UNIVERSIDAD NACIONAL "SANTIAGO ANTUNEZ DE MAYOLO"

ESCUELA DE POSTGRADO

GENERAC IÓN AUTOMÁTICA DE MALLAS TRIANGULARES CON CONTROL DE ERROR Y APLICACIONES EN LA INDUSTRÍA

Tesis para optar el grado de maestro
en Ciencias e Ingeniería
Mención en Computación e Informática

MAXIMILIANO EPIFANIO ASÍS LÓPEZ

Asesor: **Dr. JESÚS EDILBERTO ESPINOLA GONZALES**

Huaraz – Perú
2007

Nº Registro: T0105

ACTA DE SUSTENTACION DE TESIS

En la ciudad de Huaraz a los veinticuatro días del mes de noviembre del dos mil siete y siendo la una con treinticinco minutos de la tarde, los miembros del Jurado Calificador: Msc. Esmelin Niquín Alayo, Presidente; M.Sc. Jube C. Portaletino Zevallos, Secretario; Jesús E. Espinola Gonzales, vocal; se reunieron en un ambiente de la Escuela de Postgrado de la UNASAM, en cumplimiento a lo establecido en la Resolución Nº 111-2007-UNASAM-EPG, para evaluar la tesis: "Generación automática de mallas triangulares con control de error y aplicaciones en la industria", del Bachiller Maximiliano Epifanio Asis López, egresado de la Maestría en Ciencias e Ingeniería con mención en Computación e Informática.

El Presidente de Jurado otorgó al Bachiller veinte minutos para la sustentación de la tesis en mención, al término del cual, los miembros de la Comisión "Jurado formularon las preguntas correspondientes, las mismas que fueron absueltas por el bachiller. Terminado el proceso, el jurado en pleno otorgó la conformidad, por lo que se procedió con la deliberación para la calificación, obtenemdo el Bachiller la nota de diecisiete (17) considerándose APROBADO CON MENCION, y quedando expedito para obtener el grado Académico correspondiente.

Siendo las dos y treinte minutos de la tarde del mismo día, se dió por concluido el acto de sustentación, firmándose en señal de conformidad el presente acta

M.Sc. Esmelin Niquín Alayo
Presidente

M.Sc. Jube Portaletino Zevallos
Secretario

Dr. Jesús E. Espinola Gonzales
Vocal

UNIVERSIDAD NACIONAL "SANTIAGO ANTÚNEZ DE MAYOLO"
SECRETARIA GENERAL
CERTIFICA: Que la presente copia es conforme al original que tuve a la vista.
HUARAZ: 13 JUN. 2023

UNIVERSIDAD NACIONAL - JEFE DE GRADOS - Escuela de Post Grado - Santiago Antúnez de Mayolo

UNIVERSIDAD NACIONAL - DIRECTOR - ESCUELA DE POSTGRADO - SANTIAGO ANTÚNEZ DE MAYOLO

MIEMBROS DEL JURADO

Magíster Esmelin Niquin Alayo Presidente

_____

Magíster Jube Ciro Portalatino Zevallos Secretario

_____

Doctor Jesús Edilberto Espinola Gonzales Vocal

_____

ASESOR

***Doctor* Jesús Edilberto Espinola Gonzales**

AGRADECIMIENTO

- Al profesor Dr. Jesús Espinola por su apoyo en la obtención de toda la información necesaria para el desarrollo de esta tesis.
- A la Universidad Nacional Santiago Antunez de Mayolo por apoyarme en mi perfeccionamiento permanente.

A Dios,

A mis padres Hilario y Juana por toda su compresión

y apoyo en el logro de mis objetivos,

A todas las personas por su apoyo incondicional.

INDICE

Página

Resumen

Abstract

I. INTRODUCCIÓN ……....1

Objetivos ……..3

II. MARCO TEÓRICO ……..4

2.1. Antecedentes ……4

2.2. Sistema de Coordenadas Homogéneas y las

Transformaciones Geométricas afines ……..…….....5

2.3. Transformaciones Geométricas afines ……7

Transformaciones 2D …….....7

Coordenadas Homogéneas y Representación

matricial de Transformaciones 2D …….....9

Representación matricial de las Transformaciones en 3D ….....12

2.4. Proyecciones ……14

Tipos de Proyecciones …….....15

2.5. Representación Tridimensional de Objetos …….17

Modelado de Superficies …….....18

Mallas Poligonales …….....18

Ecuaciones Implícitas …….....19

Curvas y Superficies Paramétricas …….....20

Curvas y Superficies de Bézier …….....21

2.6. GENERACIÓN DE MALLA ……..43

2.7. Generación de malla no estructurada en superficies ……..….51

2.8. Métodos de generación de malla triangular en el plano …….....51

III. MATERIALES Y MÉTODOS ….……...70

3.1. Equipos ……..70

3.2. Software …….70

3.3. Métodos …….70

3.4. Herramientas computacionales ……..71

3.5. Principales formatos gráficos en CAGD …….72

IV. RESULTADOS …….....82

V. DISCUSION ……..…173

VI. CONCLUSIONES …….....176

VII. RECOMENDACIONES …….....177

VIII. REFERENCIAS BIBLIOGRAFICAS …….....178

ANEXO.....181

LISTA DE FIGURAS

Fig. Nº 01: Elementos de la Proyección ……..……..15

Fig. Nº 02: Cilindro representado por un malla poligonal ……18

Fig. Nº 03: Elipsoide modelado de forma implícita ……...19

Fig. Nº 04: Curvas de Bézier de diversos grados ….……..24

Fig. Nº 05: (a) Curva de Bézier de grado 3

(b) Polinomio de Bernstein de grado 3 …….…….26

Fig. Nº 06: Curvas de Bézier Racionales para:

(a) $w_0 = w_1 = w_2 = w_3$ (b) $w_1 > (w_0 = w_2 = w_3)$

(c) $(w_1 = w_2) < (w_0 = w_3)$ ……...28

Fig. Nº 07: Ejemplo de superficie de Bézier ……32

Fig. Nº 08: Ejemplo de funciones Base B-Spline ……..40

Fig. Nº 09 : Ejemplo de Superficie B-Spline ……..41

Fig. Nº 10: Elementos de línea y triangulares …….……44

Fig. Nº 11: Elementos cuadriláteros y tetraedros ……..….45

Fig. Nº 12: Elementos hexaédricos y prismas de base triangular . ….46

Fig. Nº 13: Malla no estructurada y una malla estructurada ……..48

Fig. Nº 14: Malla conforme y malla no conforme ……49

Fig. Nº 15: Tamaño del elemento para cada tipo de elemento ….……49

Fig. Nº 16: Mallado con el método de frente de avance …….……52

Fig. Nº 17: Triangulación Delaunay de un conjunto de puntos

en el plano ……..……..53

Fig. Nº 18: Mallado con el método de subdivisiones sucesivas……54

Fig. Nº 19: Diagrama de Voronoi de tres puntos ……..……..58

Fig. Nº 20: Criterio del circulo vació ……...60

Fig. Nº 21: Arista legal de la triangulación ……...61

Fig. Nº 22: Legalización de Arista legal de la triangulación …….….61

Fig. Nº 23: Arista ilegal (en naranja) ……...63

Fig. Nº 24: Arista legal obtenida al aplicar flip ……..64

Fig. Nº 25: Dualidad Voronoi-Delaunay …….66

Fig. Nº 26: Triángulo ficticio inicial ……..67

Fig. Nº 27: Esquema Gráfico del Algoritmo Incremental ……69

Fig. Nº 28: Ventana Principal de 3D-STUDIO MAX 7 72

Fig. Nº 29: Visualización del archivo IGES …….……..168

Fig. Nº 30: Visualización del archivo IGES del objeto vista Isométrica ……169

Fig. Nº 31: Visualización del archivo IGES del objeto vista Perspectiva……169

Fig. Nº 32: Visualización del archivo IGES del objeto vista “XY” ..170

Fig. Nº 33: Visualización de una superficie IGES que Representa al objeto …….……..171

Fig. Nº 34: Visualización isométrica del mallado …..……172

Fig. Nº 35: Vista XY del mallado …….172

RESUMEN

En esta tesis se describen los fundamentos matemáticos y algorítmicos para el desarrollo de un programa de aproximación; el cual aproxima un objeto 3D representado por una superficie de Bézier; a través de mallas triangulares con un control de error cometido.

Primeramente se digitaliza el objeto 3D en formato IGES. Luego se genera un conjunto de puntos aleatoriamente en el dominio de la superficie.

Finalmente se hace el mallado de la superficie que representa al objeto 3D, controlando el error con un modelo matemático obtenido en este trabajo; también se implementa el algoritmo de mallado obtenido, en Visual Basic.NET para visualizar el resultado.

Palabras Clave: Superficie de Bézier, Mallas Triangulares, Objeto 3D, Formato IGES.

ABSTRAC

In this thesis it is described the mathematical foundations and algorithmic for the development of an approach program; which approaches an object 3D represented by a surface of Bezier; through triangular meshes with a control of made error.

Firstly the object 3D are digitized in IGES Format. Then a group of points are generated aleatorily in the domain of the surface.

Finally the meshing of the surface that represents the object 3D is made, controlling the error with an mathematical model obtained in this work also the algorithm of meshing obtained is implemented, in Visual Basic.NET to visualize the result.

Key Words: Bézier Surfaces, Triangulate Mesh, 3D Object, IGES format.

I. INTRODUCCIÓN

Dado que el mundo real está compuesto de objetos y volúmenes en el cual interactuamos a través de su superficie, se necesita una expresión correcta de la superficie y el estudio detallado de sus propiedades.

Con el desarrollo de los ordenadores se puede hacer análisis de modelos extremadamente complejos; además los programas de cálculo y simulación por métodos numéricos se han ido sofisticando con el paso del tiempo, para modelizar de manera más aproximada los fenómenos reales; dichos programas requieren de datos que cada vez aumentan la complejidad, y la malla necesaria para el programa debe cumplir una especificaciones muy determinadas y ser capaces de modelar situaciones muy especiales de la geometría en el contexto del problema bajo estudio.

El resultado de esta necesidad ha sido que, actualmente, el proceso que generalmente consume más tiempo–hombre en el conjunto de un análisis es la preparación de datos y la generación de la malla.

Por otro lado el desarrollo de sistemas gráficos cada vez más competitivos esta ligado a la implementación en hardware de los distintos algoritmos involucrados en el proceso de modelado, síntesis y almacenamiento de los datos que representan a los objetos.

Todos los algoritmos de modelado se basan en la utilización de Superficies y volúmenes descritos mediante estructuras poligonales, que caracterizados por una descripción matemática muy sencilla, exige un tratamiento y almacenamiento de un volumen de información muy elevado. Utilizando las curvas paramétricas para

la descripción matemática de los objetos se obtiene mejores resultados y, además implica la reducción de los datos a ser manejados y almacenados.

El problema de generación de mallas en dos dimensiones consiste en aproximar el dominio de simulación por un polígono, y a este polígono dividirlo en elementos geométricos sencillos (cuadriláteros, triángulos), de tal manera que la intersección de dos elementos sea una arista, un vértice o un conjunto vacío. Cuando se tiene un dominio rectangular, la solución al problema es directa utilizando un arreglo uniforme de rectángulos, pero cuando se desea discretizar un dominio cuyo dominio no es trivial, se hace necesario un método de generación más elaborada. Aún cuando la geometría sea sencilla, el tamaño de los elementos determinara el error de aproximación del método numérico, y en ocasiones es deseable obtener una malla con elementos de distinto tamaño en diferentes zonas del dominio.

El mallado de objetos bidimensionales y tridimensionales es un tema de gran importancia en muchas aplicaciones pero se utiliza generalmente para los siguientes propósitos:

- Visualización de un objeto en un ordenador. Las primitivas básicas de visualización son los triángulos o tetraedros, por lo tanto en general, cualquier objeto para poder ser visualizado requiere previamente ser dividido (mallado) utilizando estas primitivas.

- Diseño Industrial. Para diseñar puentes, aviones, carros, barcos, etc. es muy útil tener un modelo de objeto en tres dimensiones que se utilice para simular en un ordenador el comportamiento físico del modelo.

Objetivos del estudio

Objetivo General.

El objetivo de esta tesis es, a partir de Superficies en representación paramétrica de Bézier, obtener para cada superficie una malla triangular con control de error de la malla respecto de la superficie que representa al objeto 3D. Así como la implementación de un programa de mallado de superficie que visualice dicho resultado en Visual Basic.NET.

Objetivos Específicos.

- Estudiar los fundamentos matemáticos de los diferentes tipos de mallado.
- Diseñar un programa computacional, con interfaz gráfica adecuada que pueda generar mallas a partir de Superficies Bézier.
- Presentar aplicaciones del método desarrollado en la industria del automóvil.
- El programa debe ser capaz de generar mallas a partir de las Superficies de Bézier y controlar el error cometido al aproximar al objeto a través de mallas triangulares.

II. MARCO TEÓRICO

2.1. Antecedentes

Los primeros estudios de modelación geométrica se remontan a los años 60-70 y se originaron en las grandes compañías fabricantes de automóviles. En un inicio estas compañías fueron las creadoras y las usuarias al mismo tiempo, de los primeros programas de diseño asistido por computador (CAD). Más tarde, los programas de este tipo se han ido extendiendo a todos los ámbitos de la ingeniería y del procesamiento industrial.

Gracias a los trabajos de Bézier, Coon, Gerald Farin [1], Foley [2] y otros han permitido que, en unos pocos años, todo el conocimiento matemático sobre geometría computacional se haya extendido y es ampliamente usado en todos los programas de modelación geométrica tridimensional.

La generación de mallas ha sido durante los últimos 20 años y sigue siendo hoy en día objeto de investigación por parte de multitud de grupos tanto en la universidad como en el ámbito de la explotación comercial. Actualmente, existen ya metodologías que funcionan razonablemente bien para problemas particulares. Pero el proceso de relación entre geometría del objeto y generación de malla provoca un problema que requiere procesos de adaptación del modelado geométrico, corrección de imperfecciones asociadas y preparación para la generación, que puede considerarse como un problema no resuelto de manera totalmente satisfactoria.

Los primeros trabajos sobre la generación de malla no estructurada bi y tridimensional se deben a R. Löhner [3], J. Peraire y J. Peiró. Más tarde, el

número de investigadores y grupos que se han dedicado a mejorar las metodologías de generación de malla se han ido aumentando progresivamente, pero estos programas solo son utilizados para tratar con problemas muy específicos, y por tanto, no aptos para tratar el análisis de error planteado como objetivo de esta tesis.

2.2. Sistema de Coordenadas Homogéneas y las Transformaciones Geométricas afines

Es bien conocido que, para representar dibujos, figuras u objetos tanto en dos como en tres dimensiones, se suele utilizar el sistema de coordenadas cartesianas. Este sistema es sencillo y bastante genérico a la hora de dibujar; pues todas las dimensiones se comportan de la misma manera y tienen el mismo sistema de representación, a diferencia de otros como las coordenadas polares donde existe un único eje y los objetos se representan con ejes y ángulos, lo cual si bien es útil (sobre todo en las rotaciones) no es suficientemente genérico, ya que posee dos notaciones distintas para representar los objetos (coordenadas lineales y ángulos).

Es por eso que se trata de implementar el sistema de coordenadas cartesianas, sin embargo, este sistema de coordenadas sí bien es el más eficiente a la hora de representar un objeto, no es necesariamente el más eficiente a la hora de realizar operaciones y transformaciones sobre dichos objetos. Esto es debido a que existen matrices de transformación para aplicar las transformaciones sobre los objetos como rotación y traslación, así como para transformaciones de cámara.

Algunas de estas matrices, de ser utilizado el sistema de coordenadas cartesianas necesitarían más datos para almacenar información (como la matriz de proyección) y en otros casos necesitarían cambiar la operación estándar de multiplicación por una de suma (como la matriz de translación) generando inconsistencias con el modelo multiplicativo de transformaciones.

Para simplificar el manejo de operaciones, mejorar la eficiencia y homogeneizar el manejo de matrices se utiliza el sistema de coordenadas homogéneas en el cual se añade una coordenada a cada punto.

De esta forma, cada punto en dos dimensiones se representa siguiendo la forma (x, y, w). Dos conjuntos de coordenadas homogéneas (x, y, w) y (x', y', w') representan al mismo punto si sólo sí una es múltiplo de la otra.; al menos una de las coordenadas debe ser diferente de cero, no se permite (0, 0,0). Si la coordenada $w \neq 0$, podemos dividir por ella, entonces (x, y, w) representa al mismo punto que $\left(\frac{x}{w}, \frac{y}{w}, 1\right)$; a los números $\frac{x}{w}$ y $\frac{y}{w}$ se les llama coordenadas cartesianas del punto homogéneo, según [4].

En general el punto $P(x_1, x_2, ..., x_n)$ es representado como $P(wx_1, wx_2, ..., wx_n, w)$, siendo este espacio de coordenadas de una dimensión mayor que la cartesiana.

Entonces, dado una representación en coordenadas homogéneas para un punto $P(x, y, z, w)$, se obtiene su representación cartesiana haciendo $x = \frac{x}{w}$,

$y = \frac{y}{w}$, $z = \frac{z}{w}$. En general w suele ser 1 excepto cuando se le aplican algunas matrices de proyección. Este sistema de coordenadas también hace que las matrices de transformación en tres dimensiones pasen de orden $3x3$ a ser de orden $4x4$, según [5].

2.3. Transformaciones Geométricas afines

Las transformaciones utilizadas en Computación Gráfica son: La traslación, escalamiento, y rotación.

Dichas transformaciones son utilizadas directamente por aplicaciones y en muchos paquetes de subrutinas gráficas. En general, muchas aplicaciones utilizan las transformaciones geométricas para cambiar la posición, orientación y tamaño de los objetos en un dibujo. Es conveniente, por tanto, comprender como combinar dichas transformaciones para obtener los resultados deseados, según [5].

Transformaciones 2D

- **Traslación**

Los puntos en el plano (x, y) se pueden trasladar a nuevas posiciones añadiendo cantidades de traslación a las coordenadas originales de los puntos. Para cada punto $P(x, y)$ a ser movido d_x unidades paralelamente al eje *x* y d_y unidades paralelamente al eje *y*, hacia el nuevo punto $P'(x', y')$, podemos escribir las siguientes ecuaciones

$$x' = x + d_x \qquad y' = y + d_y \quad \ldots\ldots\ldots\ldots\ldots\ldots(1)$$

Si se definen los vectores columna

$$P = \begin{bmatrix} x \\ y \end{bmatrix},\ P' = \begin{bmatrix} x' \\ y' \end{bmatrix},\ T = \begin{bmatrix} d_x \\ d_y \end{bmatrix} \qquad \text{…… (2)}$$

Entonces la ecuación (1) puede ser expresada como:

$$P' = P + T \qquad \text{….....……...……(3)}$$

Una forma de efectuar la traslación de un objeto es aplicando a cada punto del mismo la ecuación (1). No obstante, dado que cada línea del objeto está compuesta por un número infinito de puntos, esto tomaría un tiempo infinito de ejecución. Afortunadamente, se pueden trasladar todos los puntos de una línea simplemente trasladando los puntos extremos y dibujando una nueva línea entre los puntos extremos trasladados; esto se cumple también para el caso del escalamiento y la rotación, según [6].

- **Escalado**

Otra transformación aplicable a los puntos que componen a un objeto es el escalamiento, en un factor s_x en el eje *x* y un factor s_y en el eje *y*.

Puede comprobarse que si el factor relativo a un eje es menor que 1 esto resultará en una disminución del tamaño del objeto con respecto a ése eje, similarmente un factor mayor que 1 ocasionará un agrandamiento relativo al eje en cuestión. Para que el objeto conserve su proporción original debe cumplirse que $s_x = s_y$ (escalamiento uniforme) de lo contrario se habla de un escalamiento diferencial. La transformación de escalamiento puede expresarse con las siguientes multiplicaciones:

$$x' = s_x x \qquad y' = s_y y \text{ ……. (4)}$$

Nótese que estas fórmulas pueden ser expresadas en forma matricial, obteniéndose

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$P' = SP \text{ …….....(5)}$$

- **Rotación**

Los puntos también pueden ser rotados un ángulo θ *c*on respecto al origen. Una rotación se define matemáticamente como, según [4]:

$$x' = x\cos(\theta) - y\,sen(\theta) \text{ , } y' = xsen(\theta) - y\cos(\theta) \text{ ……..... (6)}$$

En forma matricial, tenemos

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -sen(\theta) \\ sen(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$P' = RP \text{ …….....(7)}$$

Coordenadas Homogéneas y Representación matricial de Transformaciones 2D

Las representaciones matriciales obtenidas hasta ahora para traslación, escalamiento y rotación son, respectivamente:

$$P' = T + P \text{ …....…….. ..(8)}$$

$$P' = SP \text{ ……....…….. ..(9)}$$

$$P' = RP \text{ …….....…….....(10)}$$

Como se puede notar en estas ecuaciones, la traslación es tratada de una forma diferente, como una adición, mientras que las otras dos transformaciones se expresan con multiplicaciones. Esto supone un problema a la hora de expresar el conjunto de transformaciones en una forma genérica y consistente. Para resolver este problema se expresan los puntos en un sistema de coordenadas homogéneas, de forma que las tres transformaciones puedan ser tratadas como multiplicaciones.

Dado que los puntos 2D se representan como vectores columna de 3 elementos, las matrices de transformación deben ser de orden 3x3. Las ecuaciones de traslación (1) pasan a ser una matriz de 3x3 en coordenadas homogéneas

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{…….....….....(11)}$$

La ecuación (11) puede ser expresada también como:

$$P' = T\left(d_x, d_y\right) P \quad \text{….....……(12)}$$

donde

$$T\left(d_x, d_y\right) = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \quad \text{……. ……. (13)}$$

Por otro lado, puede verificarse con facilidad que la transformación inversa de una traslación $T\left(d_x, d_y\right)$ no es más que $T^{-1}\left(d_x, d_y\right) = T\left(-d_x, -d_y\right)$.

Un procedimiento similar al efectuado con la traslación puede aplicarse al escalamiento, obteniendo una nueva representación matricial de la ecuación (5), de la forma siguiente

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{.....(14)}$$

Definiendo

$$S\left(s_x, s_y\right) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{.....(15)}$$

se tiene que

$$P' = S\left(s_x, s_y\right) P \quad \text{.....(16)}$$

Por otra parte, puede comprobarse que la inversa de un escalamiento $S\left(s_x, s_y\right)$ es $S^{-1}\left(s_x, s_y\right) = S\left(\frac{1}{s_x}, \frac{1}{s_y}\right)$ con $s_x \neq 0$, $s_y \neq 0$. Esto puede verificarse calculando la inversa de la matriz correspondiente. Se notar que pueden lograrse efectos de reflexión al utilizar valores negativos para s_x y/o s_y .

Similarmente, las ecuaciones de rotación (6) pueden ser representadas como

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -sen(\theta) & 0 \\ sen(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{.....(17)}$$

Donde

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -sen(\theta) & 0 \\ sen(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{..... ..... (18)}$$

obteniéndose

$$P' = R(\theta)P \quad \text{.....(19)}$$

Se sabe que, en general, la multiplicación de matrices no es conmutativa. Sin embargo, al aplicar transformaciones fundamentales de traslación, escalamiento y rotación se dan casos especiales donde el producto de matrices es conmutativo (dados A, B dos matrices cuadradas y se verifica que $AB = BA$). Una matriz de traslación seguida de otra matriz de traslación pueden conmutarse sin afectar el resultado. De forma semejante, una matriz de escalamiento seguida de otra matriz de escalamiento pueden multiplicarse en cualquier orden, así como una matriz de rotación seguida de otra matriz de rotación son conmutativas.

Otro caso donde la multiplicación de este tipo de matrices es conmutativa corresponde a tener una matriz de rotación y otra de escalamiento uniforme ($s_x = s_y$). En estos casos no es necesario preocuparse por el orden en la manipulación de las matrices, según [6].

Representación matricial de las Transformaciones en 3D

La representación de las transformaciones en 2D como matrices de 3x3 tiene un equivalente para las transformaciones 3D, las cuales son representadas como matrices de 4x4. Para permitir esto, el punto (x, y, z) será

representado en coordenadas homogéneas como (wx, wy, wz, w), con $w \neq 0$, según [6].

- **Traslación**

La matriz de traslación en 3D es una simple extensión de 2D:

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{..... (20)}$$

Al multiplicar esta matriz por el vector de puntos $(x, y, z, 1)$ queda:

$$T(d_x, d_y, d_z) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + d_x \\ y + d_y \\ z + d_z \\ 1 \end{bmatrix} \text{.. .....(21)}$$

- **Escalamiento**

La matriz de escalamiento es similarmente extendida:

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{..... (22)}$$

y al multiplicarla por el vector de puntos, queda:

$$S(s_x, s_y, s_z) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} xs_x \\ ys_y \\ zs_z \\ 1 \end{bmatrix} \text{.....(23)}$$

- **Rotación:**

En 3D, una rotación con respecto al eje z es:

$$Rz(\theta)=\begin{bmatrix}\cos(\theta) & -sen(\theta) & 0 & 0\\ sen(\theta) & \cos(\theta) & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \quad \text{..... (24)}$$

La matriz de rotación con respecto al eje x es:

$$Rx(\theta)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos(\theta) & -sen(\theta) & 0\\ 0 & sen(\theta) & \cos(\theta) & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \quad \text{.....…….. (25)}$$

La matriz de rotación con respecto al eje y es:

$$Ry(\theta)=\begin{bmatrix}\cos(\theta) & 0 & sen(\theta) & 0\\ 0 & 1 & 0 & 0\\ -sen(\theta) & 0 & \cos(\theta) & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \quad \text{.....……(26)}$$

2.4. Proyecciones.

Una proyección es una transformación matemática que convierte puntos representados en n dimensiones en puntos representados en $n-i$ dimensiones, es decir:

$$\text{Proy}: \mathbb{R}^n \rightarrow \mathbb{R}^{n-i},\ i=1,...,n-1$$

Solamente nos interesa las proyecciones del espacio 3D a 2D, de forma que al plano $2D$, donde vamos a proyectar los objetos $3D$, se le conoce como plano de proyección (PP).

Supongamos un punto P tridimensional y un plano PP, además de otro punto del espacio al que llamaremos centro de proyección (CP) o punto de vista (PV).

Si trazamos una recta que inicie del CP, pase por P y corte al PP, dicha recta se llama Visual y a la intersección de la visual con el plano de proyección se le conoce como proyección del punto P.

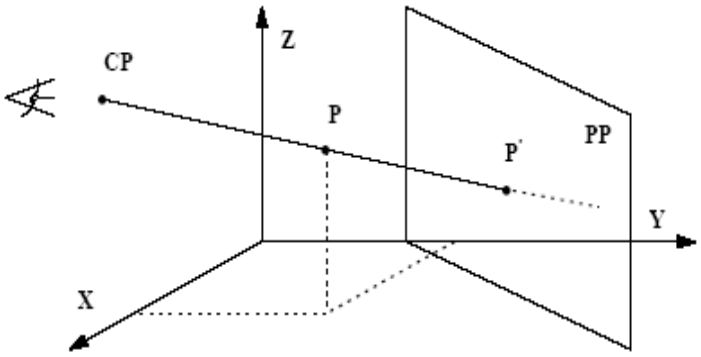


Fig. Nº 01: Elementos de la Proyección.

Tipos de Proyecciones.

Ya se mencionó que la proyección parte de un centro de proyección. Dependiendo de la distancia entre el centro de proyección y el plano, las proyecciones se dividen en dos, según [7]:

a) Perspectiva, si la distancia es finita;

b) Paralela, si la distancia es infinita.

Proyección Perspectiva

El centro de proyección ocupa un punto real del espacio llamado propio.

A esta proyección se le conoce también como proyección central. Esta clase de proyección guarda gran similitud con la visión humana y la fotografía.

El tamaño de una proyección perspectiva varía inversamente a la dirección del objeto con el centro de proyección, así pues la proyección perspectiva no es útil para captar la forma y medidas exactas de un objeto. Se puede decir que la proyección perspectiva presenta varios beneficios, tales como: proveer una visión realista y por ende una sensación de tridimensionalidad; por esto la proyección perspectiva es usada en diversos campos de investigación en los cuales se buscan que sus realizaciones sean realistas, estos son: publicidad, arquitectura, diseño, ingeniería y el ámbito artístico, según [7].

Proyecciones Paralelas.

El centro de proyección no es un punto real del espacio llamado (impropio) que se considera situado en el infinito. Todas las visuales son paralelas. Como en este tipo de proyección no tiene mucho sentido hablar de CP lo que se da es una dirección de proyección (DP), que es un vector que nos indica cual es la dirección de las visuales. A esta proyección también se le llama proyección cilíndrica.

Las proyecciones paralelas se clasifican a su vez, según la dirección de proyección en dos tipos, según [7]:

Proyección Oblicua: La dirección de proyección forma con el plano de proyección un ángulo menor de 90º y mayor de 0º

Proyección Ortogonal: También llamada proyección ortográfica: En estas proyecciones la dirección de proyección es perpendicular al plano de proyección.

2.5. Representación Tridimensional de Objetos

Se entiende por objeto tridimensional (Objeto 3D) toda aquella figura representable gráficamente que puede o no encerrar un volumen, es decir, puede ser representada desde dos puntos de vista: como un todo (sólido) que encierra un volumen o como un conjunto de Superficies que lo acotan.

En virtud de estas dos posibilidades de descripción se habla de modelado de sólidos como la representación de volúmenes completamente encerrados por Superficies y que permiten definir los conceptos dentro y fuera e incluso la intersección de diversos objetos.

La otra forma de representación de objetos es mediante el modelado de Superficies, que utilizan la descripción de las Superficies que lo delimitan pero no necesariamente encierran un volumen.

Ambas estrategias se encuentran relacionados y se emplean conjuntamente para la definición de objetos en tres dimensiones, sin embargo, el modelado de sólidos se encuentra limitado a objetos que encierran completamente un volumen , mientras que el modelado de Superficies se pueden utilizar tanto para acotar un volumen como para definir una frontera .

A continuación se da una breve descripción de la representación paramétrica de Superficies, prestando especial atención a la representación de Bézier.

Modelado de Superficies

En este tipo de representación se proporcionan información sobre el contorno del objeto, que es lo que un observador va ha poder ver.

Las tres representaciones más comunes de superficie tridimensional son: Superficies de mallas poligonales, las ecuaciones implícitas y las Superficies paramétricas, según [4].

