation.

Many representation schemes have been proposed to represent solids. Most of them are specifically intended to represent solids with planar faces, like spa-

tial enumeration [KCY93, Män88], octree [Mea82, Män88], BSP [TN87] or polyhedral B-Rep [Req80, Män88, FvDFH92, Sha02]. Some of them could represent some free-form solids, but with a more or less limited domain, depending on their implementation, like primitive instancing [Män88, MNK90, SV95], spatial decomposition [Män88, Woo03], CSG [RV77, Req80, Män88], sweeping [Mor97, AMBJ06], B-Rep with curved patches [KKM97, KGMM97], hiperpatches [Mor97] or free-form deformations [SP86, Bec98, HML00]. Finally there are some proposals of hybrid schemes that combine two or more representations in order to enhance the performance of the system [Män88, FvDFH92, Sha02].

A more specific approach to free-form solid modeling is the one from Allègre et al. [AGCA06], which presents a mixed data structure based on a CSG tree that combines point clouds, triangle meshes and skeletal implicit surfaces [BGA04]. Depending on the operation that is going to be computed, it uses the most convenient representation.

Other works intended for free-form solid modeling are the ones from Lin-sen [Lin00] and Biermann et al. [BKZ01]. These approaches use a B-Rep representation using parametric patches and subdivision surfaces respectively. Regarding boolean operations, they operate using the control point nets of the surfaces to obtain an approximate result for the boundary of the new solids.

The search for precision in the evaluation of boolean operations led to systems based on lazy evaluation and exact arithmetic, but with a high execution time, like the work of Keyser et al. [KCF+04].

Ruiz de Miras and Feito presented a different approach: the Extended Simplicial Chain model [RF97, RF99a]. This model is a representation scheme based on the decomposition of the solid volume into a set of non-disjoint cells that was initially developed for two dimensional solids bounded by conic and Bézier curves [RF97], and was later extended to three dimensional solids bounded with triangular algebraic patches.

This work extends the Extended Simplicial Chain model to triangular parametric patches, studies the necessary adaptations of the algorithms, and deals with the evaluation of boolean operations between free-form solids represented with extended simplicial chains.

B.3 The Extended Simplicial Chain model

The Extended Simplicial Chain model (from now on, it will be written as ESC model) is a formal scheme to represent and handle free-form solids [RF99a]. It extends the Simplicial Chain (SC) model proposed by Feito and Rivero to represent polyhedral solids [FR98].

With the ESC model, each solid is decomposed into simpler elements (*extended cells*) so that any modeling operation is implemented by solving the problem for each individual element and merging the results. This is a typical application of the *divide and conquer* principle [BB97].

One of the most differentiating characteristics of the ESC model is the fact that it does not force the cells to be disjoint. This makes it easier to create the cells, unlike the classical spatial decomposition scheme.

B.3.1 Extended cells

There are two main types of extended cells in any extended simplicial chain: simplices and free-form cells. Each of them contributes to the volume of the represented solid by adding or subtracting its volume to the final result, depending on its sign.

Simplices

A ***d-dimensional simplex*** is the convex hull of a set of d+1 linearly independent points. In 3D, a simplex is a tetrahedron, like the one in figure B.3.

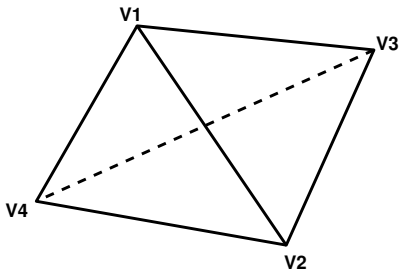


Figure B.3 Three dimensional simplex

Any point in space can be represented by a lineal combination of the vertices from a simplex. The coefficients of this lineal combination are called ***barycentric coordinates***. For three dimensional simplices these coordinates are computed in this way [Men92]:

$$\begin{pmatrix} t \\ u \\ v \\ w \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} \tag{B.1}$$

where the coefficients of the 4×4 matrix are the homogeneous coordinates of the vertices of the simplex, and $(x_p, y_p, z_p, 1)$ are the homogeneous coordinates of the point whose barycentric coordinates are (t, u, v, w). This concept will be intensively used in the point in solid test algorithm that will be explained later.

Each simplex has an associated sign, depending on the orientation of its vertices. Counterclockwise oriented vertices determine a positive simplex, whereas clockwise oriented vertices determine a negative simplex. This sign is used to determine if the volume of the simplex is added or subtracted to the final solid, and for three dimensional simplices it can be easily computed as the sign of the ***signed volume*** of the simplex [O'R98]:

$$\Delta = \frac{1}{6} \cdot \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \tag{B.2}$$

It is possible to decompose the volume of any polyhedral solid into a set of simplices sharing one vertex; this set forms a ***simplicial chain*** [FR98]. Figure B.4 shows the building process of a two dimensional solid whose area is decomposed into a set of three simplices: two of them contributes with positive area to the final solid, and one contributes with negative area. The sum of all those areas is equal to the area of the solid.

The vertex shared by all the simplices from a simplicial chain is called the ***origin*** of the chain, and therefore all the simplices that share that vertex are called ***original simplices***.

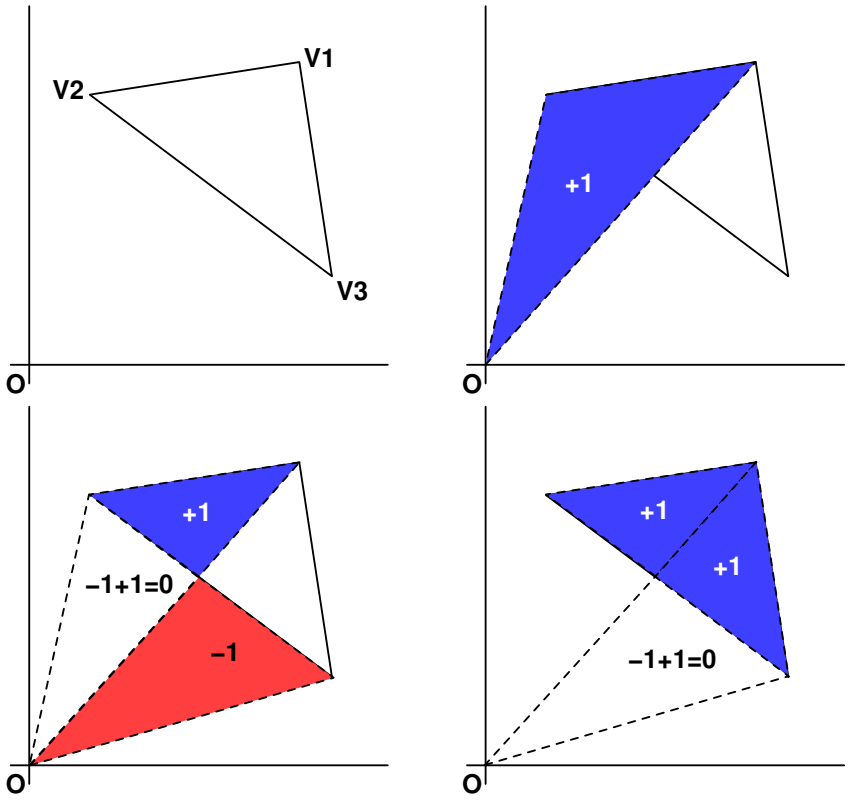


Figure B.4 Building process of a two dimensional solid from a simplicial chain that represents it

Free-form cells

Simplicial chains are not suitable to represent free-form solids, as they are composed of simplices bounded by planar elements. It is then necessary to add a new type of extended cell to allow the representation of solids bounded by curved elements. This cells are called *free-form cells* (noted ffcs).

A ***d-dimensional free-form cell*** is a set of points in $\mathbb{R}^d$, obtained as the intersection of the half-spaces defined by a d-1-dimensional free-form element and one or more planar elements of dimension d-1 that verifies [RF99a]:

- It is a closed and connected set in $\mathbb{R}^d$.

- Every point in the set belongs to the same connected component with respect to the free-form bounding element, excepting the ones from the boundary.

- The points from the set that are shared by a number of bounding elements greater than or equal to d belong to the free-form bounding element.

Every ffc has an associated sign to determine if its volume is added to or subtracted from the volume of the solid.

The definition of ffc given above does not depend on the type of free-form bounding elements used, so up to now it has been particularized using Bézier curves and conics in 2D [RF97], triangular algebraic patches [RF99a] and triangular Bézier patches [GRF03, GRF04a].

B.3.2 Definition of Extended Simplicial Chain

A d-dimensional ***extended simplicial chain***, noted ***ESC*** is defined as [RF99a]:

$$\delta = \sum_{i=1}^{n} \alpha_i \cdot E_i \tag{B.3}$$

where the α_i represent integer coefficients, and the E_i represent d-dimensional extended cells, either simplices or ffcs. The coefficient associated to an extended cell is equal to its sign.

A key concept in the ESC model is the ***function associated to an ESC***. For a given chain δ, this function is defined as:

$$\begin{aligned} f_\delta &: \mathbb{R}^d \rightarrow \mathbb{Z} \\ f_\delta(\mathbf{Q}) &= \sum_{\forall E_i / \mathbf{Q} \in E_i} \alpha_i \end{aligned} \tag{B.4}$$

the function associated to an ESC returns the sum of the coefficients associated to the extended cells that contains the point $\mathbf{Q}$. This function is used to define the ***solid associated to an ESC*** as:

$$FF_\delta = \left\{ \mathbf{Q} \in \mathbb{R}^d / f_\delta(\mathbf{Q}) \neq 0 \right\} \tag{B.5}$$

Figure B.5 shows a two dimensional solid, together with the extended cells (three original simplices and one ffc) that forms an ESC that represents it. In this case, the origin of the chain is a point placed outside the solid, the positive cells are bluish, and the negative ones are reddish.

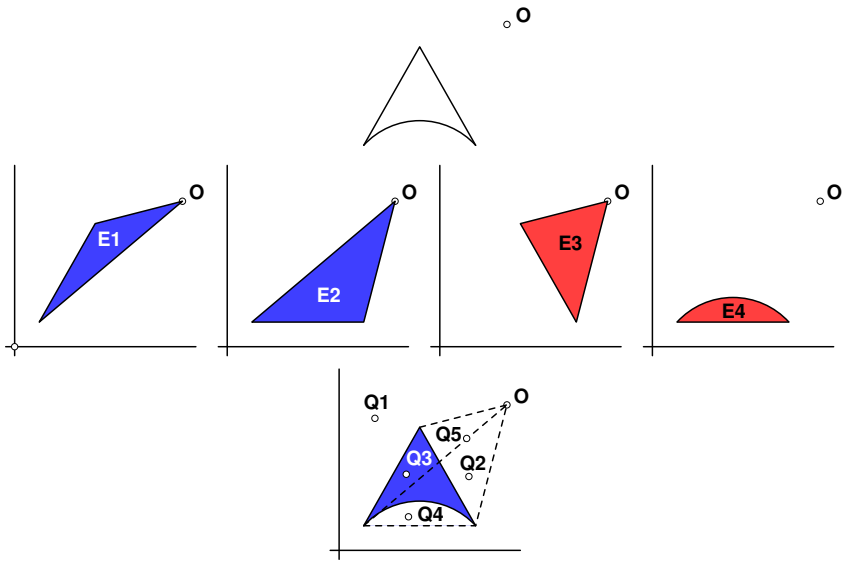


Figure B.5 ESC for a two dimensional solid. Up: solid and origin of the ESC. Middle: extended cells. Down: solid obtained by combining the area of the extended cells

The ESC represented in figure B.5 can therefore be written as:

$$\delta = E_1 + E_2 - E_3 - E_4 \tag{B.6}$$

and the function associated to δ takes these values for the points shown in the figure:

$$\begin{aligned} f_\delta(\mathbf{Q1}) &= 0; \\ f_\delta(\mathbf{Q2}) &= \alpha_2 + \alpha_3 = 1 - 1 = 0; \\ f_\delta(\mathbf{Q3}) &= \alpha_1 = 1; \\ f_\delta(\mathbf{Q4}) &= \alpha_2 + \alpha_4 = 1 - 1 = 0; \end{aligned} \tag{B.7}$$

$\mathbf{Q_5}$ represents a special case, since it is placed on the border of two extended cells (E_1 and E_2) and inside a third one (E_3). If it is considered as inside the three cells, the value of the associated function would be:

$$f_\delta(\mathbf{Q_5}) = \alpha_1 + \alpha_2 + \alpha_3 = 1 + 1 - 1 = 1 \tag{B.8}$$

This is clearly a wrong value for point $\mathbf{Q_5}$. In order to avoid this kind of situations, and following the work of Requicha [RV77], the regular function associated to an ESC is defined in the following way:

Let p and n represent the maximum and the minimum of the values of the associated function in an arbitrarily small neighbourhood of $\mathbf{Q}$:

$$\begin{aligned} p &= max\left\{f_\delta\left(\mathbf{Q}'\right) \mid \forall\ \mathbf{Q}' \in N_\varepsilon(\mathbf{Q})\right\} \\ n &= min\left\{f_\delta\left(\mathbf{Q}'\right) \mid \forall\ \mathbf{Q}' \in N_\varepsilon(\mathbf{Q})\right\} \end{aligned} \tag{B.9}$$

Then, the ***regular function associated to an ESC*** δ is defined as follows:

$$\begin{aligned} f_\delta^* &: \mathbb{R}^d \rightarrow \mathbb{Z} \\ f_\delta^*(\mathbf{Q}) &= 0 \quad if\ \mathbf{Q} \notin closure\left(interior\left(FF_\delta\right)\right) \\ f_\delta^*(\mathbf{Q}) &= p \quad if\ p > 0 \\ f_\delta^*(\mathbf{Q}) &= n \quad otherwise \end{aligned} \tag{B.10}$$

The value of the regularized version of the associated function for the point $\mathbf{Q_5}$ is 0, as it was expected.

There is not a unique ESC that represent one solid; it is then necessary to stablish a criterion to select a representative ESC for each solid. This is the reason why a ***normal extended simplicial chain*** [Rui01] is defined as the chain whose represented solid verifies:

$$\delta\ is\ normal\ iff\ f_\delta^*(\mathbf{Q}) = 1 \quad \forall\ \mathbf{Q} \in FF_\delta \tag{B.11}$$

From now on, free-form solids will be always represented by normal ESCs.

B.3.3 Operations using ESCs

There are two basic operations with ESCs: sum and product by a scalar value. This operations give as a result new ESCs, and are defined as follows:

Given two ESCs:

$$\delta = \sum_{i=1}^{n} \alpha_i \cdot E_i \tag{B.12a}$$

$$\delta' = \sum_{j=1}^{m} \alpha'_j \cdot E'_j \tag{B.12b}$$

The sum of chains δ and δ' is a new ESC formed by the sum of the cells from the two chains [Rui01]:

$$\delta + \delta' = \sum_{i=1}^{n} \alpha_i \cdot E_i + \sum_{j=1}^{m} \alpha'_j \cdot E'_j \tag{B.13}$$

The product of an ESC δ, by a scalar value λ, is a new chain whose coefficients are multiplied by the scalar value [Rui01]:

$$\lambda \cdot \delta = \sum_{i=1}^{n} (\lambda \cdot \alpha_i) \cdot E_i \tag{B.14}$$

The regularized associated functions to the new ESCs are defined as [Rui01]:

$$\begin{aligned} f_\delta +^* f_{\delta'} &= f^*_{(\delta+\delta')} \\ \lambda \cdot^* f_\delta &= f^*_{(\lambda\cdot\delta)} \end{aligned} \tag{B.15}$$

The set of ESCs is a vectorial space with respect to the regularized operations above [FR98]. From now on, all the operations will be considered as regularized.

Boolean operations

Boolean operations are solved in a straight way using ESCs, as they can be reduced to the composition of the results from the individual cells. To achieve this target, it is necessary to deal with the intersection of two whichever extended cells.

Intersecting whichever two extended cells gives as a result a set of points which has to be represented somehow. However, this set is a volume, and therefore it can be decomposed into extended cells and represented by a normal

ESC. From now on, the decomposition into extended cells of the intersection between cells E and E' will be noted as the $\boldsymbol{ExCell}(\boldsymbol{E} \cap \boldsymbol{E'})$ function [Rui01].

Given the ESCs defined in B.12a and B.12b, and their associated free-form solids FF_δ and $FF_{\delta'}$, the ESC for the intersection $FF_\delta \cap FF_{\delta'}$ is computed as [Rui01]:

$$\delta_{FF_\delta \cap FF_{\delta'}} = \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\alpha_i \cdot \alpha'_j\right) \cdot ExCell\ \left(E_i \cap E'_j\right) \tag{B.16}$$

Figure B.6 shows how the ESC for the result of the intersection of two free-form solids is computed using the extended cells of the chains from the operands. In the first row of the figure, the free-form solids and their ESCs are drawn, while the second and third rows show each of the extended cells of the new chain, and the fourth row presents the composition of the cells in the new solid.

The ESC computed in figure B.6 is:

$$\begin{aligned} \delta = & ExCell\ (S3 \cap s3) - ExCell\ (S2 \cap s3) \\ & + ExCell\ (ffc1 \cap s3) - ExCell\ (S3 \cap s1) \\ & + ExCell\ (S2 \cap s1) - ExCell\ (ffc1 \cap s1) \end{aligned} \tag{B.17}$$

The associated function f_δ takes the following values for the points shown in figure B.6:

$$\begin{aligned} f_\delta(\mathbf{Q1}) &= \alpha_1 + \alpha_2 + \alpha_4 + \alpha_5 = 1 - 1 - 1 + 1 = 0 \\ f_\delta(\mathbf{Q2}) &= \alpha_2 + \alpha_5 = -1 + 1 = 0 \\ f_\delta(\mathbf{Q3}) &= \alpha_4 + \alpha_5 = -1 + 1 = 0 \\ f_\delta(\mathbf{Q4}) &= \alpha_5 = 1 \\ f_\delta(\mathbf{Q5}) &= \alpha_2 + \alpha_3 + \alpha_5 + \alpha_6 = -1 + 1 + 1 - 1 = 0 \\ f_\delta(\mathbf{Q6}) &= \alpha_5 + \alpha_6 = 1 - 1 = 0 \end{aligned} \tag{B.18}$$

The new chain is normal, as its value is 1 for every point in the solid.

Once there is a way to compute the ESC for the intersection of two solids, the other boolean operations can be represented using sums, products by a scalar value and intersections in the following way [Rui01]:

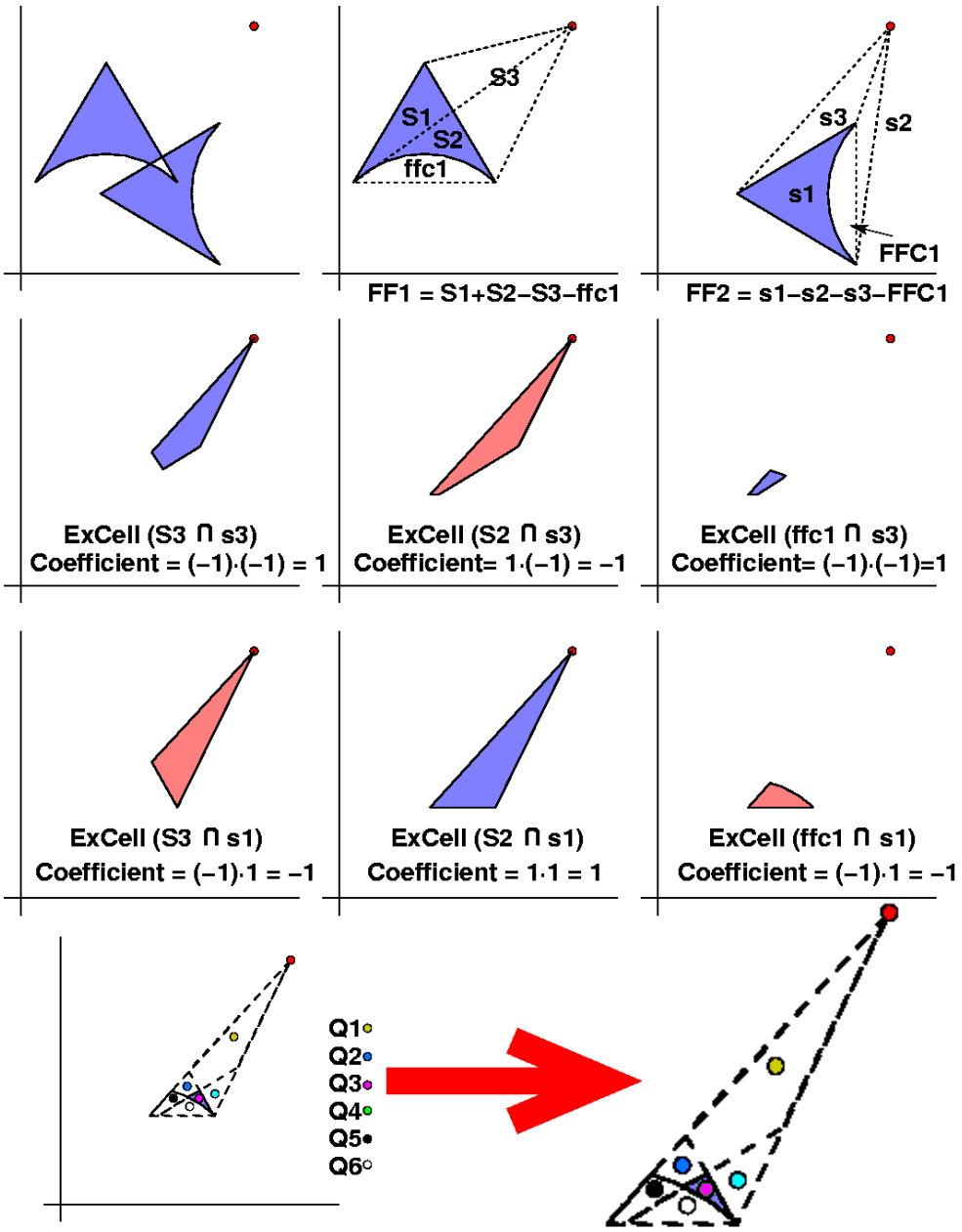


Figure B.6 First row: two free-form solids and the ESCs that represent them. Second and third rows: results of the intersection of the extended cells from the two ESCs. Fourth row: resulting solid and enlargement

- Union:

$$\delta_{FF_\delta \cup FF_{\delta'}} = \delta + \delta' - \delta_{FF_\delta \cap FF_{\delta'}} \tag{B.19}$$

- Difference:

$$\delta_{FF_\delta - FF_{\delta'}} = \delta - \delta_{FF_\delta \cap FF_{\delta'}} \tag{B.20}$$

- Complementation:

$$\delta_{FF_\delta^C} = \delta_{\mathbb{R}} - \delta \tag{B.21}$$

  considering $\delta_{\mathbb{R}}$ as the ***universe chain***, i.e. an ESC that represents the space in which the solid is included. This can be modeled as a cube or sphere that includes all the modeling space.

