UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

ESTUDO DE MÉTODOS E ALGORITMOS DE *SPLINES BEZIER, CASTELJAU E B-SPLINE*

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

JEVERSON ZOZ

BLUMENAU, AGOSTO/1999

1999/1-28

ESTUDO DE MÉTODOS E ALGORÍTMOS DE *SPLINES*, *BEZIER, CASTELJAU E B-SPLINE*

JEVERSON ZOZ

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Dalton Solano dos Reis — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Dalton Solano dos Reis

_____

Prof. Antônio Carlos Tavares

_____

Prof. Maurício Capobianco Lopes

AGRADECIMENTOS

Ao meu orientador, prof. Dalton Solano dos Reis, pela atenção dispensada a este trabalho.

A minha noiva Joseane, pela ajuda, apoio e compreensão, durante todos os meus trabalhos realizados.

Aos meus pais, Raimundo e Maria Edite, pela compreensão e apoio durante estes anos em que me dediquei à faculdade.

Ao meu Colega Cristian em especial, pelo apoio e suporte de software nas horas em que mais precisei.

Aos meus colegas da FURB, pela amizade e colaboração.

A todos aqueles que, direta ou indiretamente, contribuíram para a realização deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

RESUMO

Este trabalho apresenta um estudo realizado sobre algoritmos de *splines*, apresentando conceitos, fundamentos e o processo de funcionamento dos algoritmos de *Bezier*, *Casteljau* e *B-Splines*. A revisão bibliográfica apresenta o desenvolvimento sobre a geração de curvas destas *splines* e suas técnicas de uma maneira didática. Este documento, apresenta também, o desenvolvimento de um protótipo, que permite a geração destas curvas de *splines* através da implementação dos algoritmos. Esta implementação possibilitou a realização prática, onde podem ser analisados alguns aspectos de grande importância, como, por exemplo, a visualização da curva gerada, o tempo gasto em cada processamento da curva, a memória, entre outros, oportunizando-se a comparação entre os seus resultados. Foi utilizado como plataforma de desenvolvimento e execução, o Windows 98, utilizando-se o Borland Delphi 3.0 como ferramenta de desenvolvimento e implementação do protótipo.

ABSTRACT

This paper introduces a study on Splines Algorithms presenting concepts, fundametals and the working process of Bezier, Casteljau and B-Splines algorithms. The bibliographic review presents the development for the generation of these splines curves and its techniques in a didactic manner. This document also presents the development of a prototype which allows the generation of these splines curves through the algorithms implementation. This implementation enabled a pratical realization, where can be analyzed some very important aspects, as, for instance, visualizing the gererated curve, te time spent on each curve processing and the memory, among other things, allowing to compare its results. As a platform for its development and execution the Windows 98 was used, utilizing the Borland Delphi 3.0 as a tool for the prototype's development and implementation.

1 INTRODUÇÃO

Splines são formas de representação de modelagem geométricas, servindo como definição visual. Na maior parte das implementações de aplicativos que usam *splines*, pode-se optar por algoritmos que nem sempre são indicados, ou seja, entre os algoritmos existentes poderia haver algum deles com um tempo de execução menor e melhor, ou que consumisse menos memória, etc. Talvez isto aconteça devido a falta de informação ou estudo sobre esses algoritmos. Sendo assim será feito um estudo usando três algoritmos normalmente mais utilizados, o *Bezier, Casteljau* e *B-Splines* [OLI92]. Esses algoritmos foram escolhidos sem o uso de qualquer critério de escolha (aleatóriamente).

Sabe-se da grande diversidade de definições de algoritmos as quais permitem utilizar as mais variadas técnicas de Computação Gráfica (CG). Entre estas definições, ainda, muitas vezes encontram-se várias soluções para o mesmo problema, os quais aparentemente apresentam os mesmos resultados. Conceitualmente, ao deparar-se com um "problema" tem-se várias preocupações: o que é realmente o problema, a definição do seu escopo, qual a melhor forma de resolver, etc. Em Computação Gráfica, como em qualquer área, necessitam-se ter critérios para avaliar se as técnicas utilizadas permitem obter resultados dentro de uma faixa de valores aceitável.

Na tentativa de obter uma solução deve-se também considerar o quanto de interação terá o usuário. Pois muitas vezes, mesmo tratando-se de um processo o qual não apresente grandes dificuldades na sua construção manual, dependendo da quantidade de informações, este procedimento pode-se tornar cansativo e propenso a erros (devido a aspectos ergonômicos, fadiga, etc). Já ao utilizar procedimentos computacionais, tem-se a vantagem de se tratar de um processo automatizado e menos propenso a erros, tendo-se somente o custo da implementação das rotinas necessárias a este procedimento. Desta forma, poder-se-ia questionar o quanto estes procedimentos computacionais são eficientes e próximos da representação real [REI 97].

Serão estudados estes três algoritmos, observando-se seus conceitos, aplicações, e comportamento, além de uma implementação que permitirá a análise destes algoritmos. Esta análise poderá verificar a eficiência e a eficácia de cada algoritmo, podendo-se compará-los em diversas ocasiões.