Mallas Poligonales

Una malla poligonal no es más que un conjunto de puntos, aristas y caras planas que se usan para delimitar una región del espacio. Como ejemplo en la figura 2 se esta representado un cilindro como un conjunto de polígonos

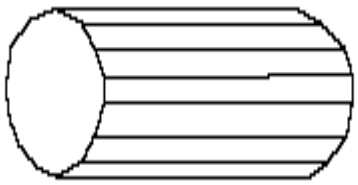


Fig. Nº 02: Cilindro representado por una malla poligonal

En la actualidad este método de representación es el más extendido a la hora de generar una imagen. La mayor parte del hardware grafico esta diseñado para procesar este tipo de representación (sobre todo mallas de triángulos) debido a que los algoritmos existentes para la generación de imágenes que usan este tipo de representación tienen bajo coste computacional comparado con otros tipos de representaciones; de hecho a la hora de generar una

imagen de un objeto modelado en cualquier otra representación, es usual transformarlo a una malla de polígonos para hacer cálculos de iluminación. Como contrapartida, una alta definición requiere una gran cantidad de polígonos para representar el objeto (especialmente en el caso de Superficies curvas) y por tanto una gran cantidad de memoria.

Ecuaciones Implícitas

En esta representación se usa una ecuación en forma implícita de segundo grado $f(x, y, z) = 0$ para definir la superficie, donde f es un polinomio cuadrático en x, y, z .

Este método tiene dos desventajas muy importantes:

Por una parte existen pocas Superficies que se pueden representar de forma implícita. A esto hay que añadir que para calcular cada punto de una superficie en esta representación pueden ser necesarios muchos cálculos, exigiendo un alto costo computacional.

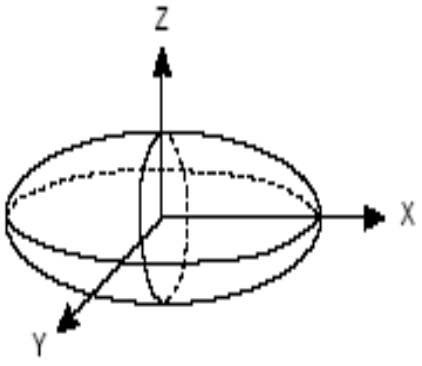


Fig. Nº 03: Elipsóide modelado de forma implícita.

Curvas y Superficies Paramétricas

Numerosas aplicaciones de representación y modelado de objetos en computación gráfica requieren el uso de curvas y Superficies con una expresión matemática precisa. Éstas suelen adoptar la forma paramétrica.

Una curva es el resultado de ajustar un conjunto dado de puntos geométricos, que junto con una base de funciones paramétricas determinan fuertemente su geometría. La expresión general que permite calcular un punto de una curva paramétrica es:

$$\vec{P}(t) = \sum_{i=0}^{n} P_i(t)\vec{P}_i$$

Donde $\vec{P}(t)$ es el punto de la curva, $\vec{P}_i(t)$ son las funciones (generalmente polinómicas) dependiendo del parámetro t que se usaran para interpolar los puntos de control y $\vec{P}_i$ es el conjunto de puntos de control que definen la curva, según [8].

Como extensión de las curvas, se define las Superficies paramétricas mediante la siguiente expresión:

$$S(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} P_{ij}(u,v)\vec{P}_{ij}$$

Donde $S(u,v)$ es el punto sobre la superficie, $P_{ij}(u,v)$ son las funciones de interpolación y $\vec{P_{ij}}$ son los puntos de control.

La representación paramétrica tienen una serie de ventajas sobre las mallas poligonales; entre ellas, cabe destacar que permiten obtener una

representación exacta a una superficie curva a partir de un conjunto pequeño de puntos de control, al contrario que el caso de la representación poligonal, donde para obtener se requiere un gran numero de polígonos y por tanto, de puntos.

Además permiten modificar la forma de la superficie con suavidad modificando la posición de uno de sus puntos de control.

La representación paramétrica más conocida y de uso más extendido son las NURBS (Non Uniform Racional B-Splines) y las curvas, superficie de Bézier.

Curvas y Superficies de Bézier

A continuación se presenta una breve introducción sobre las curvas y Superficies Bézier, así como las funciones base en las que se encuentran definidas, los polinomios de Bernstein, según [1,9].

A. Curvas de Bézier no Racionales

Las curvas de Bézier son una forma alternativa de representación de una curva polinómica. En lugar de la representación tradicional de un polinomio como: $\vec{P} = \vec{P}_0 + \vec{P}_1 t + \vec{P}_2 t^2 + ... + \vec{P}_n t^n$, se escoge otra representación en la que los puntos $\vec{P}^i$ adquieran sentido geométrico. A continuación se introduce, de manera constructiva, la definición de una curva de Bézier.

Dados los puntos $\vec{P}_0$ y $\vec{P}_1$, todos los punto $\vec{P}$ pertenecientes a $\mathbb{R}^3$ de la forma $\vec{\mathrm{P}} = \vec{\mathrm{P}}(t) = (1-t)\vec{\mathrm{P}}_0 + t\vec{\mathrm{P}}_1$, $t \in \mathbb{R}$ se denomina la línea recta a través de $\vec{P}_0$ y $\vec{P}_1$ .Para $t \in [0,1]$, el punto $\vec{P}$ estará entre los dos puntos. Por tanto

podemos considerar a $\vec{P}(t)$ como una función parametrica de $t \in [0,1]$ hacia un segmento de línea recta en $\mathbb{R}^3$.

Si realizamos la misma construcción para un grado más, o sea, creamos una nueva función de t tal que sea combinación baricéntrica de dos combinaciones baricéntricas lineales se obtiene:

$$\vec{P}_0^1(t) = (1-t)\vec{P}_0 + t\vec{P}_1$$

$$\vec{P}_1^1(t) = (1-t)\vec{P}_1 + t\vec{P}_2$$

Insertamos ambas funciones en una nueva combinación baricéntrica,

$$\vec{P}_0^2(t) = (1-t)\vec{P}_0^1 + t\vec{P}_1^1$$

y desarrollamos , obteniendo la parábola :

$$\vec{P}_0^2(t) = (1-t)^2\vec{P}_0 + 2t(1-t)\vec{P}_1 + t^2\vec{P}_2$$

Por tanto obtenemos una nueva función paramétrica, llamada parábola, que es una aplicación del intervalo $[0,1]$ a una curva de grado dos en el espacio.

Si generalizamos la construcción anterior para grado n, obtenemos:

$$\vec{P}_i^r(t) = (1-t)\vec{P}_i^{r-1}(t) + t\vec{P}_{i+1}^{r-1}(t) \ , r = 1,...,n \ , \ i = 0,...,n-r$$

Definimos la curva de Bézier de grado n, $\vec{P}^n$ como $\vec{P}_0^n(t)$

Vemos que esta curva queda perfectamente definida mediante su grado n, y su polígono de control, que son los puntos $\vec{P}_0, \vec{P}_1, ..., \vec{P}_n$.

Esta curva cumple las siguientes propiedades:

1. La curva pasa por el punto inicial y el punto final.

2. La tangente de la curva en los puntos $\vec{P}_0$ y $\vec{P}_n$ pasan, respectivamente, por los puntos $\vec{\mathrm{P}}_1$ y $\vec{P}_{n-1}$ si no coinciden con los anteriores y en general por los mas cercanos a ellos que no sean coincidentes.

3. La curva es continua y la derivada es una curva de grado $n-1$ en todo el dominio.

4. La curva es plana y esta incluida en el polígono convexo que forman sus puntos de control.

5. De las propiedades anteriores se deduce que la curva se acerca suavemente a los puntos interiores del polígono de control.

6. Se deduce también que si el polígono de control reside en un plano, la curva estará contenida en el mismo plano.

   En la siguiente figura puede observarse las propiedades de interpolación, continuidad y tangencias para curvas de Bézier de diversos grados.

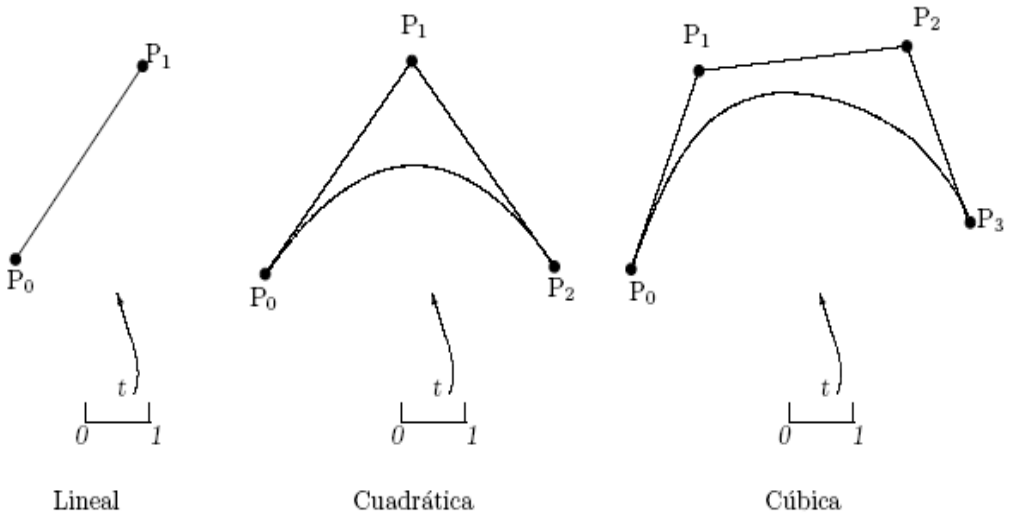


Fig. Nº 04: Curvas de Bézier de diversos grados

El método usado para introducir y definir a las curvas de Bézier es al mismo tiempo, una función recursiva que sirve para evaluarlas.

Existen otras formas de definir y evaluar este tipo de curvas.

- **Curvas de Bézier expresadas en términos de los polinomios de Bernstein.**

Se definen los polinomios de Bernstein como:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

Donde

$$\binom{n}{i} = \begin{cases} \dfrac{n!}{i!(n-i)!}, & 0 \le i \le n \\ 0 \quad , \text{ en otro caso} \end{cases}$$

Los polinomios de Bernstein cumplen las siguientes propiedades, siempre referidas al intervalo $[0,1]$:

a) $B_i^n(t) \geq 0$, $\forall t, \forall n$.

b) $B_i^n(t)$ posee exactamente un máximo se produce en $t = \frac{i}{n}$.

c) Recursividad.

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t) \text{ , con}$$

$$B_0^0(t) = 1 \text{ y } B_j^n(t) \equiv 0 \quad , \; j \notin \{0,...,n\}$$

d) El conjunto forma una partición de la unidad.

$$\sum_{i=0}^{n} B_i^n(t) \equiv 1$$

e) Derivada

$$\left[B_i^n(t)\right]' = n\left[B_{i-1}^{n-1}(t) - B_{i-1}^n(t)\right]$$

Dadas estas definiciones, podemos expresar una curva de Bézier $\vec{P}^n(t)$ como:

$$\vec{P}^n(t) = \vec{P}_0^n(t) = \sum_{i=0}^{n} B_i^n(t)\vec{P}_i \quad \text{..... (27)}$$

De la ecuación anterior se deduce que para calcular cada punto sobre la curva, se necesitan $n+1$ puntos de control $\vec{P}_i$ y $n+1$ funciones base B_i^n Para construir la curva completa, hay que calcular los puntos para todos los valores de t entre cero y 1.

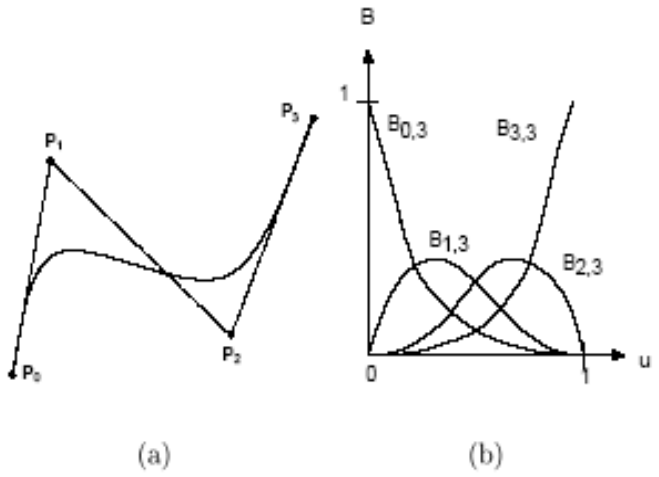


Fig. Nº 05: (a) Curva de Bézier de grado 3 (b) Polinomio de Bernstein de grado 3

En la figura (a) se muestra una curva Bézier de grado 3. Dado que es una curva de grado 3, se necesita 4 puntos de control para describirla. En la figura (b) se muestra la forma de los 4 polinomios de Bernstein de grado 3 para $0 \le t \le 1$. En este intervalo el valor que toma los polinomios es menor o igual a 1, siendo 1 solamente para B_0^3 y B_3^3 en los extremos del intervalo.

B. Curvas Bézier Racionales

La representación de curvas Bézier no Racionales no permite representar secciones cónicas de forma exacta (circunferencia, elipse, hipérbola). Esto supone una gran desventaja en herramientas CAD, donde estas secciones son ampliamente usadas.

Las Curvas de Bézier Racionales resuelven este problema representando una curva de Bézier en cuatro dimensiones y proyectándola sobre el espacio tridimensional.

La curva en 4D sería:

$$\vec{P}^{n}(t)=\sum_{i=0}^{n}B_i^n(t)\vec{P}_i\ ,\ 0\le t\le 1$$

Donde $\vec{P}_i=(w_i x_i, w_i y_i, w_i z_i, w_i)$ son los puntos de control de la curva. Proyectamos la curva dividiendo las tres primeras coordenadas de la ecuación anterior por $w_i \neq 0$, se obtiene la expresión:

$$\vec{P}^{n}(t)=\frac{\sum_{i=0}^{n}w_i B_i^n(t)\vec{P}_i}{\sum_{i=0}^{n}w_i B_i^n(t)}\ ,\ 0\le t\le 1$$

De este modo controlando el valor de los pesos w_i, podemos hacer que la curva se acerque o se aleje de los puntos de control, teniendo mayor control sobre la forma final de la curva.

En la figura se muestra un ejemplo de que como los pesos w_i afectan a la forma de la curva para un mismo conjunto de puntos de control. El caso $w_0=w_1=w_2=w_3=1$ representado en la figura (a) se corresponde con la curva Bézier no Racional. En la figura (b) se muestra el caso en que w_1 es mayor que el resto de los pesos, acercándose la curva al punto control $\vec{P}_1$. En la figura (c) se muestra el caso en que $w_1=w_2<1$ en este caso, la curva se aleja de los correspondientes puntos control, tomando una forma achatada respecto al caso no Racional.

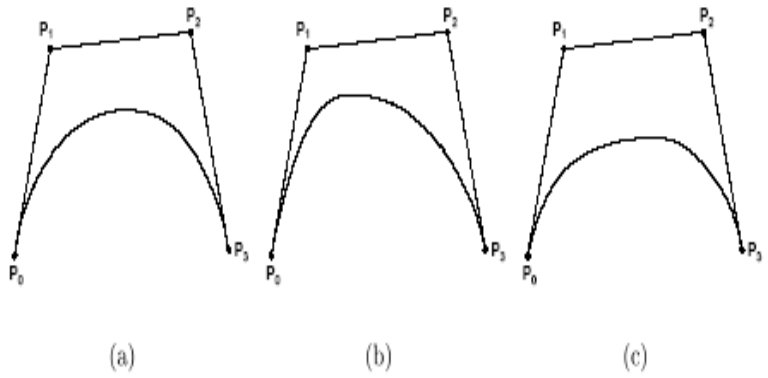


Fig. Nº 06: Curvas de Bézier Racionales para; (a) $w_0 = w_1 = w_2 = w_3$;

(b) $w_1 > (w_0 = w_2 = w_3)$ (c) $(w_1 = w_2) < (w_0 = w_3)$

C. Propiedades de las curvas de Bézier

1. La línea definida por los puntos de control es una primera aproximación a la forma de la curva .Esto implica que ningún recta puede intersectar a la curva más veces que a dicha línea.

2. Los extremos inicial y final de la curva coinciden con los puntos inicial y final de los puntos de control. es decir $\vec{P}(0) = \vec{P}_0$ y $\vec{P}(1) = \vec{P}_n$

3. Si definimos el operador diferencias en avance Δ como $\Delta\vec{P}_i = \vec{P}_{i+1} - \vec{P}_i$, obtenemos que la derivada de una curva de Bézier para $\Delta\vec{P}_i \in \mathbb{R}^3$ es :

$$\frac{d}{dt}\vec{P}^n(t) = n\sum_{i=0}^{n}\left[B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\right]\vec{P}_i = n\sum_{i=0}^{n-1}\Delta\vec{P}_i B_i^{n-1}(t) \quad (*)$$

Para las derivadas de orden superior definimos el operador diferencias en avance iterativo Δ^r como:

$$\Delta^r \vec{P}_i = \Delta^{r-1}\vec{P}_{i+1} - \Delta^{r-1}\vec{P}_i = \sum_{j=0}^{r}\binom{r}{j}(-1)^{r-j}\,\vec{P}_{i+j} \ldots\ldots\ldots\ldots\ldots\ldots\ldots(**)$$

A partir de (*) y (**) se deduce que la derivada de grado r de $\vec{P}^n(t)$ es:

$$\frac{d^r}{dt^r}\vec{P}^n(t) = \frac{n!}{(n-r)!}\sum_{i=0}^{n-r}\Delta^r \vec{P}_i B_i^{n-r}(t)$$

4. Los vectores tangentes a la curva en el comienzo y final de la curva son coincidentes a los vectores formados por los puntos de control $\vec{P}_1 - \vec{P}_0$ y $\vec{P}_n - \vec{P}_{n-1}$.

   Es decir

$$\frac{d}{dt}\vec{P}^n(0) = n\left[\vec{P}_1 - \vec{P}_0\right]$$

$$\frac{d}{dt}\vec{P}^n(1) = n\left[\vec{P}_n - \vec{P}_{n-1}\right]$$

5. La curva siempre esta contenido por el polígono mínimo convexo formado por los puntos de control

6. La otra forma de expresar las curvas de Bézier es mediante matrices en la siguiente forma :

$$\vec{P}^n(t) = \sum_{i=0}^{n}\vec{P}_i B_i^n(t) = \left[\vec{P}_0,\ldots,\vec{P}_n\right]\begin{bmatrix} B_0^n(t) \\ \vdots \\ B_n^n(t) \end{bmatrix}$$

$$\vec{P}^{n}(t)=\left[\vec{P}_0,...,\vec{P}_n\right]\begin{bmatrix} m_{00} & . & . & m_{0n} \\ . & & & . \\ . & & & . \\ m_{n0} & . & . & m_{nn} \end{bmatrix}\begin{bmatrix} t^0 \\ \vdots \\ t^n \end{bmatrix}$$

Donde los coeficientes m_{ij} valen:

$$m_{ij}=(-1)^{j-i}\binom{n}{j}\binom{j}{i}$$

D. Superficies de Bézier no Racionales

Se denomina superficie de Bézier no Racional asociada a $(n+1)(m+1)$ puntos $\vec{P}_{00},\vec{P}_{01},...,\vec{P}_{0m},\vec{P}_{10},\vec{P}_{11},...,\vec{P}_{1m},...,\vec{P}_{n0},\vec{P}_{n1},...,\vec{P}_{nm}$ de $\mathbb{R}^3$, a la superficie parametrizada, definida para $(u,v)\in[0,1]x[0,1]$, cuyos puntos $S(u,v)$ vienen dadas mediante la expresión

$$S(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}B_i^n(u)B_j^m(v)\vec{P}_{ij}$$

Siendo $B_i^n(u)=\binom{n}{i}u^i(1-u)^{n-i}$ y $B_j^m(v)=\binom{m}{j}v^j(1-v)^{m-j}$, para $i=0,...,n$, $j=0,...,n$ los polinomios de Bernstein de grado n y m.

La sucesión de puntos $\vec{P}_{00},\vec{P}_{01},...,\vec{P}_{nm}$, que determina una superficie de Bézier se denomina B-malla o malla de Bézier.

Para estudiar las propiedades de las Superficies de Bézier lo expresaremos de otra forma.

Primero se tiene que

$$S(u,v)=\sum_{j=0}^{m}\left(\sum_{i=0}^{n}B_i^n(u)\vec{P}_{ij}\right)B_j^m(v)=\sum_{j=0}^{m}\vec{P}_jB_j^m(v)$$

Donde $\vec{P}_j(u)=\sum_{i=0}^{n}B_i^n(u)\vec{P}_{ij}$ y conviene notar que, para cada j, $\vec{P}_j(u)=\sum_{i=0}^{n}B_i^n(u)\vec{P}_{ij}$, con $u\in[0,1]$, es una curva Bézier en el espacio tridimensional cuyos puntos de control son $\vec{P}_{0j},\vec{P}_{1j},...,\vec{P}_{nj}$, y que, a su vez, al escribir $S(u,v)=\sum_{j=0}^{m}\vec{P}_j(u)B_j^m(v)$, entonces, fijo el valor de u, lo que se observa es la expresión de una curva de Bézier cuyos puntos de control , que dependen de los valores de u, son $\vec{P}_0(u),...,\vec{P}_m(u)$.

Análogamente, $S(u,v)$ se puede también escribir de la forma

$$S(u,v)=\sum_{i=0}^{n}\left(\sum_{j=0}^{m}B_j^m(v)\vec{P}_{ij}\right)B_i^n(u)=\sum_{i=0}^{n}\vec{D}_jB_i^n(u)$$

Donde $\vec{D}_j(v)=\sum_{j=0}^{m}B_j^m(v)\vec{P}_{ij}$ y sobre esta otra expresión de la superficie se puede hacer comentarios iguales a los anteriores.

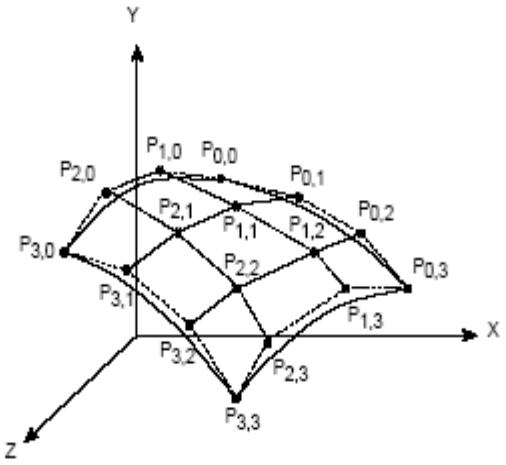


Fig. Nº 07: Ejemplo de superficie de Bézier no Racional

E. Superficies de Bézier Racionales

De la necesidad de representar cilindros, conos, esferas, etc. surge la definición de las Superficies Bézier Racionales, de forma paralela a como ocurre con las curvas de Bézier Racionales .Una superficie de Bézier Racional puede ser definido como:

$$S(u,v)=\frac{\sum_{i=0}^{n}\sum_{j=0}^{m}B_i^n(u)B_j^m(v)w_{i,j}P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m}B_i^n(u)B_j^m(v)w_{i,j}},\quad 0\le u,v\le 1$$

Siendo los $B_i^n(u)=\binom{n}{i}u^i(1-u)^{n-i}$ y $B_j^m(v)=\binom{m}{j}u^j(1-v)^{m-j}$ para $i=0,...,n$, $j=0,...,m$ los polinomios de Bernstein de grado n, y grado m respectivamente, $\{P_{ij}\}$ los puntos de control y $\{w_{ij}\}$ son los pesos, que controla la proximidad de la superficie a los distintos puntos de control.

F. Propiedades de las Superficies de Bézier

1. Cualquier superficie de Bézier $S(u,v)$, cuando $(u,v)\in[0,1]x[0,1]$, se encuentra en la envolvente convexa de la B-malla que la define.

2. Las curvas de una superficie de Bézier obtenidas para los valores $u=0,u=1$ y $v=0,v=1$,son curvas Bézier cuyos polígonos de control son los polígonos que delimitan la B-malla, es decir, los polígonos asociados a los conjuntos de puntos $\left\{\vec{P}_{00},\vec{P}_{01},...,\vec{P}_{0m}\right\}$, $\left\{\vec{P}_{n0},\vec{P}_{n1},...,\vec{P}_{nm}\right\}$, $\left\{\vec{P}_{00},\vec{P}_{10},...,\vec{P}_{n0}\right\}$, $\left\{\vec{P}_{0m},\vec{P}_{1m},...,\vec{P}_{nm}\right\}$, respectivamente. Por ello la superficie pasa por los puntos $\vec{P}_{00}$, $\vec{P}_{n0}$, $\vec{P}_{m0}$ y $\vec{P}_{nm}$, que determinan las esquinas de la malla de control .

3. Las derivadas parciales de orden k, respecto de las variables u y v, en un punto de la superficie de Bézier $S(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}B_i^n(u)B_j^m(v)\vec{P}_{ij}$ son:

$$\frac{\partial^k S(u,v)}{\partial u^k}=\frac{n!}{(n-k)!}\sum_{j=0}^{m}\left[\sum_{i=0}^{n-k}B_i^{n-k}(u)\Delta^{k,0}\vec{P}_{ij}\right]B_j^m(v)\ ;\ k=1,...,n$$

$$\frac{\partial^k S(u,v)}{\partial v^k}=\frac{m!}{(m-k)!}\sum_{i=0}^{n}\left[\sum_{j=0}^{m-k}B_j^{m-k}(v)\Delta^{0,k}\vec{P}_{ij}\right]B_i^n(u)\ ;\ k=1,...,m$$

En particular, en el contorno de la superficie

$$\frac{\partial S(0,v)}{\partial u^k}=\frac{n!}{(n-k)!}\sum_{j=0}^{m}\left[\Delta^{k,0}\vec{P}_{0j}\right]B_j^m(v)$$

$$\frac{\partial S(1,v)}{\partial u^k} = \frac{n!}{(n-k)!}\sum_{j=0}^{m}\left[\Delta^{k,0}\vec{P}_{n-k,\,j}\right]B_j^m(v)$$

$$\frac{\partial S(u,0)}{\partial v^k} = \frac{m!}{(m-k)!}\sum_{i=0}^{n}\left[\Delta^{0,k}\vec{P}_{i0}\right]B_i^n(u)$$

$$\frac{\partial S(u,1)}{\partial v^k} = \frac{m!}{(m-k)!}\sum_{i=0}^{n}\left[\Delta^{0,k}\vec{P}_{i,m-k}\right]B_i^n(u)$$

Donde se denota por $\Delta^{k,0}\vec{P}_{ij}$ a la diferencia progresiva de orden k respecto del primer subíndice i, es decir se calcula, por recurrencia, a partir de los puntos de control

$$\Delta^{1,0}\vec{P}_{ij} = \vec{P}_{i+1,j} - \vec{P}_{ij}$$

$$\Delta^{2,0}\vec{P}_{ij} = \Delta^{1,0}\left(\Delta^{1,0}\vec{P}_{ij}\right) = \vec{P}_{i+2,j} - 2\vec{P}_{i+1,j} + \vec{P}_{ij}$$

.

.

.

$$\Delta^{k,0}\vec{P}_{ij} = \Delta^{1,0}\left(\Delta^{k-1,0}\vec{P}_{ij}\right)$$

Análogamente, se denota por $\Delta^{0,k}\vec{P}_{ij}$ a la diferencia progresiva de orden k respecto al segundo subíndice j.

4. Los planos tangentes, si existen , en los puntos esquinas del contorno de una superficie de Bézier están determinados por los puntos de la B-malla que se define. Es decir:

El plano tangente en el punto $\vec{P}_{00}$ es el determinado por los puntos $\vec{P}_{00}$, $\vec{P}_{01}$ y $\vec{P}_{10}$.

El plano tangente en el punto $\vec{P}_{0m}$ es el determinado por los puntos $\vec{P}_{0m}$, $\vec{P}_{1m}$ y $\vec{P}_{0m-1}$.

El plano tangente en el punto $\vec{P}_{n0}$ es el determinado por los puntos $\vec{P}_{n0}$, $\vec{P}_{n1}$ y $\vec{P}_{n-10}$.

El plano tangente en el punto $\vec{P}_{nm}$ es el determinado por los puntos $\vec{P}_{nm}$, $\vec{P}_{n-1m}$ y $\vec{P}_{nm-1}$.

G. Curvas y Superficies B-splines y NURBS

A la vez que por su sencillez las curvas y Superficies de Bézier tienen un uso muy extendido en la computación grafica también presentan ciertos inconvenientes entre los que hay que destacar la dependencia de la forma de la curva con la totalidad de los puntos de control para los que se encuentra definida. Como alternativa se desarrollan otras definiciones de curvas paramétricas, las B-splines que están construidos por segmentos de curvas cuyos coeficientes polinomiales sólo dependen de algunos puntos de control. Dentro de las B-splines hay que destacar las NURBS (Non Uniform Racional B-spline) como caso particular.

En la siguiente sección se presenta una introducción de las curvas y Superficies B-splines, según [9].

a) Curvas B-splines no Racionales

Una curva B-spline, $C(t)$, se define como una curva polinomial parametrica a trozos en una base de funciones dadas, donde cada trozo es un segmento de curva delimitado por nudos (knot). Un nudo es el valor que toma el parámetro t en los puntos de unión de dos segmentos consecutivos y que, junto con los dos valores paramétricos extremos $t=0$ y $t=1$, se agrupan formando una secuencia no decreciente de números reales y positivos llamados vector de nudos (knot vector), cuya expresión genérica es:

$$U=\{t_0,t_1,...,t_m\}$$

Cada uno de los intervalos $[t_i,t_{i+1})$ no nulos definidos por dos nudos consecutivos es denominado intervalo de nudos (knot span). Si $t_i=t_{i+1}$, la longitud del intervalo de nudos es nula y además se dice que la multiplicidad de un nudo t_i es $(s+1)$. Si $t_i=t_{i+1}=...=t_{i+s}$. Normalmente los nudos extremos t_0 y t_m para una curva de grado p tienen una multiplicidad $(p+1)$. La multiplicidad de un nudo t_i esta directamente relacionado con el grado de continuidad con el que se unen dos segmentos adyacentes de la curva en ese valor paramétrico dado t_i, condicionando así la forma de la curva B-spline al tipo de vector de nudos escogido.

Atendiendo a la forma que puede tomar este vector de nudos existen diversas clasificaciones:

- Vector de nudos uniformes si el espaciado es un valor constante d:

  $t_i - t_{i-1} = d \ , \ \forall i \in [p, m-p-1]$

- Vector de nudos no uniforme si el espaciado entre al menos dos nudos consecutivos es distinto de la constante d :

  $t_i - t_{i-1} \neq d \ , \ \forall i \in [p, m-p-1]$

Una curva B-spline no Racional de grado p asociado al conjunto de puntos del plano o espacio $\{\vec{P}_0, \vec{P}_1, ..., \vec{P}_n\}$ y al conjunto de nodos U es una combinación lineal de los puntos $\vec{P}_i$ en la que los coeficientes son funciones B-Splines de grado p, es decir $C(t) = \sum_{i=0}^{n} N_{i,p}(t)\vec{P}_i$

Donde $\{\vec{P}_i\}$ representa el conjunto de puntos de control y $\{N_{i,p}\}$ son las funciones base B-splines de grado p.