All the ESCs obtained in this way are normal, and can be used as operands in further operations.

B.4 Applying parametric triangular patches to the ESC model

Once the basics of the ESC model have been presented, the application of the model to free-form solids bounded by parametric triangular patches is described here. To be precise, triangular Bézier patches are used in this work.

B.4.1 Triangular Bézier patches

Triangular Bézier patches (TBPs) [Far86, Far93, PBP02] are defined over a triangular parametric domain. The three parameters u, v and w verify:

$$u + v + w = 1$$
$$0 \leq u \leq 1, \quad 0 \leq v \leq 1, \quad 0 \leq w \leq 1$$

The control point net for a TBP is composed of $\frac{1}{2} \cdot (n+1) \cdot (n+2)$ points, n being the degree of the patch. Each control point is identified using three indices: i, j and k. For a n-degree TBP, the indices verify the following condition:

$$i + j + k = n$$
$$0 \leq i \leq n, \quad 0 \leq j \leq n, \quad 0 \leq k \leq n$$

Figure B.7 shows an schematic view of the distribution of the control points and the parameter values for $n = 3$.

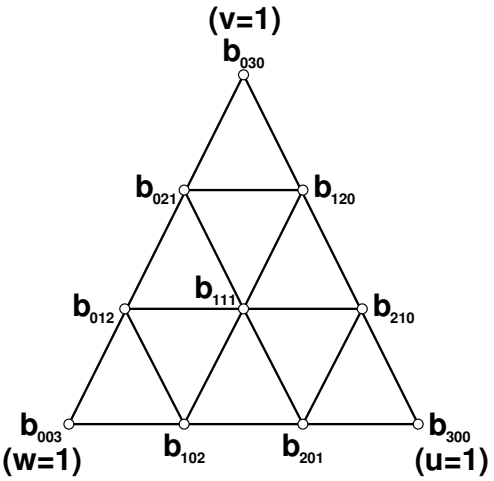


Figure B.7 Control point net layout and parametric domain for a third degree TBP

Points on the patch can be computed using the Bernstein polynomials:

$$B^n_{i,j,k}(u, v, w) = \frac{n!}{i! \cdot j! \cdot k!} \cdot u^i \cdot v^j \cdot w^k \tag{B.22}$$

calling λ the three-tuple (i, j, k) and τ the three-tuple (u, v, w), the expression to evaluate a n-degree TBP is the following [Far86]:

$$\mathbf{b}^n(\tau) = \sum_{|\lambda|=n} \mathbf{b}_\lambda \cdot B^n_\lambda(\tau) \tag{B.23}$$

where $|\lambda| = n$ stands for all the three-tuples (i, j, k) that verify $i + j + k = n$.

Points on the patch can also be computed using the De Casteljau algorithm. This algorithm recursively interpolates values from the previous step, using the control points as the starting point, and the parametric coordinates of the point

to be computed as coefficients. The general formulation for this algorithm is as follows:

$$\mathbf{b}_{\mathbf{i}}^{r}(\tau) = u \cdot \mathbf{b}_{\mathbf{i}+\mathbf{e1}}^{r-1}(\tau) + v \cdot \mathbf{b}_{\mathbf{i}+\mathbf{e2}}^{r-1}(\tau) + w \cdot \mathbf{b}_{\mathbf{i}+\mathbf{e3}}^{r-1}(\tau) \tag{B.24}$$

For a n-degree patch, the algorithm spends n steps, with r varying from 1 to n. $\mathbf{i}$ stands for the three-tuples (i, j, k) that verify $|\mathbf{i}| = n - r$ for every step of the algorithm, and **e1**, **e2** and **e3** stand for the three-tuples $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$ respectively. At the first step, $\mathbf{b}_{\mathbf{i}}^{0}(\tau) = \mathbf{b}_{\mathbf{i}}$.

For instance, let us see the steps to evaluate a third degree TBP:

First step; $r = 1$

$$\begin{aligned}
\mathbf{b}_{0,0,2}^{1}(\tau) &= u \cdot \mathbf{b}_{102} + v \cdot \mathbf{b}_{012} + w \cdot \mathbf{b}_{003} \\
\mathbf{b}_{0,1,1}^{1}(\tau) &= u \cdot \mathbf{b}_{111} + v \cdot \mathbf{b}_{021} + w \cdot \mathbf{b}_{012} \\
\mathbf{b}_{1,0,1}^{1}(\tau) &= u \cdot \mathbf{b}_{201} + v \cdot \mathbf{b}_{111} + w \cdot \mathbf{b}_{102} \\
\mathbf{b}_{1,1,0}^{1}(\tau) &= u \cdot \mathbf{b}_{210} + v \cdot \mathbf{b}_{120} + w \cdot \mathbf{b}_{111} \\
\mathbf{b}_{0,2,0}^{1}(\tau) &= u \cdot \mathbf{b}_{120} + v \cdot \mathbf{b}_{030} + w \cdot \mathbf{b}_{021} \\
\mathbf{b}_{2,0,0}^{1}(\tau) &= u \cdot \mathbf{b}_{300} + v \cdot \mathbf{b}_{210} + w \cdot \mathbf{b}_{201}
\end{aligned}$$

Second step; $r = 2$

$$\begin{aligned}
\mathbf{b}_{0,0,1}^{2}(\tau) &= u \cdot \mathbf{b}_{1,0,1}^{1}(\tau) + v \cdot \mathbf{b}_{0,1,1}^{1}(\tau) + w \cdot \mathbf{b}_{0,0,2}^{1}(\tau) \\
\mathbf{b}_{0,1,0}^{2}(\tau) &= u \cdot \mathbf{b}_{1,1,0}^{1}(\tau) + v \cdot \mathbf{b}_{0,2,0}^{1}(\tau) + w \cdot \mathbf{b}_{0,1,1}^{1}(\tau) \\
\mathbf{b}_{1,0,0}^{2}(\tau) &= u \cdot \mathbf{b}_{2,0,0}^{1}(\tau) + v \cdot \mathbf{b}_{1,1,0}^{1}(\tau) + w \cdot \mathbf{b}_{1,0,1}^{1}(\tau)
\end{aligned}$$

Third step; $r = 3$

$$\mathbf{b}_{0,0,0}^{3}(\tau) = u \cdot \mathbf{b}_{1,0,0}^{2}(\tau) + v \cdot \mathbf{b}_{0,1,0}^{2}(\tau) + w \cdot \mathbf{b}_{0,0,1}^{2}(\tau)$$

Figure B.8 shows an example of the points that are computed in each step of the De Casteljau algorithm.

A third degree TBP can be seen in figure B.9. This kind of patches have been directly implemented in hardware [ATI01, VPBM01], and have been used in the development of many interpolation schemes for the creation of free-form surfaces [Far86, Pip87, SS87, SS91, Far93, Loo94, KPS95, PBP02, KNZ03,

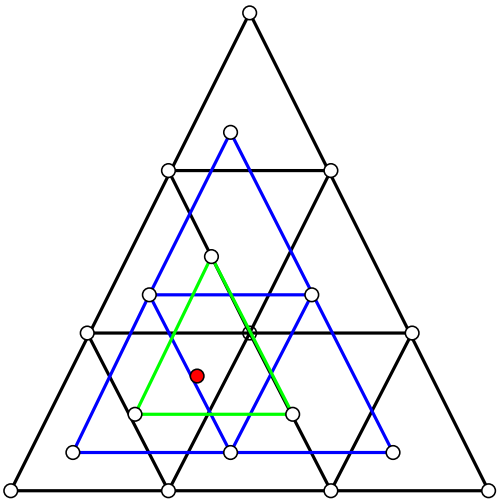


Figure B.8 Parametric distribution of the points computed in each step of the De Casteljau algorithm. In blue: first step. In green: second step. In red: resulting point

HB03, BS05, BRS05], as they are quite simple and easy to use. The triangle defined by b_{003}, b_{030} y b_{300}, in red in figure B.9, will be referred to as the ***base triangle*** of the patch.

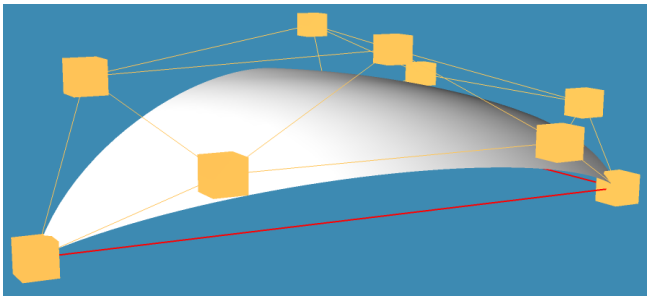


Figure B.9 Third degree triangular Bézier patch

The properties of this kind of patches are well-known and can be studied through the references [Far86, Far93, PBP02]. However, subdivision is especially important in this work, as it will be seen later.

A TBP can be easily subdivided using the De Casteljau algorithm [Sei89, PBP02, Gol03]. To obtain the control point c_{ijk} for a subpatch $\mathbf{c}^n$ defined by the parametric subdomain ($\mathbf{p}$, $\mathbf{q}$, $\mathbf{r}$) (figure B.10), it is just necessary to apply i

De Casteljau steps using the parametric coordinates of $\mathbf{p}$, then j De Casteljau steps using the parametric coordinates of $\mathbf{q}$, and finally k De Casteljau steps with the parametric coordinates of $\mathbf{r}$.

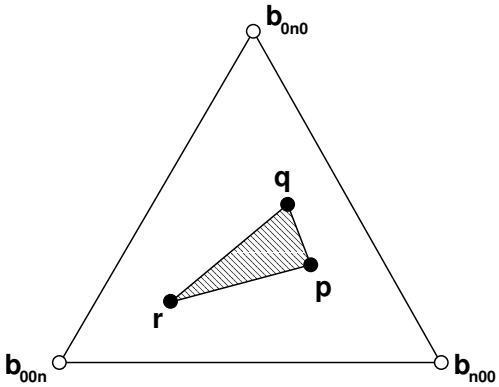


Figure B.10 Parametric representation of a subpatch

The triangles defined by the vertices of the subpatches from a regular subdivision of a TBP constitute a polyhedral approximation of the patch. Figure B.11 illustrates this point.

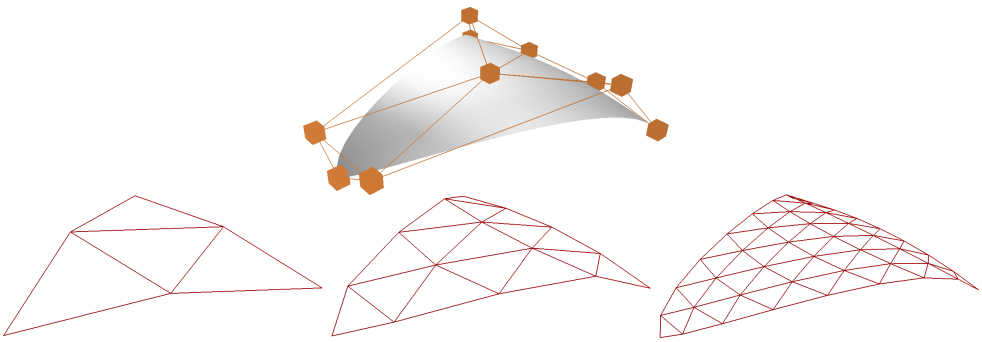


Figure B.11 Polyhedral approximations of a TBP obtained by subdivision

Li et al. [LKL+02] developed a criterion to determine in advance the number of subdivisions to be applied in order to obtain a polyhedral approximation of a TBP, depending on an accuracy threshold. However, the algorithm suffers from numerical unstability, and was discarded for its use in this work.

In every solid modeling system, computing the intersection of the surfaces that bound the solids is an essential step to evaluate boolean operations. There are two main approaches to solve this problem:

- **Analytic methods**, which try to obtain an exact mathematical expression for the intersection. However, these methods are suitable only for simple surfaces [HKE99]; trying to achieve an analytic expression for the intersection of parametric patches results in a too complex equation system.

- **Numeric methods** use different strategies to obtain an approximation of the intersection curve. They can be subdivided into three main groups: tracing methods [AMY96, LCK02, PMKM04], *subdivision methods* [HEFS85, Hoh92, HMPY97] and *lattice based methods* [Pat93]. There also exist some approaches which cannot be included in any of these groups [LJCW04, CKKK97].

To our knowledge, there is no specific work about computing the intersection of TBPs, therefore algorithms to intersect and trim this kind of patches have been developed using a levels of detail hierarchy [GRF06]. Each patch has an associated hierarchy, defined by the number of levels (***n***), and the number of points regularly evaluated per side of the parametric domain (***lod***). Figure B.12 shows the parametric representation of a hierarchy with $n = 2$ and $lod = 1$, and figure B.13 contains an example of a patch and a hierarchy with $n = 3$ and $lod = 1$. Each node of the hierarchy stores all the relevant information about a subpatch, like its control point net or its bounding box.

The algorithms to intersect and trim the patches using this data structure will be explained in detail later.

B.4.2 Free-form cell using triangular Bézier patches

Free-form cells (ffcs) were formally defined in section B.3.1. The concept of ffc is now particularized to three dimensions using TBPs as free-form bounding elements [GRF03, GRF04a].

The starting point to build ffcs is an ***initial triangulation*** that approximates the surface of a free-form solid. Then, an interpolation scheme is applied over these triangles to obtain a set of TBPs that bounds the free-form solid; in

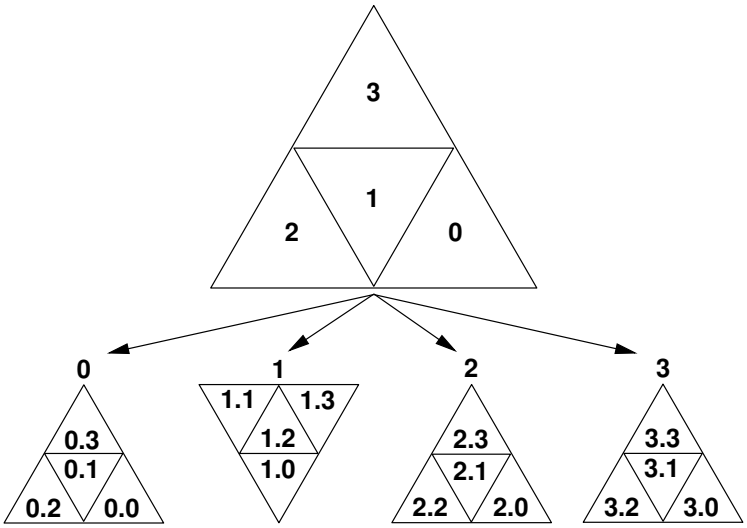


Figure B.12 Parametric subdivision example. $lod = 1$

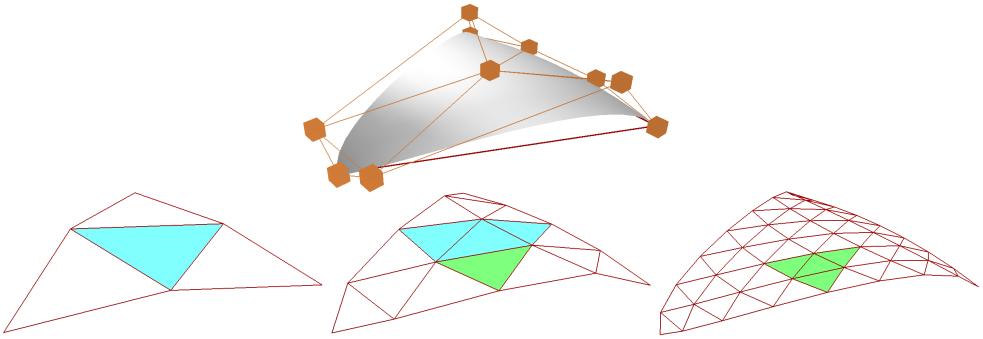


Figure B.13 A patch and its hierarchy. $n = 3; lod = 1$

this work, the interpolation scheme used is the one behind the TRUFORM© technology implemented in some graphics cards [ATI01, VPBM01].

In order to determine the planar elements that bound the ffc, the concept of plane associated to two triangles is introduced here:

> *Given two triangles t_1 y t_2 from an initial triangulation of a solid that share an edge* $\mathbf{e}$*, the* ***plane associated to t_1 and t_2****, namely $\pi(t_1, t_2)$, is defined as the plane that contains the edge* $\mathbf{e}$ *and whose normal vector is perpendicular to the weighted average of the normal vectors from t_1 and t_2.*

Once the concept of plane associated to two triangles has been established, the definition of ffc using TBPs as free-form bounding elements is as follows:

> *Let* $\mathbf{S}$ *be a free-form solid bounded by a set of triangular Bézier patches* $\mathbf{P}$*, and let* $\mathbf{b}^n$ *be a patch from* $\mathbf{P}$*, whose base triangle is* t_0*. Let* $\mathbf{b}_1^n$*,* $\mathbf{b}_2^n$ *and* $\mathbf{b}_3^n$ *be the triangular Bézier patches from* $\mathbf{P}$ *that share an edge with* $\mathbf{b}^n$*, and* t_1*,* t_2 *and* t_3 *their base triangles. A* ***free-form cell*** *of the ESC that represents the solid* $\mathbf{S}$ *is defined as the intersection of the half-spaces determined by the plane that contains* t_0*, the planes* $\pi(t_0, t_1)$*,* $\pi(t_0, t_2)$ *and* $\pi(t_0, t_3)$*, and the patch* $\mathbf{b}^n$ *and all its neighbouring patches that are partially or totally included in the half-spaces determined by the planes.*

In the definition above, two patches are neighbours if they share at least one vertex. Figure B.14 shows an example of ffc; the neighbouring patches are drawn in wireframe mode to distinguish them better, and the associated planes are drawn as four-edged polygons. The base triangle of the patch is also named the ***base triangle of the ffc***, and is drawn in red in figure B.14.

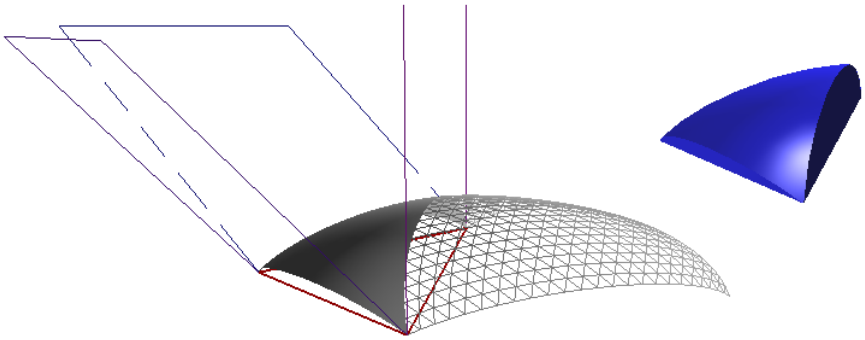


Figure B.14 Ffc bounded by triangular Bézier patches. Left: bounding elements. Right: enclosed volume

Different weights can be used to compute the weighted average of the normal vectors; the use of the triangle area as averaging weight slightly reduces the number of patches that are included as bounding elements of a ffc.

The sign of a ffc depends on its position with respect to the plane that contains the base triangle of the main patch. Positive cells are placed on the positive half-space defined by the plane, while negative cells are placed on the

negative half-space. If the patches go through the plane, then two different cells are considered (figure B.15).

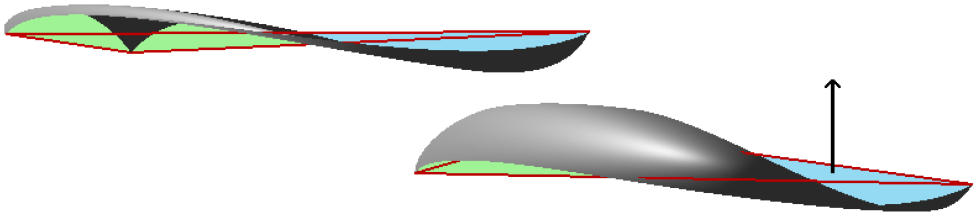


Figure B.15 Two ffcs that share their bounding elements

B.4.3 Building the ESC for a free-form solid

Once the ffc bounded with TBPs has been defined, it is possible to buid an ESC for a free-form solid bounded by a set of this kind of patches. The starting point is an initial triangulation that approximates the surface of the free-form solid, together with a set of TBPs obtained from that triangulation by applying an interpolation scheme.

The extended cell creation is as follows:

- One original simplex is created for each triangle of the initial triangulation. The simplex is defined by the three vertices of the triangle, and an origin point **O** shared by all the original simplices.