Al polígono determinado por el conjunto $\{\vec{P}_0, \vec{P}_1, ..., \vec{P}_n\}$ se le denomina polígono de control o polígono spline.

b) Curvas B-splines Racionales

Las curvas B-splines no Racionales anteriormente descritas se encuentran limitadas para la representación de ciertos tipos de curvas cerradas como círculos, elipses hipérbolas, etc. para los que es preciso introducir las curvas B-splines Racionales, definidas en una base de funciones Racionales, que a su vez son cociente de dos funciones Racionales, típicamente polinomios:

$$C(t) = \frac{\sum_{i=0}^{n} N_{i,p}(t) w_i \vec{P_i}}{\sum_{i=0}^{n} N_{i,p}(t) w_i}, \ 0 \le t \le 1$$

Donde los $\{\vec{P_i}\}$ son los puntos de control, los $\{w_i\}$ son denominados pesos y representan cuánto control ejerce un punto $\vec{P_i}$ dado sobre la curva y las $\{N_{i,p}(t)\}$ son las funciones base B-spline de grado p.

Si el vector de nudos para el que se define una curva B-spline Racional es no uniforme el tipo de curva descrita recibe el nombre de NURBS**.**

Dados un vector de nudos $U = \left\{ \underbrace{t_0,...,t_0}_{p+1}, t_{p+1},...,t_{m-p-1}, t_m, \underbrace{t_m,...,t_m}_{p+1} \right\}$. Los B-Splines de grado p son definidos como la siguiente función:

Para $p = 0$

$$N_{i,0}(t) = \begin{cases} 1 \ , \ si \ \ t_i \le t < t_{i+1} \\ 0 \ , \ en \quad otro \ caso \end{cases}$$

y para $p > 0$ y $i = 0,..., n$

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)$$

Donde, si algún denominador es cero porque hay nodos múltiples, se adopta el convenio de anular el termino correspondiente en la expresión de $N_{i,p}(t)$.

Propiedades de los B-splines

- $$\begin{cases} N_{i,p}(t) = 0 \ , \ \forall t \notin \left[t_i , t_{i+p+1} \right] \\ N_{i,p}(t) \geq 0 \ , \ \forall t \in \left[t_i , t_{i+p+1} \right] \end{cases}$$

  El intervalo $\left[t_i , t_{i+p+1} \right]$ se denomina soporte de B-Spline $N_{i,p}(t)$

- Para una intervalo arbitrario de nodo $\left[t_r , t_{r+1} \right)$ esta contenido en todos los soportes $\left[t_i , t_{i+p+1} \right]$ para los que se verifica que $r - p \leq i \leq r$. Además $\sum_{i=r-p}^{r} N_{i,p}(t) = 1, \ \forall t \in \left[t_r , t_{r+1} \right)$.

- Los B-splines de grado p son funciones polinomiales a trozos de orden menor o igual a $p+1$.

- Todas las derivadas de $N_{i,p}(t)$ existen en el interior del intervalo de nodo .En un nodo $N_{i,p}(t)$ es $p-k$ veces continuamente diferenciable, donde k es la multiplicidad de los nodos. Además, para cualquier conjunto de nodos , en los puntos donde es derivable se verifica que:

$$\frac{d}{dt}\left[N_{i,p}(t) \right] = (p) \left[\frac{1}{t_{i+p} - t_i} N_{i,p-1}(t) - \frac{1}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t) \right]$$

Propiedades de las curvas NURBS

- Una primera aproximación de la forma que toma la curva es la línea que une los puntos de control, denominada polígono de control.

- En una curva NURBS, para un valor $t \in [t_i, t_{i+1})$ $\forall i \in [p, m-p-1]$, la curva está contenida en el polígono mínimo convexo definido por el conjunto de los puntos de control $\{\vec{P}_{i-p}, ..., \vec{P}_i\}$.

- Normalmente la multiplicidad de los nodos de los extremos en el vector de los nodos es superior al grado de la curva en una unidad, por eso la curva NURBS interpola los puntos extremos $C(0) = \vec{P}_0$ y $C(1) = \vec{P}_n$.

- No existe ningún plano que intersecte más veces la curva que su polígono de control.

- Una curva NURBS es una generalización de las curvas Bézier para el caso particular en que el número de puntos de control coinciden con el grado de la curva y el vector de nodos toma la forma $U = \{0, ..., 0, 1..., 1\}$, es decir, no hay nudos interiores.

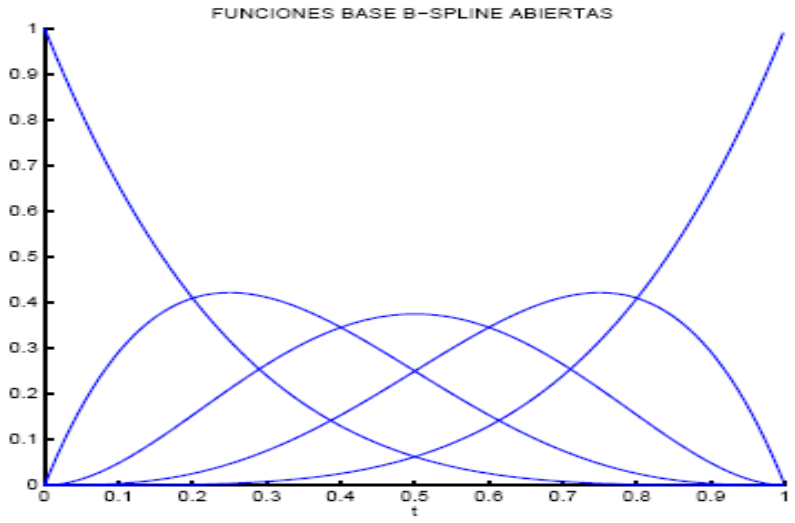


Fig. Nº 08 : Ejemplo de funciones Base B-Spline

c) Superficies B-spline

Una superficie B-spline es definido como:

$$S(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)\vec{P}_{ij}\ ,\ 0\le u,v\le 1$$

Donde $\{N_{i,p}(u)\}$ y $\{N_{j,q}(v)\}$ son las funciones base B-spline de grado p y q respectivamente definidos para los dos vectores nodos:

$$U=\left\{\underbrace{0,...,0}_{p+1},u_{p+1},...,u_{s-p-1},\underbrace{1,...,1}_{p+1}\right\}$$

$$V=\left\{\underbrace{0,...,0}_{q+1},v_{p+1},...,v_{s-p-1},\underbrace{1,...,1}_{q+1}\right\}$$

Y los puntos $P_{ij}=(x_{ij},y_{ij},z_{ij})$ constituyen los puntos de control.

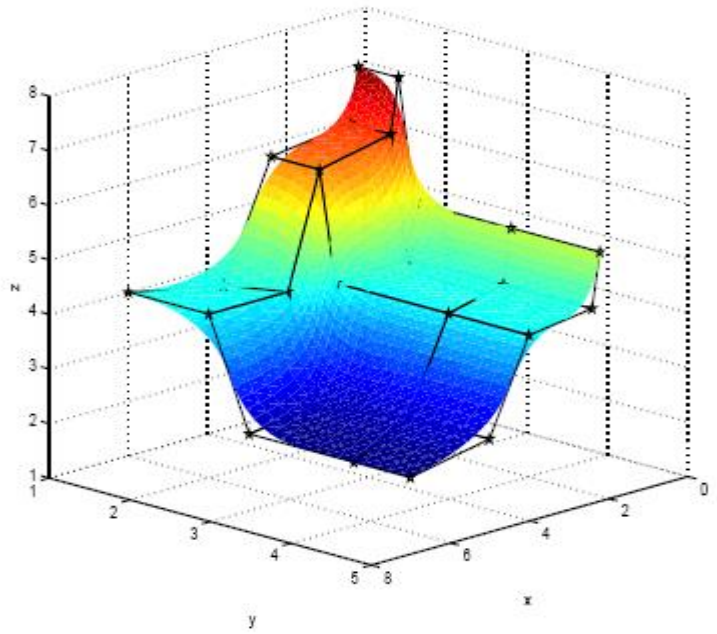


Fig. Nº 09: Ejemplo de Superficie B-Spline

d) Superficies NURBS

Una superficie NURBS Racional de grado p en la dirección u y de grado q en la dirección v viene dada por:

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{l} N_{i,p}(u) N_{j,q}(v) w_{ij} \vec{P}_{ij}}{\sum_{i=0}^{n}\sum_{j=0}^{l} N_{i,p}(u) N_{j,q}(v) w_{ij}}, \quad 0 \le u,v \le 1$$

Donde $S(u,v)$ es un punto de la superficie para unos valores dados de u y v, $\vec{P}_{ij}$ son los puntos de control que definen una malla, $\{N_{i,p}(u)\}$ y $\{N_{j,q}(v)\}$ son las funciones base B-splines de grado p y q respectivamente definidos para los dos vectores nodos:

$$U = \left\{ \underbrace{0,...,0}_{p+1}, u_{p+1},...,u_{s-p-1}, \underbrace{1,...,1}_{p+1} \right\}$$

$$V = \left\{ \underbrace{0,...,0}_{q+1}, v_{p+1},...,v_{s-p-1}, \underbrace{1,...,1}_{q+1} \right\}$$

Propiedades de las Superficies NURBS

- Una aproximación planar poliédrica, resultado de unir los diversos puntos de control a lo largo de las dos direcciones marcadas por los dos valores parametritos u y v , es una referencia de la forma final que va a tomar la superficie resultante.
- La superficie $S(u,v)$ está contenida en el polígono mínimo convexo definido por el conjunto de puntos de control.

- Como consecuencia de las condiciones de contorno impuestas para su definición, las Superficies Bézier interpolan los cuatro puntos en las esquinas: $S(0,0)=\vec{P}_{00}$, $S(1,0)=\vec{P}_{n0}$, $S(0,1)=\vec{P}_{0l}$ y $S(1,1)=\vec{P}_{nl}$.

- Contrario a lo que ocurre con las curvas NURBS, las Superficies NURBS pueden tener un número de intersecciones mayor que las que puede presentar su polígono de control con la misma recta.

- De forma similar a como ocurre con las curvas, las Superficies NURBS Racionales son un caso particular de las Superficies NURBS cuando los dos vectores de nudos para los que se definen son :

  $U=\{0,...,0,1,...,1\}$ y

  $V=\{0,...,0,1,...,1\}$.

2.6. Generación de Malla

Se entiende por generación de malla al proceso por el cual a partir de la geometría que define al modelo y de unas indicaciones dadas por el usuario, se procede a la construcción automática de una malla que define adecuadamente a esa geometría con los condicionantes del programa de análisis a usar.

Para la correcta obtención de esta malla hay que tener en cuanta una serie de factores que pueden venir dados por el usuario o que pueden calcularse automáticamente los cuales son:

- Tipos de elementos a Generar: Unidimensionales, triángulos, cuadriláteros, tetraedros, hexaedros.

- Malla estructurado o no estructurado
- Tamaño del elemento en cada zona
- Transición del tamaño del elementos entre zonas
- Comprobación de la calidad final de los elementos generados

En el siguiente apartado se comentara cada uno de estos factores y las posibles soluciones. También se describen los algoritmos usados en los diversos tipos de generación de mallas.

Tipos de elementos

Según las particularidades del análisis y aplicación a efectuar, puede ser necesario obtener elementos de distinto grado. Los elementos básicos, que pueden presentase en una generación de malla son:

- Elementos de barra de dos nodos
- Elementos de superficie triángulos de tres nodos o cuadriláteros de 4 nodos.
- Elementos de volumen tetraedros de 4 nodos o hexaedros de 8 nodos.

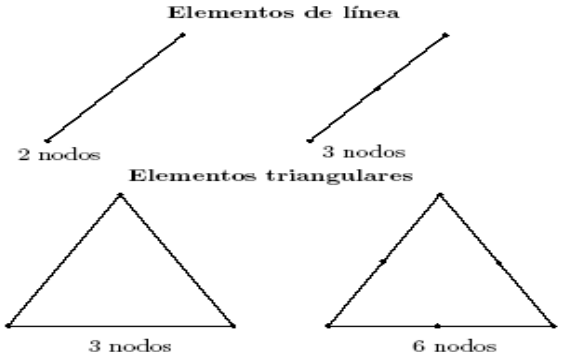


Fig. Nº 10: Elementos de línea y triangulares

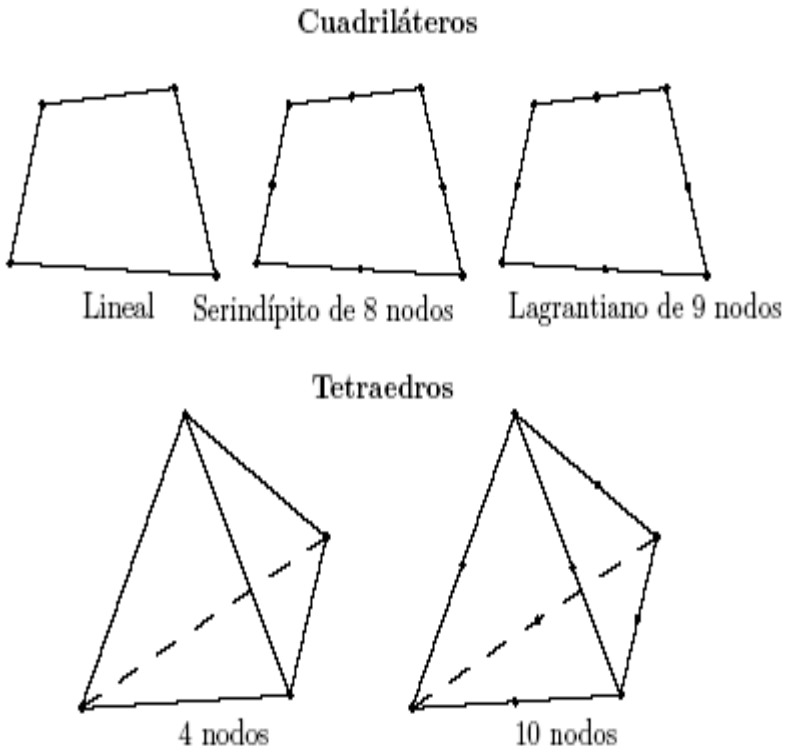


Fig. Nº 11: Elementos cuadriláteros y tetraedros

Si se desea elementos de mayor grado se pueden generar:

- Elementos de barra de tres nodos
- Elementos de superficie de triángulos de 6 nodos o cuadriláteros de 8 o 9 nodos.
- Elementos de volumen tetraedros de 10 nodos
- Elementos de volumen hexaedros de 20 o de 27 nodos

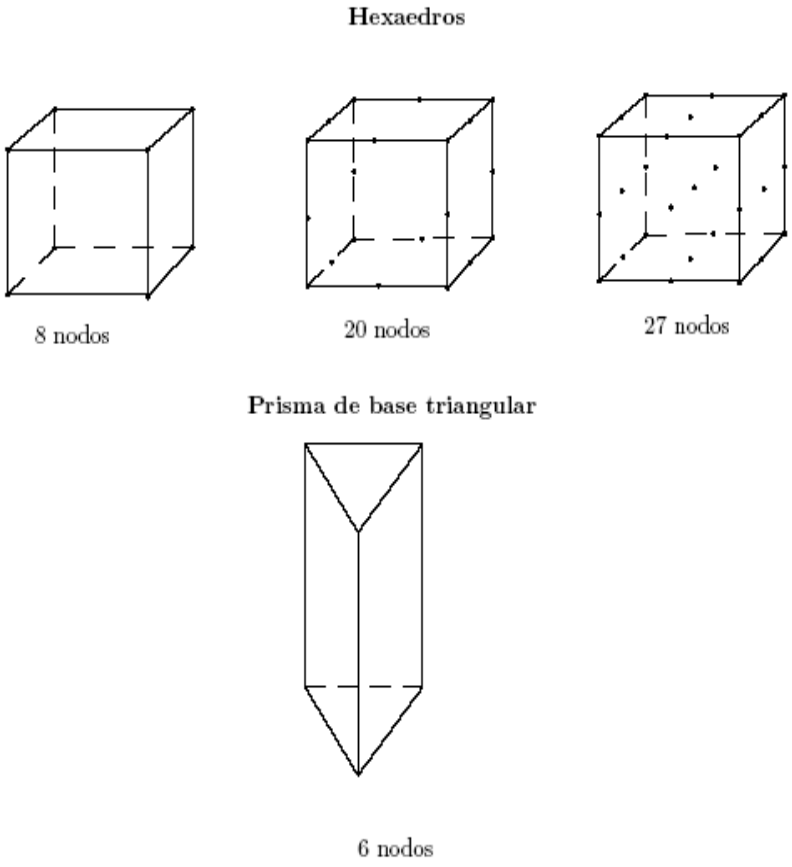


Fig. Nº 12: Elementos hexaédricos y prismas de base triangular.

Malla estructurada y no estructurada

Una malla es estructurada:

Si todos sus nodos interiores tienen el mismo número de elementos que lo contienen; o sea que el número de elementos que contienen ese nodo como una de sus conectividades es constante para todos los nodos del interior del dominio.

La característica de una malla de este tipo es la regularidad. Esto significa que se puede encontrar una submalla que se repite un número arbitrario de veces para formar la malla total.

Los algoritmos de generación de malla estructurada posicionan los nodos de manera ordenada en el interior de las Superficies o volúmenes .El grave inconveniente reside en que para poder ser generada, se debe restringir la topología de las entidades geométricas que componen el modelo. De alguna manera la regularidad y la repetición de la malla resultante debe estar ya reflejada en la composición de la geometría original.

Malla no estructurada:

Es cualquier malla que no cumpla la definición anterior. Los algoritmos de generación de malla no estructurada son más complejos y no tienen garantía absoluta de éxito pero pueden ser usados en modelos de geometría arbitrariamente compleja sin intervención del usuario en la definición de los parámetros de mallado.

Es posible mezclar malla estructurada y no estructurada en un mismo modelo, ver [5].

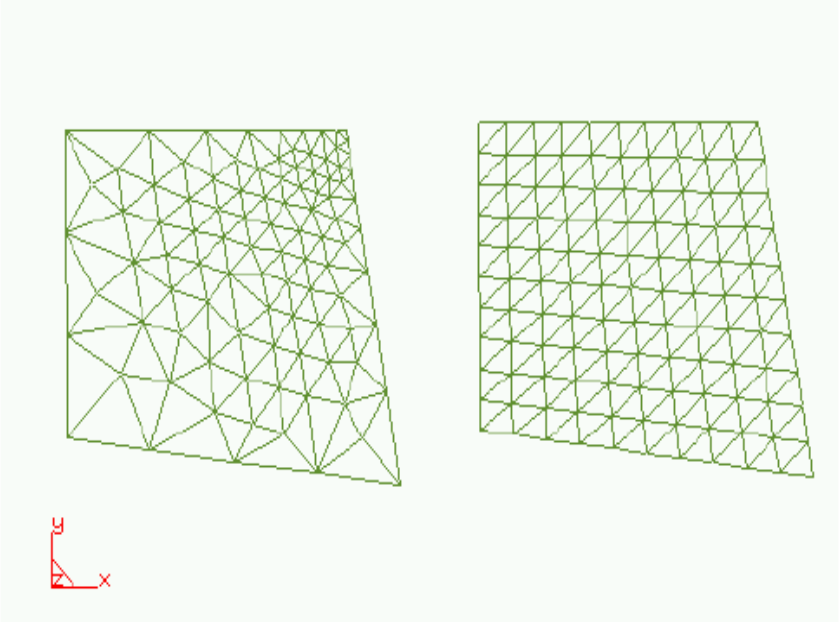


Fig. Nº 13: Malla no estructurada y una malla estructurada

Malla conforme y malla no conforme

Malla conforme:

Se entiende por malla conforme aquella en la que todos sus elementos comparten sus nodos entre elementos vecinos y en al que cada lado de un elemento coincide con el lado de otro elemento y no con una parte de ese lado.

Malla no conforme:

La malla no conforme es aquella en la que sus elementos son independientes entre si.

Ejemplo de ambas mallas puede verse en la siguiente figura

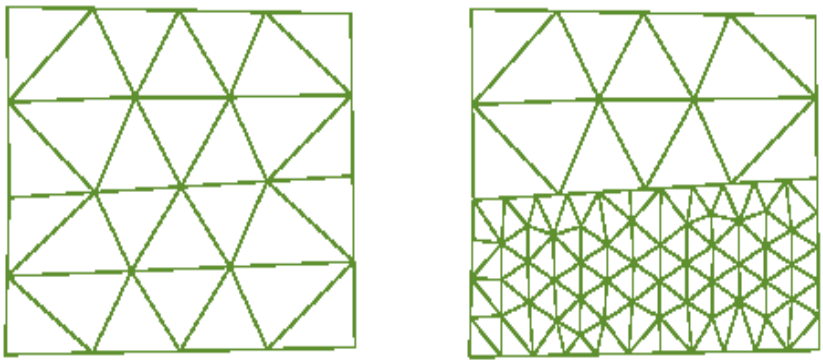


Fig. Nº 14: Malla conforme (izquierda) y malla no conforme (derecha)

En general, para modelar adecuadamente un sólido continuo, es necesario el uso de malla conforme.

Tamaño del elemento para malla no estructurado

Se entiende por tamaño del elemento la longitud de un elemento lineal o la longitud media de una arista de un elemento de otro tipo .

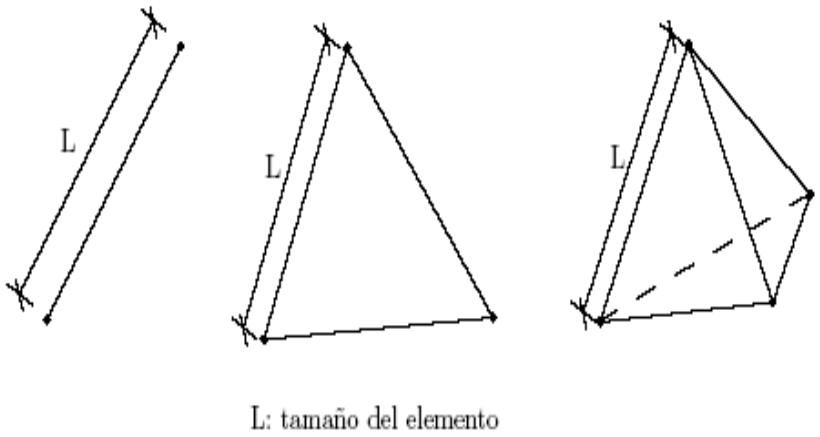


Fig. Nº 15: Tamaño del elemento para cada tipo de elemento.

El tamaño de la malla es muy importante, porque depende de ello que la aproximación del modelo sea exitosa.

Esta información puede provenir de varias fuentes.

- En el momento de generar la malla, el usuario define un tamaño medio que se asignara a todas las entidades que no tenga ningún tamaño predefinido.
- El usuario puede asignar manualmente tamaños a cualquier entidad geométrica.
- Existe una función que asigna tamaños máximos a las entidades en función a su magnitud y forma geométrica
- Otra función asigna tamaños máximos según un criterio de curvatura o de máximo error cordal entre el modelo y la malla resultante
- Mediante otra función se suavizan las diferencias de tamaño entre entidades cercanas para que estas diferencias no superen un determinado criterio de progresión geométrica. Esta función es útil después de haber asignado tamaños con algunas de los criterios anteriores.

Transición del tamaño del elemento entre zonas

Cuando se tiene asignados diferentes tamaños a diferentes zonas de la malla, los elementos tendrán un tamaño más pequeño cerca de una de las zonas y mayor cerca de la otra. El paso de uno de los tamaños al otro puede hacerse de distintas formas y dependerá del algoritmo de mallado y criterios de optimización que se este utilizando.

2.7. Generación de malla no estructurada en Superficies

Existen varias posibilidades para realizar este mallado:

- Transformar la superficie a una superficie equivalente en un plano y realizar una generación 2D en él y aplicar la transformación inversa
- Generar la malla directamente en el espacio.
- Generar la malla en el espacio paramétrico y evaluar en la malla generada con la superficie que representa al objeto

En esta tesis se desarrolla el último método, y por lo tanto se describirán los métodos de mallado en el plano y se elige el elemento triángulo en la descripción de todos los métodos.

2.8. Métodos de generación de malla triangular en el plano

Las ideas básicas para la generación de mallas en geometrías generales pueden clasificarse en tres tipos:

- El primero que mencionaremos consiste en la modificación del dominio original por la remoción sucesiva de triángulos a partir de su borde. Esta idea es comúnmente conocida como el método frontal, según [10] o método de frente avance debido a la forma en que evoluciona la frontera del dominio durante el proceso de generaciones; el método frontal fue utilizado con éxito para la generación de mallas de elementos finitos en dos dimensiones y ha sido extendido al caso tridimensional.

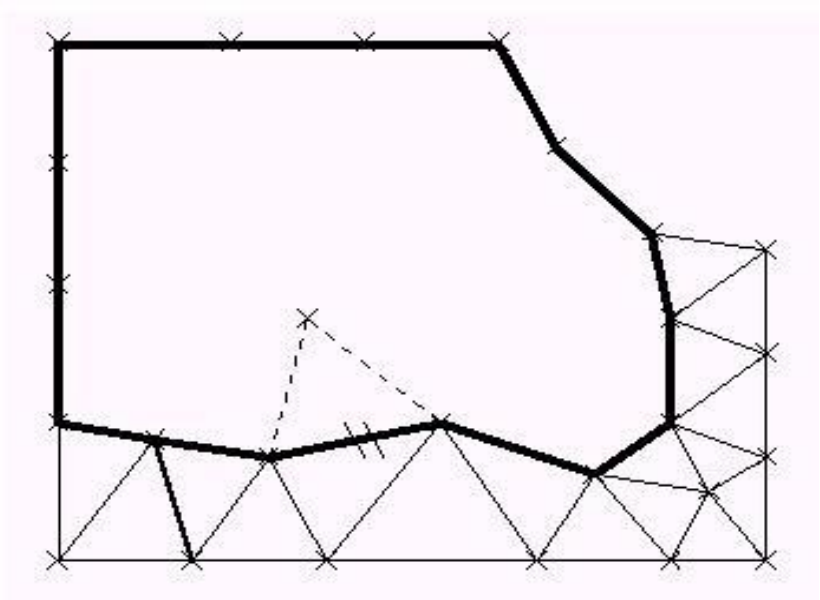


Fig. Nº 16: Mallado con el método de frente de avance

- Otra alternativa a la generación de mallas consiste en generar puntos sobre el dominio de alguna manera, y luego conectarlos para formar una triangulación que satisfaga alguna propiedad de optimalidad. Por ejemplo puede hallarse la triangulación que maximice el mínimo de los ángulos de cada par de triángulos adyacentes. Esta triangulación es conocida como Triangulación Delaunay de un conjunto de puntos. La condición del mínimo ángulo es equivalente a la del círculo circunscrito: el circulo que pasa por los tres vértices de cada triangulo no contiene a otros puntos de la triangulación. Esta propiedad se utiliza para la construcción de la triangulación Delaunay.

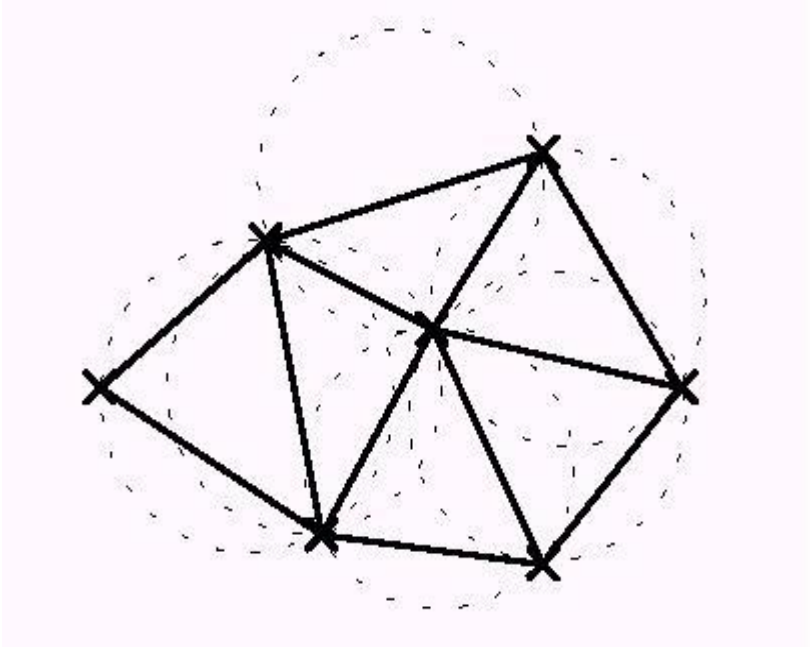


Fig. Nº 17: Triangulación Delaunay de un conjunto de puntos en el plano

- Un tercer método se basa en la utilización de una cuadricula que se superpone al dominio, las celdas exteriores se descartan, mientras que el resto se puede subdividir sucesivamente de acuerdo al grado de discretización deseado en cada región del dominio. Finalmente se dividen las celdas interiores en triángulos manteniendo la compatibilidad con las celdas vecinas. Las celdas que intersecan a la frontera requieren un tratamiento especial (pueden triangularse por ejemplo utilizando el método frontal). Esta técnica es conocida como el método de árboles cuaternarios (en tres dimensiones: árboles octales), debido a la estructura de datos implícita en la subdivisión sucesiva de cada celda, y ha sido utilizado en generación de mallas de elementos finitos tanto en dos como en tres dimensiones.