- One or two ffcs are created for each triangle of the initial triangulation. The ffcs will be bounded by the patches and planes associated to that triangle just as explained above.

The coefficient associated to each extended cell is computed as described before: the coefficient for each simplex is the sign of its signed volume, while the coefficient for each ffc is the sign of the half-space defined by the plane that contains the base triangle in which the cell is contained. Figure B.16 shows an original simplex and a ffc for a triangle.

The ESC formed by the extended cells computed in this way is normal. Figure B.17 shows how the volume of the positive (in blue) and negative (in red) cells is combined step by step to obtain the represented solid.

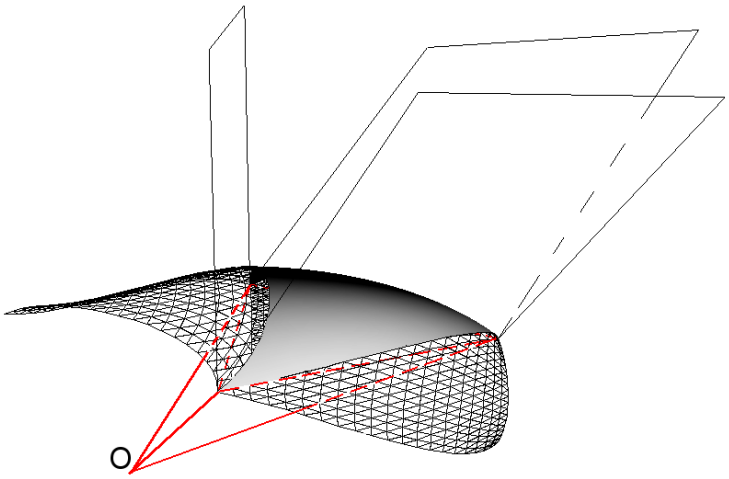


Figure B.16 Original simplex and ffc

B.5 Point in solid test

The point in solid test is an essential operation in Solid Modeling because many algorithms, like boundary evaluation, ray-casting visualization or conversion into other representations, rely on it. It is therefore necessary to develop a robust and efficient algorithm to perform this operation.

The problem of determining the position of a point with respect to a free-form solid can be stated as:

> *Given a point and a free-form solid, determine whether the point is inside, outside or on the boundary of the solid.*

Classic algorithms to solve this problem are based on the Jordan curve theorem [Jor87, Tve80]. This theorem states that a closed line homeomorphic to a circumference in $\mathbb{R}^2$ divides the plane into two disjoint subsets, the *inside* and the *outside*, bounded by the curve. As a result, a half-line starting from a point will intersect the curve an odd number of times if the point is in the inside subset, and an even number of times otherwise (figure B.18). In three dimensions, the same principle can be applied: a closed surface homeomorphic to a sphere in $\mathbb{R}^3$ divides the space into two disjoint subsets, the *inside* and the *outside*, bounded by the surface. Therefore a half-line starting from a point will intersect the surface an odd number of times if the point is in the inside subset, and an even number of times otherwise.

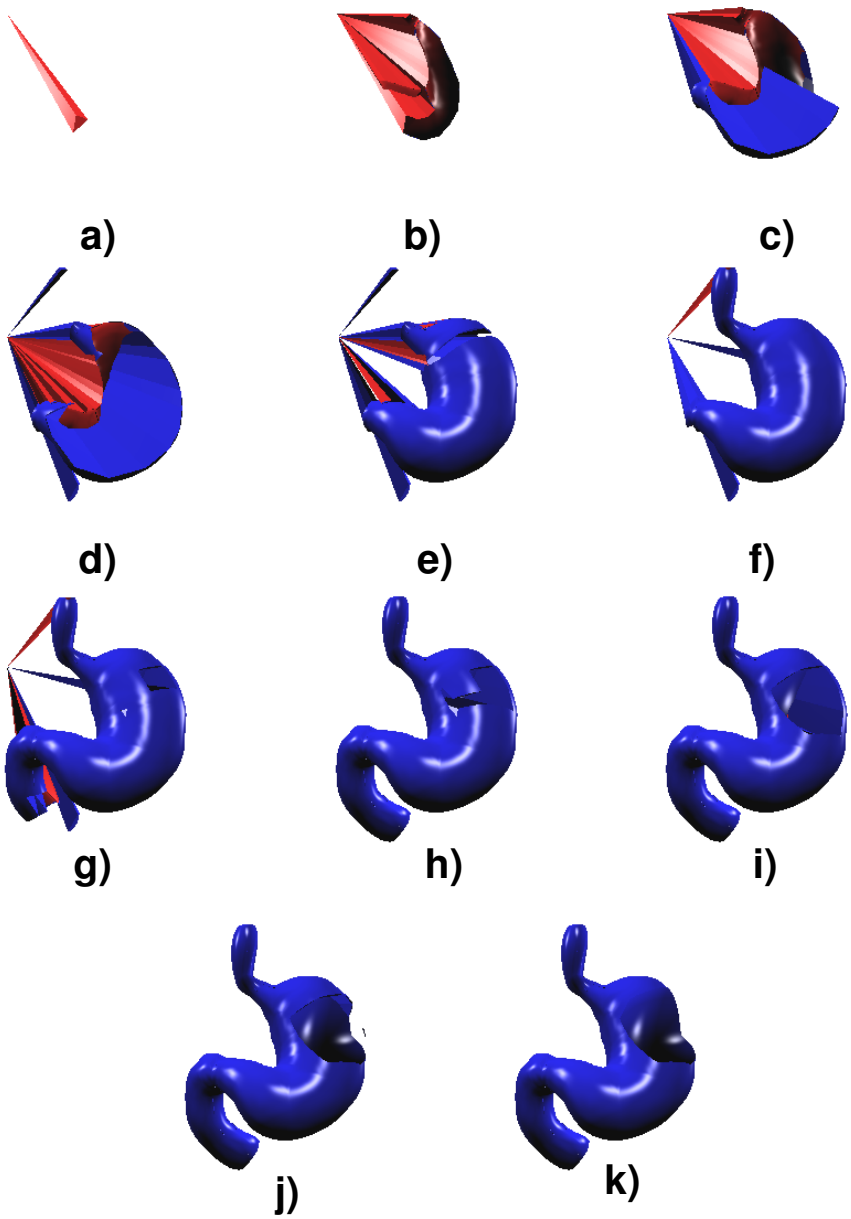


Figure B.17 Steps in the creation of a volume from an ESC

However, using this theorem to check the inclusion of a point in a free-form solid frequently implies solving complex equation systems [HPY96, PS85,

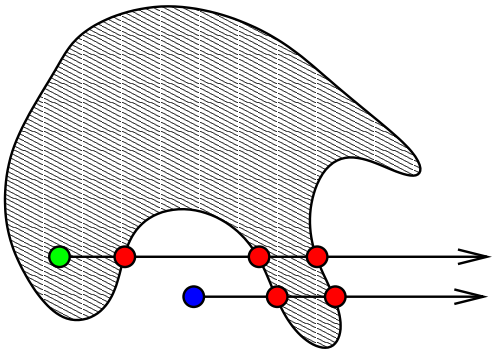


Figure B.18 Verifying the Jordan curve theorem for a two dimensional free-form solid

YS91], with a lot of special cases [Hui97] and numerical unstability [KGMM97, KM96].

B.5.1 General algorithm for point in solid testing

The ESC model allows the implementation of a simple and robust point in solid test algorithm, based on the decomposition of the problem into simpler tests performed on the extended cells that form the chain that represents the solid [RF97, RF99b, GRF04a]. It is based on the concepts of function and solid associated to an ESC given in section B.3.2.

To check the inclusion of a point in a free-form solid modeled as a normal ESC it is just necessary to check the inclusion of the point in each of the extended cells, and add the coefficients of the cells that contain the point. If the sum equals 1, the point is inside the solid. See algorithm B.1. Special cases will be dealt with later.

It is therefore necessary to study how the inclusion of a point in an extended cell can be computed.

B.5.2 Point in simplex test

Determining whether a point is inside a simplex is quite easy. The barycentric coordinates of the point with respect to the vertices of the simplex help a lot in this process. In three dimensions, the barycentric coordinates of a point

```
Algorithm inclusionESC ( escSolid, P )

  accum ← 0

  Foreach extended cell E in escSolid do
    If P ∈ E then
      accum ← accum + sign ( E )

  If accum = 1 then
    Return inside

  Else
    Return outside
```

Algorithm B.1 General algorithm to compute the point in solid test

$\mathbf{P} = (x_p, y_p, z_p)$ with respect to a simplex $\{\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3, \mathbf{T}_4\}$ are computed as explained in section B.3.1.

As the barycentric coordinates are the coefficients of the linear interpolation:

$$\mathbf{P} = t \cdot \mathbf{T}_1 + u \cdot \mathbf{T}_2 + v \cdot \mathbf{T}_3 + w \cdot \mathbf{T}_4 \tag{B.25}$$

their values determine the position of the point with respect to the simplex:

- If all the barycentric coordinates are greater than zero, the point is inside the simplex.
- If at least one of the barycentric coordinates is negative, the point is outside the simplex.
- If one of the barycentric coordinates equals zero and the other ones are positive, the point is on the face defined by the vertices whose corresponding coordinates are positive.

- If two of the barycentric coordinates equal zero and the other ones are positive, the point is on the edge defined by the vertices whose corresponding coordinates are positive.

- If three of the barycentric coordinates equal zero, the point is equal to the only vertex whose corresponding coordinate is positive.

If the point is on a face, edge or vertex, and the barycentric coordinate corresponding to the origin of the ESC is positive, then the point is on an original edge or face, or is equal to the origin of the ESC.

B.5.3 Point in ffc test

An algorithm to test the inclusion of a point in a ffc defined using TBPs as stated in section B.4.2 is presented here. This algorithm is based on computing the intersections of a ray cast from the point in the direction of the normal vector of the ffc's base triangle (figure B.19).

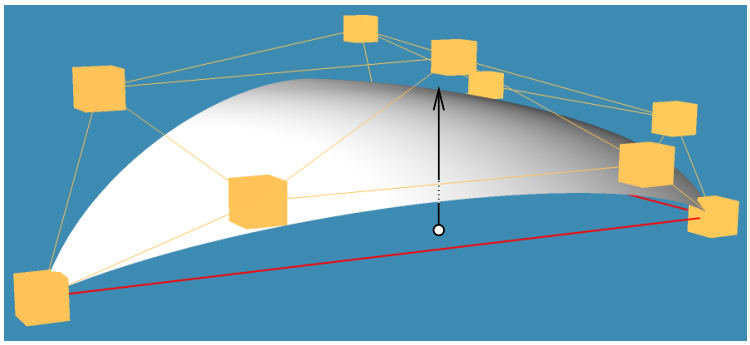


Figure B.19 Ray cast from a point in the direction of the normal vector of an ffc's base triangle

It is not necessary to compute the intersections accurately, but only their number and if the ray is coming inside or outside the ffc. To obtain this information, the levels of detail hierarchy explained in section B.4.1 is used; see algorithm B.2.

The *triangulation* function in algorithm B.2 returns the triangulation stored in a level of the hierarchy that is determined by the *triangulationLevel* for the parametric subdomain defined in *domain*. The *rayTrianglesInt* procedure in the same algoritm computes the set p of intersections of the ray with the set of

```
Algorithm rayTBPInt ( ray, patch )

  triangulationLevel ← 1
  domain ← patch.domain

  Repeat
    t ← triangulation ( patch, triangulationLevel, domain )
    rayTrianglesInt ( ray, t, p, newD )
    domain ← newD

    If p ≠ ∅ then
      triangulationLevel ← triangulationLevel + 1
  Until ( p = ∅ ) or ( triangulationLevel = maxLevel )

  Return p
```

Algorithm B.2 Algorithm to compute the ray-Triangular Bézier Patch intersection

triangles t, together with the parametric subdomains where the intersections have been found. These subdomains are stored in the parameter *newD*.

The intersections are labelled as *inwards* or *outwards* depending on the angle between the ray and the normal vector of the intersected triangle, see figure B.20. The intersections that are outside the bounding elements of the ffc are discarded, and they are replaced with the nearest intersections to the plane that contains the base triangle (figure B.21).

The point is classified as *inside* the ffc if it is placed between an inwards and an outwards intersection; otherwise it is classified as *outside*.

The algorithm can be optimized in the following ways:

- If there is just one outwards intersection with the patch, and the point is between the intersection and the base triangle, it is not necessary to continue, as the point is clearly inside.

- Perform a preliminary test using the planes that bound the ffc together with a plane parallel to the plane that contains the base triangle of the

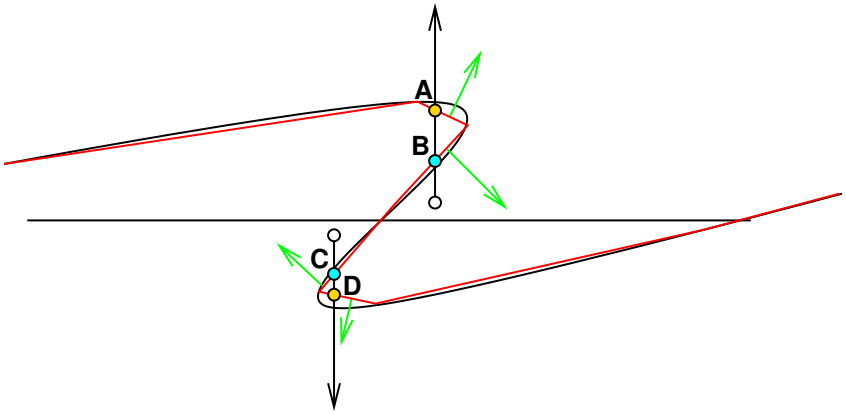


Figure B.20 Inwards (in blue) and outwards (in yellow) intersections

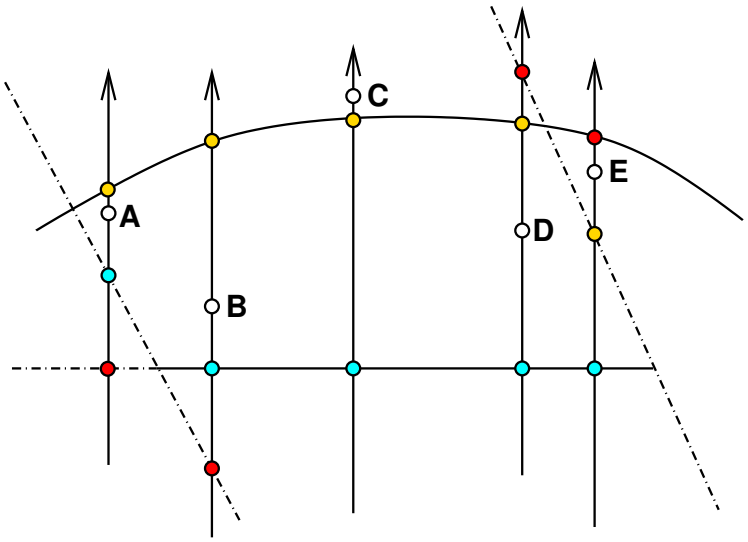


Figure B.21 Schematic view of the point selection process. Discarded intersections are in red; inwards points are in blue, and outwards points are in yellow

ffc and contains the more distant control points of the TBPs (figure B.22).

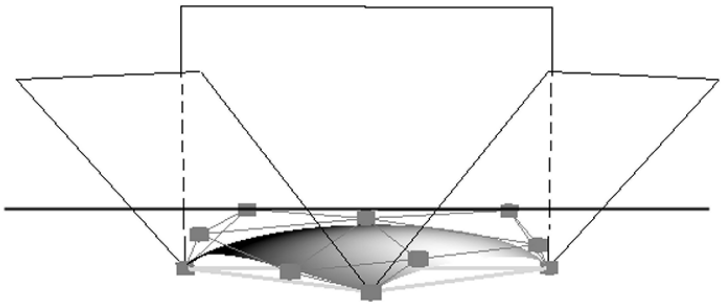


Figure B.22 Planes used in the preliminary test

- If the ray intersects a triangle in a vertex, it is not necessary to continue, as the vertices of the triangles are points on the TBP.

- If the point is equal to one of the intersections with the planes, it is on the boundary of the ffc. If the point is equal to an intersection with a vertex of a triangle, then the point is on the boundary of the solid.

Algorithm B.3 summarizes the described process. P stands for a point, and *cell* stands for a ffc.

B.5.4 Point in solid test for a free-form solid bounded with triangular Bézier patches

Once the algorithms to test the inclusion of a point in a simplex and a ffc have been obtained, the general algorithm B.1 to perform the point in solid test can be detailed to free-form solids bounded with TBPs. The representation of the solid is by means of an ESC as described previously.

The algorithm has very few special cases, and these special cases are easily detected and solved; therefore the algorithm is simple and robust. Figure B.23 presents a portion of an initial triangulation, together with their corresponding simplices and ffcs; the points represent all the special cases. The special cases can be handled as follows:

- If a point is on an original face of a simplex, like point **Q1** in figure B.23, it is shared by two simplices (**T5** and **T6** in the figure). In this

```
Algorithm pointInclusionFfc ( P, cell )

  If ( P is outside cell.planes ) then
    Return OUTSIDE

  r ← ray ( P, cell.base.normal )

  For each tbp ∈ cell.patches do
    intsPatch ← rayTBPIntersection ( r, ptb )
    labelIntersections ( intsPatch )
    ints ← ints ∪ intsPatch

  DiscardIntersections ( ints, cell.planes, r )

  pos ← pointPosition ( P, ints )

  If ( pos = between inwards and outwards ) then
    Return INSIDE

  Else
    Return OUTSIDE
```

Algorithm B.3 Algorithm to compute the inclusion test for a point in a ffc

case, each simplex contributes with $+\frac{1}{2}$ or $-\frac{1}{2}$ to the sum, instead of $+1$ or -1.

- If a point is on an original edge, like point **Q2**, it is shared by many simplices. This problem is solved by adding $+1$ (or -1) just once if there is at least one positive (or negative) simplex sharing the point.

- If a point is on a triangle from the initial triangulation, like point **Q3**, it is shared by a simplex and a ffc (**T6** and ffc6 in the figure). Each cell contributes with $+\frac{1}{2}$ or $-\frac{1}{2}$ to the sum, instead of $+1$ or -1.

- If a point is on an edge of the initial triangulation, like point **Q4**, it is shared by two simplices (**T1** and **T6**) and two ffcs (ffc1 and ffc6). In

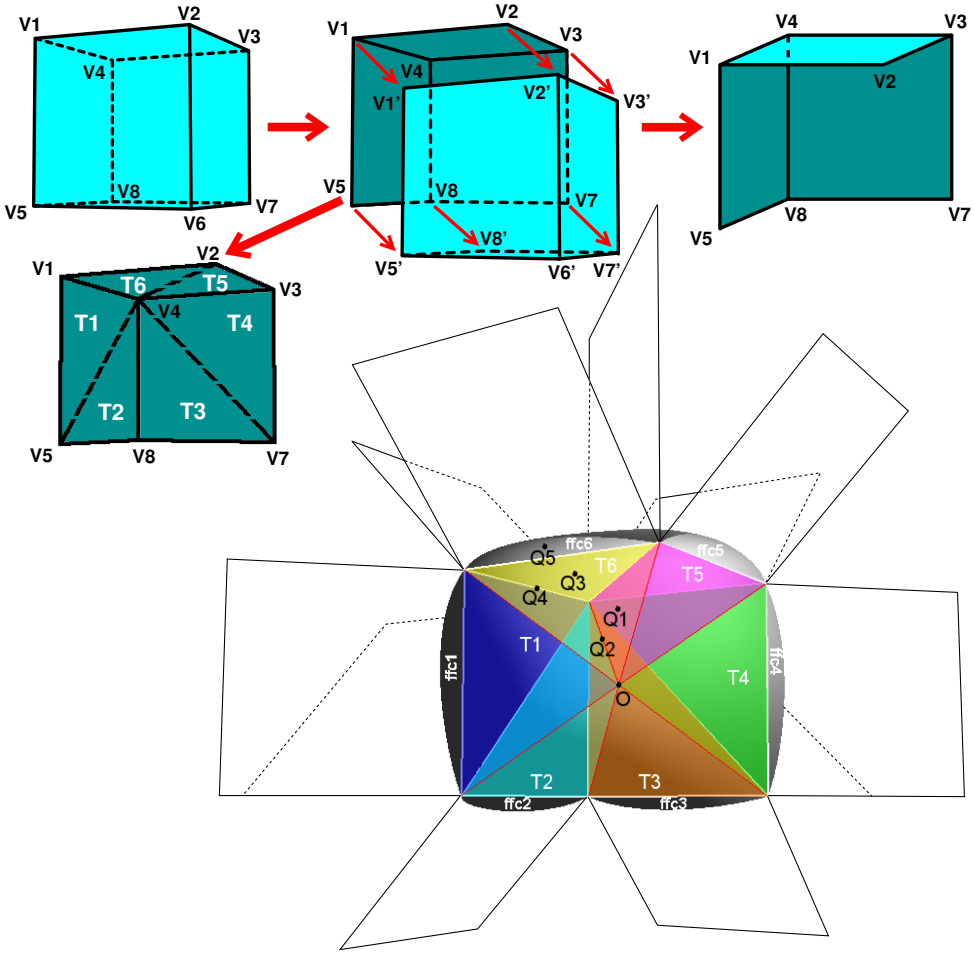


Figure B.23 Up: how the portion of the initial triangulation is obtained. Down, left: portion of the initial triangulation. Down, right: original simplices, free-form cells and special cases. The origin of the chain is a point between the observer and the triangulation

this case, $+1$ or -1 will be added just once only when the simplex and the ffcs associated to the same triangle have the same sign.