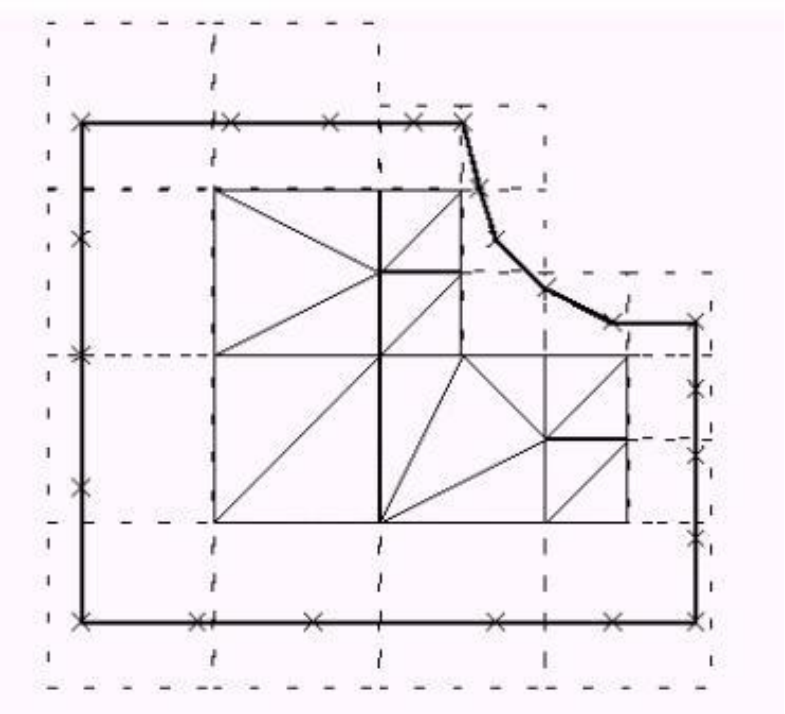


Fig. Nº 18: Mallado con el método de subdivisiones sucesivas

Algunos métodos surgen de combinar estas ideas básicas: frontal con Delaunay, Delaunay con frontal, Árboles Octales con Delaunay, etc.

Generación frontal

El método de generación frontal apareció para resolver la necesidad de analizar problemas cuyas geometrías eran lo suficientemente complejos como para no poder discretizar con malla estructurada.

Este tipo de generación de malla recubre el dominio con triángulos empezando de su contorno.

La generación frontal (advancing front) se basa en un algoritmo que va creando los elementos uno a uno hasta que cubran todo el interior del objeto a mallar.

Este proceso de creación puede esquematizarse en los siguientes pasos:

1. El punto de partida es una malla de contorno; es decir una aproximación poligonal del contorno, que esta formado por elementos lineales de dos nodos. El cual toma el nombre de frente activo

2. El avance debe empezar en una de las caras del frente activo. Se escoge la más pequeña de ellas porque optimizan mejor el futuro tamaño de los elementos.

3. A partir de esa cara se calcula la posición óptima de un punto tal que unido forme un triangulo.

4. A la lista de posibles puntos de formación del nuevo elemento, deben añadirse los puntos del frente activo más cercanos al calculado. Esta lista debe ordenarse según un criterio de distancia al punto creado. Todo punto del frente cuya distancia D, al punto calculado sea menor que $D < f(h)$ siendo $f(h)$ una función que depende de la altura del elemento calculado, se considera más optimo que este punto.

5. Para cada uno de los puntos de la lista ordenada, se calcula si el elemento formado con este punto cumple con una serie de criterios mínimos de calidad, como el de ángulo mínimo aceptable. También se comprueba si sus caras intersetan con cualquiera de las caras del frente. El primer punto de la lista que cumpla con todos estos criterios queda aceptado y se crea un nuevo elemento.

6. Se actualiza el frente quitando la cara, o caras usadas y añadiendo las nuevas caras que se han creado.

7. Mientras queden caras en el frente, se repite el proceso desde el paso 2.

Triangulación de Delaunay

El método de Delaunay es un método alternativa para la generación de malla no estructurada triangular en un dominio.

Comenzaremos por dar una idea al respecto que podría tener esta triangulación ya que podría servir como una primera aproximación a la definición formal. Dicha aproximación se realiza destacando sus características más importantes.

En la triangulación de Delaunay:

- Todos los puntos están conectadas entre sí y forman el mayor número de triángulos posibles sin que se crucen sus aristas (imprescindible para pues se trate de una triangulación)
- Los triángulos se definen de forma que los puntos mas próximos están conectadas entre si por una arista.
- Esto impida que los triángulos formados serán los mas regulares posibles, es decir de modo que se maximice sus ángulos menores y se minimice la longitud de sus lados.

Estas propiedades hacen a esta triangulación muy interesante en varios campos.

Además, si se tiene en cuenta que existen algoritmos optimizados para resolver estas triangulaciones a gran velocidad para un elevado número de de puntos, la elección del tipo de triangulación en este caso es clara.

La triangulación de Delaunay puede caracterizarse de varias maneras. Todas ellas pueden servir para definirla y para encontrar métodos que sirvan para calcularla.

A continuación se dan tres definiciones.

Definición 1: maximización de ángulos

Dado que todas la posibles triangulaciones sobre un mismo conjunto finito de puntos en el plano cuentan necesariamente con el mismo número de triángulos (y que por tanto, todas los posibles vectores de ángulos correspondientes a esas triangulaciones tendrían el mismo número de elementos), podría definirse la triangulación de Delaunay como la que posee el vector de ángulo mayor de entre todas las posibles triangulaciones, entendiéndose que los vectores han sido ordenados mediante orden lexicográfico; es decir, si para todas las posibles triangulaciones de una nube de puntos definimos un vector cuyos elementos son los ángulos formados por las aristas de la triangulación (ordenados de mayor a menor), la triangulación de Delaunay seria aquella en la que todas los elementos del vector fueran mayores o iguales que los de cualquier otra triangulación.

Definición 2: a partir del diagrama de Voronoi

Otra posible definición de la triangulación de Delaunay podría hacerse partiendo del diagrama de Voronoi, que queda definido de la siguiente manera:

Sea $P = \{p_1, p_2, ..., p_n\}$ un conjunto de n puntos distintos en el plano. Definimos el diagrama de Voronoi de P como la subdivisión del plano en n regiones, cada una correspondiente a un punto de P, cumpliendo que un punto q de P pertenece a la región correspondiente al punto p_i si y solamente sí $D(q, p_i) < D(p, p_j)$ para cada punto p_j en P, con $j \neq i$

Donde D es la distancia euclidiana.

En el siguiente grafico se muestra el diagrama de Voronoi de tres puntos

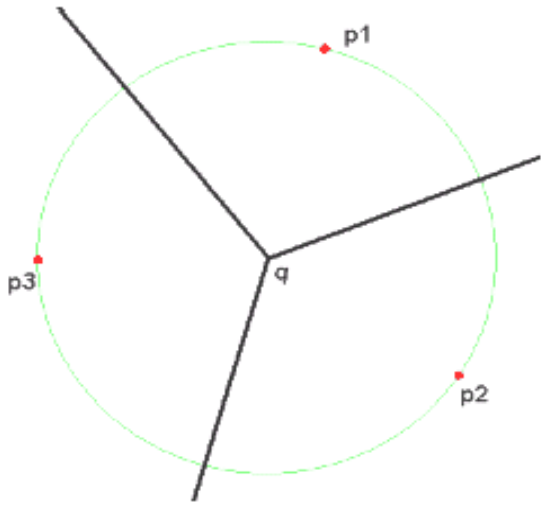


Fig. Nº 19: Diagrama de Voronoi de tres puntos

Este diagrama posee una serie de propiedades.

- Cada vértice del diagrama de Voronoi debe ser centro de un círculo que contenga al menos tres puntos de la nube en su contorno
- Cada región definida por éste diagrama es convexa.

- El bisector entre dos regiones del plano formará parte del diagrama de Voronoi si y sólo si puede trazarse un círculo con centro en el bisector y que contenga en su contorno a los dos puntos (uno de cada región) sin contener a ningún otro punto de la nube en ningún otro lugar.

Una vez conocida el diagrama de Voronoi, existen una serie de propiedades que permiten calcular a partir de él una triangulación de Delaunay de forma directa. Por ello el diagrama de Voronoi y la triangulación de Delaunay son duales.

- La triangulación de Delaunay es el grafo de líneas rectas dual al diagrama de Voronoi.
- Cada triángulo de la triangulación se corresponde con un vértice del diagrama.
- La triangulación y el diagrama tienen el mismo número de aristas y se corresponden entre sí.
- Cada nodo de la triangulación se corresponde con una región del diagrama.
- El contorno de la triangulación es equivalente al cierre convexo de la nube de puntos.
- No es posible encontrar ningún punto de la nube en el interior de los triángulos formados por la triangulación.

Definición 3: propiedades respecto a circunferencias

La tercera posible definición de la triangulación de Delaunay podría efectuarse indicando sus propiedades:

- Tres puntos de la nube de puntos son vértices de un mismo triángulo de la triangulación de Delaunay si y sólo si puede trazarse un círculo cuyo contorno contenga esos tres puntos y no contenga otros puntos de la nube en su interior.

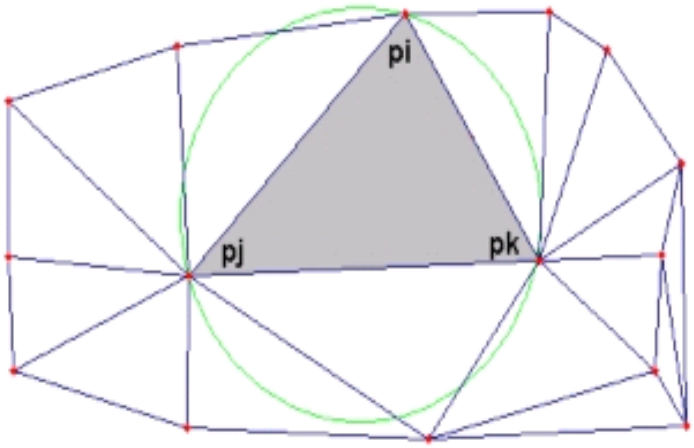


Fig. Nº 20: Criterio del círculo vacío

- Dos puntos de la nube definen una arista de la triangulación si y sólo si es posible trazar un círculo cuyo contorno contenga a esos dos puntos pero en su interior no contenga ningún otro punto de la nube.

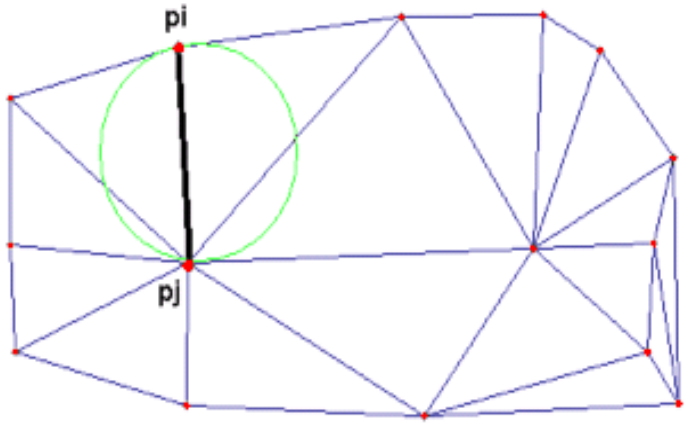


Fig. Nº 21: Arista legal de la triangulación

Con estas dos propiedades podemos caracterizar la triangulación de Delaunay de la siguiente manera:

Sea *P* un conjunto de puntos en el plano y *T* una triangulación de *P*. *T* es una Triangulación de Delaunay de *P*, si y solamente si, la circunferencia circunscrita de cualquier triángulo de *T* no contiene puntos de *P*.

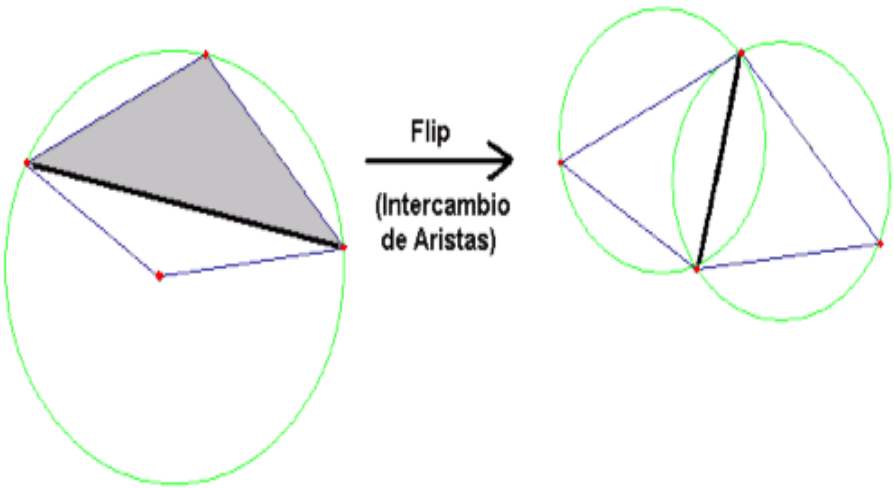


Fig. Nº 22: Legalización de Arista ilegal de la triangulación

Se puede observar, que en la figura de la izquierda, la circunferencia circunscrita al triángulo sombreado contiene a otro punto en su interior, la arista que pertenece a los dos triángulos (línea más gruesa) es una *arista ilegal*. Realizando un intercambio de aristas o *flip* se consigue una triangulación válida, tal y como se observa en el dibujo de la derecha. Hemos convertido una arista ilegal en legal.

Se puede generalizar y afirmar que una triangulación *T* de un conjunto *P* de puntos en el plano es una Triangulación de Delaunay, si y solamente si, todas las aristas son legales.

Algoritmos

Cualquiera de las tres definiciones vistas anteriormente puede servirnos como base para definir un algoritmo que nos permita calcular la triangulación de Dalaunay de un conjunto de puntos.

Partir de una triangulación cualquiera (Algoritmo 1)

La definición 1 no es directamente aplicable como algoritmo, ya que exigiría calcular todas las posibles triangulaciones de una nube de puntos para tomar después aquella a la que corresponda el vector de ángulo mayor.

Sin embargo, basándonos en una característica de toda triangulación (todas las posibles triangulaciones de una nube de puntos tienen el mismo número de triángulos) se puede partir de una triangulación cualquiera para ir modificándola: bastará con cambiar los orígenes y destinos de las aristas para obtener una triangulación a partir de cualquier otra.

Y es aquí donde cobran importancia las propiedades indicadas en las definiciones 1 y 3.

Dada una triangulación cualquiera, puede comprobarse para cada uno de sus triángulos que se cumple la propiedad de la definición 3 (circunferencia circunscrita al triángulo). En caso de no ser así, existirá un punto en el interior de la circunferencia que compartirá una arista con el triángulo circunscrito.

Dicha arista compartida sería una arista ilegal, y abría que realizar sobre ella lo que se denomina una operación de flip o legalización de arista. Esta operación consiste en cambiar esa arista ilegal por una arista legal (que cumpla la propiedad de la circunferencia circunscrita) y se realizará de la siguiente forma:

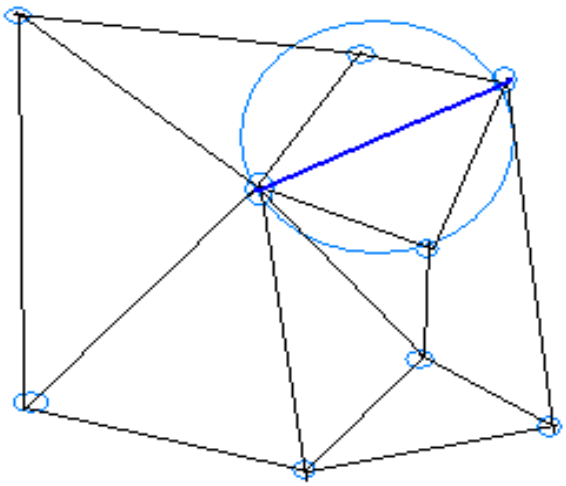


Fig. Nº 23: Arista ilegal (en azul)

Se puede observarse en la figura anterior que los tres puntos del triángulo sobre el que se circunscribe la circunferencia forman junto con el punto interior a la circunferencia pero ajeno al triángulo, un cuadrilátero. En este caso, la arista ilegal sería una de las diagonales del cuadrilátero (la mostrada de rojo) y bastaría con cambiar dicha diagonal por la diagonal opuesta para completar la operación de flip. La arista legalizada es de menor longitud que la arista ilegal, por lo que, en realidad, la comprobación del círculo se asegura de que se verifique que los puntos mas próximos están conectados entre si por una arista.

En este proceso se destruye dos triángulos (los que comparten la arista ilegal) y aparecen dos triángulos nuevos tras el intercambio de aristas (slip, aquellos que competen la arista legalizada).

Este proceso de intercambio de arista se muestra en la siguiente figura:

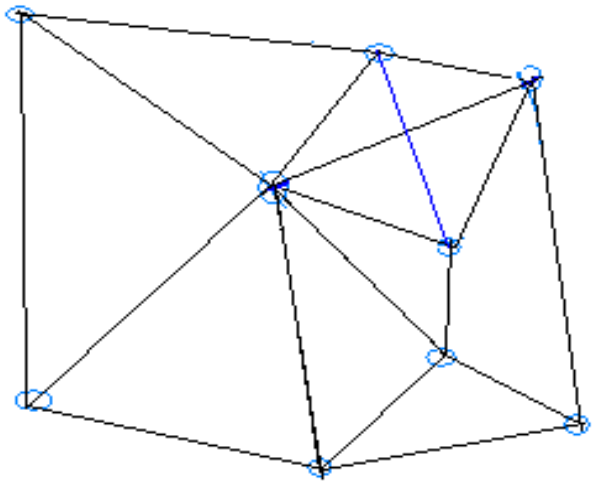


Fig. Nº 24: Arista legal obtenida al aplicar flip (en azul)

Puede observarse que también se cumple la definición 1, ya que, al realizar un flip o legalizado de una arista, se maximizan los ángulos de los dos triángulos afectados por la operación. Con esto finaliza la operación de flip sobre esa arista.

El algoritmo quedaría así:

***Entrada*:** Un conjunto *P* de puntos en el plano.

Salida: La triangulación de Delaunay de *P*.

1. Obtener una triangulación *T* cualquiera de *P*.

2. mientras existan aristas ilegales en *T*

Hacer el flip correspondiente de la arista.

3. Fin

Como hemos comentado no vamos partir de una triangulación dada, sino que vamos a construir la triangulación de Delaunay de manera incremental a medida que van insertándose nuevos puntos.

Partir del diagrama de Voronoi (Algoritmo 2)

También puede construirse un algoritmo útil para calcular triangulaciones de Dalaunay a partir del diagrama de Voronoi de un conjunto de puntos.

Para realizar la conversión de un diagrama de Voronoi a una triangulación de Delaunay, es suficiente con recordar y aplicar las propiedades de ambos diagramas:

- Cada triángulo perteneciente a la triangulación de Delaunay se corresponde con un vértice (dual) de Voronoi, que además coincide con el circuncentro del triángulo.
- Cada vértice de la triangulación se corresponde con una única región de Voronoi.
- Cada arista de la triangulación puede calcularse a partir de una arista de Voronoi, ya que son perpendiculares entre si.

De esta forma obtendríamos:

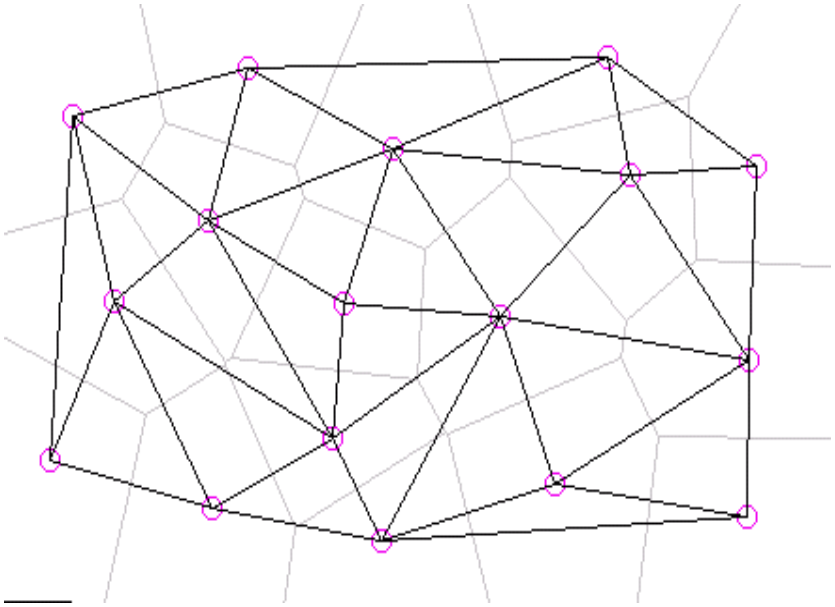


Fig. Nº 25: Dualidad Voronoi-Delaunay

Construcción incremental aleatoria (Algoritmo 3)

Aquí se describe otra manera de aplicar las propiedades de la triangulación de Delaunay (al igual que el algoritmo 1) pero con un comienzo distinto.

En este caso, se partirá de un triángulo ficticio lo suficientemente grande como para englobar a toda la nube de puntos que se desea triangular en su

interior. Después, se comienzan a triangular los puntos de la nube desde cero, uno por uno, tomando el triángulo ficticio como parte de la triangulación ya realizada.

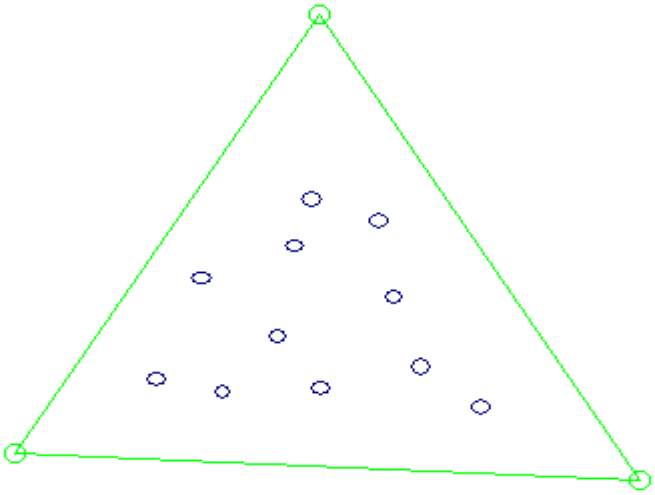


Fig. Nº 26: Triángulo ficticio inicial (en verde)

Una vez realizada toda la triangulación, tan sólo resta descartar todas las aristas que contengan en algún extremo a alguno de los puntos del triángulo ficticio inicial.

El algoritmo se puede resumir así:

triangulacion_delaunay (P)

***Entrada*:** Un conjunto *P* de puntos en el plano.

Salida: La Triangulación de Delaunay de *P*.

Paso 1: Seleccionar un punto de la nube aún no triangulado

Paso 2: Averiguar dentro de qué triángulo (ficticio o no) se encuentra ese punto, éste es un típico problema de localización de punto.

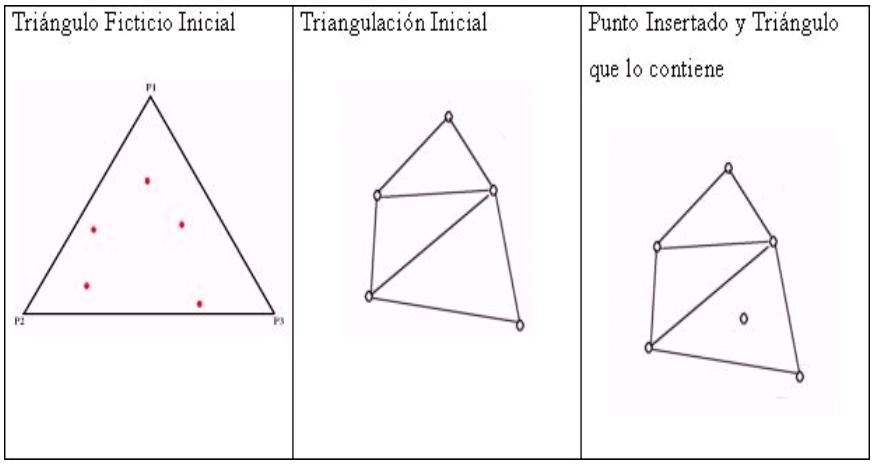
Paso 3: Trazar una arista desde el punto a triangular a cada uno de los tres vértices del triángulo que lo contiene (dado que las aristas del triángulo son los puntos más cercanos al introducido)

Paso 4: Averiguar si es *legal* cada una de las tres aristas del triángulo (ficticio o no) que contiene al punto.

Paso 5: En caso negativo, realizar un *flip* de la(s) arista(s) *ilegal(es)* y comprobar la legalidad de las otras dos aristas del triángulo (ficticio o no) afectado para cada arista legalizada.

Paso 6: Descartar las aristas de la triangulación en cuyos extremos se encuentren vértices del triángulo ficticio inicial.

Gráficamente



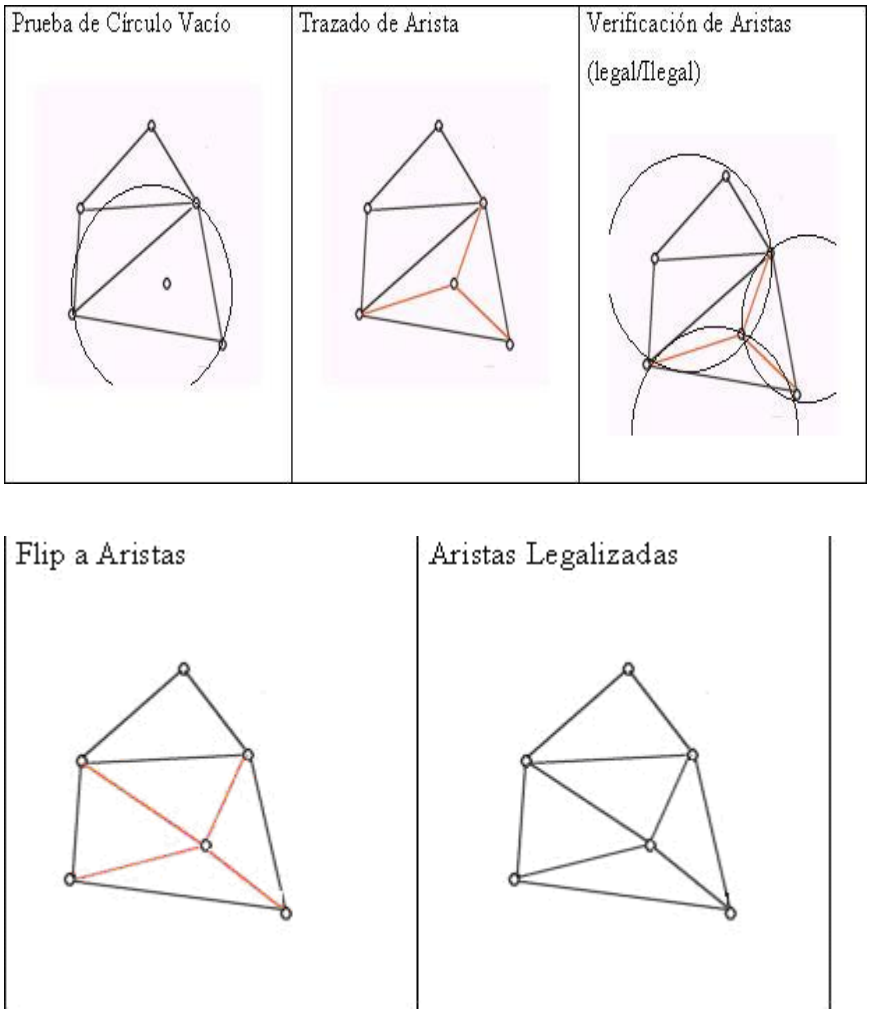


Fig .Nº 27: Esquema Gráfico del Algoritmo Incremental.

III. MATERIALES Y MÉTODOS

El hardware necesario para generar el mallado es esencial, pues en las aplicaciones graficas de calidad, se necesitan muchos triángulos para desplegar objetos tridimensionales. Por lo anterior se necesita equipamiento computacional con capacidad grafica y de procesamiento

3.1. Equipos

- Una Computadora Pentium IV
- Una impresora
- Un digitalizador 3D
- Bibliografía especializada

3.2. Software

- Windows XP profesional
- Software Visual Basic.net 2003 y Sql Server 2000
- Software de diseño y animación 3D-STUDIO MAX 7

3.3. Métodos

El procedimiento para la aproximación de Superficies a través de mallas triangulares con el control de error que se utilizó en esta tesis se puede resumir en los siguientes pasos:

Primero:

Digitalización de un pieza o un objeto en forma de puntos 3D, dichos puntos estarán en formato IGES que será leído y almacenado en una tabla de base

de datos el cual será parte del software que se implementa en este tesis, para realizar la aproximación a través de mallado.

Segundo:

Creación de las Superficies que aproximan a la nube de puntos que representa al objeto real, a través de mallados de superficie Bézier formado a través de triángulos. El mallado se realiza haciendo una verificación de control de error para cada uno de los elementos de la malla obtenida. Es decir se verifica que si el mallado final se ajusta lo suficiente al objeto que se obtuvo en la primera etapa.

3.4. Herramientas computacionales

En este trabajo se utiliza la herramienta 3D-STUDIO MAX 7, para poder importar el archivo IGES y visualizar la superficie al cual se aplica el algoritmo de mallado desarrollado en este trabajo.

El 3D-STUDIO MAX 7 es una herramienta de modelado y animación. El entorno tiene las siguientes características:

En la parte superior se encuentra, como en los programas comunes, la barra de menú, en la cual están todas las diversas funciones del programa.

Inmediatamente abajo se encuentra a manera de solapas, las barras de herramientas más usadas.

Luego el área de dibujo y al final una barra de herramientas que más adelante se detallan sus funciones.

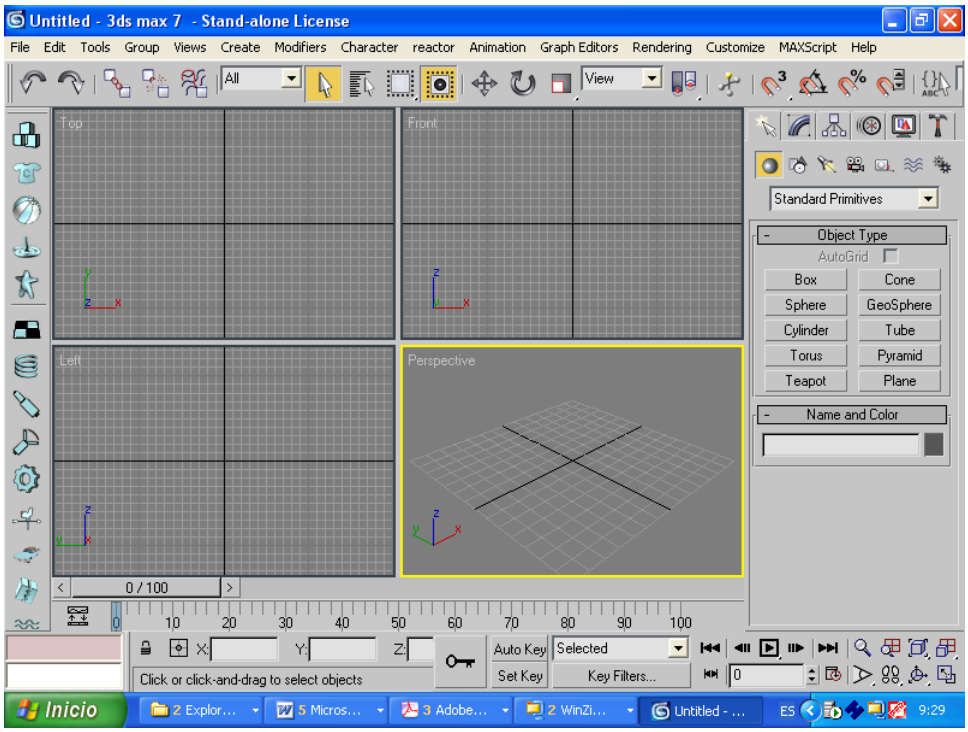


Fig. Nº 28: Ventana Principal de 3D-STUDIO MAX 7

3.5. Principales formatos gráficos en CAGD

Ráster y Vectorial son las dos estructuras básicas para almacenar y manipular datos de un objeto tridimensional en un ordenador. Los paquetes CAD, GIS o de Diseño gráfico más importantes disponibles hoy en día están basados de manera primordial en una de las dos estructuras.