- If the point is on a plane that bounds a ffc, it is shared by two ffcs, like **Q5**. In this case, each ffc contributes with $+\frac{1}{2}$ or $-\frac{1}{2}$ to the sum, instead of $+1$ or -1.

Algorithm B.4 describes the proposed algorithm.

```
Algorithm pointInSolidTest ( P, S )

  inclusion ← 0.0

  For each tr ∈ S.triangles do
    tet ← simplex associated to tr
    ffcs ← free-form cells associated to tr

    sT ← sign ( tet )

    For each ffc ∈ ffcs do
      sFfc ← sign ( ffc )
      resFfc ← pointInclusionFfc ( P, ffc )
      inclusion ← inclusion + processResFfc ( resFfc,
                                              sFfc, sT )

    resTet ← pointInclusionSimplex ( P, tet )
    inclusion ← inclusion + processResTet ( resTet, sT )

  If ( inclusion = 1 ) then
    Return INSIDE

  Else
    Return OUTSIDE
```

Algorithm B.4 Algorithm to test the inclusion of a point in a free-form solid

The function *processResFfc* in algorithm B.4 returns the value that has to be added to the *inclusion* value according to the special cases described above and the value returned by the *pointInclusionFfc* algorithm. The function *processResTet* works similarly to *processResFfc*.

As most of the volume of the solids is represented with simplices, the algorithm to test the inclusion of a point in a ffc is called only occasionally. This makes the process very fast, as the algorithm to test the inclusion in a simplex is straightforward.

In every ESC, the number of simplices and ffcs is linearly dependendent to the number of triangles in the original triangulation. Therefore, the computational complexity of the main algorithm is lineal with respect to the number

of triangles in the original triangulation that approximates the boundary of the solid.

The algorithm has been tested with the models in figure B.24, using the interpolation scheme from Vlachos et al. to obtain the TBPs that bound the free-form solids [ATI01, VPBM01]. Table B.1 shows information about the models.

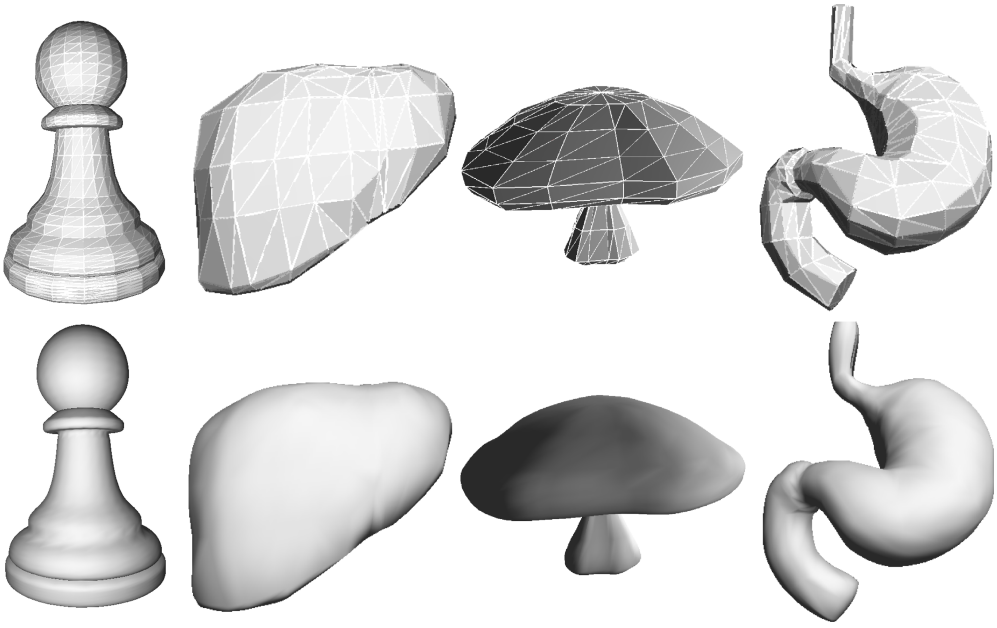


Figure B.24 Up: initial triangulations. Down: free-form solids

Mesh	Vertices	Triangles
Pawn	1202	2399
Liver	139	274
Mushroom	226	448
Stomach	269	534

Table B.1 Vertex and face information from the models in figure B.24

To avoid numerical precision problems, a threshold has been used as usual in the implementation of the algorithms. Tests have been performed using a regularly distributed point cloud contained in the bounding box of each solid.

Tables B.2, B.3 and B.4 show the number of intersections ray-TBP that should be computed applying the Jordan curve theorem and the described algorithm. Note that the number of intersections that need to be computed is dramatically reduced using the new algorithm, as many points are outside the ffcs.

Number of points	Int. Jordan	Int./Pt. Jordan	Int. ESC	Int./Pt. ESC
1000 (10×10×10)	611	0.611	**58**	**0.058**
1728 (12×12×12)	1116	0.646	**229**	**0.133**
2744 (14×14×14)	1767	0.644	**378**	**0.138**
3375 (15×15×15)	2208	0.654	**399**	**0.118**
4096 (16×16×16)	2697	0.658	**609**	**0.149**
4913 (17×17×17)	3251	0.662	**610**	**0.124**

Table B.2 Results of the inclusion tests with the stomach model

Number of points	Int. Jordan	Int./Pt. Jordan	Int. ESC	Int./Pt. ESC
1000 (10×10×10)	816	0.816	**147**	**0.147**
1728 (12×12×12)	1466	0.848	**303**	**0.175**
2744 (14×14×14)	2349	0.856	**566**	**0.206**
3375 (15×15×15)	2872	0.851	**681**	**0.202**
4096 (16×16×16)	3558	0.869	**741**	**0.181**
4913 (17×17×17)	4321	0.880	**940**	**0.191**

Table B.3 Results of the inclusion tests with the liver model

Number of points	Int. Jordan	Int./Pt. Jordan	Int. ESC	Int./Pt. ESC
1000 (10×10×10)	621	0.621	**30**	**0.030**
1728 (12×12×12)	1089	0.630	**130**	**0.075**
2744 (14×14×14)	1781	0.649	**138**	**0.050**
3375 (15×15×15)	2212	0.655	**136**	**0.040**
4096 (16×16×16)	2715	0.663	**166**	**0.041**
4913 (17×17×17)	3296	0.671	**136**	**0.028**

Table B.4 Results of the inclusion tests with the chess pawn model

Efficiency tests have been performed using a Pentium IV 2.4 GHz PC with 512 MB of RAM memory. Table B.5 shows the time taken to load the data, build the extended cells that form the ESC and create the levels of detail hierarchy for all the TBPs that bound the solid. As it can be seen, the extended cells building process is very fast, whereas the creation of the levels of detail

hierarchies takes more or less time depending on the number of levels in the hierarchies (in these tests, each level of the hierarchies contained 8 parametric subdivisions).

Mesh	Load	ESC Creation	T. level 1	T. levels 2 and 3
Stomach	0.300	0.076	0.360	29.336
Liver	0.083	0.040	0.180	14.696
Mushroom	0.213	0.063	0.300	24.420

Table B.5 Time taken in the preliminary tasks (in seconds)

Table B.6 shows interesting data about the use of TBPs in the building of the ffcs. It can be seen that usually two or three patches take part in the definition of each ffc.

Mesh	Patches	ffcs	Patches/ffc
Stomach	534	818	2.587
Liver	274	422	2.600
Mushroom	448	694	2.687

Table B.6 Use of TBPs in the building of ffcs

Table B.7 shows time information for a test with a regularly distributed point cloud with 100×100×100 points. Figure B.25 shows the result of these tests. It can be seen that the average number of rays that have to be cast per point is very low, as well as the average time to compute the inclusion of a point.

Mesh	Total time	Time/Pt.	Rays	Rays/Pt.
Stomach	19m 31s	0.001171s	64050	0.064050
Liver	8m 32s	0.000512s	142886	0.142886
Mushroom	15m 19s	0.000919s	64156	0.064156

Table B.7 Times for a test with 100×100×100 points

B.6 Boolean operations

Boolean operations are a key element in any solid modeling system, as they allow the users to create and/or edit solid models. As explained in section B.3.3, the result of a boolean operation between two solids modeled with ESCs

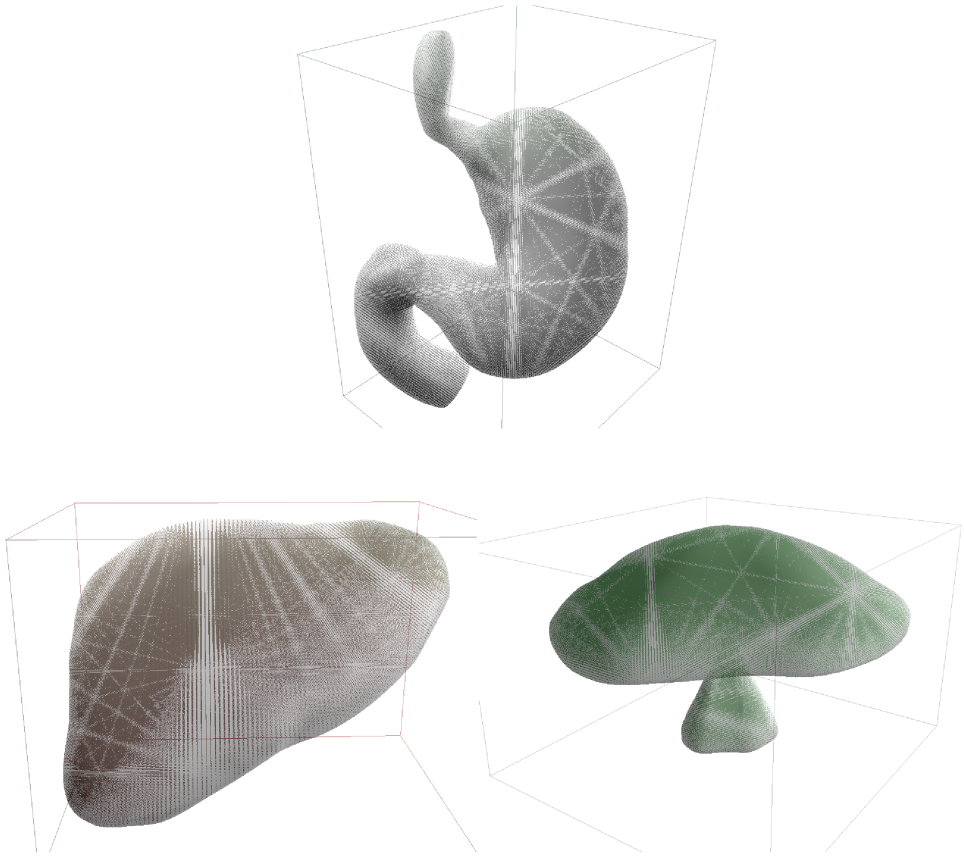


Figure B.25 Results of the inclusion test with a set of 100×100×100 points

can be represented in another ESC. In practice, it is desirable to minimize the number of extended cells that intersect, so the origin of the ESCs should be placed far from each other, like in figure B.26

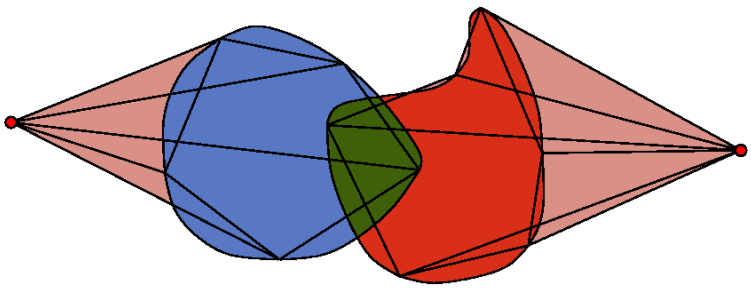


Figure B.26 An example of desirable position of the origin of the ESCs for a boolean operation

As the result of the boolean operations is modeled as a new ESC, it can be used as an operand for a new operation, and all the algorithms proposed for ESCs can be applied on it. The ESC model is therefore a closed representation scheme that allows a CSG-like modeling process as shown in figure B.27.

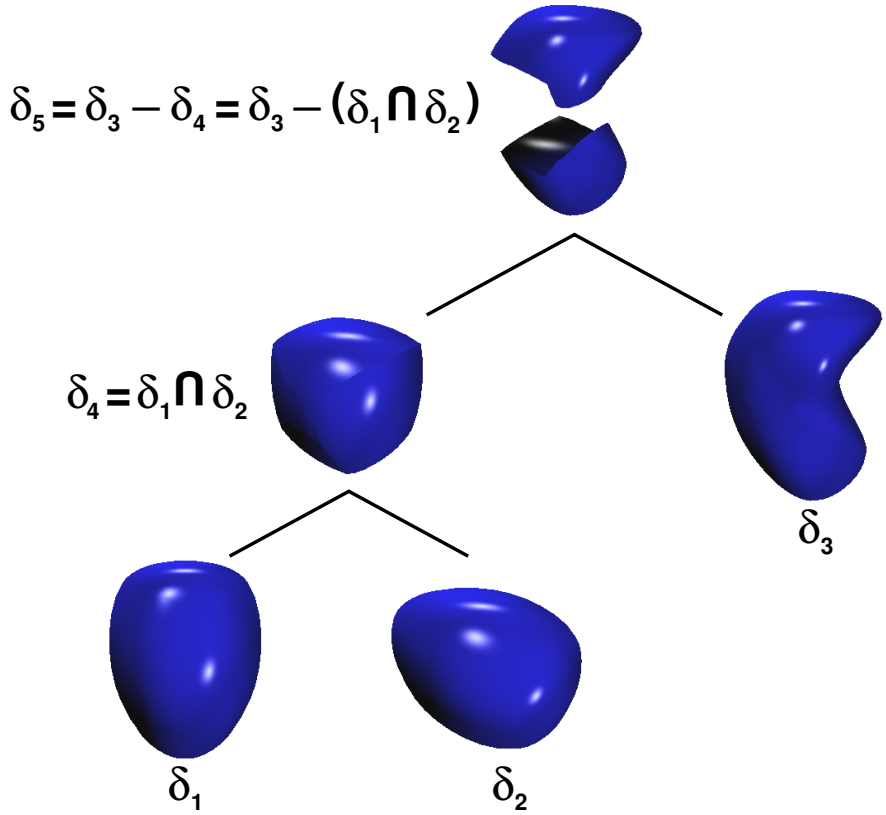


Figure B.27 ESC computation for a CSG solid

B.6.1 Visualization of non-evaluated boolean operations

The computation of $ExCell$ $(E \cap E')$ can become a difficult task, especially when one of the extended cells is a ffc. However, it is possible to delay the evaluation of this operation, as it can be possible that the result of the intersection of two cells does not contribute to the volumen of the final result. Considering this, the possibility of visualizing the result of a boolean operation without effectively evaluating it becomes a desirable advantage in the modeling process.

The relevance of the visualization of non-evaluated boolean operations can be noticed through the amount of papers that give solutions to this problem [HG96, SL98, ET00, SLJ02, GKMV03, HR05, GRF05]. It is clear that having a preview of the result of an operation is very useful to the users of any modeling system.

Instead of computing the result of $ExCell$ $(E \cap E')$, it is possible to combine the result of the algorithms over E and E'. A visualization algorithm has been developed in this way, using ray casting [Gla93]. This algorithm uses intensively the ray-TBP intersection algorithm as described before, and is summarized in algorithm B.5.

```
Algorithm visualizeESC ( chain, image )

  For each pixel p ∈ image do
    ray ← ray that goes through p
    ints ← ∅

    For each cell E ∈ chain do
      ints.orderedInsertion ( ray.intersections ( E ) )

    If ints ≠ ∅ then
      P ← ints.first
      inside ← inclusionESC ( chain, P )

      While ( not inside ) and ( P ≠ ints.last ) do
        P ← ints.next
        inside ← inclusionESC ( chain, P )

      If inside then
        p.color ← computeColor ( P.material )

      Else
        p.color ← background color

    Else
      p.color ← background color
```

Algorithm B.5 Algorithm to visualize a free-form solid modeled with an ESC

Figures B.28 and B.29 show some examples of visualization of non-evaluated boolean operations between free-form solids using the algorithm described above.

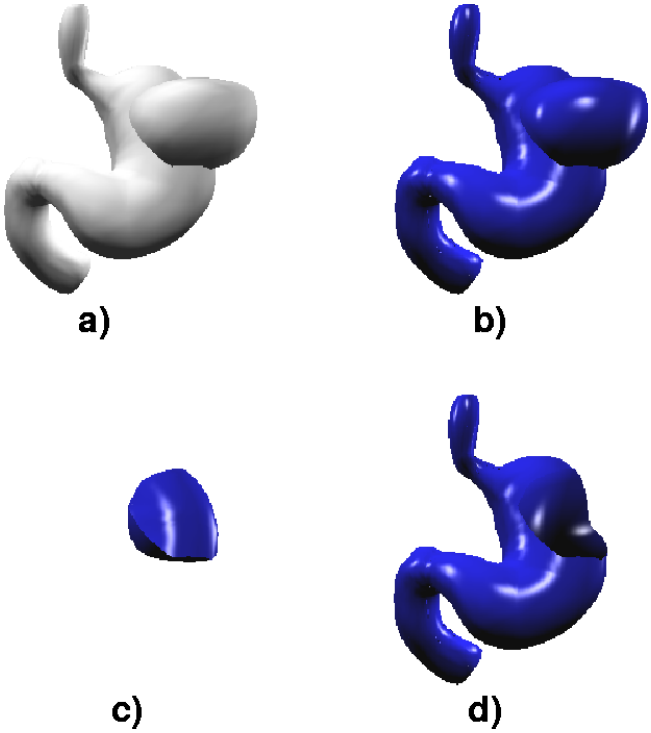


Figure B.28 a) Two free-form solids. b) Union. c) Intersection. d) Difference

The ray-casting algorithm can be adapted to visualize not only a complete ESC, but also a portion of it, by indicating the indices for the starting and ending cells of the portion. This is how figure B.17 was created.

B.6.2 Boundary evaluation

In CAD/CAM environments it is very important to determine the boundary of a solid modeled using boolean operations. This task can be especially difficult when the primitives used are free-form. Robustness and accuracy problems are usual in these situations [For97, KGMM97, KCF+04].

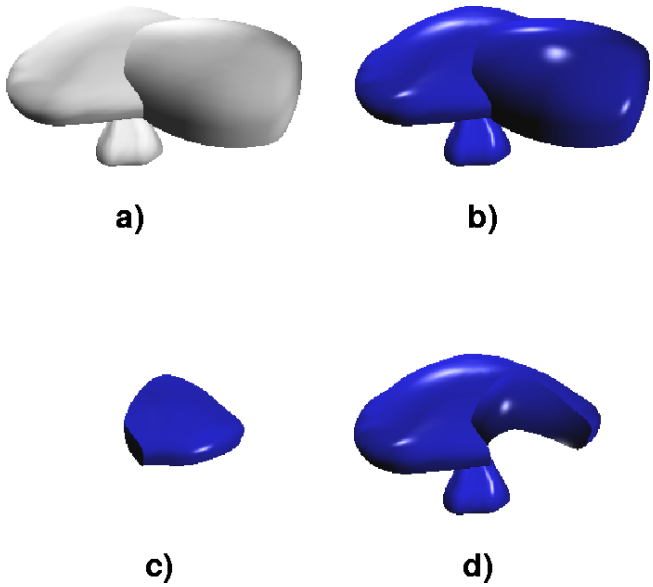


Figure B.29 a) Two free-form solids. b) Union. c) Intersection. d) Difference

Intersection of triangular Bézier patches

The first step in a boundary evaluation process is determining the intersection curves among the surfaces that bound the solids that are being combined in a boolean operation. Although there exist many references in surface intersection [AMY96, GK97, LCK02, LJCW04, Muk05], to our knowledge there were no specific works on the intersection of TBPs, so an algorithm to intersect this kind of patches using the levels of detail hierarchy already implemented was developed [GRF06].

Following [Key00], the TBPs are considered in ***general position***, i.e. their position is such that an infinitesimal perturbation on the input data will not change the branching decisions in the algorithms.

The proposed algorithm takes as starting point the levels of detail hierarchies from the patches. Then successive refinement steps through the hierarchies are taken, checking the bounding boxes of the nodes until two leaf nodes whose triangles can probably intersect are reached. The intersection is then computed between the triangles at the leaf nodes, and the resulting segments form a piecewise linear approximation of the intersection curve. See algorithm B.6.

```
Algorithm intersectionTBPs ( P1, P2, nMax )

  If P1.BBox ∩ P2.BBox ≠ ∅ then
    If ( P1.level = nMax ) and ( P2.level = nMax ) then
      i ← intersectTriangles ( P1.base, P2.base )
      addSegments ( i )

    Else
      If ( P1.level ≠ nMax ) and ( P2.level ≠ nMax ) then
        Pm ← patch with smaller bounding box
        PM ← patch with bigger bounding box

        For each subpatch SP ∈ PM.children do
          intersectionTBPs ( SP, Pm, nMax )

      Else
        If P1.nivel = nMax then
          For each subpatch SP ∈ P2.children do
            intersectionTBPs ( P1, SP, nMax )

        Else
          For each subpatch SP ∈ P1.children do
            intersectionTBPs ( SP, P2, nMax )

  Else
    Return ∅
```

Algorithm B.6 Algorithm to compute the intersection between two TBPs

The triangle intersection algorithm does not differ from the classic ones in the references [Möl97]. The segments from the intersections have to be ordered to detect closed components and loops; this is done in the *addSegments* routine, described in algorithm B.7.

In algorithm B.7, the segments that may create loops are stored separately. These are segments that share an end with more than one segment, or segments that complete a closed component (figure B.30).

```
Algorithm addSegments ( i )

  If there are no stored segments
    store i

  Else
    If ∃ s ∈ segments / s ∩ i ≠ ∅ then
      take out s from segments
      p ← intersection ( i, s )
      {i1, i2} ← divide ( i, p )
      {s1, s2} ← divide ( s, p )
      addSegments ( i1, i2, s1, s2 )

    Else
      If ∃ s ∈ segments / s links with i then
        If there may appear loops then
          store i separately

        Else
          store i linked with s
          reorder segments

      Else
        store i
```

Algorithm B.7 Algorithm to insert in order the intersection segments

The segments are reordered in algorithm B.7 when necessary to make all the segments be oriented in the same direction (figure B.31). This is to make the component detection process easier.

Once the segments are computed and ordered, the component detection process (algorithm B.8) divides the intersection curves into components. A ***component*** is formed by a sequence of connected segments that can be open or closed, such that an ending point of a segment is equal to an ending point of the following segment, and no ending point is shared by more than two segments. In algorithm B.8, the *checkComponents* module looks for pairs of segment sequences whose ending points coincide and can be joined into a bigger one; if so, the sequences are joined, and the search continues until no more sequences can be joined.

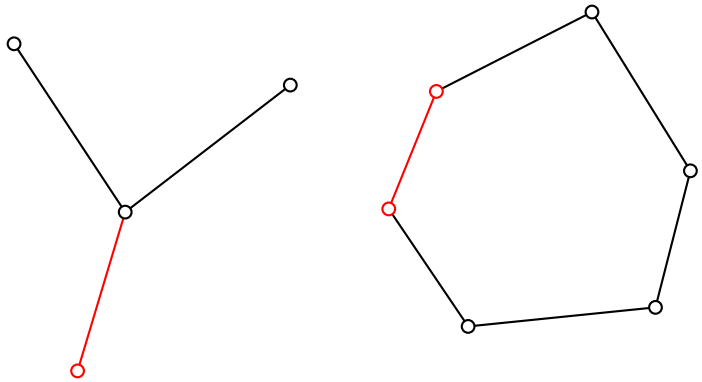


Figure B.30 Segments that are stored separately in algorithm B.7 (in red)

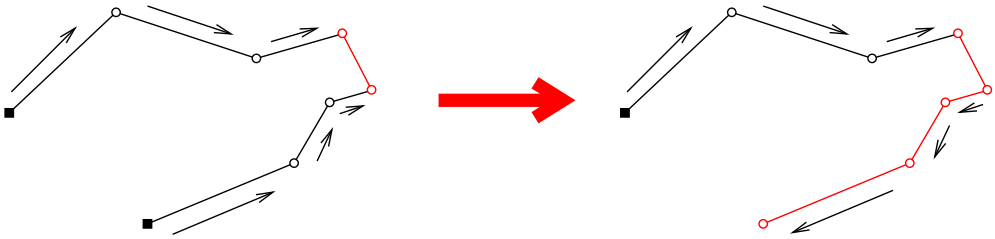


Figure B.31 Reordering of a segment chain after a new segment is added. Left: the new segment (in red) induces the reordering process. Right: result after the reordering

The final result of this process is a set of components that divide the TBPs into closed regions.

Trimming of triangular Bézier patches

The second step of the boundary evaluation process is the patch trimming operation. It consist of deleting some of the regions bounded by the components obtained in the previous step, following a trimming criterion. This criterion is defined by some kind of spatial partition, such as the ones determined by a plane or a solid. The case of trimming a TBP using a solid bounded by

```
Algorithm buildComponents ( segments )

  c ← newComponent ( segments.first )

  While ( segments ≠ ∅ ) do
    s ← segments.next

    If s links with c then
      join s to c

    Else
      storeComponent ( c, components )
      c ← newComponent ( s )

  s ← segmentsStoredSeparatedly.first

  While ( segmentsStoredSeparatedly ≠ ∅ ) do
    For each c ∈ components do
      If s links with c then
        join a copy of s to c

        If a loop appears then
          c2 ← cutLoop ( c )
          storeComponent ( c2, components )

    s ← segmentsStoredSeparatedly.next

  checkComponents ( components )
```

Algorithm B.8 Algorithm to detect components in the intersection of two TBPs

TBPs is detailed here, as it is the most complex situation that can appear. The trimming process includes the following steps (figure B.32):

a) Compute the intersection curves and its components between the patch and all the patches that bound the solid, as described before.

b) Combine the components from all the patches and the original boundaries of the patch. In this step, the intersection points among the curves

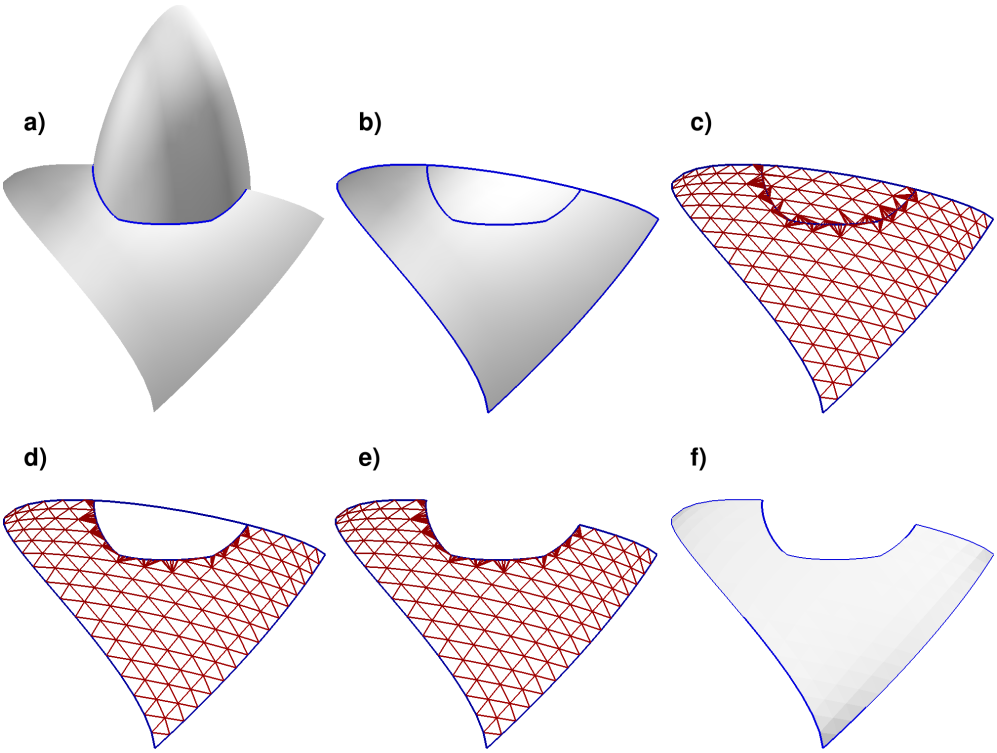


Figure B.32 Steps in the trimming of a triangular Bézier patch

are computed; after that, the curves are divided into pieces using the obtained points. As the curves are approximated by piecewise linear elements, the intersections are computed easily.

c) Tessellate the parametric subdomain of the subpatches from the leaf nodes of the levels of detail hierarchy that are intersected by the curve pieces obtained in the previous step. To do this, the barycentric coordinates of the intersection points among the base triangles of the subpatches and the curves are computed; the tessellation vertices are then the vertices of the subdomain together with these points. The tessellation algorithm can be any of the classic ones [O'R98, Las96].

   As a result, all the parametric subdomains stored in the hierarchy are now completely inside or completely outside of the solid.

d) Delete the subpatches from the leaf nodes of the hierarchy that are inside or outside, depending on the trimming criterion. Propagate the

deletion process upwards in the hierarchy, deleting those nodes whose children have been completely deleted and updating the bounding boxes when necessary. Algorithm B.9 describes this process.

To check whether a subpatch is inside or outside the solid, the inclusion test algorithm described in section B.5 is applied to the barycenter of its base triangle.

e) Delete the boundaries that are inside or outside the solid, depending on the trimming criterion.

f) Define the new boundaries for the trimmed patch

```
Algorithm trim ( patch, obj, criterion, maxN )

  If patch.level ≠ maxN then
    For each subpatch sp ∈ patch.children do
      trim ( sp, obj, criterion, maxN )

      If sp.status = toBeDeleted then
        delete sp
        patch.status ← trimmed

    If patch.nChildren = 0 then
      patch.status ← toBeDeleted

  Else
    vInc ← pointInSolidTest ( patch.base.baric, obj )

    If vInc = criterion then
      patch.status ← toBeDeleted
```

Algorithm B.9 Algorithm to delete the subpatches during the trimming process

The result of the process is a trimmed patch that keeps the same curvature as the original patch, as the subpatches have been computed using subdivision over the original patch [Sei89]. The trimmed patch can be trimmed again using the same algorithm.

Figures B.33, B.34 and B.35 show examples of trimmed patches. Table B.8 shows how many patches were involved, how many triangle-triangle intersections were computed, how many segments were obtained and how much time took to obtain the figures.

Figure	Patches	Triangle intersections	Segments	Time (seconds)
B.33	31	502624	303	0.44
B.34	13	821392	344	0.67
B.35	43	1305920	836	1.20

Table B.8 Computational data for figures B.33, B.34 and B.35

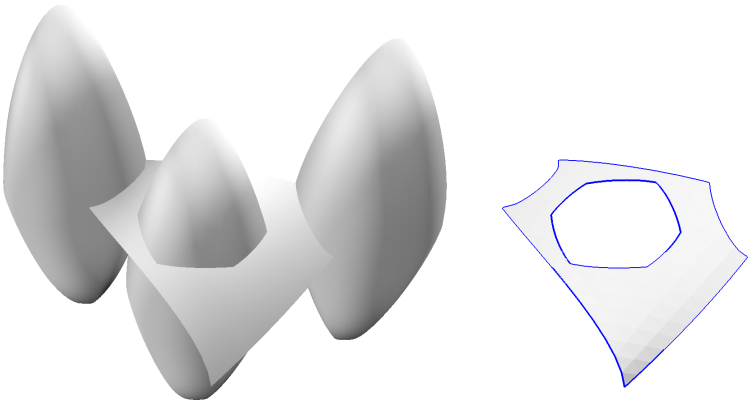


Figure B.33 Trimming examples (I). Left: initial position. Right: trimmed patch

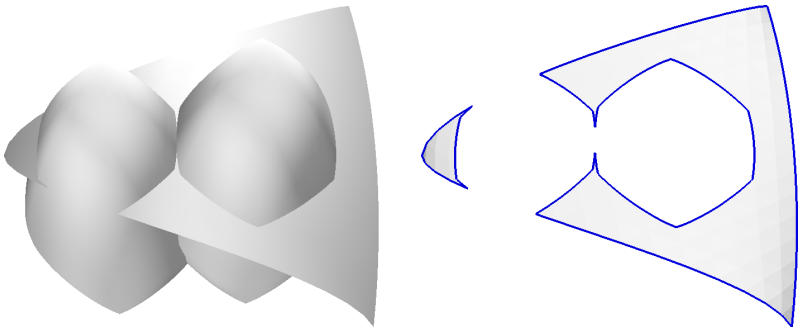


Figure B.34 Trimming examples (II). Left: initial position. Right: trimmed patch

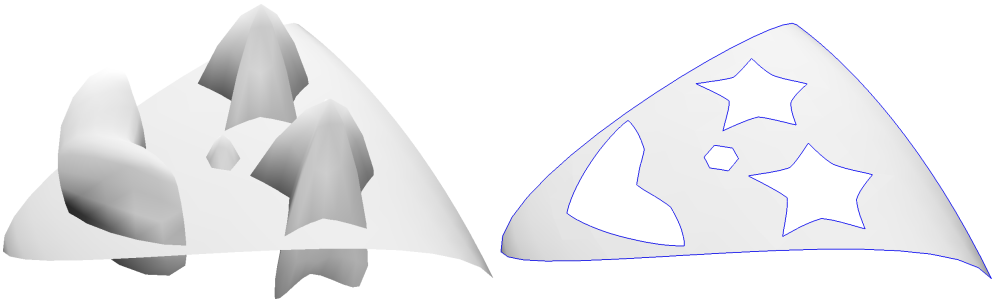


Figure B.35 Trimming examples (III).
Left: initial position. Right: trimmed patch

Building the B-Rep representation of a boolean operation

The B-Rep of the result of a boolean operation between two free-form solids bounded with TBPs can be obtained by trimming the patches from each operand using the other operand as trimming solid, and selecting the patches for the new solid according to the operation that is being performed. As the trim is based on barycentric coordinates, using the algorithm described above to trim the patches results in a more or less accurate approximation to the result. The accuracy of the result depends on the number of levels and divisions in the levels of detail hierarchy; this effect can be seen in figure B.36, where the same operation has been performed with one level and one division (B.36.a), one level and three divisions (B.36.b) and two levels and three divisions (B.36.c).

It is therefore necessary to perform an additional operation on the trimmed patches, consisting on correcting their geometry to assure that the set of selected patches defines a valid solid. This implies removing holes and intersections among the patches. To make it simpler, the trimmed patches are decomposed into triangular patches using the triangular subpatches that are stored in their hierarchies. Taking this as starting point, several strategies have been studied, considering the interpolation scheme that is used to obtain the TBPs [VPBM01].

1. The first strategy is the simplest one: as the base triangles of the patches do constitute a correct closed triangle mesh that approximates the boundary of the new solid, it is taken as the new initial triangulation of the solid, and the interpolation scheme is applied over it. However,

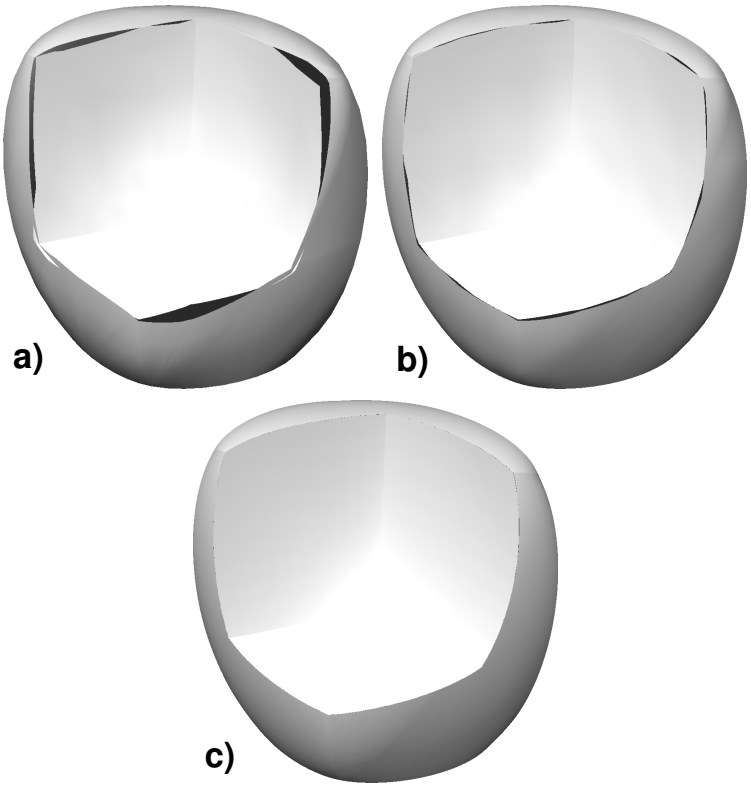


Figure B.36 Results obtained with different setups of the levels of detail hierarchies. a) One level and one parametric division per level. b) One level and three parametric divisions per level. c) Two leveles and three parametric divisions per level

the result depends on the interpolation scheme used; applying the selected scheme [VPBM01], the surface of the new solid suffers from an undesirable smoothing effect, as figure B.37 shows.

2. The second strategy is more complex. It implies modifying the triangulation formed by the base triangles of the patches, in order to reduce the smoothing effect of the previous attempt. It consists on the following steps (figure B.38):

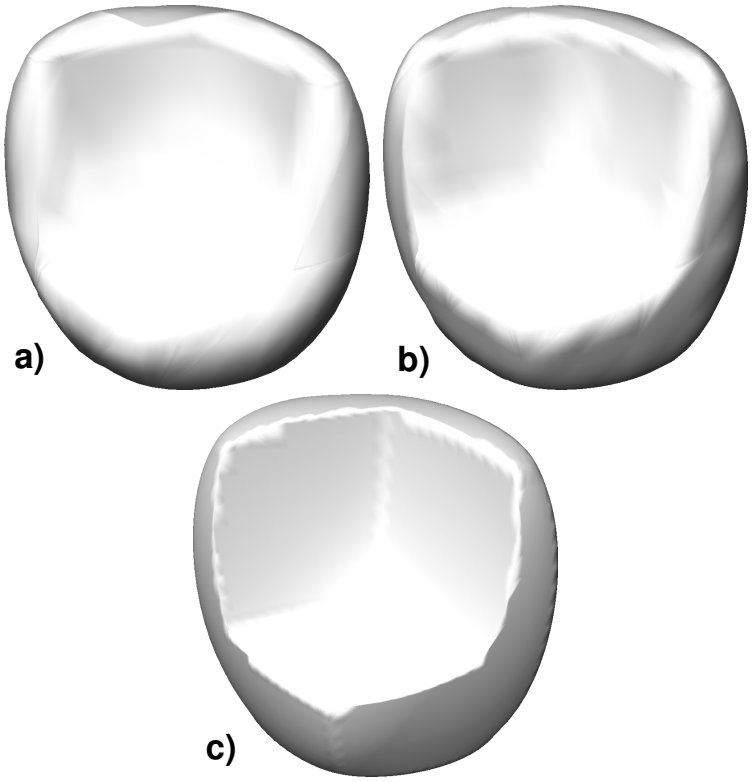


Figure B.37 Results applying the first strategy to correct the B-Reps from figure B.36

- First of all, move the vertices from the base triangles to the corners of the TBPs (figure B.38.b).

- Build two new triangles to fill the hole that has appeared in the previous step (figure B.38.c).

- Compute two new patches associated to the new triangles (figure B.38.d).

When the patches intersect, it is necessary to swap the vertices of the triangles, as shown in figure B.39.

This solution produces visually better results (figure B.40). However, there appear too many special cases, like patches rotated with

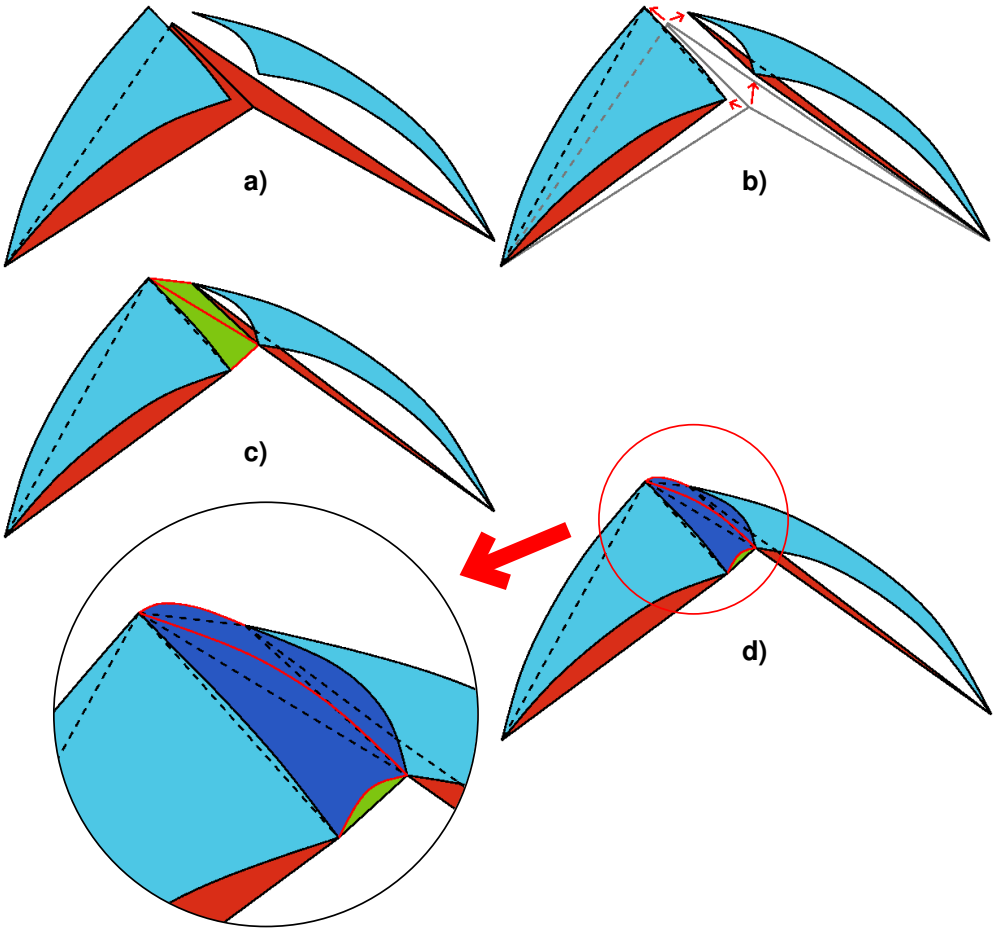


Figure B.38 Steps in the second strategy to correct the B-Reps

respect to their neighbours; some of them are even produced by the vertex swapping operation. Therefore this solution was discarded.

3. The third strategy modifies the control point of the patches to make the holes disappear (figure B.41). It consists of the following steps:

   – Substitute the control points from the vertices of the patches by the average of the corresponding points from all the patches that should share that vertex. Figures B.41.b and B.41.c.