Los datos Ráster vienen en forma de píxeles individuales y cada posición espacial o elemento de resolución tiene un píxel asociado donde el valor del píxel indica un atributo como color, elevación, o número de identificación, etc. Los datos de un Ráster se adquieren normalmente mediante un escaneo óptico, una cámara CCD (dispositivo de acoplamiento de carga) digital u otros dispositivos de entrada.

Los datos vectoriales tienen la forma de puntos y líneas que están geométricamente y matemáticamente asociados. Los puntos están almacenados usando coordenadas. La forma vectorial de almacenamiento es flexible y eficiente para representar datos de los objetos y necesitan menos recursos del sistema para su manipulación y almacenamiento.

Cada programa de modelado geométrico tiene su propio formato de base de datos interna. Esto se hace así para facilitar y optimizar el almacenado de las entidades que maneja el código. El inconveniente de este sistema de organización de datos es que cada programa solo entiende su propio formato de base de datos y no el de otros; ello conlleva a definir unos formatos de fichero de intercambio, aceptados por todos los programas, que son los que se van a usar para importar y/o exportar los modelos.

A lo largo de los años, una serie de instituciones y entidades han ido definiendo varios de estos estándares y algunos de ellos son aceptados y usados por la mayoría de los programas de CAD y de tratamiento geométrico. En seguida se describen brevemente los principales:

Formato DXF (Drawing eXchange Format)

Este formato se creó y se usa mayoritariamente para el intercambio de geometrías bidimensionales. Soporta la definición de entidades tridimensionales pero de una manera un tanto precaria. En concreto, no soporta las Superficies NURBS.

Los ficheros con formato DXF (formato de intercambio de dibujo) permiten el intercambio de dibujos entre AutoCAD y otros programas. Los archivos DXF pueden ser en formato ASCII o binario.

Todas las implementaciones de AutoCAD aceptan el formato DXF ASCII (sin embargo la compatibilidad hacia atrás no esta garantizada). Los archivos DXF ASCII pueden fácilmente traducirse a los formatos de otros sistemas de CAD o ser importados desde otros programas de análisis especializado.

Los fichares DXF ASCII son más comunes que el formato binario, por eso se usa simplemente el termino “DXF” cuando nos referimos a fichero DXF ASCII y el termino “DXF binario” cuado nos referimos al formato binario. Ambos, DXF ASCII y el formato de fichero binario, contienen una descripción completa de la salida grafica de AutoCAD.

Organización y detalles del fichero

Esencialmente un archivo DXF se compone de pares de códigos y valores asociados. Los códigos conocidos como códigos de grupo, son un número

entero e indican el tipo de valor que sigue.

Usando estos códigos de grupo y pares de valores, un fichero DXF se organiza en secciones, que se componen de regiones, que a la vez esta compuesto de un código de grupo y de un lado. En el archivo DXF cada código de grupo y cada valor están en su propia línea de texto.

Código de grupo

El código de grupo y su valor asociado define un aspecto específico de un objeto de una entidad. La línea que sigue inmediatamente al código de grupo es ese valor asociado. Este valor puede ser una cadena de caracteres, un número entero, un número real como la coordenada X de un punto.

Los códigos de grupo especiales se usan como separadores de ficheros, como marcadores para el comienzo y final de sección, para tablas y también para el final de archivo

Estructura del fichero DXF

La organización total de un archivo DXF se divide en varias secciones como se indican continuación:

Sección header

En esta sección se encuentra la información general sobre el dibujo. Contienen variables que representan el estado de la configuración de AutoCAD. Otras variables establecen las unidades de medida de ángulos, desplazamientos, escalas, etc. Cada parámetro contiene un nombre de variable y su valor asociado.

Sección classes

Contiene la información para las clases definidas por la aplicación. Una definición de clase se fija permanentemente en al clase de jerarquía.

Sección tables.

Esta sección contiene lista de información usadas en el resto del dibujo, como la lista de tipo de líneas, nombre de las capas, tipos de letra, y vistas preestablecidas de dibujo

Sección blocks

Contiene elementos del dibujo predefinidos que deben estar presentes en el dibujo. El formato de entidades en la sección de bloques es idéntico a las entidades de la sección entidad.

Sección entities.

Esta sección contiene los objetos gráficos (entidades) que hay en el dibujo, incluyendo las referencias a bloques.

Sección object

Contienen los objetos no gráficos de dibujo. Todos los objetos que no son entidades o registros de tabla de símbolos se almacenan en esta sección.

SET (Standar Echange et de Transfert).

Su desarrollo comienza en 1983 y viene a ser un estándar Francés para intercambio de datos CAD.

STEP (Standar of the Exchange of Product model data).

Este formato fue introducido por la International Organization for Standardization (ISO) a fin de liberar un modelo de Intercambio CAD el cual debería ser aceptado como estándar alrededor del mundo. STEP es el más moderno interfaz CAD, pero no es muy utilizado alrededor del mundo,

lo que más usan es el IGES por la presencia que tiene este formato desde 1979 y esta ampliamente difundido.

SAT (Standar ACIS Text) and SAB (Standar ACIS Binary)

Son formatos de archivo usados para almacenar y transferir datos en ACIS cual es una flexible librería C++ y SCHEME producido por tecnología espacial para propósitos de manipulación geométrica.

Formato IGES (Initial Graphics Exchange Specification)

Este formato es inherentemente tridimensional y se puede considerar como el estándar para todos los programas que trabajan con geometrías complejas en el espacio. Soporta gran cantidad de entidades diferentes, lo cual es, en algunos casos, un inconveniente por la gran complejidad que entraña. Soporta las líneas y Superficies NURBS.

Las entidades que soporta este formato son: geometría alambica 2D y 3D, Superficies recortadas y no recortadas (regladas, paramétricas, NURBS y otras), subfiguras, grupos, capas, textos, etiquetas, símbolos, anotaciones, acotaciones y sólidos B-Rep (Representación de frontera).

IGES fue desarrollado originalmente para el cambio del esbozo de datos 2D/3D, modelos, texto, datos dimensionados y una limitada clase de Superficies.

La forma geométrica de un producto es descrita usando estas 2 partes (geometría y topología). La parte no geométrica puede ser dividida en anotaciones, definiciones y organización.

La categoría de anotación contiene dimensiones, borrador de notaciones, texto.

La categoría de definición sirve para definir propiedades específicas de entidades individuales o compuestas.

La categoría de organización define grupos de geometría, anotación o elementos de propiedad.

IGES es actualmente la interfaz CAD más usada hoy en día. IGES fue desarrollado en 1979 por la US National Bureau de estandarización. Entre los años 1979-1994, muchas aplicaciones se han estado ya realizando, los cuales necesitan datos CAD, tal como elementos finitos o generación de mallas. La extensión de estos archivos es ".igs". Un archivo IGES es estructurado en cinco secciones principales: sección inicio, sección global, directorio de entrada, parámetro de datos y fin de sección.

- En la sección inicio uno puede escribir cosas como el directorio de localización del archivo. Este debería tener al menos una línea.
- En la sección global, se encuentra la información general las cuales sirven para leer el archivo actual, donde, cuando y por quien fue generado el archivo. Este también puede especificar el parámetro delimitador como también el registro delimitador.
- El directorio de entrada provee un resumen general de las diferentes componentes del archivo IGES y este indica también un registro, en el parámetro de datos, el cual contiene información completa acerca de todos lo parámetros. Por ejemplo, en el directorio de entrada se puede

ver que entidades tenemos para tratar con Superficies NURBS. La información como puntos de control, secuencia de nodos y pesos no pueden ser encontrados en el directorio de entrada. Toda entidad tiene su propia identificación la cual varía de 100 a 514 a lo más. En la siguiente tabla se resume el número de entidades correspondiente a unos cuantos objetos geométricos importantes.

Entidad	ID numero
Circular arc	100
Curva B-spline Rational	126
Curva compuesta	102
Línea	110
Superficie de revolución	120
Cilindro tabulado	112
Superficie B-spline Racional	128

- Es en el parámetro de datos que se encuentran los valores de todos los parámetros necesarios para cada entidad. Esta sección cambia de una entidad a otra pero esta tiene en general la siguiente forma, suponiendo que el delimitador de parámetros es una coma (,) y el delimitador de registro es un punto y coma (;): Número de entidad, parámetro 1, parámetro 2,…, parámetro N; después del delimitador de registro, cualquier comentario puede ser agregado.

- El fin de sección consiste solo de una simple línea. Las columnas 33-72 no son usados en esta sección. Aquí se encuentra cuatro campos correspondientes a las cuatro secciones formadas.

Campo	Columna	Sección
1	1-8	Comienzo
2	9-16	Global
3	17-24	Directorio de entrada
4	25-32	Datos de parametro

Por último la utilización de este formato está dirigida a aquellos usuarios que necesitan integrar datos de otros sistemas CAD. Para un intercambio eficiente y preciso de datos de diseño creados en diversos sistemas CAD es necesario un estándar para el intercambio de datos digitales, este es uno de los estándares mundiales.

Formato VDA (Verband Der Automobil industrie).

Este es un formato de archivo que tiene su origen en la asociación manufacturera de vehículos alemana VDA. Este formato es reconocido por su capacidad para manipular libremente Superficies 3D cuales son frecuentemente desarrollados en diseño de vehículos; su extensión es ".VDA".

Es un formato tridimensional simplificado en el cual todas las Superficies tienen que convertirse a unas entidades básicas que tienen forma analítica. Su mayor ventaja es la simplicidad, lo cual evita muchos errores en el traspaso. Su inconveniente es que, al pasar de un sistema a otro, ya no se

tiene exactamente la misma entidad que en el original. No soporta las Superficies NURBS.

El código encargado de leer estos formatos debe ser capaz de transformar todas las entidades que no estén directamente soportadas en el sistema a otras que si lo estén. .En algunos casos, ello implica una aproximación, pero si se tiene como entidad básica a las NURBS, casi cualquier entidad soportada en esos formatos puede ser descrita exactamente como una superficie de este tipo.

IV. RESULTADOS

El cliente entrega un objeto o una pieza realizada en metal, plástico, barro, yeso u otro material y desea un modelo CAD generado a partir de la misma.

Se puede realizar una clasificación inicial del objeto en dos grupos:

- **Los objetos de animación,** son modelos artísticos que se utilizan para realizar animaciones 3D y que no precisan una precisión elevada.
- **Los objetos de diseño industrial**, los cuales precisan un tratamiento más estricto, debido a que el modelo 3D que se obtenga serán introducidos en alguna parte del proceso CAD/CAM/CAE, tolerándose muy poco error entre el objeto original y el modelo obtenido.

El interés de este trabajo es el segundo tipo de objetos, por que para el primer tipo existen muchas herramientas de software que permite obtener resultados adecuados al no exigir una elevada precisión.

En este trabajo se leerán ficheros IGES en el cual se encuentra almacenado los datos del objeto como Superficies NURBS para luego realizar un programa CAD utilizando la plataforma de desarrollo del Visual Basic.net 2003.

4.1. Algoritmo para el mallado

El algoritmo utilizado en esta tesis para el mallado del dominio paramétrico de la superficie de Bézier es el algoritmo de Triangulación de Delaunay y se construye de la siguiente manera:

Se ordenan los puntos de datos según una clave compuesta primero por su coordenada x y luego por su coordenada y. Esta ordenación tiene como fin

disminuir los tiempos de búsqueda de los puntos que cumplan las condiciones de mínimas distancias o máximos ángulos.

Para la implementación del mallado del conjunto de puntos generados aleatoriamente en el dominio paramétrico se ha utilizando el tercer algoritmo de Delaunay pero con una pequeña modificación; es decir el algoritmo queda así:

Paso 1: Creación de un triángulo ficticio, el cual abarca todos los puntos generados, es decir, encierra el conjunto de puntos usando una triangulación ficticia.

Paso 2: Avanza por incremento, insertando nuevos puntos a la triangulación existente.

Paso 3: Por cada punto de la nube, buscar el triángulo que lo contiene, es decir la búsqueda es hecha por todos los triángulos cuyos círculos inscritos contienen el nuevo punto. Para ello se realiza los siguientes pasos:

- Sea T el triángulo que contiene al punto p a insertar. Unir p con cada uno de los vértices del triangulo T, creando tres nuevos triángulos.
- Por cada uno de los tres triángulos, generados al insertar un punto se realiza el test del círculo circunscrito; es decir se legalizan las aristas.

Paso 4: Una vez insertado todos los puntos se borran todos los lados del triangulo ficticio.

4.2. Control de error para el mallado

Esta propuesta se concentra en encontrar una estimación matemática que permita modelar los detalles finos de las Superficies de los objetos; para ello, se propone un método para la generación de la Superficie de Bézier que incorporen en su construcción mallas triangulares, tal que se aproxime a la geometría de la superficie del objeto físico obtenido del fichero IGES, la generación de malla se realiza a partir de un conjunto de puntos desorganizados que se generan aleatoriamente en el dominio paramétrico de la superficie de Bézier del objeto original.

EL modelo que se plantea en esta tesis para resolver el problema de aproximar la superficie de un objeto a través de mallas triangulares; es decir el modelo matemático para controlar el error cometido al aproximar la superficie del objeto es el siguiente:

Dado un punto $\vec{P_i} \in S(u,v)$ de la superficie y un punto $\vec{P}$ cercano a la superficie $S(u,v)$ nuestro problema es calcular $\left\| S(u,v) - \vec{P} \right\|$ tal que sea mínimo.

Es decir dado un punto $\vec{P}$ cercano a la superficie, el sistema a resolver es:

$$\text{Minimizar } f(u,v) = \left\| S(u,v) - \vec{P} \right\|$$

Que es equivalente a minimizar

$$f(u,v) = \left\| S(u,v) - \vec{P} \right\|^2$$

Si igualamos las derivadas parciales a cero obtenemos:

$$\begin{bmatrix} \frac{\partial f}{\partial u}(u,v) \\ \frac{\partial f}{\partial v}(u,v) \end{bmatrix} = 0$$

Desarrollando tenemos:

$$A = \begin{bmatrix} 2\left(S(u,v) - \vec{P}\right)\frac{\partial S(u,v)}{\partial u} \\ 2\left(S(u,v) - \vec{P}\right)\frac{\partial S(u,v)}{\partial v} \end{bmatrix} = 0$$

El cual es un sistema de ecuaciones no lineales y lo resolveremos utilizando el método de Newton-Rapson para un sistema no lineal.

Es decir utilizaremos el siguiente algoritmo:

Paso 1: Tomar k=1

Paso 2: Mientras $k \leq N$, hacer los pasos 3-7

Paso 3: Calcular A y el jacobiano J en el punto $x = (u_0, v_0)$

Paso 4: Resolver el sistema para y : $Jy = -A$

Paso 5: Tomar $x = x + y$

Paso 6: Si $\|y\| < tol$, entonces salida de la solución aproximada

Paso 7: $k = k + 1$

Paso 8: termina la iteración

Donde

N es el número de iteraciones.

tol es el error cometido al resolver el sistema.

$x = (u_0, v_0)$ es el punto inicial .

J es el matriz jacobiano y su expresión es:

$$J = 2\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$a = \left(\frac{\partial S(u,v)}{\partial u} \right)^2 + \left(S(u,v) - \vec{P} \right) \frac{\partial^2 S(u,v)}{\partial u^2}$$

$$b = \frac{\partial S(u,v)}{\partial v} \frac{\partial S(u,v)}{\partial u} + \left(S(u,v) - \vec{P} \right) \frac{\partial^2 S(u,v)}{\partial v \partial u}$$

$$c = \frac{\partial S(u,v)}{\partial u} \frac{\partial S(u,v)}{\partial v} + \left(S(u,v) - \vec{P} \right) \frac{\partial^2 S(u,v)}{\partial u \partial v}$$

$$d = \left(\frac{\partial S(u,v)}{\partial v} \right)^2 + \left(S(u,v) - \vec{P} \right) \frac{\partial^2 S(u,v)}{\partial v^2}$$

En este trabajo el punto $\vec{P}$ considerado como cercano a la superficie es obtenido calculando el centroide de cada triángulo del mallado realizado al objeto.

4.3. Ejecución del programa implementado

Para prueba del programa se ha utilizado el fichero IGES que representa la capota del carro como se puede visualizar en 3D Studio Max 7.

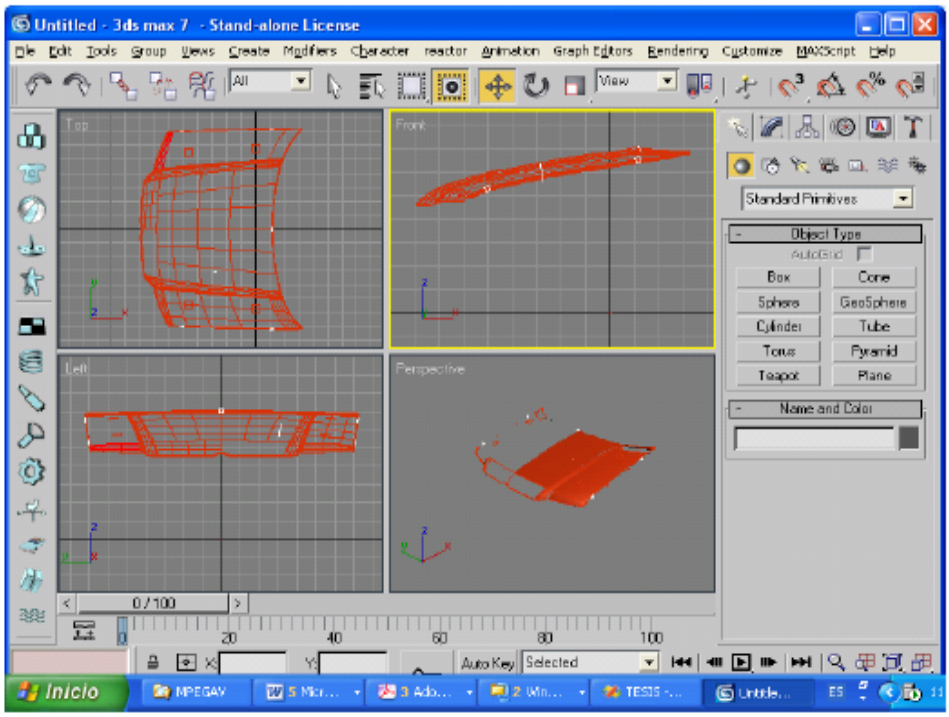


Fig. Nº 29: Visualización del archivo IGES del objeto

En las siguientes figuras se muestran las diferentes vistas del archivo IGES del objeto en estudio en el programa implementado

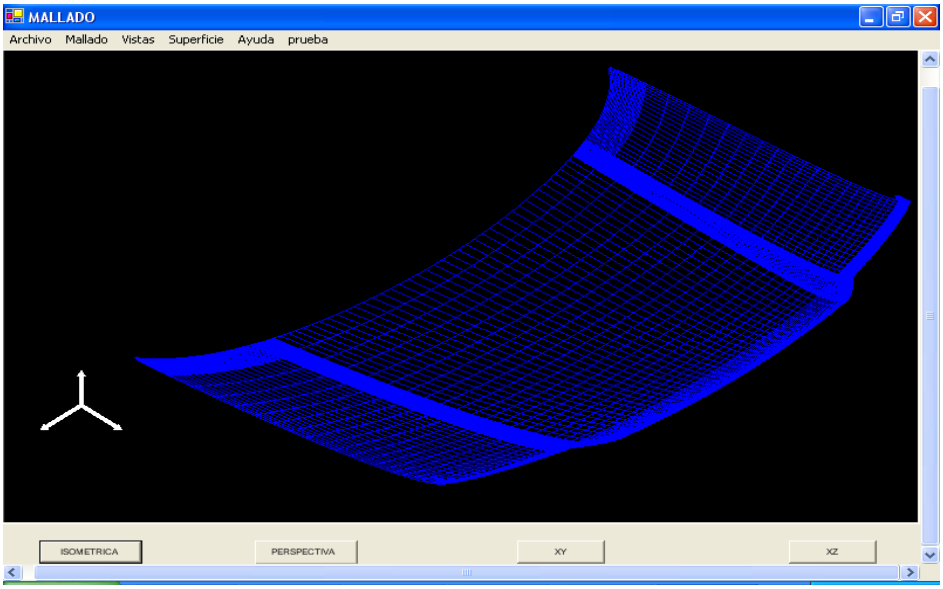


Fig. Nº 30: Visualización del archivo IGES del objeto vista Isométrica

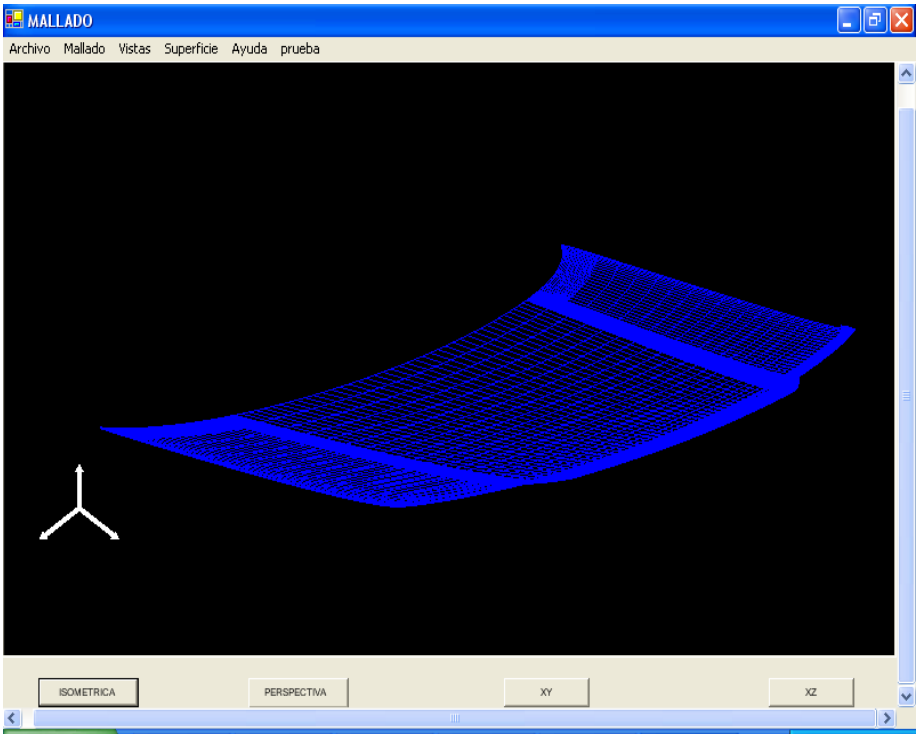


Fig. Nº 31: Visualización del archivo IGES del objeto vista Perspectiva

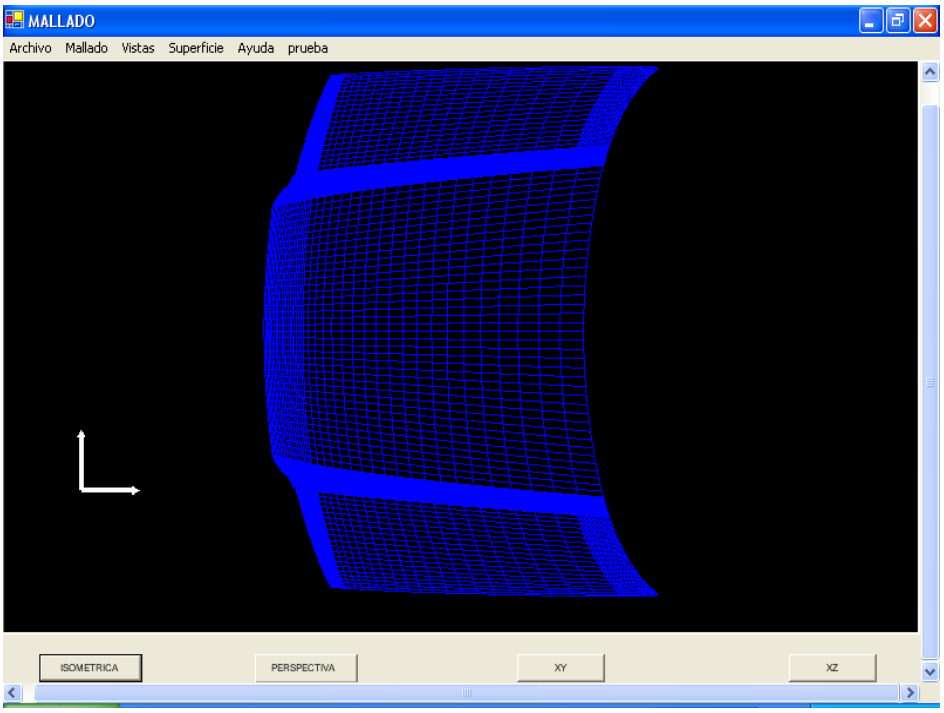


Fig. Nº 32: Visualización del archivo IGES del objeto vista “XY”

Luego tan solo tomamos una sola superficie para realizar el mallado con el programa implementado el cual visualizamos en 3D Studio Max 7

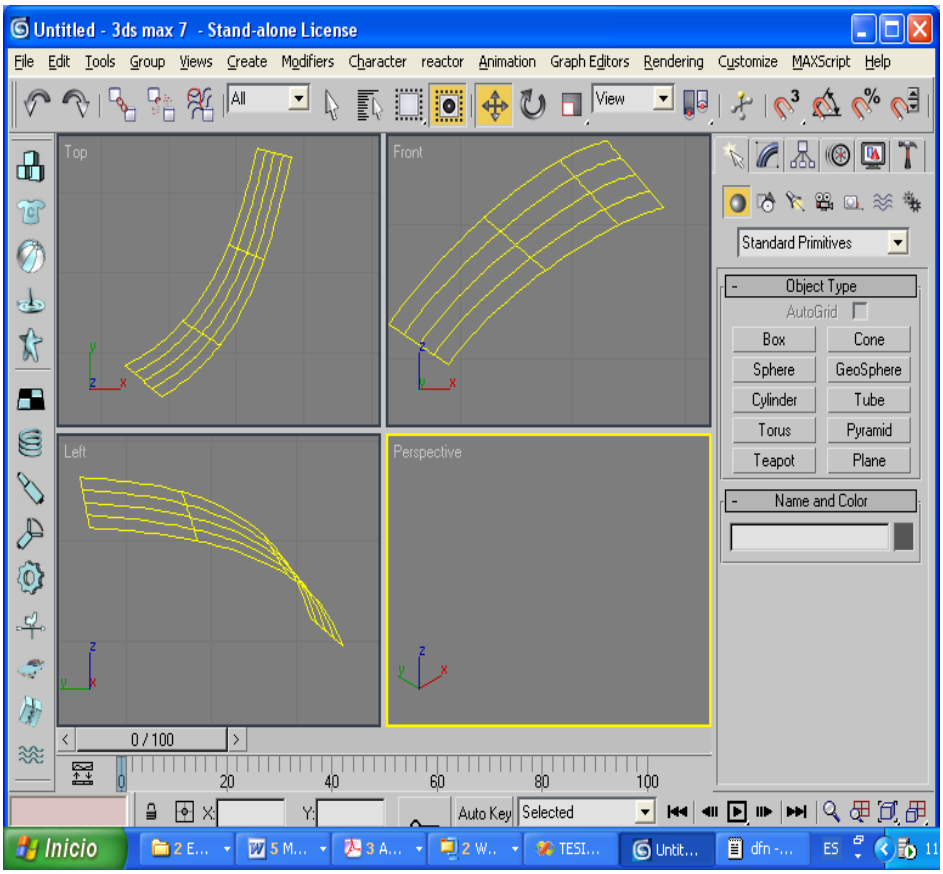


Fig. Nº 33: Visualización de una superficie IGES que representa al objeto

En las figuras siguientes se muestran las diferentes vistas del mallado con un error de aproximadamente de 5% en el programa implementado:

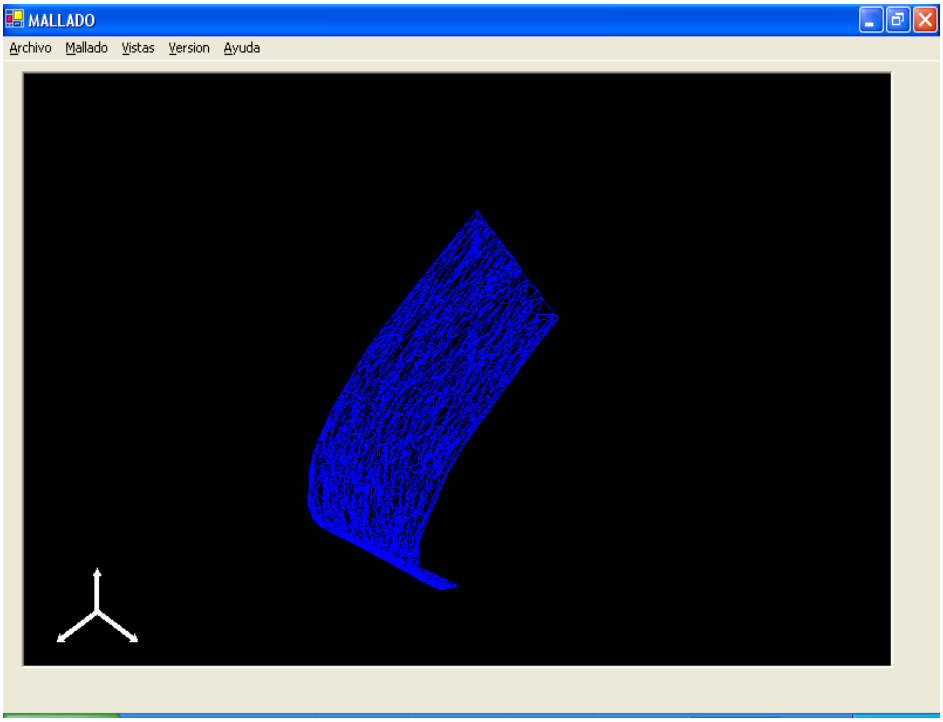


Fig. Nº 34: Visualización isométrica del mallado

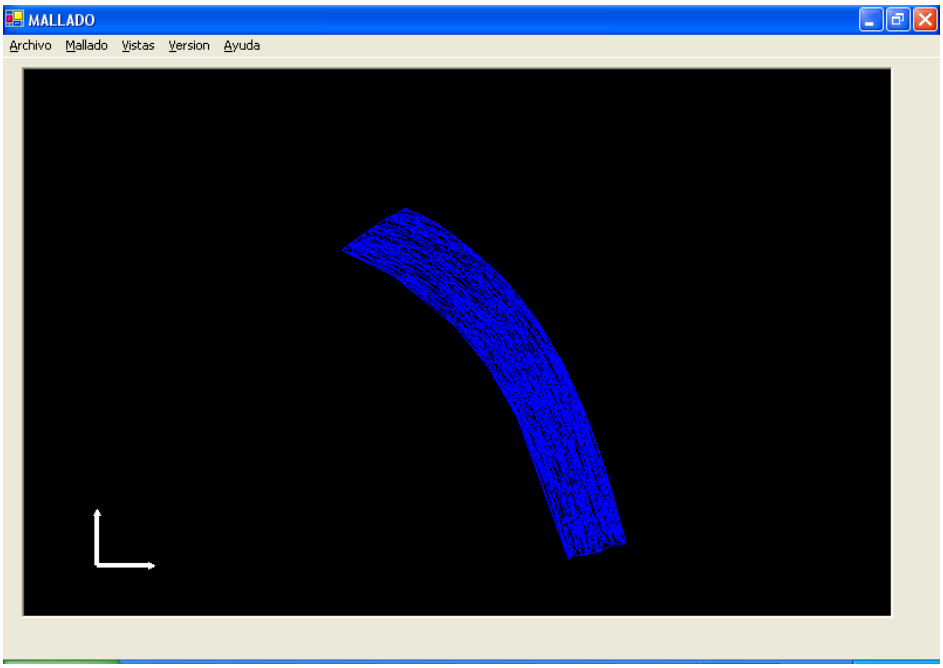


Fig. Nº 35: Vista XY del mallado

V. DISCUSION

5.1. Elección del elemento para el mallado

El número de posibles mallados sobre un conjunto de datos es muy grande. La selección del más adecuado se hace mediante criterios de optimización.

La elección del triángulo como elemento base de construcción de mallas aproximantes a la Superficies, en esta tesis es por las propiedades tan importantes que tiene el triángulo.

Las propiedades con las que tiene el triangulo como elemento de la malla, podemos mencionar los siguientes:

- El triángulo es el polígono con menor número de vértices.
- Queda definido completamente por las longitudes de sus lados, las cuales además deben cumplir con la desigualdad triangular.
- EL triángulo es el polígono geométricamente indeformable, es además necesariamente convexo.
- La proyección de un punto interior sobre el perímetro es monótona, lo que quiere decir que los puntos proyección sobre el perímetro están ordenados y su relación de orden es del ángulo de las rectas proyectantes.
- Al ser el polígono más simple es imposible descomponerlo en varios y por lo tanto es necesariamente conexo.
- Un triángulo siempre define un interior y un exterior al mismo.

- El triángulo es el que presenta una mayor relación entre su perímetro y su área, con respecto a otro polígonos regulares.
- De todos los triángulos con un perímetro dado, el triángulo equilátero es el que hace mínimo el perímetro frente a su área.
- Como el triángulo equilátero es a la vez equiángulo; en este trabajo se impone que los elementos del mallado sean lo más equilátero o equiángulo posible.

Por las razones arriba mencionadas; en este trabajo se eligió hacer el mallado con el criterio de Delaunay el cual optimiza la malla bajo la condición de maximizar los ángulos de los triángulos formados.

5.2. Ventajas del método de Delaunay

- Gran velocidad por el número de operación. El número de operaciones por elemento es menos debido a que en la creación del elemento por este método solo se compara círculos con puntos, en contraposición al avance frontal donde se realiza el cálculo de gran número de intersecciones.
- Siempre se obtiene una malla. Independiente de la calidad de los elementos, siempre se llega al final con este método, en contraposición al avance frontal donde puede llegarse a situaciones de no avance.
- Especialmente útil en el caso de que se disponga a priori de los nodos de la futura malla y únicamente se quiera crear los elementos.

5.3. Elección del Lenguaje de Programación

En este trabajo, la elección del lenguaje de programación ha quedado determinada, en gran parte, por el objetivo que tiene la aplicación y por las características que presenta el lenguaje. Para el desarrollo de la aplicación se ha elegido el lenguaje Visual Basic.NET.

Para el desarrollo de aplicaciones con interfaz grafica amigable al usuario, es imprescindible elegir un lenguaje de programación Visual. De este modo se cuenta con la ventaja de que los elementos utilizado para la comunicación con la aplicación son ampliamente conocido (ventanas, botones, menús, etc.) y fácil de incorporar en la aplicación.

Además, es deseable que el proceso de implementación estuviese acorde con la apariencia moderna del lenguaje. Es decir, que se tratase de un lenguaje de programación orientado a objetos para aprovechar las ventajas de este tipo de metodología de programación (Herencia, Polimorfismo, encapsulamiento). .

Por ultimo la elección de Visual Basic.Net como lenguaje de programación fue porque dicho lenguaje se encuentra integrado con otros lenguajes de Microsoft Visual Studio.NET. No solo permite desarrollar componentes de aplicaciones en distintos lenguajes de programación, sino que también permite heredar las clases escritas en otros lenguajes.

VI. CONCLUSIONES

- Se ha planteado un modelo matemático adecuado (ver 4.2) a partir de las Superficies de Bézier, para el control de error en la aproximación de objetos 3D a través de mallas triangulares.
- Se ha creado un algoritmo que permite generar mallas automáticamente para Superficies de Bézier controlando el error. Esto último muy importante en las aplicaciones industriales donde se aplica el método de elementos finitos.
- Se ha creado un software con prestaciones gráficas que implementa el algoritmo, desarrollado en este trabajo, que realiza el mallado con control de error a partir de Superficies de Bézier.
- La aplicación desarrollada lee la información geométrica de las Superficies desde ficheros en formato IGES. Luego realiza el proceso de mallado, y presenta el resultado de forma gráfica; también se puede obtener los resultados numéricos
- El mallado que se obtiene es de tipo conforme con control de error, por lo que es muy importante para procesos de simulación en la industria, donde se usa el método de los elementos finitos.

VII. RECOMENDACIONES

- Se recomienda el uso de los resultados de este trabajo en procesos de simulación industrial. Ya que el mallado obtenido es un mallado conforme y con control de error.
- Se recomienda utilizar los mallados resultados de este trabajo para procesos de renderizado en software grafico, por permitir una visualización más próxima a la realidad.
- El modelamiento de detalles finos de las Superficies de los objetos 3D, es un problema en investigación. Se recomienda a partir de este trabajo seguir la investigación de mallado de Superficies.
- El alto costo computacional, de los diferentes métodos de mallado especialmente 3D, sugiere utilizar Hardware con capacidad grafica y procesamiento elevado para futuras trabajos de investigación en este tema.

VIII. REFERENCIAS BIBLIOGRAFICAS

[1]. FARIN, GERALD (1993). *"Curves and Surfaces for Computer Aided Geometric Design A practical Gide".* 3rd Ed. Academic Press. Boston.

[2]. FOLEY ET AL. (1983) *"Fundamentals of interactive computer graphics"*. Editorial Addison Wesley. Boston.

[3]. ROGERS D. F., ADAMS J. A (1990). "*Mathematical Elements for Computer Graphics".* Editorial McGraw Hill. 2nd Ed. Annapolis. USA.

[4]. FOLEY, J., A. VAN DAM, S. FEINER, Y J. HUGHES (1996). *"Computer Graphics: Principles and Practice"*. 2nd ed. in C, editorial Addison-Wesley. Boston.

[5]. GEORGE P.L. (1991). "*Automatic Mesh Generation".* Edit JOHN Wiley & Sons, New York.

[6]. PORTELA, A. AND CHARAFI (2002). "A. *Finite Elements Using Maple"*. Editorial Springer. Berlin.

[7]. LANCATER P. AND SALKAUSKAS (1997). "*Curve and surface Fitting, And Introsucttion"* Editorial Academic Press. New York.

[8]. ROGERS DAVID F. (2001). "*An Introduction to Nurbs"*. Academic Press San Diego, Annapolis, USA.

[9]. LO S. H. (*1685*) "*A new mesh generation scheme for arbitrary planar domain, Int. J. Numer .Methods"*. Editorial Academic Press. New York.

[10]. BURDEN, R.L. Y FAIRES, J.D. (2002). "A*nálisis Numérico"*. Séptima edición. Editorial Internacional Thompson Editores. México.

[11]. DOBSON, RICK (2002). "*Programación de Microsoft Sql Server 2000 con Microsoft Visual Basic.Net."* Editorial McGraw Hill. España.

[12]. CARVALHO P, FIGUEIREDO L, GOMES J. (2003), "*Mathematical Optimization in Graphics and Vision. Monografías del IMCA"*, Perú.

[13]. NIEVES ANTONIO Y FEDERICO C. DOMÍNGUEZ (2003). "*Métodos numéricos Aplicaciones a la Ingeniería"*. Segunda edición. Compañía editorial Continental. México.

[14]. HEARN, DONALD Y PAULINE M. (2001) "*Gráficas por computadora"*. Segunda edición. Editorial Prentice Hall. México.

[15]. IGES/PDES (1991) *"Organization: The Initial Graphics Exchange Specification"* (IGES) Version 5.1. National Computer Graphics Association. Virginia - USA.

[16]. KREIYSZIG, E.(1999) *"Matemáticas Avanzadas para Ingeniería"*. Editorial Limusa. México

[17]. MORTENSON E. M.(1985) "*Geometric Modeling"*. Editorial John Wiley & Sons. USA.

[18]. R. LÖHNER AND P. PARIKH (1988). "*Generation of three-dimensional unstructured grids by the advancing front method. Int. j. numer. Methods fluids"*. Editorial John Wiley & Sons. USA.

[19]. NAKAMURA, S. (2001) "*Métodos Numéricos Aplicados con Software".* Editorial Prentice Hall. México

[20]. PATRIKALAKIS N. M. (1989). *"Aproximate Conversion of Rational Splines. Computer Aided Geometric Design"* .Sexta edición. Editorial Springer-Verlag.Berlin

[21]. VALLIERE D. (1990). "*Computer Aided Design in Manufacturing"*. Prentice Hall. México

[22]. PRAUTZSCH H., BOEHM W. PALUSZNY M. (2002). "*Methods of Bézier and B-Spline* ".Editorial Springer-Verlag.Berlin.

A N E X O S

ANEXO 1

Introducción a Visual Basic .NET

MICROSOFT .NET

Es un entorno de desarrollo multilenguaje diseñado por Microsoft para simplificar la construcción, distribución y ejecución de aplicaciones para Internet .Tiene fundamentalmente tres componentes: Una maquina virtual (CLR: Common Language Runtime) que procesa código escrito en un lenguaje intermedio (MSIL: Microsoft Intermediate Language), una biblioteca de clases (biblioteca .NET) y ASP.NET que proporciona los servicios necesarios para crear aplicaciones Web.

Visual Basic.NET

Es uno de los lenguajes de programación de alto nivel que pertenece al paquete. NET (otros lenguajes son C#, C/C++, etc) Visual Basic.NET es independiente de la plataforma (lo mismo podemos decir respecto a los demás lenguajes incluidos en .NET); esto quiere decir que el código producido por el compilador Visual Basic.NET puede transportarse a cualquier plataforma (Intel, Sparc, Motorota, etc.) que tenga instalada una máquina virtual de .NET y ejecutarse.

Introducción a Gráficos en Visual Basic NET

Las clases que producen gráficos vectoriales en VSNET están contenidas en los espacios System.Drawing y System.Drawing.Drawing2D:

```
Imports System.Drawing

Imports System.Drawing.Drawing2D
```

El lienzo

El primer paso para poder dibujar gráficos es crear un objeto Graphics, que viene a ser el lienzo donde expresaremos nuestro arte. El objeto Graphics carece de constructor público, por tanto no se puede instanciar con la palabra clave *New*.

Se obtiene llamando al método *CreateGraphics*. El siguiente ejemplo crea tres objetos Graphics en el formulario, en un *PictureBox* y en un botón:

```
Dim Lienzo1 As Graphics = Me.CreateGraphics

Dim Lienzo2 As Graphics = PictureBox1.CreateGraphics

Dim Lienzo3 As Graphics = Button1.CreateGraphics
```

Una vez utilizado el objeto Graphics es conveniente cerrarlo invocando el método Dispose

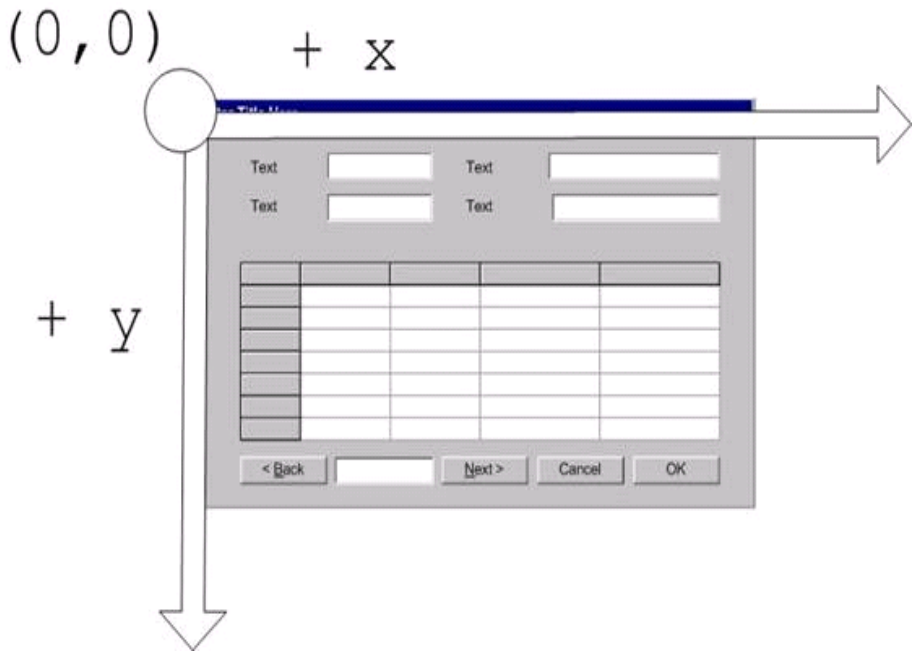
Ejemplo :

```
gr.Dispose()
```

Los ejes de coordenadas

Cualquier formulario o control contiene tres sistemas de coordenadas, llamados "coordenados de dispositivo", "coordenados de página" y "coordenados de mundo".

En los tres sistemas los valores X positivos aumentan hacia la derecha y los valores Y positivos aumentan hacia abajo, según muestra el siguiente esquema:



Las coordenadas de dispositivo y las coordenadas de página tienen su origen inamovible en el extremo superior izquierdo del área cliente del formulario o control. El origen de las coordenadas de mundo, en cambio, aunque inicialmente se sitúa en el extremo superior izquierdo del área cliente, puede moverse a cualquier punto del área cliente del formulario o control.

Coordenadas de dispositivo (device coordinate system) Son las coordenadas por defecto inamovibles y medidas en píxeles

Coordenadas de página (page coordinate system). También son inamovibles, pero admiten diferentes unidades de medida, que se establecen llamando al método PageUnit del objeto Graphics. Este método toma como argumento un miembro de la enumeración GraphicsUnit, que especifica la unidad de medida que se utilizará:

El valor por defecto es Pixel, de modo que, si no se modifica, las coordenadas de página coinciden con las de dispositivo.

Coordenadas de mundo (world coordinate system). Salvo en la orientación de los valores X e Y, que en ningún caso se puede cambiar, son totalmente personalizables: pueden moverse y girarse y se pueden definir unidades de medida personalizadas. Las coordenadas de mundo son las utilizadas por el objeto Graphics. Si no se personalizan, a través de las Transformaciones, coinciden con las coordenadas de página.

El color

VSNET nos ofrece la estructura Color que nos permite utilizar colores predefinidos con sólo escribir su nombre: Color.Blue, Color.Green, etc. Color es una estructura, no una clase, por tanto no admite instanciación con New.

El lapicero

Antes de dibujar tenemos que definir un lapicero o "pen". Al construirlo podemos definir sólo el color o el color y el grosor:

```
Dim Lápiz As New Pen(Color.Blue)

Dim Lápiz As New Pen(Color.Blue, 10)
```

También podemos crear un lapicero sin construirlo llamando a la enumeración Pens. Veremos un ejemplo más abajo.

La brocha

Las brochas se encargan de colorear los dibujos. VSNET ofrece varios tipos de brocha:

- SolidBrush. Rellena la forma con un solo color
- HatchBrush. Rellena la forma con dos colores según un modelo predefinido.
- TextureBrush. Rellena la forma con una textura
- LinearGradientBrush. Rellena un rectángulo con dos colores y transición graduada de uno a otro.
- PathGradientBrusg. Rellena la forma con colores que emanan desde el centro y los vértices de la forma y van graduando su color hasta coincidir con la emanación del color vecino.

Formas Básicas

Punto

Un punto es un objeto compuesto por un par de valores de tipo Integer que representan las coordenadas X e Y en el eje de coordenadas. Se construye así (para un punto situado en x=10 y y=10:

```
Dim MiPunto As Point = New Point(10, 10)
```

Línea

Una línea se define por dos puntos y el trazo que los une. El método que dibuja líneas se llama DrawLine:

```
    Dim Lápiz As New Pen(Color.Blue, 10)
    Dim Punto1 As Point = New Point(10, 10)
    Dim Punto2 As Point = New Point(100, 100)
    Lienzo.DrawLine(p, t1, t2)
```

Rectángulo

Para VSNET, un rectángulo es un polígono de cuatro lados tal que, dado un lado cualquiera, existe otro paralelo a él, y además todos los vértices forman un ángulo recto. Un cuadrado, por tanto, sólo es un caso particular de rectángulo que se distingue porque todos sus lados miden lo mismo.

El objeto Rectangle se compone de las coordenadas de su vértice superior izquierdo, la anchura y la altura. Para dibujarlo invocamos al método DrawRectangle que acepta como parámetros el lapicero y un objeto Rectangle:

```
Lienzo.DrawRectangle(Lápiz, New Rectangle(100, 100, 200, 100))
```

Similar a DrawLines, disponemos del método DrawRectangles, que espera un array de rectángulos y los dibuja todos a la vez.

Elipse

El método DrawEllipse toma los mismos argumentos que el método DrawRectangle, pero en vez de dibujar un rectángulo, dibuja una elipse circunscrita en él. Para dibujar un círculo, por tanto, hay que definir un cuadrado

ANEXO 2

Código fuente de la aplicación

Módulo para lectura del archivo IGES y el almacenamiento en base de Datos.

```
Imports System
Imports System.Data.SqlClient
Imports System.IO
Module LECTURA
Public Dato(100, 100) As Punto
Public PControl(10000) As bezierDato
Public PUNCONTROL(10000) As bezierDato
Public NUMsup = 3
Public Structure punto1
Public x As Single
Public y As Single
Public z As Single
Public h As Single
Public cod As Integer
End Structure
Public matriz(100, 100) As punto1
Public matriz1(1000, 1000) As punto1
Public matriz2(10, 10) As punto1
Public matriz3(10, 10) As punto1
Public MAT(100, 100) As punto1
Public cod() As Integer
Public k1(1000) As Integer
```

```
Public k2(1000) As Integer
Public m1(1000) As Integer
Public m2(1000) As Integer
Public control(100, 100) As punto1
Public Structure bezierDato
Public PP () As puntouv
Public control (,) As punto1
Public Peso (,) As Double
Public K1 As Integer
Public K2 As Integer
Public m1 As Integer
Public m2 As Integer
Public p1 As Integer
Public p2 As Integer
Public p3 As Integer
Public p4 As Integer
Public p5 As Integer
End Structure
Function leerDato() As String
Dim strSQL As String
Dim Comando As SqlCommand
Dim d1 As Form1 = New Form1
d1.SqlConnection1.Open ()
Dim jjj As Integer
```

```
For jjj = 0 To 1000
ReDim PControl (jjj).PP (1000)
ReDim PControl(jjj).control(100, 100)
Next
ReDim PControl(10000).PP(1000)
Dim r, i, j As Integer
Dim k1, k2, N2, N1 As Integer
Dim A, C, B As Integer
Dim m1, m2 As Integer
Dim cond As Boolean
Dim sr As StreamReader
Dim cadena As String
Dim cad As String
Dim cad1 As String
Dim cadena1 As String
Dim cadena2(64) As Char
Dim s As Integer
Dim q As Integer
Dim I1, I2 As Integer
Dim iii As Integer
Dim t1, t2 As Integer
Try
sr = New StreamReader("dfn25003tm15181099.igs")
cadena = sr.ReadLine
```

```
s = 0
While cadena <> Nothing
cad = cadena.Substring(0, 3)
cad1 = cadena.Substring (3, 1)
If cad.CompareTo ("128") = 0 And cad1 = "," Then
cadena = Trim(cadena)
cond = True
q = 0
Dim bandera As Boolean
bandera = True
Do
cadena1 = cadena.Substring(0, 68)
cad = cadena1.Substring (0, 3)
If cad.CompareTo ("128") = 0 Then
I1 = 0
I2 = 0
cadena1 = cadena.Substring(4, 64)
strSQL="Insert Into SUPERFICIE2
values('"Trim(s)"',"""Trim(Val(cadena1.Substring(0,1)))"',"'Trim(Val(caden
a1.Substring(2, 1))) "',"""Trim(Val(cadena1.Substring(4, 1))) "',"'
Trim(Val(cadena1.Substring(6, 1)))"')"
Comando = New SqlCommand (strSQL, d1.SqlConnection1)
Comando.ExecuteNonQuery ()
Try
```

```
Comando.ExecuteNonQuery ()
MessageBox.Show ("Alta correcta", Application.ProductName,
MessageBoxButtons.OK, MessageBoxIcon.Information)
Catch
MessageBox.Show ("Error en Alta” Err.Description.ToString (),
Application.ProductName, MessageBoxButtons.OK,
MessageBoxIcon.Information)
End Try
PControl(s).K1 = Val (cadena1.Substring (0, 1))
PControl(s).K2 = Val(cadena1.Substring(2, 1))
PControl(s).m1 = Val(cadena1.Substring(4, 1))
PControl(s).m2 = Val(cadena1.Substring(6, 1))
PControl(s).p1 = Val(cadena1.Substring(8, 1))
PControl(s).p2 = Val(cadena1.Substring(10, 1))
PControl(s).p3 = Val(cadena1.Substring(12, 1))
PControl(s).p4 = Val(cadena1.Substring(14, 1))
PControl(s).p5 = Val(cadena1.Substring(16, 1))
cadena1 = cadena.Substring(22, 42)
cadena1 = Trim(cadena1)
N1 = 1 + PControl(s).K1 - PControl(s).m1
N2 = 1 + PControl(s).K2 - PControl(s).m2
A = N1 + 2 * PControl(s).m1
B = N2 + 2 * PControl(s).m2
C = (1 + PControl(s).K1) * (1 + PControl(s).K2)
```

```
End If
j = 0
cadena1 = Trim (cadena1)
If cadena1.EndsWith (";") Then
cond = False
End If
r = 0
Dim longitud As Boolean
longitud = True
Do
While r < cadena1.Length
longitud = False
If cadena1.Substring(r, 1) = "," Then
q = q + 1
If q <= A + 1 Then
PControl(s).PP (q).u = Val (cadena1.Substring (0, r))
End If
If q > A + 1 And q <= A + B + 1 Then
PControl(s).PP(q).v = Val(cadena1.Substring(0, r))
End If
If q > A + B + 1 And q <= A + B + C + 2 Then
If I1 <= PControl(s).K1 Then
PControl(s).control(I2, I1).h = Val(cadena1.Substring(0, r))
I1 = I1 + 1
```

```
Else
I2 = I2 + 1
I1 = 0
End If
End If
If q = A + B + C + 2 Then
I1 = 0
I2 = 0
End If
If q > A + B + C + 2 And q <= A + B + 4 * C + 2 Then
iii = iii + 1
If I1 <= PControl(s).K1 Then
If bandera = True Then
If iii = 1 Then
PControl(s).control (I2, I1).x = Val (cadena1.Substring (0, r))
‘Dato (t1, t2).x = PControl(s).control (I1, I2).x
t1 = t1 + 1
End If
Else
bandera = True
iii = 2
End If
If iii = 2 Then
PControl(s).control (I2, I1).y = Val (cadena1.Substring (0, r))
```

```
End If
If iii = 3 Then
PControl(s).control (I2, I1).z = Val (cadena1.Substring (0, r))
I1 = I1 + 1
iii = 0
End If
Else
I2 = I2 + 1
I1 = 0
iii = 0
t2 = t2 + 1
bandera = False
PControl(s).control (I2, I1).x = Val (cadena1.Substring (0, r))
End If
End If
Dim dd As Integer
Dd = cadena1. Length - r - 1
Cadena1 = cadena1. Substring (r + 1, dd)
If cadena1.Length = 0 Then
longitud = False
End If
r = 0
Else
r = r + 1
```

```
End If
End While
r = 0
Loop While longitud
Console.WriteLine (cadena1)
cadena = sr.ReadLine
Loop While cond
For i = 0 To PControl(s).K1
For j = 0 To PControl(s).K2
strSQL = "Insert Into CONTROL2 values('" Trim(Str(s)) "'," "" Trim(Str(PControl(s).control(j, i).x)) "','" Trim(Str(PControl(s).control(j, i).y)) "'," "" Trim(Str(PControl(s).control(j,i).z)) "','" Trim(Str(PControl(s).control(j, i).h)) "')"
Comando = New SqlCommand (strSQL, d1.SqlConnection1)
Comando.ExecuteNonQuery ()
Next
Next
s = s + 1
End If
cadena = sr.ReadLine
End While
Console.WriteLine(r)
Catch ex As Exception
Console.WriteLine ("Error:" + ex.Message)
```

```
Finally
If Not sr Is Nothing Then
sr.Close()
End If
End Try
d1.SqlConnection1.Close ()
End Function
End Module
Function Leecontrol () As String
Dim iii As Integer
For iii = 0 To 1000
ReDim PUNCONTROL(iii).PP(1000)
ReDim PUNCONTROL (iii).control(10, 10)
Next
ReDim PUNCONTROL(10000).PP(100)
Dim strSQL As String
Dim strSQL1 As String
strSQL = "SELECT *FROM SUPERFICIE2"
Dim Comando As SqlDataAdapter
Dim d As Form1 = New Form1
Comando = New SqlDataAdapter(strSQL, d.SqlConnection1)
Dim MiDataSet As DataSet
MiDataSet = New DataSet
Comando.Fill(MiDataSet, "SUPERFICIE2")
```

```
Dim MiTabla As DataTable = MiDataSet.Tables("SUPERFICIE2")
Dim columna As DataColumn
Dim fila As DataRow
Dim i1 As Integer
Dim v As Integer
v = MiTabla.Rows.Count ()
ReDim cod(v), k1(v), k2(v), m1(v), m2(v)
i1 = 0
For Each fila In MiTabla.Rows
Cod (i1) = fila (0)
k1 (i1) = fila (1)
k2 (i1) = fila (2)
m1 (i1) = fila (3)
m2 (i1) = fila (4)
i1 = i1 + 1
Next
Dim i2, j2 As Integer
strSQL1 = "SELECT *FROM CONTROL2"
Dim Comando1 As SqlDataAdapter
Comando1 = New SqlDataAdapter (strSQL1, d.SqlConnection1)
Dim MiDataSet1 As DataSet
MiDataSet1 = New DataSet
Comando1. Fill(MiDataSet1, "CONTROL2")
Dim MiTabla1 As DataTable = MiDataSet1.Tables("CONTROL2")
```

```
Dim columna1 As DataColumn
Dim fila1 As DataRow
Dim V1 As Integer
V1 = MiTabla1.Rows.Count()
Dim cod1(V1), X(V1), Y(V1), Z(V1), W(V1), cod2(V1), X1(V1), Y1(V1),
Z1(V1), W1(V1) As Double
i1 = 0
For Each fila1 In MiTabla1.Rows
cod1(i1) = fila1(0)
X(i1) = fila1(1)
Y(i1) = fila1(2)
Z(i1) = fila1(3)
W(i1) = fila1(4)
i1 = i1 + 1
Next
i1 = 0
For Each fila1 In MiTabla1.Rows
cod2(i1) = fila1(0)
X1(i1) = fila1(1)
Y1(i1) = fila1(2)
Z1(i1) = fila1(3)
W1(i1) = fila1(4)
i1 = i1 + 1
Next
```

```
Dim k3 As Integer
k3 = 0
For i2 = 0 To k1(0)
For j2 = 0 To k2(0)
matriz(i2, j2).x = X(k3)
matriz(i2, j2).y = Y(k3)
matriz(i2, j2).z = Z(k3)
matriz(i2, j2).h = W(k3)
matriz(i2, j2).cod = cod1(k3)
k3 = k3 + 1
Next
Next
Dim i As Integer
Dim suma, suma1 As Integer
suma1 = 0
suma = 0
Dim L, PPPP As Integer
L = k1.Length - 2
PPPP = k2.Length
For i = 0 To k1.Length - 2
suma = suma + k1(i)
suma1 = suma1 + k2(i)
Next
k3 = 0
```

```
For i2 = 0 To 20
For j2 = 0 To 20
MAT(i2, j2).x = X(k3)
MAT(i2, j2).y = Y(k3)
MAT(i2, j2).z = Z(k3)
MAT(i2, j2).h = W(k3)
MAT(i2, j2).cod = cod1(k3)
k3 = k3 + 1
Next
Next
Dim S As Integer
Dim J As Integer
Dim JJ As Integer
JJ = 0
For S = 0 To v
For i = 0 To k2(S)
For J = 0 To k1(S)
PUNCONTROL(S).control(i, J).x = X(JJ)
PUNCONTROL(S).control(i, J).y = Y(JJ)
PUNCONTROL(S).control(i, J).z = Z(JJ)
JJ = JJ + 1
Next
Next
Next
```

```
End Function
```