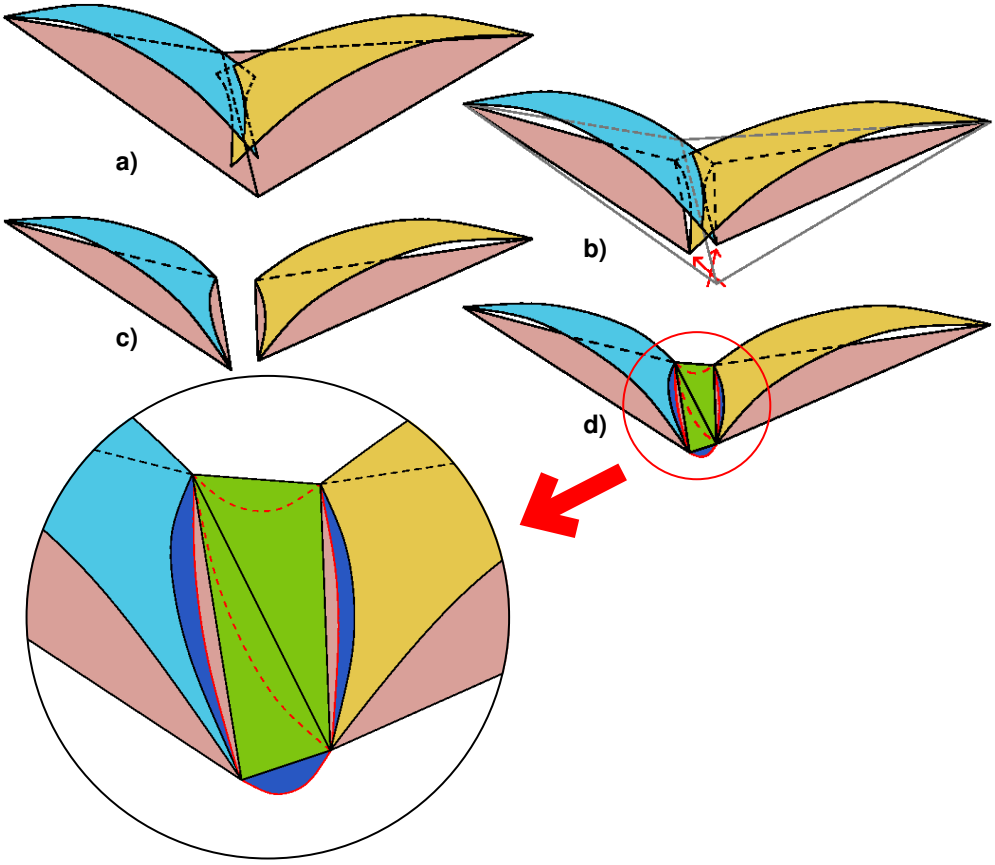


Figure B.39 Steps in the second strategy to correct the B-Reps when the patches intersect

– Substitute the two inner control points from the edges of the patches by the average of the points from the patches that should share that edge. Figures B.41.d and B.41.e.

– Substitute the vertices of the triangles with the vertices of the patches. Figure B.41.f.

The visual appearance of the results is the best of the three strategies. See figure B.42.

Figures from B.44 to B.56 show more examples of evaluation of boolean operations using the third strategy proposed. Figure B.43 shows the original primitives used.

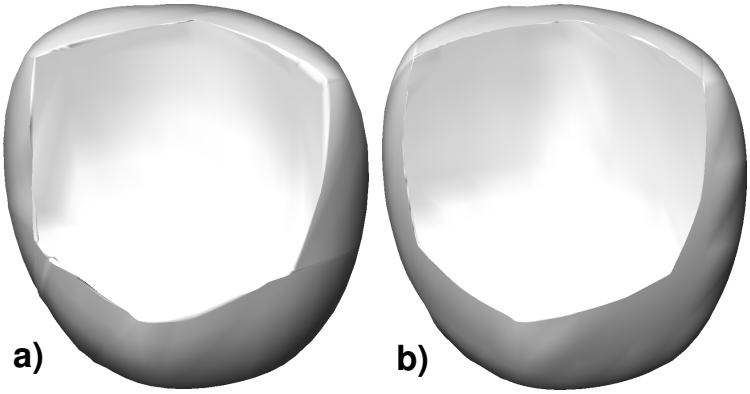


Figure B.40 Results obtained by applying the second strategy to correct the B-Reps from figure B.36. Result from figure B.36.c is not included, as there are no visual differences

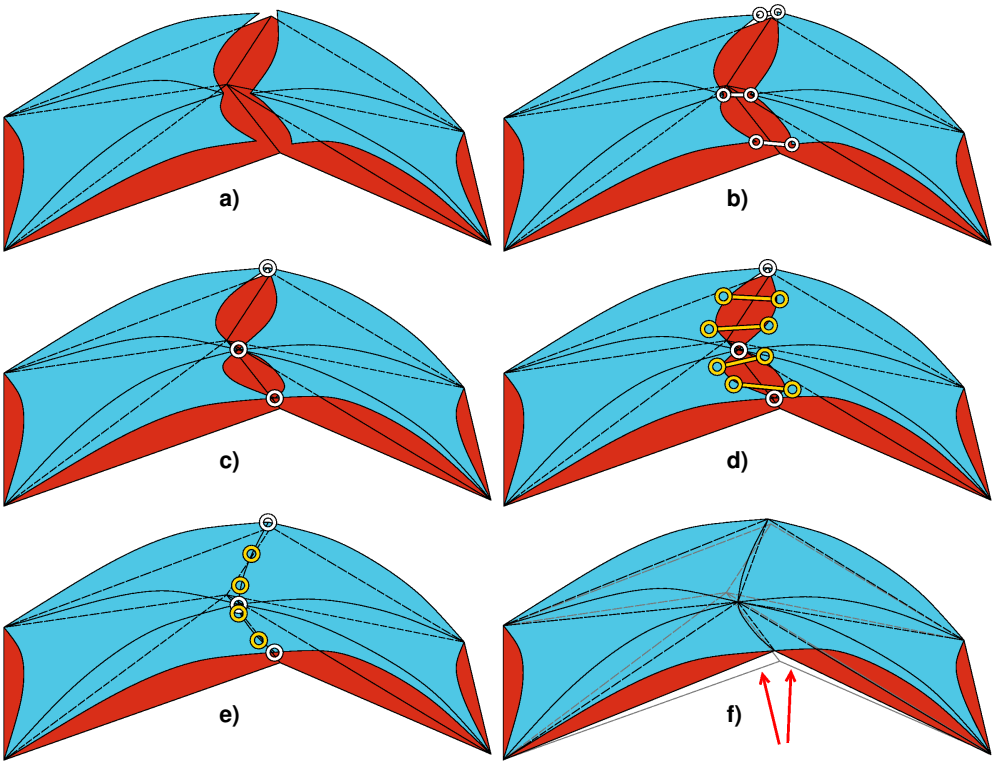


Figure B.41 Steps in the third strategy to correct the B-Reps

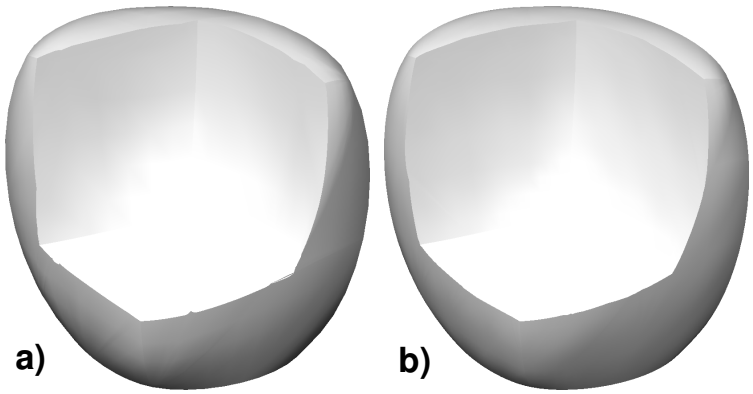


Figure B.42 Results obtained by applying the third strategy to correct the B-Reps from figure B.36. Result from figure B.36.c is not included, as there are no visual differences

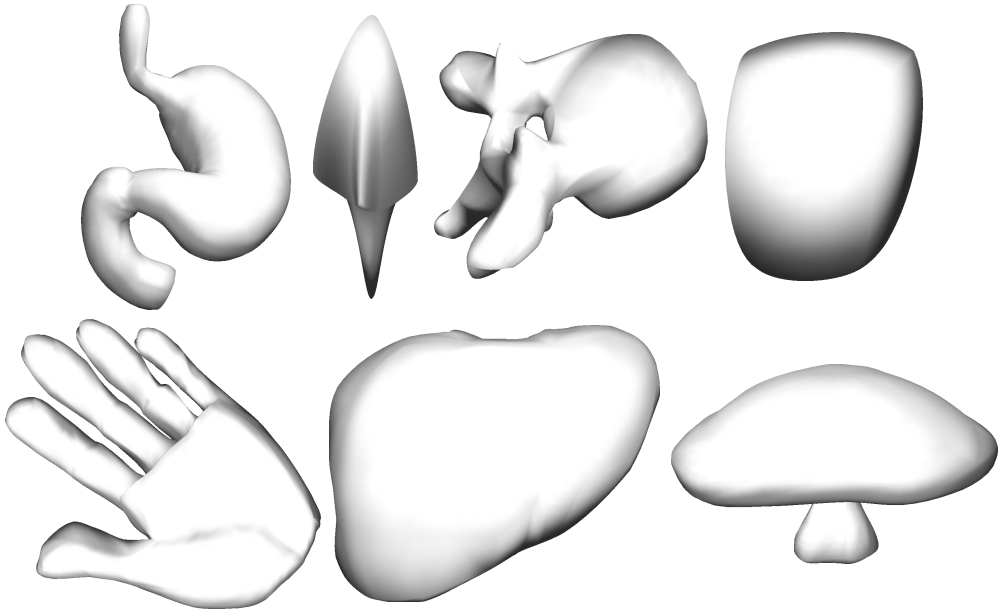


Figure B.43 Models used in the evaluation of boolean operation tests. From left to right and from top to bottom: stomach (534 TBPs), spearhead (20 TBPs), vertebra (298 TBPs), simple prism (12 TBPs), hand (768 TBPs), liver (274 TBPs) and mushroom (448 TBPs)

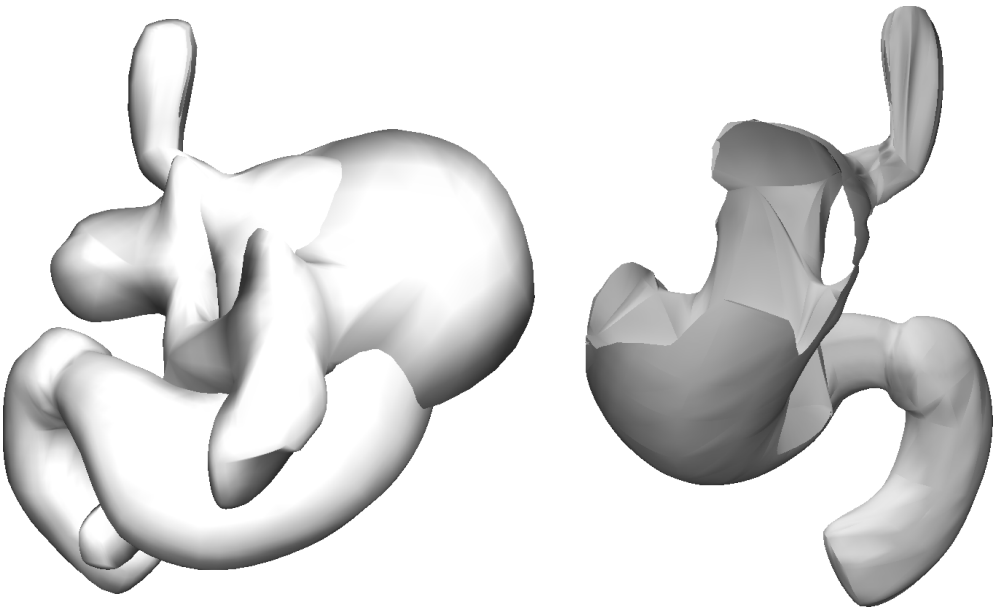


Figure B.44 Difference between the stomach model and the vertebra model. Left: initial setup. Right: result of the operation

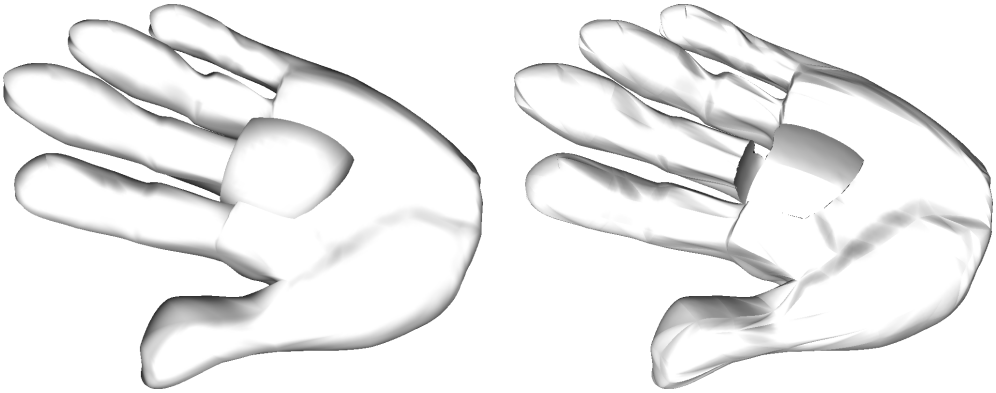


Figure B.45 Difference between the hand model and the simple prism. Left: initial setup. Right: result of the operation

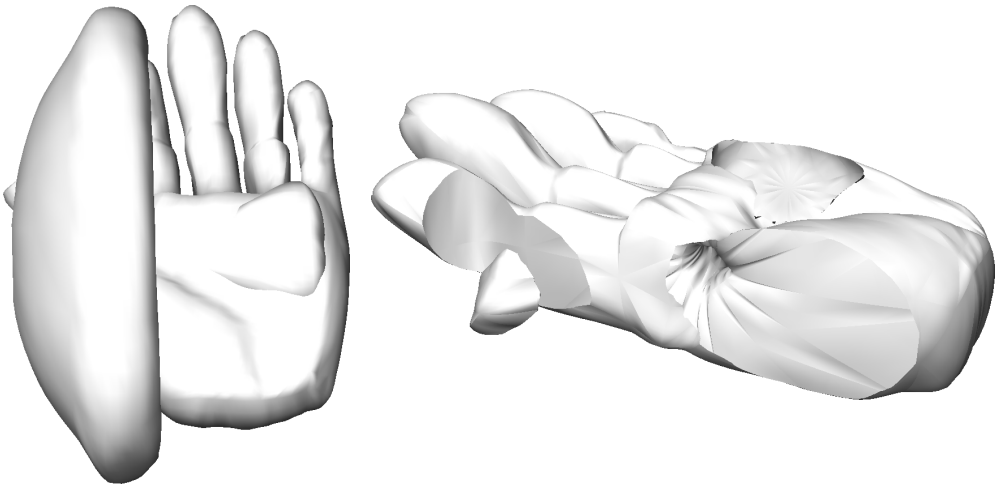


Figure B.46 Difference between the hand model and the mushroom model. Left: initial setup. Right: result of the operation

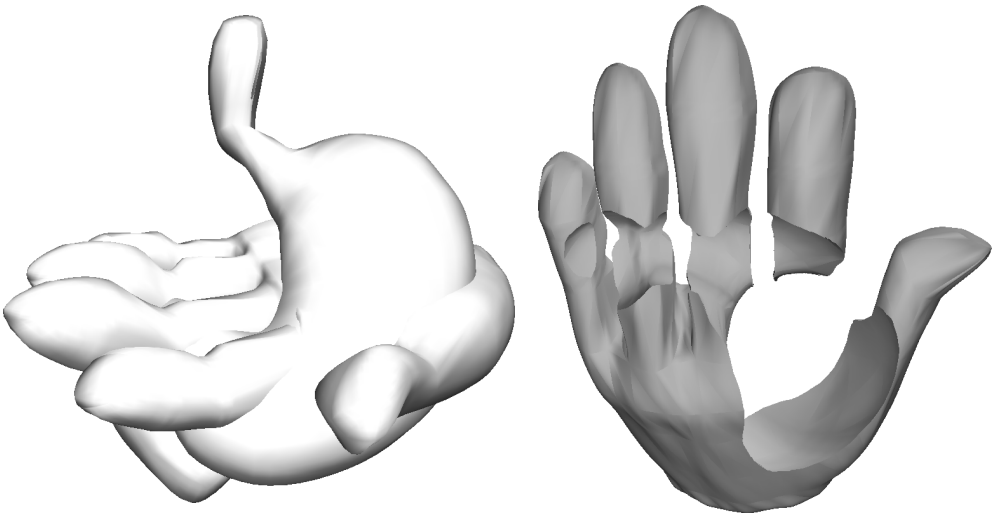


Figure B.47 Difference between the hand model and the stomach model. Left: initial setup. Right: result of the operation

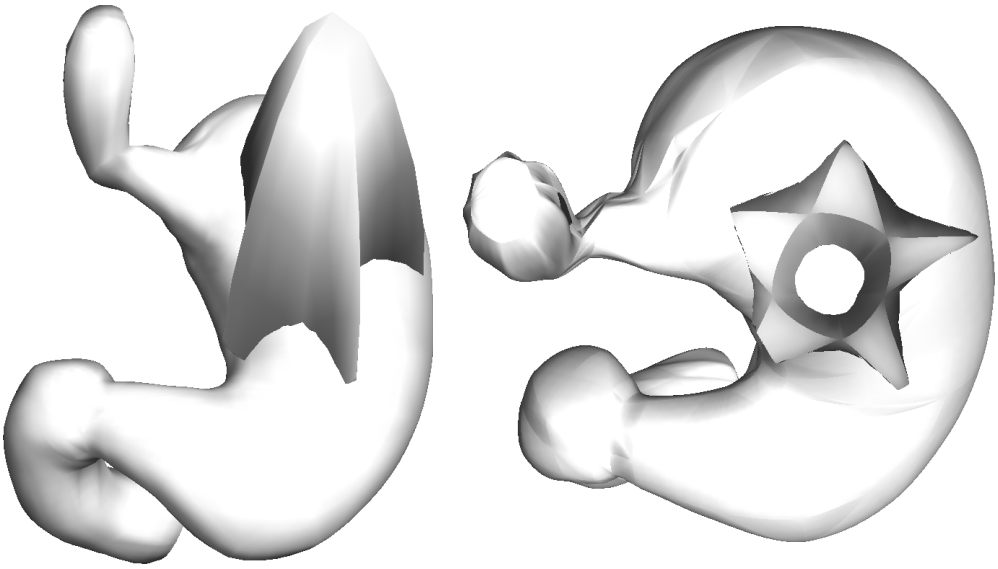


Figure B.48 Difference between the stomach model and the spearhead model. Left: initial setup. Right: result of the operation

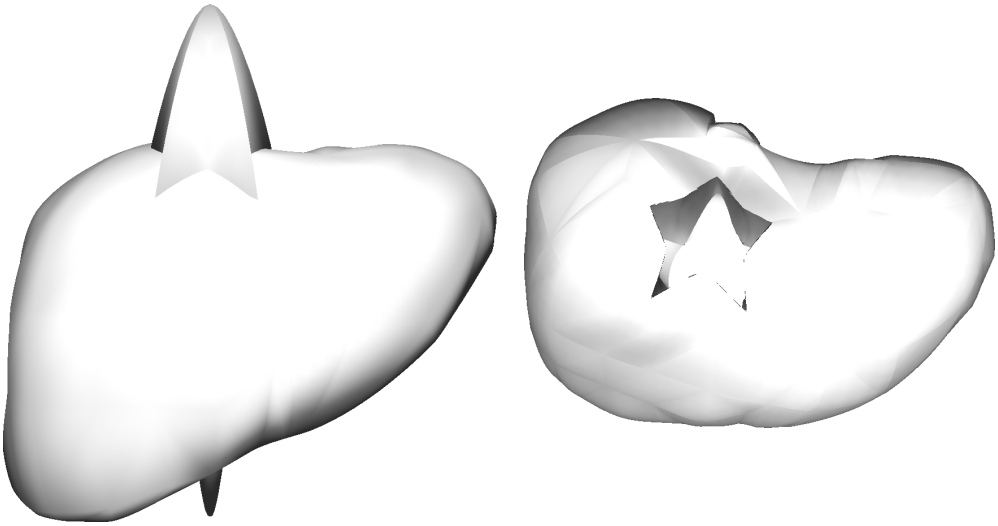


Figure B.49 Difference between the liver model and the spearhead model. Left: initial setup. Right: result of the operation

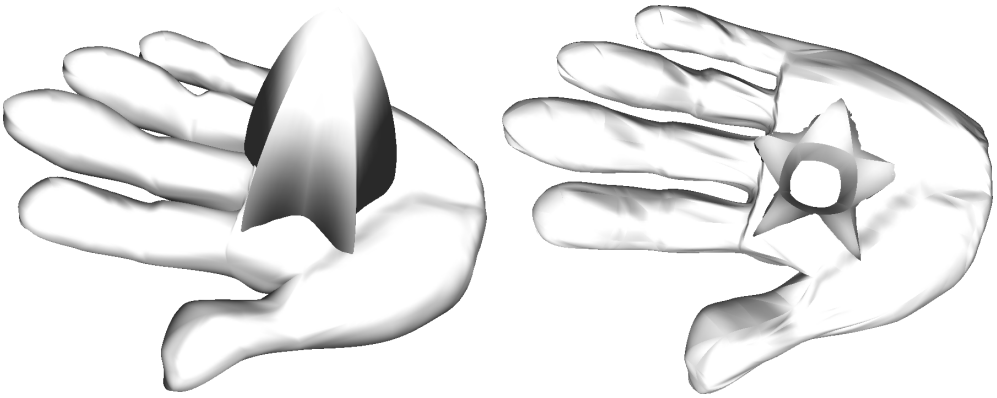


Figure B.50 Difference between the hand model and the spearhead model. Left: initial setup. Right: result of the operation