Módulo que contiene las funciones para la implementación de triangulación de Delaunay en el plano.

```
Option Strict Off
Option Explicit On
Module Module3
Public Structure dVertice
Public x As Single
Public y As Single
Public z As Single
End Structure
Public Structure dTriangulo
Public v0 As Integer
Public v1 As Integer
Public v2 As Integer
Public precal As Integer
Public xc As Double
Public yc As Double
Public r As Double
Public cx As Double
Public cy As Double
Public vv0 As Double
Public uu0 As Double
Public vv1 As Double
```

```
Public uu1 As Double
Public vv2 As Double
Public uu2 As Double
End Structure
Public Const maxVertices As Int64 = 500000
Public Const. maxTriangulo As Integer = 500000
Public Vertice(maxwelliens) As dVertice
Public Vertice1(maxVertices) As dVertice
Public Triangulo(maxTriangulo) As dTriangulo
Public Npuntos As Integer
Public ntri As Integer
Public Function puntos() As Integer
Dim i, j As Integer
For i = 1 To 400
Vertice(i).x = 0
Vertice(i).y = 0
Next
Dim ali As Random = New Random
Dim t, s As Single
For j = 0 To 300
Vertice(Npuntos).x = CType(ali.NextDouble, Single)
Vertice(Npuntos).y = CType(ali.NextDouble, Single)
Triangulo(j).cx = Vertice(Npuntos).x
Triangulo(j).cy = Vertice(Npuntos).y
```

```
Npuntos = Npuntos + 1
Next j
Return Npuntos
End Function
Private Function InCirculo(ByVal xp As Double, ByVal yp As Double, ByVal x1 As Double, ByVal y1 As Double, ByVal x2 As Double, ByVal y2 As Double, ByVal x3 As Double, ByVal y3 As Double, ByRef xc As Double, ByRef yc As Double, ByRef r As Double, ByVal j As Integer) As Boolean
Dim resul As Boolean
Dim eps As Double
Dim m1 As Double
Dim m2 As Double
Dim mx1 As Double
Dim mx2 As Double
Dim my1 As Double
Dim my2 As Double
Dim dx As Double
Dim dy As Double
Dim rsqr As Double
Dim drsqr As Double
eps = 0.00001
InCirculo = False
If Triangulo(j).precal = 1 Then
```

```
xc = Triangulo(j).xc
yc = Triangulo(j).yc
r = Triangulo(j).r
rsqr = r * r
dx = xp - xc
dy = yp - yc
drsqr = dx * dx + dy * dy
Else
If System.Math.Abs(y1 - y2) < eps And System.Math.Abs(y2 - y3) < eps
Then
InCirculo = False
Return InCirculo
Exit Function
End If
If System.Math.Abs(y2 - y1) < eps Then
m2 = -(x3 - x2) / (y3 - y2)
mx2 = (x2 + x3) / 2
my2 = (y2 + y3) / 2
xc = (x2 + x1) / 2
yc = m2 * (xc - mx2) + my2
ElseIf System.Math.Abs(y3 - y2) < eps Then
m1 = -(x2 - x1) / (y2 - y1)
mx1 = (x1 + x2) / 2
my1 = (y1 + y2) / 2
```

```
xc = (x3 + x2) / 2
yc = m1 * (xc - mx1) + my1
Else
m1 = -(x2 - x1) / (y2 - y1)
m2 = -(x3 - x2) / (y3 - y2)
mx1 = (x1 + x2) / 2
mx2 = (x2 + x3) / 2
my1 = (y1 + y2) / 2
my2 = (y2 + y3) / 2
If (m1 - m2) <> 0 Then
xc = (m1 * mx1 - m2 * mx2 + my2 - my1) / (m1 - m2)
yc = m1 * (xc - mx1) + my1
Else
xc = (x1 + x2 + x3) / 3
yc = (y1 + y2 + y3) / 3
End If
End If
dx = x2 - xc
dy = y2 - yc
rsqr = dx * dx + dy * dy
r = System.Math.Sqrt(rsqr)
dx = xp - xc
dy = yp - yc
drsqr = dx * dx + dy * dy
```

```
Triangulo(j).precal = 1
Triangulo(j).xc = xc
Triangulo(j).yc = yc
Triangulo(j).r = r
Triangulo(j).uu0 = x1
Triangulo(j).vv0 = y1
Triangulo (j).uu1 = x2
Triangulo (j).vv1 = y2
Triangulo (j).uu2 = x3
Triangulo (j).vv2 = y3
End If
If drsqr <= rsqr Then
InCirculo = True
End If
End Function
Private Function Posicion(ByVal xp As Double, ByVal yp As Double, ByVal x1 As Double, ByVal y1 As Double, ByVal x2 As Double, ByVal y2 As Double) As Integer
Dim ecuacion As Double
ecuacion = ((yp - y1) * (x2 - x1)) - ((y2 - y1) * (xp - x1))
If ecuacion > 0 Then
Posicion = -1
Return Posicion
ElseIf ecuacion = 0 Then
```

```
Posicion = 0
Return Posicion
Else
Posicion = 1
Return Posicion
End If
End Function
Public Function Triangulacion(ByVal NVert As Integer) As Integer
Dim completa (maxTriangulo) As Boolean
Dim Caras(2, maxTriangulo * 3) As Integer
Dim NUmCaras As Integer
Dim xmin As Double
DIM xmax As Double
DIM ymin As Double
DIM ymax As Double
DIM xmid As Double
DIM ymid As Double
DIM dx As Double
Dim dy As Double
Dim dmax As Double
Dim i, j, k As Short
Dim xc As Double = 0.0
Dim yc As Double = 0.0
Dim r As Double = 0.0
```

```
Dim inc As Boolean
xmin = Vertice(1).x
ymin = Vertice(1).y
xmax = xmin
ymax = ymin
For i = 2 To NVert
If Vertice(i).x < xmin Then
xmin = Vertice(i).x
End If
If Vertice(i).x > xmax Then
xmax = Vertice(i).x
End If
If Vertice(i).y < ymin Then
ymin = Vertice(i).y
End If
If Vertice(i).y > ymax Then
ymax = Vertice(i).y
End If
Next i
dx = xmax - xmin
dy = ymax - ymin
If dx > dy Then
dmax = dx
Else
```

```
dmax = dy
End If
xmid = (xmax + xmin) / 2
ymid = (ymax + ymin) / 2
Vertice(NVert + 1).x = (xmid - 2 * dmax)
Vertice(NVert + 1).y = (ymid - dmax)
Vertice(NVert + 2).x = (xmid)
Vertice(NVert + 2).y = (ymid + 2 * dmax)
Vertice(NVert + 3).x = (xmid + 2 * dmax)
Vertice(NVert + 3).y = (ymid - dmax)
Triangulo(1).v0 = NVert + 1
Triangulo(1).v1 = NVert + 2
Triangulo(1).v2 = NVert + 3
Triangulo(1).precal = 0
completa(1) = False
ntri = 1
For i = 1 To NVert
NUmCaras = 0
j = 0
Do
j = j + 1
If completa (j) <> True Then
inc = InCirculo(Vertice(i).x, Vertice(i).y, Vertice(Triangulo(j).v0).x,
Vertice(Triangulo(j).v0).y, Vertice(Triangulo(j).v1).x,
```

```
Vertice(Triangulo(j).v1).y,                    Vertice(Triangulo(j).v2).x,
Vertice(Triangulo(j).v2).y, xc, yc, r, j)
If (xc + r) < Vertice (i).x Then
completa(j) = True
Else
If inc Then
Caras(1, NUmCaras + 1) = Triangulo(j).v0
Caras(2, NUmCaras + 1) = Triangulo(j).v1
Caras(1, NUmCaras + 2) = Triangulo(j).v1
Caras (2, NUmCaras + 2) = Triangulo(j).v2
Caras (1, NUmCaras + 3) = Triangulo(j).v2
Caras (2, NUmCaras + 3) = Triangulo(j).v0
NUmCaras = NUmCaras + 3
Triangulo(j).v0 = Triangulo(ntri).v0
Triangulo(j).v1 = Triangulo(ntri).v1
Triangulo(j).v2 = Triangulo(ntri).v2
Triangulo(j).precal = Triangulo(ntri).precal
Triangulo(j).xc = Triangulo(ntri).xc
Triangulo(j).yc = Triangulo(ntri).yc
Triangulo(j).r = Triangulo(ntri).r
Triangulo(ntri).precal = 0
completa(j) = completa(ntri)
j = j - 1
ntri = ntri - 1
```

```
End If

End If

End If

Loop While j < ntri

For j = 1 To NUmCaras - 1

If Caras(1, j) <> 0 And Caras(2, j) <> 0 Then

For k = j + 1 To NUmCaras

If Caras (1, k) <> 0 And Caras (2, k) <> 0 Then

If Caras (1, j) = Caras (2, k) Then

If Caras (2, j) = Caras (1, k) Then

Caras (1, j) = 0

Caras (2, j) = 0

Caras (1, k) = 0

Caras(2, k) = 0

End If

End If

End If

Next k

End If

Next j

For j = 1 To NUmCaras

If Caras(1, j) <> 0 And Caras(2, j) <> 0 Then

ntri = ntri + 1

Triangulo(ntri).v0 = Caras(1, j)
```

```
Triangulo(ntri).v1 = Caras(2, j)
Triangulo(ntri).v2 = i
Triangulo(ntri).precal = 0
completa (ntri) = False
End If
Next j
Next i
i = 0
Do
i = i + 1
If Triangulo(i).v0 > NVert Or Triangulo(i).v1 > NVert Or Triangulo(i).v2 >
NVert Then
Triangulo (i).v0 = Triangulo(ntri).v0
Triangulo (i).v1 = Triangulo(ntri).v1
Triangulo (i).v2 = Triangulo(ntri).v2
Triangulo(i).uu0 = Vertice(Triangulo(j).v0).x
Triangulo(i).vv0 = Vertice(Triangulo(j).v0).y
Triangulo(i).uu1 = Vertice(Triangulo(j).v1).x
Triangulo(i).vv1 = Vertice(Triangulo(j).v1).y
Triangulo(i).uu2 = Vertice(Triangulo(j).v2).x
Triangulo(i).vv2 = Vertice(Triangulo(j).v2).y
i = i - 1
ntri = ntri - 1
End If
```

```
Loop While i < ntri
Triangulacion = ntri
Return Triangulacion
End Function
Public Function ordena(ByRef Vertice() As dVertice, ByVal pri As Integer,
ByVal ult As Integer) As dVertice()
Dim i, c, j As Integer
Dim piv, aux As Double
i = pri
j = ult
c = CType((pri + ult) / 2, Integer)
piv = Vertice(c).x
Do While i <= j
While (Vertice(i).x < piv)
i = i + 1
End While
While (Vertice(j).x > piv)
j = j - 1
End While
If (i <= j) Then
aux = Vertice(i).x
Vertice(i).x = Vertice(j).x
Vertice(j).x = aux
i = i + 1
```

```
j = j - 1
End If
Loop
If (pri < j) Then
ordena(Vertice, pri, j)
End If
If (i < ult) Then
ordena (Vertice, i, ult)
End If
End Function
End Module
```

Modulo para la implementación del mallado del objeto ·3D con control de error .

```
Module Module1
Public Structure d3Vertice
Public x As Single
Public y As Single
Public z As Single
End Structure
Dim px(90), py(90), cx, cy As Single
Public TRAS = 20
Public Structure d3Triangulo
Public v0 As Integer
Public v1 As Integer
```

```
Public v2 As Integer
Public uu0 As Double
Public vv0 As Double
Public uu1 As Double
Public vv1 As Double
Public uu2 As Double
Public vv2 As Double
End Structure
Public Const maxVertices As Int64 = 50000
Public Const maxTriangulo As Integer = 90000
Public Const maxParticion As Integer = 3
Public Vertice3D (maxVertices) As d3Vertice
Public d(maxParticion, maxParticion) As Double
Public Triangulo3d (maxTriangulo) As d3Triangulo
Public Npunto As Integer
Public nvert As Integer
Public u(maxParticion) As Single
Public v(maxParticion) As Single
Public u0, u1, v0, v1 As Double
Public Const n1 = 4
Public p1(n1, n1) As Punto
Public p(n1, n1) As Punto
Public puntoReal(n1) As Single
Public puntoProy(n1) As Single
```

```
Public puntoPAn As Single
Public Const pasos = 20
Public sup(6000) As bezierDato
Public supB(1000, 1000) As Punto
Public supv0(pasos, pasos) As Punto
Public supv1(pasos, pasos) As Punto
Public supv2(pasos, pasos) As Punto
Public m, n As Integer
Public Structure Punto
Public x As Single
Public y As Single
Public z As Single
End Structure
Public Structure puntouv
Dim u As Single
Dim v As Single
End Structure
Public uv(pasos, pasos) As puntouv
Function factorial(ByVal n As Integer) As Double
If (n <= 0) Then
Return 1
Else
Return n * factorial(n - 1)
End If
```

```
End Function
Function Ni(ByVal n As Integer, ByVal i As Integer) As Double
Dim co As Double
co = factorial(n) / (factorial(i) * factorial(n - i))
Return co
End Function
Function BAse(ByVal n As Integer, ByVal i As Integer, ByVal t As Double)
As Double
Dim bas As Double
Dim ti As Double
Dim tni As Double
If (t = 0 And i = 0) Then
ti = 1.0
Else
ti = Math.Pow(t, i)
End If
If (n = i And t = 1) Then
tni = 1.0
Else
tni = Math.Pow(1 - t, n - i)
End If
bas = Ni(n, i) * ti * tni
Return bas
End Function
```

```
Function SuperB(ByVal n As Integer, ByVal p As Integer, ByVal m As Integer, ByVal q As Integer, ByRef supB(,) As Punto, ByVal H As Graphics)
Dim up As Double
Dim vp As Double
Dim NU(1000), NV(1000) As Double
Dim ui, vi As Double
Dim l, k As Integer
Dim puntoreal As Punto
Dim UU(1000), VV(1000) As Double
Dim pi, pj As Int16
Dim pp As Punto
NM(UU, p, n)
NM(VV, q, m)
Dim temp(100) As Punto
For pj = 1 To pasos
v1 = pj / pasos
vp = Span(m, q, v1, VV)
For pi = 1 To pasos
u1 = pi / pasos
up = Span(n, p, u1, UU)
ui = up - p
'u1 = u1 + pi / 1200
NN1(up, u1, p, UU, NU)
```

```
NN1 (vp, v1, q, VV, NV)
puntoreal.x = 0
puntoreal.y = 0
puntoreal.z = 0
For l = 0 To q
For k = 0 To p
puntoreal.x = 0
puntoreal.y = 0
puntoreal.z = 0
vi = vp - q + 1
puntoreal.x = puntoreal.x + NU(k) * MAT(ui + k, vi).x
puntoreal.y = puntoreal.y + NU(k) * MAT(ui + k, vi).y
puntoreal.z = puntoreal.z + NU(k) * MAT(ui + k, vi).z
pp.x = pp.x + NV(l) * puntoreal.x
pp.y = pp.y + NV(l) * puntoreal.y
pp.z = pp.z + NV(l) * puntoreal.z
Next
Next
supB (pj, pi).x = pp.x
supB(pj, pi).y = pp.y
SupB(pj, pi).z = pp.z
Next
Next
Return supB
```

```
End Function
Public Function Particion(ByVal n As Integer, ByVal m As Integer, ByVal
u0 As Double, ByVal u1 As Double, ByVal v0 As Double, ByVal v1 As
Double) As puntouv(,)
Dim Division(maxParticion, maxParticion) As puntouv
Dim i, j As Integer
For i = 0 To n
u(i) = 0
Next
For j = 0 To m
v(j) = 0
Next
For i = 0 To n
u(i) = Math.Abs((u1 - u0)) * (i / n)
For j = 0 To m
v(j) = Math.Abs((v1 - v0)) * (j / m)
Division (i, j).u = u(i)
Division (i, j).v = v(j)
Next j
Next i
Return Division
End Function
Public Function BezierN(ByVal u As Single, ByVal v As Single, ByVal nn
As Integer, ByVal mm As Integer, ByVal control() As bezierDato) As Punto
```

```
Dim PPP As Punto
Dim pi, pj As Integer
Dim i, j As Integer
Dim bu, bv As Double
Dim s As Integer
PPP.x = 0
PPP.y = 0
PPP.z = 0
For s = 0 To 0
'k1.Length - 2
For j = 0 To k1(s)
bv = BAse(k1(s), j, v)
For i = 0 To k2(s)
bu = BAse(k2(s), i, u) * bv
PPP.x = PPP.x + control(s).control (i, j).x * bu
PPP.y = PPP.y + control(s).control (i, j).y * bu
PPP.z = PPP.z + control(s).control (i, j).z * bu
Next
Next
Next
Return PPP
End Function
Public Function Norma(ByVal B As Punto, ByVal PP1(,) As puntouv,
ByVal nn As Integer, ByVal mm As Integer, ByVal control() As bezierDato)
```

```
Dim k, i, j As Integer
Dim aa, bb, cc As Single
Dim max As Double = 200
Dim s As Integer
For s = 0 To 0
For i = 0 To maxParticion
For j = 0 To maxParticion
aa = Math.Abs(BezierN(PP1(i, j).u, PP1(i, j).v, nn, mm, control).x - B.x)
bb = Math.Abs(BezierN(PP1(i, j).u, PP1(i, j).v, nn, mm, control).y - B.y)
cc = Math.Abs(BezierN(PP1(i, j).u, PP1(i, j).v, nn, mm, control).z - B.z)
max = aa
If bb > max Then
max = bb
End If
If cc > max Then
max = cc
End If
d(i, j) = max
Next j
Next i
Next
Return d
End Function
```

```
Public Function Minimo(ByRef u0 As Double, ByRef u1 As Double, ByRef
v0 As Double, ByRef v1 As Double, ByRef d(,) As Double) As Double
Dim i As Integer
Dim men1 As Double
Dim men2 As Double
Dim dd As Double
Dim k1, k2 As Integer
Dim pos1u, pos2u, pos1v, pos2v As Double
Dim nn, j As Integer
men2 = 800
men1 = 600
pos1v = 2
pos2v = 1
For i = 0 To maxParticion
For j = 0 To maxParticion
If i = j Then
If d(i, j) <= men1 Then
men2 = men1
pos2v = pos1v
pos2u = pos1u
men1 = d(i, j)
pos1v = j
pos1u = i
Else
```

```
If d(i, j) < men2 Then
men2 = d(i, j)
pos2v = j
pos2u = i
End If
End If
End If
Next
Next
If u(pos1u) < u(pos2u) Then
u0 = u(pos1u)
u1 = u(pos2u)
Else
u0 = u(pos2u)
u1 = u(pos1u)
End If
If v(pos1v) < v(pos2v) Then
v0 = v(pos1v)
v1 = v(pos2v)
Else
v0 = v(pos2v)
v1 = v(pos1v)
End If
Return men1
```

```
End Function
Public Function MenNorma(ByVal rr() As Double, ByVal n As Integer) As
Double
Dim k, j As Integer
Dim may As Double
Dim menor As Double
menor = 1000
For k = 1 To n
If rr(k) < menor Then
menor = rr(k)
End If
Next
Return menor
End Function
Public Function DerivadaBeU(ByVal u As Single, ByVal v As Single,
ByVal nn As Integer, ByVal mm As Integer) As Punto
Dim D1 As Punto
Dim pi, pj As Integer
Dim i, j As Integer
Dim bu, bv As Double
Dim s As Integer
D1.x = 0
D1.y = 0
D1.z = 0
```

```
For s = 0 To 0
For j = 0 To k1(s)
bv = BAse(k1(s), j, v)
For i = 0 To k2(s)
bu = BAse(k2(s), i, u) * bv
If u <> 0 And u <> 1 Then
D1.x = D1.x + (i - nn * u) / (u * (1 - u)) * PUNCONTROL(s).control (i, j).x
* bu
D1.y = D1.y + (i - nn * u) / (u * (1 - u)) * PUNCONTROL(s).control (i, j).y
* bu
D1.z = D1.z + (i - nn * u) / (u * (1 - u)) * PUNCONTROL(s).control (i, j).z
* bu
End If
Next
Next
Next
Return D1
End Function
Public Function DerivadaBeV(ByVal u As Single, ByVal v As Single,
ByVal nn As Integer, ByVal mm As Integer) As Punto
Dim D2 As Punto
Dim pi, pj As Integer
Dim i, j As Integer
Dim bu, bv As Double
```

```
Dim S As Integer
D2.x = 0
D2.y = 0
D2.z = 0
For S = 0 To 0
For j = 0 To k1(S)
bv = BAse(k1(S), j, v)
For i = 0 To k2(S)
bu = BAse(k2(S), i, u) * bv
If v <> 0 And v <> 1 Then
D2.x = D2.x + (j - k1(S) * v) / (v * (1 - v)) * PUNCONTROL(S).control(i,
j).x * bu
D2.y = D2.y + (j - k1(S) * v) / (v * (1 - v)) * PUNCONTROL(S).control(i,
j).y * bu
D2.z = D2.z + (j - k1(S) * v) / (v * (1 - v)) * PUNCONTROL(S).control(i,
j).z * bu
End If
Next
Next
Next
Return D2
End Function
Public Function DerivadaBeUV(ByVal u As Single, ByVal v As Single,
ByVal nn As Integer, ByVal mm As Integer) As Punto
```

```
Dim D12 As Punto
Dim pi, pj As Integer
Dim i, j As Integer
Dim bu, bv As Double
Dim S As Integer
D12.x = 0
D12.y = 0
D12.z = 0
For S = 0 To 0
For j = 0 To k1(S)
bv = BAse(k1(S), j, v)
For i = 0 To k2(S)
bu = BAse(k2(S), i, u) * bv
If u <> 0 And u <> 1 And v <> 0 And v <> 1 Then
D12.x = D12.x + ((i - k2(S) * u) / (u * (1 - u))) * ((j - k1(S) * v) / (v * (1 -
v))) * PUNCONTROL(S).control(i, j).x * bu
D12.y = D12.y + ((i - k2(S) * u) / (u * (1 - u))) * ((j - k1(S) * v) / (v * (1 -
v))) * PUNCONTROL(S).control(i, j).y * bu
D12.z = D12.z + ((i - k2(S) * u) / (u * (1 - u))) * ((j - k1(S) * v) / (v * (1 -
v))) * PUNCONTROL(S).control(i, j).z * bu
End If
Next
Next
Next
```

```
Return D12
End Function

Public Function DerivadaBeVU(ByVal u As Single, ByVal v As Single,
ByVal nn As Integer, ByVal mm As Integer) As Punto
Dim D12 As Punto
Dim pi, pj As Integer
Dim i, j As Integer
Dim bu, bv As Double
Dim S As Integer
D12.x = 0
D12.y = 0
D12.z = 0
For S = 0 To 0
For j = 0 To k1(S)
bv = BAse(k1(S), j, v)
For i = 0 To k2(S)
bu = BAse(k2(S), i, u) * bv
If u <> 0 And u <> 1 And v <> 0 And v <> 1 Then
D12.x = D12.x + ((j - k1(S) * v) / (v * (1 - v))) * ((i - k2(S) * u) / (u * (1 -
u))) * PUNCONTROL(S).control (i, j).x * bu
D12.y = D12.y + ((j - k1(S) * v) / (v * (1 - v))) * ((i - k2(S) * u) / (u * (1 -
u))) * PUNCONTROL(S).control (i, j).y * bu
```

```
D12.z = D12.z + ((j - k1(S) * v) / (v * (1 - v))) * ((i - k2(S) * u) / (u * (1 - u))) * PUNCONTROL(S).control (i, j).z * bu
End If
Next
Next
Next
Return D12
End Function
Public Function DerivadaBeUU(ByVal u As Single, ByVal v As Single, ByVal nn As Integer, ByVal mm As Integer) As Punto
Dim D11 As Punto
Dim pi, pj As Integer
Dim i, j As Integer
Dim bu, bv As Double
Dim S As Integer
D11.x = 0
D11.y = 0
D11.z = 0
For S = 0 To 0
For j = 0 To k1(S)
bv = BAse(k1(S), j, v)
For i = 0 To k2(S)
bu = BAse(k2(S), i, u) * bv
If u <> 0 And u <> 1 Then
```

```
D11.x = D11.x + ((i - k2(S) * u) * (i - k2(S) * u) - k2(S) * u * u - i * (1 - 2 *
u)) / (u * u * (1 - u) * (1 - u)) * PUNCONTROL(S).control (i, j).x * bu
D11.y = D11.y + ((i - k2(S) * u) * (i - k2(S) * u) - k2(S) * u * u - i * (1 - 2 *
u)) / (u * u * (1 - u) * (1 - u)) * PUNCONTROL(S).control (i, j).y * bu
D11.z = D11.z + ((i - k2(S) * u) * (i - k2(S) * u) - k2(S) * u * u - i * (1 - 2 *
u)) / (u * u * (1 - u) * (1 - u)) * PUNCONTROL(S).control (i, j).z * bu
End If
Next
Next
Next
Return D11
End Function
Public Function DerivadaBeVV(ByVal u As Single, ByVal v As Single,
ByVal nn As Integer, ByVal mm As Integer) As Punto
Dim D22 As Punto
Dim pi, pj As Integer
Dim i, j As Integer
Dim bu, bv As Double
Dim S As Integer
D22.x = 0
D22.y = 0
D22.z = 0
For S = 0 To 0
For j = 0 To k1(S)
```

```
bv = BAse(k1(S), j, v)
For i = 0 To k2(S)
bu = BAse(k2(S), i, u) * bv
If v <> 0 And v <> 1 Then
D22.x = D22.x + ((j - k1(S) * v) * (j - k1(S) * v) - k1(S) * v * v - j * (1 - 2 *
v)) / (v * v * (1 - v) * (1 - v)) * PUNCONTROL(S).control (i, j).x * bu
D22.y = D22.y + ((j - k1(S) * v) * (j - k1(S) * v) - k1(S) * v * v - j * (1 - 2 *
v)) / (v * v * (1 - v) * (1 - v)) * PUNCONTROL(S).control (i, j).y * bu
D22.z = D22.z + ((j - k1(S) * v) * (j - k1(S) * v) - k1(S) * v * v - j * (1 - 2 *
v)) / (v * v * (1 - v) * (1 - v)) * PUNCONTROL(S).control (i, j).z * bu
End If
Next
Next
Next
Return D22
End Function
Public Function NewtonNoLineal(ByVal u0 As Double, ByVal v0 As
Double, ByVal nn As Integer, ByVal mm As Integer, ByVal P As Punto,
ByVal tol As Double, ByVal n As Integer, ByVal control() As bezierDato)
As puntouv
Dim i As Integer
Dim T As Punto
Dim T1 As Punto
Dim a, b, c, d, DU, DV, DUV, DVU As Double
```

```
Dim F1, F2 As Double
Dim J1, J2, J3, J4 As Double
Dim Nor As Double
Dim INVJ1, INVJ2, INVJ3, INVJ4 As Double
Dim X1 As puntouv
Dim X As puntouv
i = 1
While (i <= n)
T.x = P.x - BezierN (u0, v0, nn, mm, control).x
T.y = P.y - BezierN (u0, v0, nn, mm, control).y
T.z = P.z - BezierN (u0, v0, nn, mm, control).z
F1 = -2 * (T.x * Module1.DerivadaBeU (u0, v0, nn, mm).x + T.y *
Module1.DerivadaBeU (u0, v0, nn, mm).y + T.z * Module1.DerivadaBeU
(u0, v0, nn, mm).z)
F2 = -2 * (T.x * Module1.DerivadaBeV (u0, v0, nn, mm).x + T.y *
Module1.DerivadaBeV (u0, v0, nn, mm).y + T.z * Module1.DerivadaBeV
(u0, v0, nn, mm).z)
DU = DerivadaBeU (u0, v0, nn, mm).x * DerivadaBeU (u0, v0, nn, mm).x +
DerivadaBeU (u0, v0, nn, mm).y * DerivadaBeU (u0, v0, nn, mm).y +
DerivadaBeU (u0, v0, nn, mm).z * DerivadaBeU (u0, v0, nn, mm).z
DUV = DerivadaBeU (u0, v0, nn, mm).x * DerivadaBeV (u0, v0, nn, mm).x
+ DerivadaBeU (u0, v0, nn, mm).y * DerivadaBeV (u0, v0, nn, mm).y +
DerivadaBeU (u0, v0, nn, mm).z * DerivadaBeV (u0, v0, nn, mm).z
```

```
DVU = DerivadaBeV (u0, v0, nn, mm).x * DerivadaBeU (u0, v0, nn, mm).x
+ DerivadaBeV (u0, v0, nn, mm).y * DerivadaBeU (u0, v0, nn, mm).y +
DerivadaBeV (u0, v0, nn, mm).z * DerivadaBeU (u0, v0, nn, mm).z
DV = DerivadaBeV (u0, v0, nn, mm).x * DerivadaBeV (u0, v0, nn, mm).x +
DerivadaBeV(u0, v0, nn, mm).y * DerivadaBeV(u0, v0, nn, mm).y +
DerivadaBeV(u0, v0, nn, mm).z * DerivadaBeV(u0, v0, nn, mm).z
T1.x = BezierN (u0, v0, nn, mm, control).x - P.x
T1.y = BezierN (u0, v0, nn, mm, control).y - P.y
T1.z = BezierN (u0, v0, nn, mm, control).y - P.z
J1 = 2 * DU + 2 * (T1.x * DerivadaBeUU (u0, v0, nn, mm).x + T1.y *
DerivadaBeUU (u0, v0, nn, mm).y + T1.z * DerivadaBeUU (u0, v0, nn,
mm).z)
J2 = 2 * DVU + 2 * (T1.x * DerivadaBeVU (u0, v0, nn, mm).x + T1.y *
DerivadaBeVU (u0, v0, nn, mm).y + T1.z * DerivadaBeVU (u0, v0, nn,
mm).z)
J3 = 2 * DUV + 2 * (T1.x * DerivadaBeUV (u0, v0, nn, mm).x + T1.y *
DerivadaBeUV (u0, v0, nn, mm).y + T1.z * DerivadaBeUV (u0, v0, nn,
mm).z)
J4 = 2 * DV + 2 * (T1.x * DerivadaBeVV (u0, v0, nn, mm).x + T1.y *
DerivadaBeVV (u0, v0, nn, mm).y + T1.z * DerivadaBeVV (u0, v0, nn,
mm).z)
If (J1 * J4 - J2 * J3) <> 0 Then
INVJ1 = J4 / (J1 * J4 - J2 * J3)
End If
```

```
If (J1 * J4 - J2 * J3) <> 0 Then
INVJ2 = -J2 / (J1 * J4 - J2 * J3)
End If
If (J1 * J4 - J2 * J3) <> 0 Then
INVJ3 = -J3 / (J1 * J4 - J2 * J3)
End If
If (J1 * J4 - J2 * J3) <> 0 Then
INVJ4 = J1 / (J1 * J4 - J2 * J3)
End If
X1.u = INVJ1 * F1 + INVJ2 * F2
X1.v = INVJ3 * F1 + INVJ4 * F2
X.u = u0
X.v = v0
Nor = Math.Sqrt(X1.u * X1.u + X1.v * X1.v)
If Nor < tol Then
Return X
Else
u0 = u0 + X1.u
v0 = v0 + X1.u
End If
i = i + 1
End While
End Function
```

```
Public Function PMALLADONEWTON2(ByVal H As Graphics, ByVal
control() As bezierDato) As d3Vertice()
Dim strSQL As String
Dim strSQL1 As String
strSQL = "SELECT *FROM SUPERFICIE2"
Dim Comando As SqlDataAdapter
Dim ventana As Form1 = New Form1
Comando = New SqlDataAdapter (strSQL, ventana.SqlConnection1)
Dim MiDataSet As DataSet
MiDataSet = New DataSet
Comando.Fill(MiDataSet, "SUPERFICIE2")
Dim MiTabla As DataTable = MiDataSet.Tables("SUPERFICIE2")
Dim columna As DataColumn
Dim fila As DataRow
Dim i1 As Integer
Dim vv As Integer
vv = MiTabla.Rows.Count()
Dim cod(vv), k1(vv), k2(vv), m1(vv), m2(vv) As Integer
i1 = 0
For Each fila In MiTabla.Rows
cod(i1) = fila(0)
k1(i1) = fila(1)
k2(i1) = fila(2)
m1(i1) = fila(3)
```

```
m2(i1) = fila(4)
i1 = i1 + 1
Next
Dim pen1 As Pen = New Pen(Color.Red, 1000)
Dim pen2 As Pen = New Pen(Color.Blue, 0.01)
Dim N, I As Integer
Dim b As Punto
Dim v() As Double
Dim u() As Double
Dim PuntoRealV0 As Punto
Dim PuntoRealV1 As Punto
Dim PuntoRealV2 As Punto
Dim PProyectaV0 As Punto
Dim PProyectaV1 As Punto
Dim PProyectaV2 As Punto
Dim PuntoPanV0 As Punto
Dim PuntoPanV1 As Punto
Dim PuntoPanV2 As Punto
Dim PuntoPan1V0 As Punto
Dim PuntoPan1V1 As Punto
Dim PuntoPan1V2 As Punto
Dim pi, pj, k, j As Integer.
Dim bu0, bv0 As Double
Dim bu1, bv1 As Double.
```

```
Dim bu2, bv2 As Double.
Dim normaMin As Double
Dim PP (100, 100) As puntouv.
Dim u0, u1, v0, v1 As Double
Dim s(10000000) As Double
Dim r As Integer
Dim mennorma1 As Double
Dim normamenor1(100000) As Double
Do
Npunto = puntos()
ordena(Vertice, 1, Npunto)
N = Triangulacion(Npunto)
nn = k1(0)
mm = k2(0)
SubBezier3D2(nn, mm, control, N)
b.x = 0
b.y = 0
b.z = 0
For k = 1 To N
r = 0
mennorma1 = 10000
Dim x0, x1, x2, y1, y0, y2 As Double
b.x = (Vertice3D(Triangulo3d(k).v0).x + Vertice3D(Triangulo3d(k).v1).x +
Vertice3D(Triangulo3d(k).v2).x) / 3
```

```
b.y = (Vertice3D(Triangulo3d(k).v0).y + Vertice3D(Triangulo3d(k).v1).y +
Vertice3D(Triangulo3d(k).v2).y) / 3
b.z = (Vertice3D (Triangulo3d(k).v0).z + Vertice3D(Triangulo3d(k).v1).z +
Vertice3D(Triangulo3d(k).v2).z) / 3
If Triangulo3d (k).uu0 = Triangulo3d (k).uu1 Then
u0 = Math.Abs (Triangulo3d (k).uu0)
u1 = Math.Abs (Triangulo3d (k).uu2)
Else
u0 = Math.Abs (Triangulo3d(k).uu0)
u1 = Math.Abs (Triangulo3d(k).uu1)
End If
If Triangulo3d (k).vv0 = Triangulo3d(k).vv1 Then
v0 = Math.Abs(Triangulo3d(k).vv0)
v1 = Math.Abs(Triangulo3d(k).vv2)
Else
v0 = Math.Abs (Triangulo3d (k).vv0)
v1 = Math.Abs(Triangulo3d(k).vv1)
End If
NewtonNoLineal (u0, v0, nn, mm, b, 0.12, 3, control)
Do
'partiicon que se llama
PP = Particion(maxParticion, maxParticion, u0, u1, v0, v1)
Norma(b, PP, nn, mm, control)
s(r) = Minimo(u0, u1, v0, v1, d)
```

```
If s(r) > mennorma1 Then
Exit Do
End If
'r = r + 1
If s(r) <= mennorma1 Then
mennorma1 = s(r)
normamenor1(k) = mennorma1
r = r + 1
End If
Loop While (Math.Abs(u1 - u0) > 0.5) And (v1 - v0) > 0.5
Next k
normaMin = MenNorma(normamenor1, k - 1)
Loop While (normaMin > 0.05)
For k = 1 To N
PProyectaV0.x = MATVis (1, 1) * Vertice3D (Triangulo3d (k).v0).x +
MATVis(1, 2) * Vertice3D(Triangulo3d(k).v0).y + MATVis(1, 3) *
Vertice3D(Triangulo3d(k).v0).z
PProyectaV1.x = MATVis (1, 1) * Vertice3D (Triangulo3d (k).v1).x +
MATVis (1, 2) * Vertice3D (Triangulo3d (k).v1).y + MATVis (1, 3) *
Vertice3D(Triangulo3d(k).v1).z
PProyectaV2.x = MATVis (1, 1) * Vertice3D(Triangulo3d(k).v2).x +
MATVis(1, 2) * Vertice3D(Triangulo3d(k).v2).y + MATVis(1, 3) *
Vertice3D(Triangulo3d(k).v2).z
```