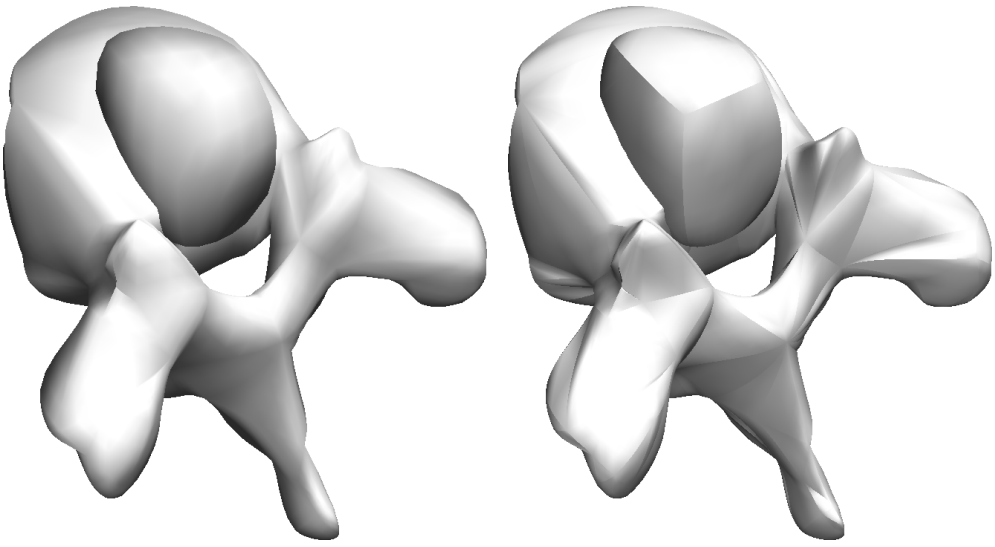


Figure B.51 Union between the vertebra model and the simple prism. Left: initial setup. Right: result of the operation

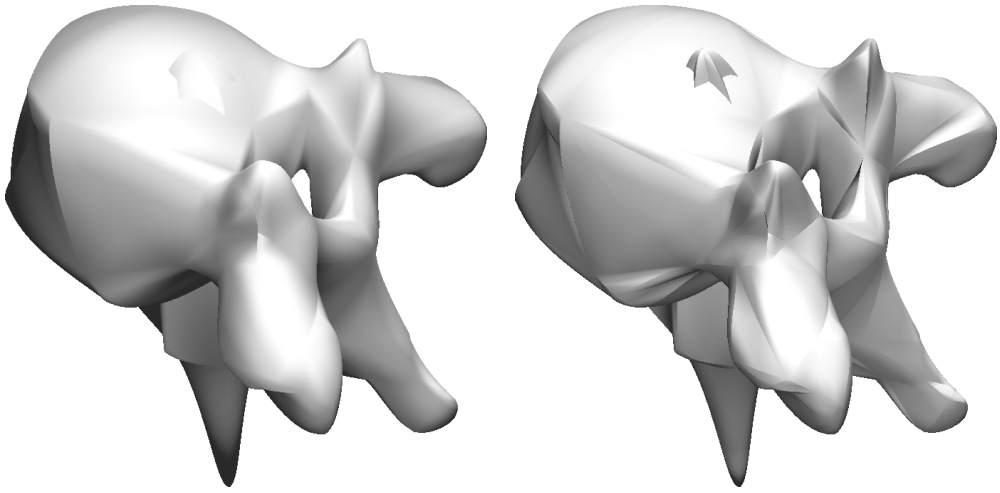


Figure B.52 Union between the vertebra model and the spearhead model. Left: initial setup. Right: result of the operation

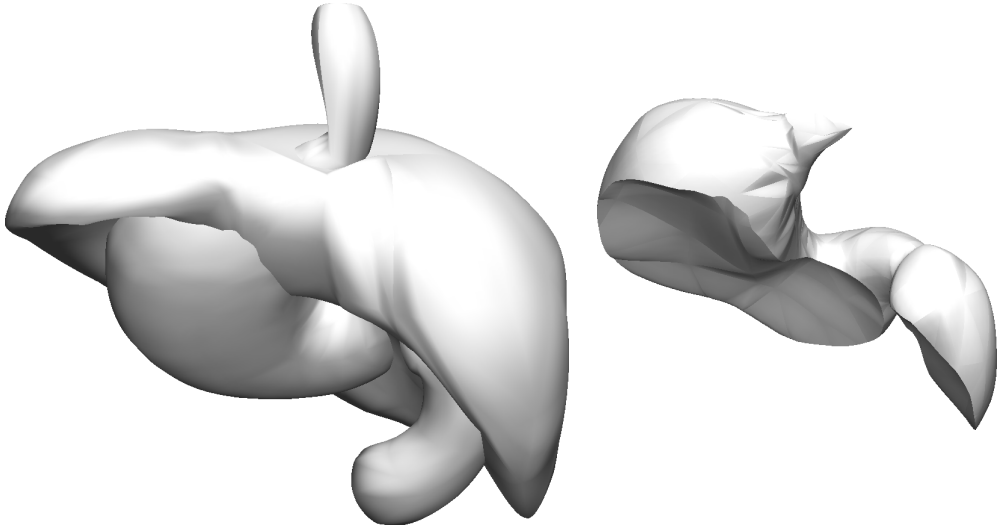


Figure B.53 Intersection between the stomach model and the liver model. Left: initial setup. Right: result of the operation

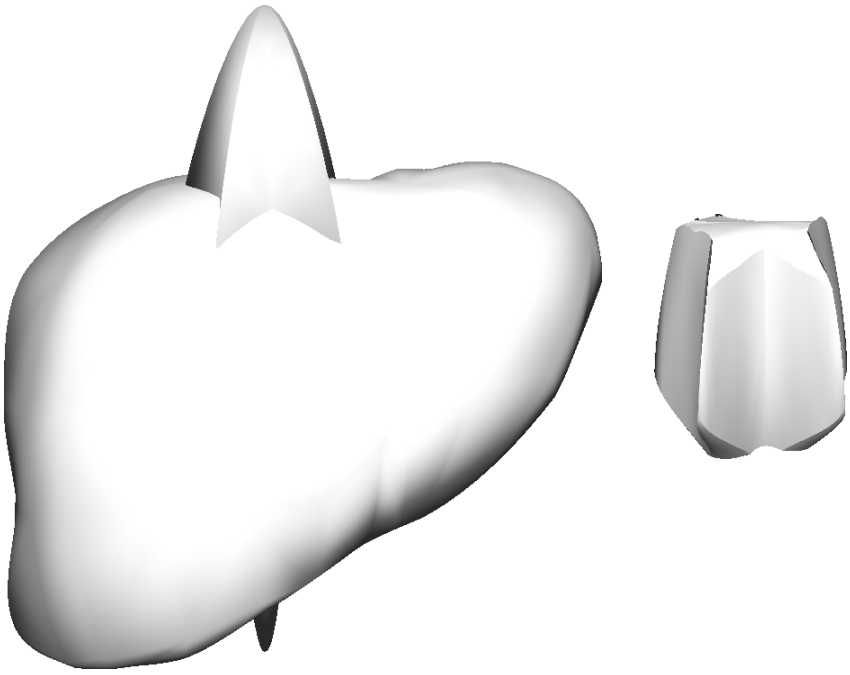


Figure B.54 Intersection between the liver model and the spearhead model. Left: initial setup. Right: result of the operation

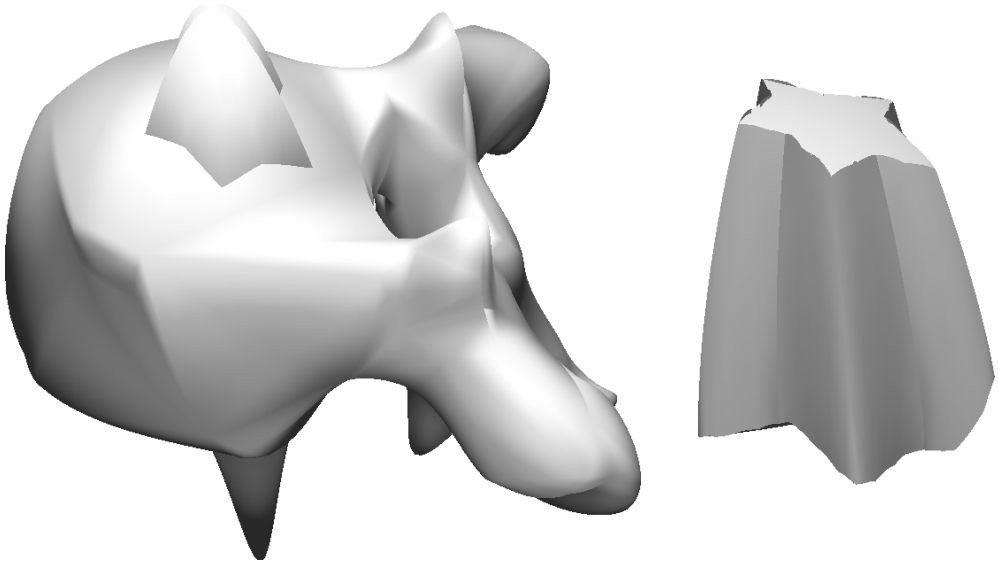


Figure B.55 Intersection between the vertebra model and the spearhead model. Left: initial setup. Right: result of the operation

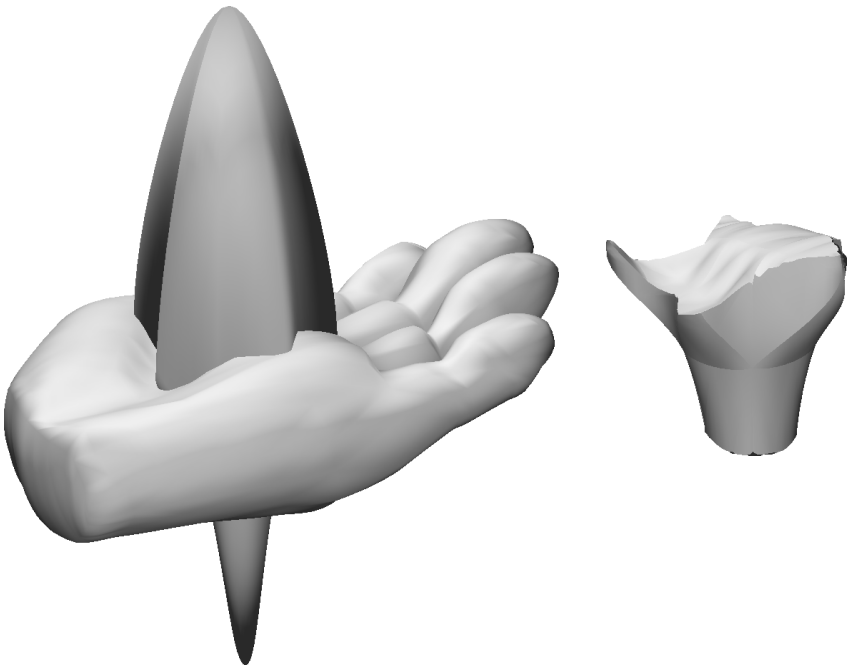


Figure B.56 Intersection between the hand model and the spearhead model. Left: initial setup. Right: result of the operation

B.6.3 Building the ESC for the result of a boolean operation

It is essential for any solid representation scheme that the result of any operation on solid models created using it can be modeled using the same scheme. Therefore the building of an ESC for the result of a boolean operation whose boundary has been computed as described above is studied here.

The formal representations proposed previously to build the ESCs (formulae B.16, B.19 and B.20) give rise to very long chains after a few operations, due to the fact that it is necessary to evaluate $ExCell\,(E \cap E')$ for each pair of extended cells from the operands. However, many of these intersections will not be of interest, as their volume will not contribute to the final solid, so it is desirable that some kind of simplification strategy could be applied on the ESCs.

Ruiz proposed a simplification strategy consisting of evaluating the boundary of the result before building the ESC [Rui01]. This strategy is applied here to reduce the number of extended cells.

The first approach to build the ESC for the result of a boolean operation takes advantage of the levels of detail hierarchies used to represent the TBPs. It takes the subpatches stored in the leaf nodes of the hierarchy as simple patches, and builds a simplex and one or two ffcs for each of these new TBPs. See figure B.57. This solution has been successfully implemented.

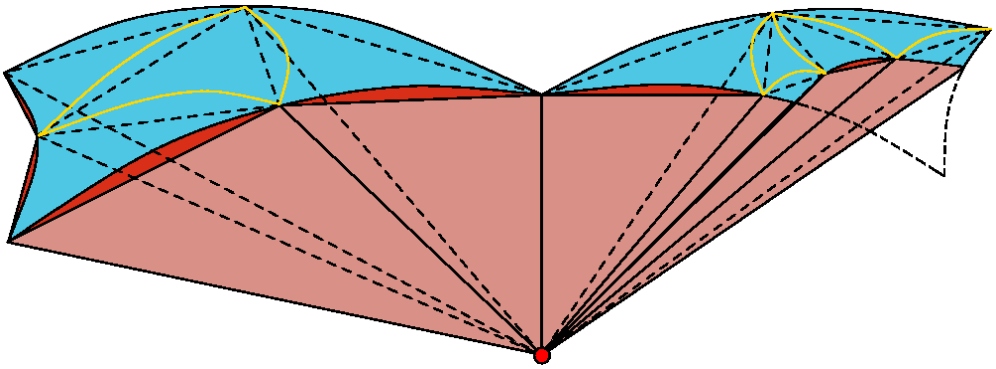


Figure B.57 First approach to build the ESC for the result of a boolean operation

A somewhat more elaborated solution consists of building a simplex and one or two ffcs for each (sub)patch that has not been trimmed. The patches that have not been trimmed will then produce the same number of extended cells than before the operation, while the trimmed ones will produce a variable number of extended cells, depending on the size and distribution of the trim (figure B.58). In this case, it could be necessary to apply subdivision on some patches to avoid cracks in the representation; this issue is known to be usual in polyhedral representations using levels of detail [LRC+03].

One of the major advantages of these two approaches is that the resulting ESCs contain only simplices and ffcs as defined previously; therefore the algorithms described above can be applied on them as usual.

Another approach to build the ESC for the result of a boolean operation is based on the definition of a new type of extended cell, called trimmed cell. A ***trimmed cell*** is the result of the intersection of two extended cells (simplices, ffcs or trimmed cells); it can be then bounded by either planar or curved elements from the boundaries of these cells. Figure B.59 shows some examples of trimmed cells.

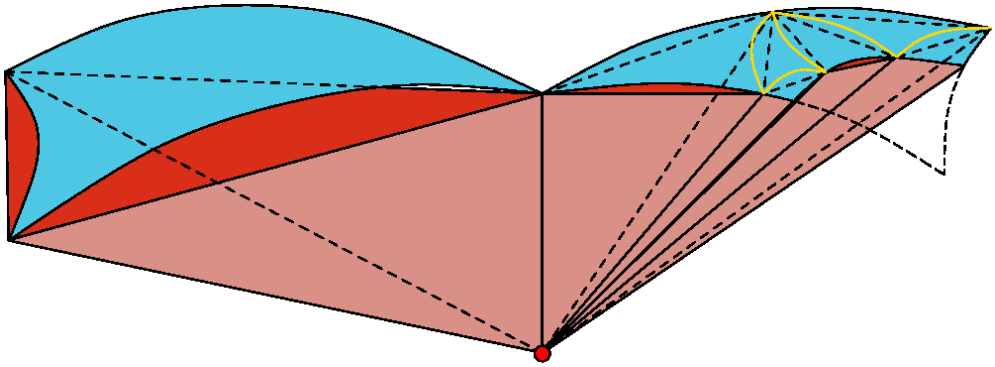


Figure B.58 Second approach to build the ESC for the result of a boolean operation

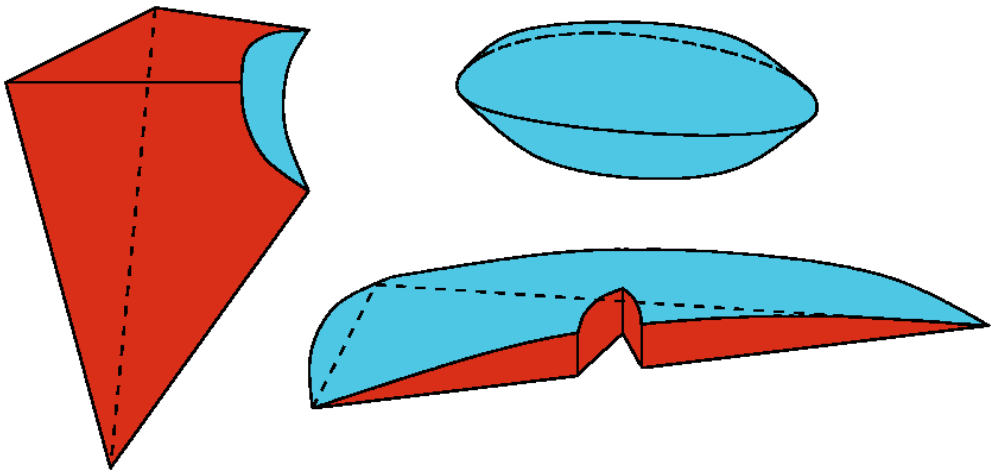


Figure B.59 Trimmed cells examples

Trimmed cells verify the same properties as the simplices and ffcs, and they can be included as elements in an ESC. The main drawback of using trimmed cells is the huge number of special situations that can appear when computing the intersection of extended cells. Therefore, the determination of the bounding elements of a trimmed cell can become very difficult.

The ESC for the result of a boolean operation using trimmed cells can be computed by intersecting every pair of extended cells from the operands, and building as many trimmed cells as necessary. The new ESC will include all these new cells, together with the ones from the operands, just like the formal equations B.16, B.19 and B.20 describe.

B.7 Conclusions and future work

Here the results, conclusions and future work are summarized.

B.7.1 Results

The results are directly related with the main goals in section B.1.1:

- The first goal of this work was the extension of the ESC model to free-form solids bounded with triangular parametric patches. This goal has been achieved by defining a new type of free-form cell, bounded with triangular Bézier patches (section B.4.2), together with a point in solid test for ffcs (section B.5.3) [GRF03].

- The second goal consisted of developing algorithms to compute the point in solid test and the visualization of solids modeled using ESCs. The algorithm to check the inclusion of a point in a free-form solid has been presented in section B.5 [GRF04a], while the visualization of solids can be solved both by drawing the base triangles from the levels of detail hierarchies described in section B.4.1, or by using ray-casting, as described in section B.6.1 [GRF05].

- The third goal included the development of algorithms to compute evaluated and non-evaluated boolean operations on free-form solids modeled using ESCs. This goal was fulfilled by developing robust algorithms to intersect and trim triangular Bézier patches as seen in section B.6.2 [GRF06]. Solutions to merge a set of trimmed patches into a boundary representation of the result of a boolean operation have been also presented, and strategies to build a new ESC that models the new solid have been described in section B.6.3.

The algorithms presented in this work have been implemented in C++ [Str97] and tested with several free-form solids, as seen in figures all over this document. The modeling application runs on GNU/Linux systems, and uses OpenGL [SWND05] for the rendering process. The interface was designed using GLUI [RSB] (figure B.60).

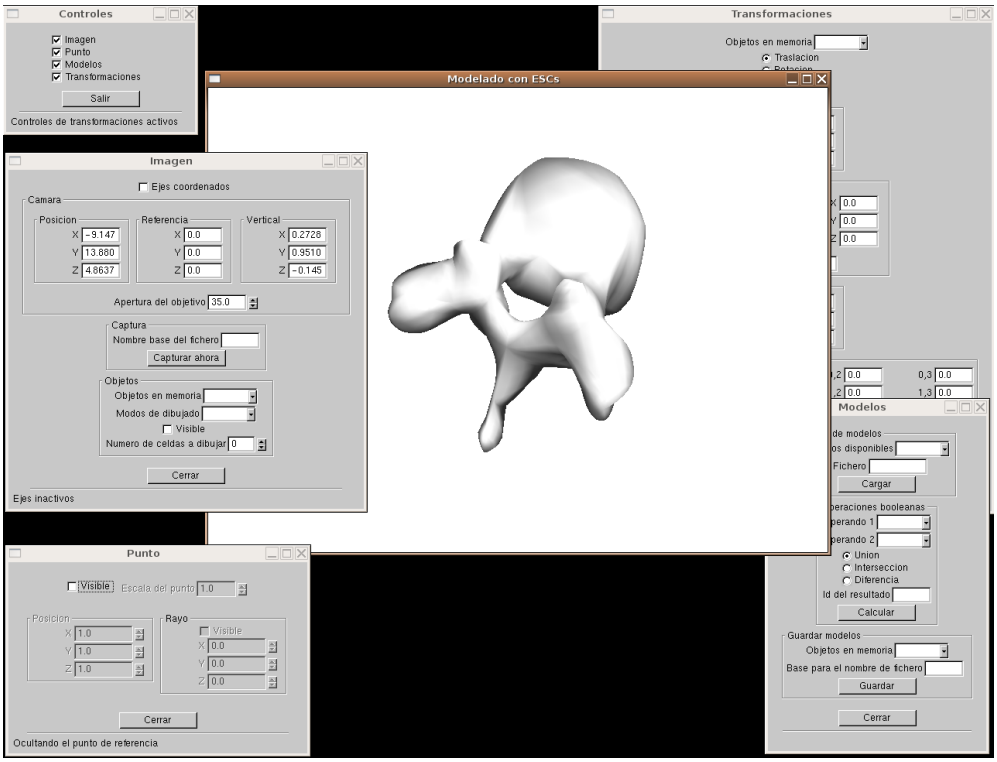


Figure B.60 Screenshot of the modeling application

B.7.2 Future work

There are several open topics related to the ESC model:

- As the ESC model represents not only the boundary, but also the volume of the solids, it would be desirable to have some method to represent information about the materials. This would make the ESC model suitable for modeling heterogeneous solids.