PProyectaV0.y = MATVis (2, 1) * Vertice3D(Triangulo3d(k).v0).x + MATVis(2, 2) * Vertice3D(Triangulo3d(k).v0).y + MATVis(2, 3) * Vertice3D(Triangulo3d(k).v0).z

PProyectaV1.y = MATVis (2, 1) * Vertice3D(Triangulo3d(k).v1).x + MATVis(2, 2) * Vertice3D(Triangulo3d(k).v1).y + MATVis(2, 3) * Vertice3D(Triangulo3d(k).v1).z

PProyectaV2.y = MATVis (2, 1) * Vertice3D (Triangulo3d(k).v2).x + MATVis(2, 2) * Vertice3D(Triangulo3d(k).v2).y + MATVis(2, 3) * Vertice3D(Triangulo3d(k).v2).z

PProyectaV0.z = MATVis (3, 1) * Vertice3D (Triangulo3d(k).v0).x + MATVis(3, 2) * Vertice3D(Triangulo3d(k).v0).y + MATVis(3, 3) * Vertice3D(Triangulo3d(k).v0).z

PProyectaV1.z = MATVis (3, 1) * Vertice3D (Triangulo3d(k).v1).x + MATVis(3, 2) * Vertice3D(Triangulo3d(k).v1).y + MATVis(3, 3) * Vertice3D(Triangulo3d(k).v1).z

PProyectaV2.z = MATVis (3, 1) * Vertice3D (Triangulo3d(k).v2).x + MATVis(3, 2) * Vertice3D(Triangulo3d(k).v2).y + MATVis(3, 3) * Vertice3D(Triangulo3d(k).v2).z

PuntoPanV0.x = PProyectaV0.x

PuntoPanV1.x = PProyectaV1.x

PuntoPanV2.x = PProyectaV2.x

PuntoPanV0.y = PProyectaV0.y

PuntoPanV1.y = PProyectaV1.y

PuntoPanV2.y = PProyectaV2.y

```
PuntoPanV0.z = PProyectaV0.z
PuntoPanV1.z = PProyectaV1.z
PuntoPanV2.z = PProyectaV2.z
H.DrawLine (pen2, PuntoPanV0.x, PuntoPanV0.y, PuntoPanV1.x, PuntoPanV1.y)
H.DrawLine (pen2, PuntoPanV1.x, PuntoPanV1.y, PuntoPanV2.x, PuntoPanV2.y)
H.DrawLine (pen2, PuntoPanV0.x, PuntoPanV0.y, PuntoPanV2.x, PuntoPanV2.y)
Next
Return Vertice3D
End Function
Public Function SubBezier3D2 (ByVal nn As Integer, ByVal mm As Integer, ByVal control () as bezierDato, ByRef numtri As Integer)
Dim xx, yy As Single
Dim PProyectaV0 As Punto
Dim PProyectaV1 As Punto
Dim PProyectaV2 As Punto
Dim PuntoRealV0 As Punto
Dim PuntoRealV1 As Punto
Dim PuntoRealV2 As Punto
Dim PuntoPanV0 As Punto
Dim PuntoPanV1 As Punto
Dim PuntoPanV2 As Punto
```

```
Dim PuntoPan1V0 As Punto
Dim PuntoPan1V1 As Punto
Dim PuntoPan1V2 As Punto
Dim PPAn1V As Punto
Dim pi, pj, k As Integer
Dim i, j As Integer
Dim u1, v1 As Double
Dim bu0, bv0 As Double
Dim bu1, bv1 As Double.
Dim bu2, bv2 As Double.
Dim sup As Integer
Dim b As Punto
Dim s As Integer
Vertice3D(Triangulo3d(1).v0).x = 0
Vertice3D(Triangulo3d(1).v0).y = 0
Vertice3D(Triangulo3d(1).v0).z = 0
Vertice3D(Triangulo3d(1).v1).x = 0
Vertice3D(Triangulo3d(1).v1).y = 0
Vertice3D(Triangulo3d(1).v1).z = 0
Vertice3D(Triangulo3d(1).v2).x = 0
Vertice3D(Triangulo3d(1).v2).y = 0
Vertice3D(Triangulo3d(1).v2).z = 0
For k = 1 To numtri
PuntoRealV0.x = 0
```

```
PuntoRealV0.y = 0

PuntoRealV0.z = 0

PuntoRealV1.x = 0

PuntoRealV1.y = 0

PuntoRealV1.z = 0

PuntoRealV2.x = 0

PuntoRealV2.y = 0

PuntoRealV2.z = 0

For s = 0 To 0

'k1.Length(-2)

For j = 0 To k1(s)

bv0 = BAse(k1(s), j, Vertice(Triangulo(k).v0).y)

bv1 = BAse(k1(s), j, Vertice(Triangulo(k).v1).y)

bv2 = BAse(k1(s), j, Vertice(Triangulo(k).v2).y)

For i = 0 To k2(s)

bu0 = BAse(k2(s), i, Vertice(Triangulo(k).v0).x) * bv0

bu1 = BAse(k2(s), i, Vertice(Triangulo(k).v1).x) * bv1

bu2 = BAse(k2(s), i, Vertice(Triangulo(k).v2).x) * bv2

PuntoRealV0.x = PuntoRealV0.x + control(s).control(i, j).x * bu0

PuntoRealV0.y = PuntoRealV0.y + control(s).control (i, j).y * bu0

PuntoRealV0.z = PuntoRealV0.z + control(s).control (i, j).z * bu0

PuntoRealV1.x = PuntoRealV1.x + control(s).control (i, j).x * bu1

PuntoRealV1.y = PuntoRealV1.y + control(s).control (i, j).y * bu1

PuntoRealV1.z = PuntoRealV1.z + control(s).control (i, j).z * bu1
```

```
PuntoRealV2.x = PuntoRealV2.x + control(s).control (i, j).x * bu2
PuntoRealV2.y = PuntoRealV2.y + control(s).control(i, j).y * bu2
PuntoRealV2.z = PuntoRealV2.z + control(s).control (i, j).z * bu2
Next
Next
Triangulo3d (k).v0 = Triangulo (k).v0
Triangulo3d (k).v1 = Triangulo (k).v1
Triangulo3d(k).v2 = Triangulo(k).v2
Vertice3D(Triangulo3d(k).v0).x = PuntoRealV0.x
Vertice3D(Triangulo3d(k).v0).y = PuntoRealV0.y
Vertice3D(Triangulo3d(k).v0).z = PuntoRealV0.z
Vertice3D(Triangulo3d(k).v1).x = PuntoRealV1.x
Vertice3D (Triangulo3d (k).v1).y = PuntoRealV1.y
Vertice3D (Triangulo3d (k).v1).z = PuntoRealV1.z
Vertice3D (Triangulo3d (k).v2).x = PuntoRealV2.x
Vertice3D (Triangulo3d (k).v2).y = PuntoRealV2.y
Vertice3D (Triangulo3d(k).v2).z = PuntoRealV2.z
Triangulo3d (k).uu0 = Triangulo(k).uu0
Triangulo3d(k).uu1 = Triangulo(k).uu1
Triangulo3d(k).uu2 = Triangulo(k).uu2
Triangulo3d(k).vv0 = Triangulo(k).vv0
Triangulo3d(k).vv1 = Triangulo(k).vv1
Triangulo3d(k).vv2 = Triangulo(k).vv2
Next
```

```
Next k

End Function
```

Código en el botón para la Vista Isométrica

```
Private Sub Button1_Click_1(ByVal sender As System. Object, ByVal e As

System.EventArgs) Handles Button1.Click

'Dim PuntoReal(3, 3) As Punto

Dim I, J As Integer

Dim x1, x2, y1, y2 As Double

Dim g1 As Graphics = PictureBox1.CreateGraphics()

Dim pen8 As New Pen(Color.Blue, 0.01)

PictureBox1.Refresh()

PictureBox1.CreateGraphics.Clear(Color.Black)

MATVis(1, 1) = Math.Sqrt(2) / 2

MATVis(1, 2) = Math.Sqrt(2) / 2

MATVis(1, 3) = 0

MATVis (2, 1) = -1 / 2

MATVis (2, 2) = 1 / 2

MATVis(2, 3) = Math.Sqrt(2) / 2

MATVis (3, 1) = 1 / 2

MATVis (3, 2) = -1 / 2

MATVis(3, 3) = Math.Sqrt(2) / 2

x1 = Me.PictureBox1.Width / 12

y1 = 3 * (Me.PictureBox1.Height / 4)

g1.TranslateTransform(x1, y1)
```

```
Ejes(g1)
x1 = Me.PictureBox1.Width / 2
y1 = Me.PictureBox1.Height / 3
g1.TranslateTransform(x1, -y1)
g1.ScaleTransform(0.48, 0.2)
g1.TranslateTransform(-10, -260)
subbezier(sup, g1)
'subB_Spline(sup, TESIS.LECTURA.PControl(0).k1,
TESIS.LECTURA.PControl(0).k2, TESIS.LECTURA.PControl(0).m1,
TESIS.LECTURA.PControl(0).m2, g1)
'subB_Spline(sup, k1, k2, m1, m2, g1)
"subB_Spline(supB, 3, 4, 4, 3, g1)
"SuperficeB(3, 4, 3, 4, u, v, S(, ))
'SuperB(3, 4, 3, 4, supB, g1)
Dim s, ns As Integer
Dim d As Double
s = 0
For s = 0 To k1.Length - 2
For J = 0 To pasos
I = 0
Do
g1.DrawLine(pen8, sup(s).control(I, J).x, sup(s).control(I, J).y,
sup(s).control(I + 1, J).x, sup(s).control(I + 1, J).y)
I = I + 1
```

```
Loop While I < pasos
Next
For I = 0 To pasos
J = 0
While (J < pasos)
g1.DrawLine(pen8, sup(s).control(I, J).x, sup(s).control(I, J).y,
sup(s).control(I, J + 1).x, sup(s).control(I, J + 1).y)
J = J + 1
End While
Next
Next
End Sub
```

Código en el botón para la Vista en Perspectiva

```
Private Sub Button2_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
Dim I, J As Integer
Dim x1, x2, y1, y2 As Double
Dim g1 As Graphics = PictureBox1.CreateGraphics()
Dim pen8 As New Pen(Color.Blue, 1)
PictureBox1.Refresh()
PictureBox1.CreateGraphics.Clear(Color.Black)
MATVis (1, 1) = -Math.Sqrt (2) / 2
MATVis (1, 2) = Math.Sqrt (2) / 2
MATVis (1, 3) = 0
```

```
MATVis (2, 1) = -1 / 2
MATVis(2, 2) = -1 / 2
MATVis(2, 3) = Math.Sqrt(2) / 2
MATVis(3, 1) = 1 / 2
MATVis(3, 2) = 1 / 2
MATVis(3, 3) = Math.Sqrt(2) / 2
x1 = Me.PictureBox1.Width / 12
y1 = 3 * (Me.PictureBox1.Height / 4)
g1.TranslateTransform(x1, y1)
Ejes(g1)
x1 = Me.PictureBox1.Width / 2
y1 = Me.PictureBox1.Height / 3
g1.TranslateTransform(x1, -y1)
g1.ScaleTransform (0.4, 0.2)
g1.TranslateTransform (-200, -200)
subbezier(sup, g1)
Dim s As Integer
For s = 0 To k1.Length - 2
For J = 0 To pasos
I = 0
Do
g1.DrawLine(pen8, sup(s).control(I, J).x, sup(s).control(I, J).y,
sup(s).control(I + 1, J).x, sup(s).control(I + 1, J).y)
I = I + 1
```

```
Loop While I < pasos
Next
For I = 0 To pasos
J = 0
While (J < pasos)
g1.DrawLine(pen8, sup(s).control(I, J).x, sup(s).control(I, J).y,
sup(s).control(I, J + 1).x, sup(s).control(I, J + 1).y)
J = J + 1
End While
Next
Next
End Sub
```

Código en el botón para la Vista en el plano XY

```
Private Sub Button3_Click_1(ByVal sender As System. Object, ByVal e As
System.EventArgs) Handles Button3.Click
Dim I, J As Integer
Dim x1, x2, y1, y2 As Double
Dim g1 As Graphics = PictureBox1.CreateGraphics ()
Dim pen8 As New Pen(Color.Blue, 1)
PictureBox1.Refresh ()
PictureBox1.CreateGraphics.Clear (Color.Black)
MATVis (1, 1) = 1
MATVis (1, 2) = 0
```

```
MATVis (1, 3) = 0

MATVis (2, 1) = 0

MATVis (2, 2) = 1

MATVis (2, 3) = 0

MATVis (3, 1) = 0

MATVis(3, 2) = 0

MATVis(3, 3) = 1

x1 = Me.PictureBox1.Width / 12

y1 = 3 * (Me.PictureBox1.Height / 4)

g1.TranslateTransform(x1, y1)

Ejes(g1)

x1 = Me.PictureBox1.Width / 2

y1 = Me.PictureBox1.Height / 3

g1.TranslateTransform(x1, -y1)

g1.ScaleTransform (0.3, 0.3)

g1.TranslateTransform (10, 100)

'Ejes(g1)

subbezier(sup, g1)

Dim s As Integer

For s = 0 To k1.Length - 2

For J = 0 To pasos

I = 0

Do
```

```
g1.DrawLine(pen8, sup(s).control(I, J).x, sup(s).control(I, J).y,
sup(s).control(I + 1, J).x, sup(s).control(I + 1, J).y)
I = I + 1
Loop While I < pasos
Next
For I = 0 To pasos
J = 0
While (J < pasos)
g1.DrawLine(pen8, sup(s).control(I, J).x, sup(s).control(I, J).y,
sup(s).control(I, J + 1).x, sup(s).control(I, J + 1).y)
J = J + 1
End While
Next
Next
End Sub
```

Código en el botón para la Vista en el plano XZ

```
Private Sub Button4_Click (ByVal sender As System. Object, ByVal e As
System.EventArgs) Handles Button4.Click
Dim I, J As Integer
Dim x1, x2, y1, y2 As Double
Dim g1 As Graphics = PictureBox1.CreateGraphics()
Dim pen8 As New Pen(Color.Blue, 1)
PictureBox1.Refresh()
PictureBox1.CreateGraphics.Clear(Color.Black)
```

```
MATVis(1, 1) = 1
MATVis(1, 2) = 0
MATVis (1, 3) = 0
MATVis (2, 1) = 0
MATVis (2, 2) = 0
MATVis (2, 3) = 1
MATVis (3, 1) = 0
MATVis (3, 2) = -1
MATVis (3, 3) = 0
x1 = Me.PictureBox1.Width / 12
y1 = 3 * (Me.PictureBox1.Height / 4)
g1.TranslateTransform(x1, y1)
Ejes(g1)
x1 = Me.PictureBox1.Width / 2
y1 = Me.PictureBox1.Height / 3
g1.TranslateTransform(x1, -y1)
g1.ScaleTransform (0.7, 0.8)
g1.TranslateTransform (150, -500)
'Ejes(g1)
subbezier(sup, g1)
Dim s As Integer
For s = 0 To k1.Length - 2
For J = 0 To pasos
I = 0
```

```
Do
g1.DrawLine(pen8, sup(s).control(I, J).x, sup(s).control(I, J).y,
sup(s).control(I + 1, J).x, sup(s).control(I + 1, J).y)
I = I + 1
Loop While I < pasos
Next
For I = 0 To pasos
J = 0
While (J < pasos)
g1.DrawLine(pen8, sup(s).control(I, J).x, sup(s).control(I, J).y,
sup(s).control(I, J + 1).x, sup(s).control(I, J + 1).y)
J = J + 1
End While
Next
Next
End Sub
```

Código en el botón para la Vista en el plano YZ

```
Private Sub MenuItem11_Click(ByVal sender As System. Object, ByVal e
As System.EventArgs) Handles MenuItem11.Click
Dim g As Graphics = PictureBox1.CreateGraphics()
'Dim pen As New Pen(Color.Red, 2)
Dim x1, y1 As Double
Dim i, j As Integer
PictureBox1.Refresh ()
```

```
PictureBox1.CreateGraphics.Clear (Color.Black)
MATVis (1, 1) = 0
MATVis (1, 2) = 1
MATVis (1, 3) = 0
MATVis (2, 1) = 0
MATVis(2, 2) = 0
MATVis(2, 3) = 1
MATVis(3, 1) = 1
MATVis(3, 2) = 0
MATVis(3, 3) = 0
x1 = Me.PictureBox1.Width / 12
y1 = 3.6 * (Me.PictureBox1.Height / 4)
g.TranslateTransform(x1, y1)
Ejes(g)
x1 = Me.PictureBox1.Width / 2
y1 = Me.PictureBox1.Height / 3
g.TranslateTransform(x1, -y1)
g.ScaleTransform (14, 14)
g.TranslateTransform (-420, -470)
PMALLADONEWTON2 (g, PUNCONTROL)
End Sub
```

ANEXO 3

Información geométrica en la representación B-spline de la capota en formato IGES

A Continuación se muestra parte de la información geométrica en la representación B-spline de la capota en formato IGES

start record go heres 1

1h,,1h;,20hcatia - iges product,41hdfn 250.03 ind.tm15 mod. 18.1g 1

0.99,57hibm catia iges - catia solutions v4release 2.0 refresh 01,40hcatg2

ia solutions v4release 2.0 refresh 01,32,75,6,75,15,,1.0,2,2hmm,1000, g

1.0,13h991019.175845,0.0001,1000.0,,17hdassault systemes,9,0,13h991019.1g 4

75710; g 5

124 1 0 1 0 0 0 000000001D 1

4061413 0 0 0 0 0 000020301D 555

406 0 0 16007PROP6007 0D 556

1281414 0 2 0 0 0 000020001D 557

128 200 2 27 0SUR00035 178D 558

1263390 0 1 0 0 0 000030001D 1167

126 200 3 6 0COS00107 302D 1168

1263396 0 1 0 0 0 000030001D 1169

126 200 3 6 0COS00108 303D 1170

1263402 0 1 0 0 0 000030001D 1171

126 200 3 24 0COS00109 304D 1172

1263426 0 1 0 0 0 000030001D 1173

126 200 3 28 0COS00110 305D 1174

1023454 0 1 0 0 0 001010000D 1175

102 200 2 1 0FAC00019 300D 1176

126 200 2 2 0FAC00021 306D 1222

1263596 0 1 0 0 0 001010501D 1223

126 200 2 2 0FAC00021 307D 1224

1263598 0 1 0 0 0 001010501D 1225

126 200 2 2 0FAC00021 308D 1226

1263600 0 1 0 0 0 001010501D 1227

126 200 2 6 0FAC00021 309D 1228

1263606 0 1 0 0 0 001010501D 1229

126 200 2 17 0FAC00021 310D 1230

1443685 0 1 0 0 0 000020001D 1247

144 200 2 1 0FAC00021 304D 1248

4063686 0 0 0 0 0 000020301D 1249

406 0 0 16007PROP6007 0D 1250

1263687 0 2 0 0 0 001010001D 1251

126 200 5 5 0SUR00068 306D 1252

1263692 0 2 0 0 0 001010001D 1253

126 200 5 5 0SUR00068 307D 1254

1183697 0 2 0 0 0 001030001D 1255

118 200 5 1 1SUR00068 308D 1256

4063698 0 0 0 0 0 000020301D 1257

406 0 0 16007PROP60070D 1258

1263699 0 1 0 0 0 000030001D 1259

1263709 0 1 0 0 0 000030001D 1261

402 0 0 21 7 VDAFS 0D 1338

124,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0,0;1P 1

124,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0; 3P 2

126,10,5,0,0,1,0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,1.0, 1317P 3854

2.0,2.0,2.0,2.0,2.0,2.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1317P 3855

1.0,-585.2267308,693.239919,491.2519359,-584.5966539,1317P 3856

693.2682327,491.7042718,-583.9548981,693.299604,492.1407129, 1317P 3857

-583.3060175,693.3332019,492.5639656,-582.6485245,693.3690451,1317P 3858

492.9754818,-581.9852506,693.4066075,493.3770305,-578.390339,1317P 3859

693.6101939,495.5534045,-574.6256048,693.864283,497.4369768, 1317P 3860

-570.8673855,694.1190753,499.2970402,-567.140831,694.3606163,1317P 3861

501.2420319,-563.4097809,694.6054856,503.1695045,0.0,2.0,1317P 3862

0.5755177821,0.5755177821,0.5809979044,0,0;1317P 3863

126,5,5,1,0,1,0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,1.0,1.0, 1319P 3864

1.0,1.0,1.0,1.0,1.0,1.0,-563.4097809,694.6054856,503.1695045, 1319P 3865

-563.4402346,695.4051085,503.1543201,-563.4706883,696.2047315,1319P 3866

503.1391357,-563.501142,697.0043544,503.1239513,-563.5315957,1319P 3867

697.8039774,503.1087669,-563.5620494,698.6036003,503.0935825,1319P 868

0.0,1.0,0.01973360569,0.01973360569,0.999610509,0,0; 1319P 3869

126,10,5,0,0,1,0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,1.0, 1321P 3870

2.0,2.0,2.0,2.0,2.0,2.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1321P 3871

1.0,-563.5620494,698.6036003,503.0935825,-565.9608277, 1321P 3872

698.4286484,501.9035061,-568.3391094,698.2607291,500.6709064,1321P 3873

-570.7184129,698.0981045,499.4411142,-573.1025438,697.9357485,1321P 3874

498.2196073,-575.4736388,697.7668463,496.9742836,-577.4967862,1321P 3875

697.6227298,495.9117056,-579.5104428,697.4738473,494.8317878,1321P 3876

-581.5034591,697.315916,493.7148543,-583.4549945,697.1434341,1321P 3877

492.5216481,-585.3122585,696.9500026,491.196741,0.0,2.0, 1321P 3878

0.01973360569,0.01973360569,0.999610509,0,0; 1321P 3879

126,5,5,0,0,1,0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,1.0,1.0, 1323P 3880

1.0,1.0,1.0,1.0,1.0,1.0,-585.3122585,696.9500026,491.196741, 1323P 3881

-585.2951313,696.2079866,491.2077924,-585.278015,695.4659702,1323P 3882

491.2188376,-585.2609095,694.7239535,491.2298766,-585.2438148,1323P 3883

693.98364,491.2409094,-585.2267308,693.239919,491.2519359,0.0,1323P .3884

1.0,0.01973360569,0.01973360569,0.999610509,0,0;1323P 3885

102,4,1317,1319,1321,1323,0,0; 1325P 3886

142,0,1313,0,1325,2,0,0; 1327P3887

144,1313,1,0,1327,0,2,1331,101; 1329P3888

406,2,20,20,0,0; 1331P 3889

126,3,3,0,0,1,0,0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1333P 3890

-585.2267182,-693.2399223,491.25192,-580.2196638,-694.2700083,1333P 3891

491.26273,-574.5062123,-694.7016705,489.1439857,-569.1184673, 1333P 3892

-695.219212,490.1682984,0.0,1.0,0.01973360569,0.01973360569, 1333P 3893

0.999610509,0,0; 1333P 3894

124,1.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,1.0,0.0,1,5,0; 1335P 3895

402,298,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61, 1337P 3896

63,65,67,69,71,81,83,85,87,89,91,97,111,113,115,117,119,121,127, 1337P 3897

139,141,143,145,151,169,171,173,175,181,193,195,197,199,201,207, 1337P 3898

251,253,255,257,259,261,263,265,267,269,271,273,275,277,279,281, 1337P 3899

283,289,293,301,305,313,321,329,333,341,349,357,365,369,373,381, 1337P 3900

389,397,405,413,421,429,433,441,449,457,465,473,481,489,493,501, 1337P 3901

509,517,525,529,533,537,541,545,549,553,557,561,583,585,587,589, 1337P 3902

591,593,599,615,617,619,625,645,647,649,651,653,659,663,667,687, 1337P 3903

689,691,693,695,701,721,723,725,727,729,735,751,753,755,761,781, 1337P 3904

783,785,787,789,795,799,819,821,823,825,827,833,837,841,845,849, 1337P 3905

853,873,875,877,879,881,887,891,913,915,917,919,921,923,929,937,1337P 3906

945,949,953,955,957,959,961,963,965,967,969,971,973,975,977,979,1337P 3907

981,983,985,987,989,991,993,995,997,999,1001,1003,1005,1007, 1337P 3908

1009,1011,1013,1015,1017,1019,1021,1023,1025,1027,1029,1031,1337P 3909

1033,1035,1037,1039,1041,1043,1045,1047,1049,1051,1053,1055, 1337P 3910

1057,1059,1061,1063,1065,1067,1069,1071,1087,1089,1091,1093, 1337P 3911

1099,1111,1113,1115,1117,1123,1135,1137,1139,1141,1143,1145, 1337P 3912

1151,1163,1165,1167,1169,1171,1173,1179,1191,1193,1195,1197, 1337P 3913

1199,1205,1209,1213,1233,1235,1237,1239,1241,1247,1259,1261, 1337P 3914

1263,1265,1271,1291,1293,1295,1297,1299,1305,1317,1319,1321, 1337P 3915

1323,1329,1333,0,0; 1337P 3916

S 1G 5D 1338P 3916 T 1