- Conversion from the ESC representation to other standard representations, like octree or spatial enumeration. This could be done by applying the point in solid test [RF02].

- Algorithms to evaluate boolean operations have been presented in this work. Using them, the step by step evaluation of CSG trees is straightforward; however, developing an algorithm to evaluate a complete CSG tree at once would be very useful.

- Trimmed patch simplification. Sometimes the subpatches obtained during the trimming operation can be regrouped. Having an algorithm to combine these subpatches into a simple one would simplify the boundary of the solids while keeping the accuracy.

- Study the existing hardware related works using triangular Bézier patches [VPBM01, ATI01, BS05, BRS05], and use the capabilities of graphics hardware to improve the performance of the algorithms.

B.7.3 Conclusions

The Extended Simplicial Chain model has been presented as a closed representation scheme for free-form solid modeling. This scheme has been adapted to parametric triangular patches, which are a very popular type of surface in CAD/CAM environments; algorithms to perform the point in solid test, the visualization and evaluation of boolean operations have also been described in detail, together with future developments.

243

Bibliografía

[ATI01] ATI Technologies, Inc. Truform white paper. 2001

[AMY96] K. Abdel-Malek and H.-J. Yeh. Determining intersection curves between surfaces of two solids. *Computer Aided Design*, 28:539–549, 1996.

[AMBJ06] K. Abdel-Malek , D. Blackmore and K. Joy Swept volumes: Foundations, perspectives, and applications. 2006. Submitted to International Journal of Shape Modeling.

[AGCA06] R. Allègre , E. Galin , R. Chaine and S. Akkouche. The HybridTree: mixing skeletal implicit surfaces, triangle meshes and point sets in a free-form modeling system. *Graphical Models*, 68(1):42–64, 2006.

[AB90] N.M. Aziz and R. Bata. Bezier surface/surface intersection. *IEEE Computer Graphics and Applications*, 10:50–58, 1990.

[BGA04] A. Barbier , E. Galin and S. Akkouche. Complex skeletal implicit surfaces with levels of detail. *Journal of the WSCG*, 12:35–42, 2004.

[Bec98] D. Bechmann. Multidimensional free-form deformation tools. state of the art report.. In *Proceedings of Eurographics'98*, 1998.

[BKZ01] H. Biermann , D. Kristjansson and D. Zorin. Approximate boolean operations on free-form solids. In *ACM Siggraph*, pages 185–194, Los Angeles, USA, 2001.

[BS05] T. Boubekeur and C. Schlick. Generic mesh refinement on GPU. In *Graphics Hardware 2005 Proceedings*, pages 99–104, 2005. SIGGRAPH/Eurograhpics.

243
243

[BRS05] T. Boubekeur , P. Reuter and C. Schlick. Scalar tagged PN triangles. In *Eurographics 2005 Short Papers Proceedings*, pages 17–20, 2005.

[BB97] G. Brassard and P. Bratley. *Fundamentos de Algoritmia*. Prentice Hall , 1997.

[BJ93] W.F. Bronsvoort and F.W. Jansen. Feature modelling and conversion–Key concepts to concurrent engineering. *Computers in Industry*, 21(1):61–86, 1993.

[BN85] P. Brunet and I. Navazo. Geometric modelling using exact octree representation of polyhedral objects. In *Proceedings of Eurographics'85*, 1985.

[BA87] P. Brunet and D. Ayala. Extended octree representation of free form surfaces. *Computer Aided Geometric Design*, 4(1–2):141–154, 1987.

[BN90] P. Brunet and I. Navazo. Solid representation and operation using extended octrees. *ACM Transactions on Graphics*, 9(2):170–197, 1990.

[Bus05] S.R. Buss. *3-D Computer Graphics. A mathematical introduction with OpenGL*. Cambridge University Press , 2005.

[CT02] P. Cano and J.C. Torres. Representation of polyhedral objects using sp-octrees. In *Proceedings of WSCG 2002*, pages 95–101, 2002.

[CTV03] P. Cano , J.C. Torres and F. Velasco. Progressive transmission of polyhedral solids using a hierarchical representation scheme. In *Proceedings of WSCG 2003*, 2003.

[CKKK97] N.W. Cho , N.K. Kim , Y. Kim and S.-H. Kang. An evolutionary method for general surface-surface intersection problems. *Computers & Industrial Engineering*, 33:573–576, 1997.

[dC63] P. de Casteljau Courbes et surfaces à poles. Technical Report, André Citroën Automobiles SA, 1963.

[ET00] G. Erhart and R.F. Tobler General purpose Z-Buffer CSG rendering with consumer level hardware. Technical Report, VRVis Zentrum für Virtual Reality und Visualisierung Forschungs GmbH, 2000. http://www.vrvis.at.

[Far86] G. Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3:83–127, 1986.

[Far93] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design. A Practical Guide*. Academic Press, San Diego , 1993.

[FCS95] F.R. Feito , F.A. Conde and R.J. Segura. *Informática Gráfica. Teoría y Práctica*. Universidad de Jaén, Jaén , 1995.

[Fei95] F.R. Feito *Modelado de sólidos y álgebra de objetos gráficos*. PhD thesis, Universidad de Granada, 1995.

[FT97] F.R. Feito and J.C. Torres. Inclusion test for general polyhedra. *Computers & Graphics*, 21(1):23–30, 1997.

[FR98] F.R. Feito and M. Rivero. Geometric modelling based on simplicial chains. *Computers & Graphics*, 22(5):611–619, 1998.

[FvDFH92] J.D. Foley , A. van Dam , S.K. Feiner and J.F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley , 1992.

[For97] S. Fortune. Polyhedral modelling with multiprecision integer arithmetic. *Computer Aided Design*, 29(2):123–133, 1997.

[GRF03] Á.-L. García , J. Ruiz and F.R. Feito. Free-form solid modelling based on extended simplicial chains using triangular Bézier patches. *Computers & Graphics*, 27(1):27–39, 2003.

[GRF04a] Á.-L. García , J. Ruiz and F.R. Feito. Point in solid test for free-form solids defined with triangular Bézier patches. *The Visual Computer*, 20(5):298–313, 2004.

[GRF05] Á.-L. García , J. Ruiz and F.R. Feito. Algebraic representation of CSG solids built from free-form primitives. In V. Skala, editor, *WSCG'2005 Short Papers*, pages 1–4, Plzen (Czech Republic), 2005.

[GRF06] Á.-L. García , J. Ruiz and F.R. Feito. Adaptive trimming of cubic triangular Bézier patches. In P. Brunet , N. Correia and G. Baranoski, editors, *SIACG'2006 Proceedings*, pages 149–158, Santiago de Compostela (Spain), 2006.

[Gar82] I. Gargantini. Linear octrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing*, 20, 1982.

[Gla93] A.S. Glassner, editor *An Introduction to Ray Tracing* Academic Press , 1993.

[Gol03] R. Goldman. *Pyramid Algorithms*. Morgan Kaufmann Publishers , 2003.

[GK97] T.A. Grandine and F.W. Klein. A new approach to the surface intersection problem. *Computer Aided Geometric Design*, 14:111–134, 1997.

[GKMV03] S. Guha , S. Krishnan , K. Munagala and S. Venkatasubramanian. Application of the two-sided depth test to CSG rendering. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 177–180, 2003. ACM, ACM Press.

[HR05] J. Hable and J. Rossignac. Blister: GPU-based rendering of boolean combinations of free-form triangulated shapes. *ACM Transactions on Graphics*, 24(3):1024–1031, 2005. Proceedings of ACM Siggraph 2005.

[HB03] S. Hahmann and G.-P. Bonneau. Polynomial surfaces interpolating arbitrary triangulations. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):99–109, 2003.

[HG96] J.-M. Hasenfratz and D. Ghazanfarpour. Rendering CSG scenes with general antialiasing. In *CSG 96 Set-theoretic Solid Modelling: Techniques and Applications*, pages 275–289, Winchester, U.K., 1996. Informations Geometers Ltd..

[HKE99] H.-S. Heo , M.-S. Kim and Gershon Elber. The intersection of two ruled surfaces. *Computer Aided Design*, 32:33–50, 1999.

[HML00] G. Hirota , R. Maheshwari and M.C. Lin. Fast volume-preserving free-form deformation using multi-level optimization. *Computer Aided Design*, 32(8–9):499–512, 2000.

[HR96] C.M. Hoffmann and J.R. Rossignac. A roadmap to solid modeling. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):3–10, 1996.

[HEFS85] E.G. Houghton , R.F. Emnett , J.D. Factor and C.L. Sabharwal. Implementation of a divide-and-conquer method for intersection of parametric surfaces. *Computer Aided Geometric Design*, 2:173–183, 1985.

[Hoh92] M.E. Hohmeyer *Robust and Efficient Surface Intersection for Solid Modeling*. PhD thesis, University of California at Berkeley, 1992.

[HPY96] C.Y. Hu , N.M. Patrikalakis and X. Ye. Robust interval solid modelling. Part I: Representations. *Computer Aided Design*, 28(10):807–817, 1996.

[HMPY97] C.Y. Hu , T. Maekawa , N.M. Patrikalakis and X. Ye. Robust interval algorithm for surface intersections. *Computer Aided Design*, 29:617–627, 1997.

[Hui97] K.C. Hui. A robust point inclusion algorithm for regions bounded by parametric curve segments. *Computer Aided Design*, 29(11):771–778, 1997.

[Jor87] C. Jordan. *Course D'Analyse*. École Polytechnique de Paris , 1887.

[KCY93] A. Kaufman , D. Cohen and R. Yagel. Volume graphics. *IEEE Computer*, 26(7):51–64, 1993.

[KKM97] J.C. Keyser , S. Krishnan and D. Manocha. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic. In C.M. Hoffmann and W. Bronsvort, editors, *Fourth Symposium on Solid Modeling and Applications*, pages 42–55, Atlanta, GA, 1997. Solid Modeling Association, ACM Press.

[Key00] J.C. Keyser *Exact Boundary Evaluation for Curved Solids*. PhD thesis, University of North Carolina at Chapel Hill, 2000.

[KCF+04] J. Keyser , T. Culver , M. Foskey , M. Krishnan and D. Manocha. ESOLID–a system for exact boundary evaluation. *Computer Aided Design*, 36(2):175–193, 2004.

[KNZ03] N. Kohlmüller , G. Nürnberger and F. Zeilfelder. Construction of cubic 3d spline surfaces by lagrange interpolation at selected points. In T. Lyche , M.-L. Mazure and L.L. Schumaker, editors, *Curve and Surface Design: Saint-Malo 2002*, pages 245–254, Brentwood, USA, 2003. Nashboro Press.

[KPS95] A. Kolb , H. Pottmann and H.-P. Seidel Fast and fair surface reconstruction. Technical Report, Universität Erlangen, 1995.

[KM96] S. Krishnan and D. Manocha. Efficient representations and techniques for computing B-Reps of CSG models with NURBS primitives. In *CSG 96 Set-theoretic Solid Modelling: Techniques and Applications*, pages 101–122, Winchester, U.K., 1996. Informations Geometers Ltd..

[KGMM97] S. Krishnan , M. Gopi , D. Manocha and M. Mine. Interactive boundary computation of boolean combinations of sculptured solids. *Computer Graphics Forum*, 16(3):67–78, 1997. Proceedings of Eurographics'97.

[Kur73] K. Kuratowski. *Introducción a la teoría de conjuntos y a la topología*. Vicens-Vives, 2nd edition , 1973.

[Las96] Michael J. Laszlo. *Computational Geometry and Computer Graphics in C++*. Prentice Hall, 1st edition , 1996.

[LCK02] K.Y. Lee , D.Y. Cho and T.W. Kim. A tracing algorithm for surface/surface intersections on surface boundaries. *Journal of Computer Science and Technology*, 17:843–850, 2002.

[LKL+02] Y.-Q. Li , Y.-L. Ke , W.S. Li , Q.-S. Peng and J.-R. Tan. Termination criterion for subdivision of triangular Bézier patch. *Computers & Graphics*, 26:67–74, 2002.

[LJCW04] X. Li , H. Jiang , S. Chen and X. Wang. An efficient surface-surface intersection algorithm based on geometry characteristics. *Computers & Graphics*, 28:527–537, 2004.

[Lin00] L. Linsen. Netbased modelling. In *Spring Conference on Computer Graphics (SCCG)*, pages 259–266, Budmerice (Slovakia), 2000.

[Loo94] C. Loop. A G^1 triangular spline surface of arbitrary topological type. *Computer Aided Geometric Design*, 11:303–330, 1994.

[LRC+03] D. Luebke , M. Reddy , J.D. Cohen , A. Varshney and B. Watson et al.. *Level of Detail for 3D Graphics*. Morgan Kaufmann Publishers , 2003.

[Män88] M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press , 1988.

[Mea82] D. Meagher. Geometric modeling using octree enconding. *Computer Graphics and Image Processing*, 19, 1982.

[Men92] J.P. Menon *Constructive shell representations for free-form surfaces and solids*. PhD thesis, Cornell University, 1992.

[Möl97] T. Möller. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2:25–30, 1997.

[MNK90] M. Morozumi , H. Nakamura and Y. Kijima. A primitive-instancing interactive 3-D modeling system for spatial design studies. In Malcolm McCullough , William J. Mitchell and Patrick Purcell, editors, *The Electronic Design Studio. Proceedings of the 3rd CAAD Futures conference*, pages 457–468, 1990. MIT Press.

[Mor97] M.E. Mortenson. *Geometric Modeling*. John Wiley & Sons , 1997.

[Muk05] H. Mukundan Surface-surface intersection with validated error bounds. Master's thesis, Massachusetts Institute of Technology, 2005.

[OSF03a] C.J. Ogayar , R.J. Segura and F.R. Feito. Técnicas para el cálculo de inclusión de puntos en mallas de triángulos. estudio comparativo. In *XIII Congreso español de Informática Gráfica (CEIG 2003)*, 2003.

[OSF03b] C.J. Ogayar , R.J. Segura and F.R. Feito. Point in solid tests for triangle meshes. comparative study. In *Proceedings of Eurographics 2003*, 2003.

[Pat93] N. M. Patrikalakis. Surface-to-surface intersections. *IEEE Computer Graphics and Applications*, 13:89–95, 1993.

[PM02] N.M. Patrikalakis and T. Maekawa. *Shape Interrogation for Computer Aided Desing and Manufacturing*. Springer Verlag , 2002.

[PMKM04] N.M. Patrikalakis , T. Maekawa , K.H. Ko and H. Mukundan. Surface to surface intersections. *Computer-Aided Design and Applications*, 1:449–458, 2004.

[Pip87] B.R. Piper. *Geometric Modeling: Algorithms and New Trends*, chapter Visually Smooth Interpolation with Triangular Bézier Patches, pages 221–233. SIAM, Philadelphia, 1987.

[PBP02] H. Prautzsch , W. Boehm and M. Paluszny. *Bézier and B-Spline Techniques*. Springer Verlag , 2002.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. Springer Verlag, New York , 1985.

[RSB] P. Rademacher , N. Stewart and B. Baxter GLUI user interface library. http://glui.sourceforge.net.

[Rap95] A. Rappoport. Geometric modeling: a new fundamental framework and its practical implications. In *Third ACM Symposium on Solid Modeling and its Applications*, pages 31–41, 1995. ACM, ACM Press.

[RSB96] A. Rappoport , A. Sheffer and M. Bercovier. Volume-preserving free-form solids. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):19–27, 1996.

[RS97] A. Rappoport and S. Spitz. Interactive boolean operations for conceptual design of 3-d solids. In *24th Conference on Computer Graphics & Interactive Techniques*, pages 269–278, Los Angeles, USA, 1997. ACM.

[RV77] A.A.G. Requicha and H.B. Voelcker Constructive solid geometry. Tech. Memo 25, Production Automation Project, Univ. Rochester, Rochester, N.Y., 1977.

[Req80] A.A.G. Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Computing Surveys*, 12(4):437–464, 1980.

[RUL00] J. Revelles , C. Ureña and M. Lastra. An efficient parametric algorithm for octree traversal. In *Proceedings of WSCG 2000*, pages 212–219, 2000.

[RF00] M.L. Rivero and F.R. Feito. Boolean operations on general planar polygons. *Computers & Graphics*, 24(6):881–896, 2000.

[Riv02] M.L. Rivero *Algoritmos para las operaciones booleanas en 2D y 3D, bajo un sistema de representacin formal*. PhD thesis, Universidad de Granada, 2002.

[RTA93] J.R. Rossignac , J. Turner and G. Allen, editors *Second ACM Symposium on Solid Modeling and Applications*, 1993. ACM, ACM Press.

[RDG01] M. Roth , P. Diezi and M.H. Gross. Ray tracing triangular Bézier patches. *Computer Graphics Forum*, 20, 2001.

[O'R98] Joseph O'Rourke. *Computational Geometry in C* Cambridge University Press, 2nd edition , 1998.

[RF97] J. Ruiz and F.R. Feito. Inclusion test for curved edge polygons. *Computers & Graphics*, 21(6):815–824, 1997.

[RF98] J. Ruiz and F.R. Feito. B-rep-octree conversion for free-form solids. In *VIII Spanish Conference on Computer Graphics*, 1998. in spanish.

[RF99a] J. Ruiz and F.R. Feito. Mathematical free-form solid modeling based on extended simplicial chains. In *WSCG '99: VII Conference on Computer Graphics, Visualization and Interactive Digital Media*, pages 241–248, Plzen-Bory, Czech Republic, 1999.

[RF99b] J. Ruiz and F.R. Feito. Inclusion test for free-form solids. *Computers & Graphics*, 23(2):255–268, 1999.

[Rui01] J. Ruiz *Free-Form Solid Modeling*. PhD thesis, Universidad de Granada, Granada, Spain, 2001.

[RF02] J. Ruiz and F.R. Feito. Direct and robust voxelization and poligonalization of free-form CSG solids. In *3DVPT 2002: 1st International Symposium on 3D Data Processing, Visualization and Transmission*, 2002.

[SP86] T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. In *Proceedings of the SIGGRAPH 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 151–160, 1986. ACM, ACM Press.

[Sei89] H.P. Seidel. *Mathematical methods in computer aided geometric design*, chapter A general subdivision theorem for Bézier triangles, pages 573–581. Academic Press, 1989.

[SV95] V. Shapiro and D.L. Vossler. What is a parametric family of solids?. In *Third ACM Symposium on Solid Modeling and its Applications*, pages 43–54, 1995. ACM, ACM Press.

[Sha02] V. Shapiro. Solid modeling. In G. Farin , J. Hoschek and M.-S. Kim, editors, *Handbook of Computer Aided Geometric Design*. North-Holland, 2002.

[SS87] L.A. Shirman and C.H. Séquin. Local surface interpolation with Bézier patches. *Computer Aided Geometric Design*, 4:279–295, 1987.

[SS91] L.A. Shirman and C.H. Séquin. Local surface interpolation with Bézier patches: errata and improvements. *Computer Aided Geometric Design*, 8:217–221, 1991.

[SWND05] D. Shreiner , M. Woo , J. Neider and T. Davis. *OpenGL Programming Guide*. Addison-Wesley, 5th edition , 2005.

[SL98] N. Stewart and G. Leach. An improved Z-Buffer CSG rendering algorithm. In *Workshop on Graphics Hardware*, pages 25–30, 1998. Eurographics / Siggraph.

[SLJ02] N. Stewart , G. Leach and S. John. Linear-time CSG rendering of intersected convex objects. In *WSCG 2002: 10th International Conference on Computer Graphics, Visualization and Interactive Digital Media*, pages 437–444, Plzen-Bory, Czech Republic, 2002.

[Str97] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 3rd edition , 1997.

[Stü88] W. Stürzlinger. Ray-tracing triangular trimmed free-form surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 4:202–214, 1988.

[TN87] W.C. Thibault and B.F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *Proceedings of the ACM SIGGRAPH'87*, pages 153–162, New York, USA, 1987. ACM, ACM Press.

[Tor92] J.C. Torres *Representación abstracta de sistemas gráficos. Teoría de objetos gráficos*. PhD thesis, Universidad de Granada, 1992.

[Tve80] H. Tverberg. A proof of the Jordan curve theorem. *Bulletin of the London Mathematical Society*, 12:34–38, 1980.

[vdBBV02] E. van den Berg , W.F. Bronsvoort and J.S.M. Vergeest. Free-form feature modelling: concepts and prospects. *Computers in Industry*, 49(2):217–233, 2002.

[VPBM01] A. Vlachos , J. Peters , C. Boyd and J.L. Mitchell. Curved PN triangles. In *2001 ACM Symposium on Interactive 3D Graphics*, pages 159–166, New York, USA, 2001. ACM, ACM Press.

[Whi57] H. Whitney. *Geometric Integration Theory*. Princeton University Press, Princeton, USA , 1957.

[Woo03] Y. Woo. Fast cell-based decomposition and applications to solid modeling. *Computer Aided Design*, 35(11):969–977, 2003.

[YS91] J.Q. Ying and N. Sugie. A point-inclusion algorithm for a domain with boundary composed of algebraic curve segments. *Systems and Computers in Japan*, 22:1823–1829, 1991.

253

Índice de Términos

253
253