As comparações incluem [REI 97]:

a) **ordem de Complexidade:** é um fator que está relacionado diretamente com a velocidade de execução do algoritmo;
b) **necessidade de Memória**: nesta avaliação é verificado o espaço de memória exigido pela execução de um determinado procedimento, onde o tamanho e a quantidade das variáveis são os fatores a serem considerados;
c) **fator de Precisão**: considera-se quão mais próximos os valores de entrada podem demonstrar os resultados os quais estes valores propõem-se a representar;
d) **aspecto Visual**: este fator é diretamente visível e é relacionado com a precisão da curva, ou seja, é analisada a suavidade da curva, que deve possuir um acabamento minucioso;
e) **facilidade de Implementação**: este fator é julgado altamente subjetivo, podendo depender da familiaridade do implementador com algumas estruturas exigidas pelo método, da filosofia de implementação e do ambiente físico disponível para o mesmo.

A comparação será analisada graficamente, concluindo um estudo sobre a capacidade de cada algoritmo, mostrando e comprovando a importância de uma boa escolha por parte do usuário, permitindo ao protótipo adequar-se à necessidade da aplicação. Este estudo mostrará a necessidade de uma boa escolha do algoritmo, na hora da aplicação deste.

1.1 MOTIVAÇÃO

Ao passar dos tempos, vem exigindo-se cada vez mais na qualidade dos softwares, ou seja, os softwares devem ser mais ágeis e rápidos, ocupando menos memória. Este estudo é realizado, a fim de proporcionar uma melhor avaliação dos algoritmos propostos verificando sua adequação às exigências do mercado, pois através de uma análise comparativa, é que se pode concluir qual o melhor algoritmo a ser utilizado para uma determinada situação, adequando-o para a exigência do software.

1.2 OBJETIVOS

Os principais objetivos principais deste trabalho são:

a) desenvolver um protótipo de software que possibilite a comparação destes algoritmos, podendo-se chegar a conclusões concretas;
b) comprovar as diferenças entre os três algoritmos propostos;
c) avaliar os três algoritmos de *splines* (*Bezier, Casteljau* e *B-Splines*), em nível prático;

E como objetivos secundários tem-se:

a) realizar um levantamento bibliográfico e estudo teórico sobre os algoritmos de *splines Bezier, Casteljau* e *B-Splines*;
b) possibilitar uma escolha mais direcionada do algoritmo a ser utilizado para determinada ocasião e
c) mostrar a importância de se fazer esta escolha.

1.3 RELEVÂNCIA

Este trabalho de conclusão de curso apresenta os seguintes aspectos tecnológicos relevantes:

a) permite o estudo de diferentes algoritmos de *splines*;
b) possibilita a demonstração do funcionamento de diferentes algoritmos de *splines*;
c) permite a identificação e avaliação visual do algoritmo mais indicado em diferentes ocasiões e
d) envolve técnicas de grande uso na Computação Gráfica.

1.4 RESULTADOS

O resultado deste trabalho é a apresentação e avaliação comparativa de um estudo sobre três diferentes algoritmos de *splines*, sendo *Bezier, Casteljau* e *B-Splines*, bem como, a especificação e implementação de um protótipo de software comparativo, que permite:

a) a demonstração de diferentes algoritmos de *splines: Bezier, Casteljau* e *B-Splines*;

b) a identificação do algoritmo mais indicado em diferentes casos através de testes realizados, referentes a: ordem de complexidade, necessidade de memória, fator de precisão, aspecto visual e facilidade de implementação.

1.5 ORGANIZAÇÃO DO TEXTO

O capítulo 1 apresenta a introdução, motivação, os objetivos, a relevância, resultados e a organização do trabalho, ou seja a estrutura geral do trabalho.

O capítulo 2 é dedicado a Conceitos Básicos de *Splines*, descrevendo conceitos, tipos e idéias gerais de *Splines*.

No capítulo 3 as *Splines* são detalhadas, ou seja, é o estudo das três *Splines* propostas, explicando-se um pouco mais sobre cada uma delas, incluindo seus algoritmos e características.

No capítulo 4 é mostrado como foi desenvolvido o protótipo do software, mencionando-se uma abordagem geral, as especificações do protótipo, o funcionamento e a implementação.

No capítulo 5, são apresentados os testes comparativos, onde são colocados em forma de tabelas e gráficos, os resultados obtidos e comparados entre os três algoritmos propostos.

E por fim, o capítulo 6, onde são expostas as conclusões, analisando as dificuldades, os resultados do trabalho, avaliações e extensões.

2 CONCEITOS BÁSICOS DE *SPLINES*

Este capítulo é dedicado a Algoritmos de *splines* em geral, onde é apresentada uma introdução de curvas e superfícies, os conceitos e principais características de cada um dos algoritmos.

2.1 CURVAS E SUPERFÍCIES

A representação de formas complexas bidimensionais ou tridimensionais, utilizando-se simplesmente de elementos lineares como segmentos de reta ou poliedros de faces planas, pode exigir uma quantidade enorme desses elementos para se atingir um grau de realismo satisfatório [PRE88].

Imagine então um processo interativo em que um projetista vai, sucessivamente modificando a forma de uma peça, introduzindo novos elementos, substituindo outros e avaliando o resultado dessas modificações numa tela gráfica. Ele procura, através dessa seqüência de ajustes obter um padrão final desejado. Numa situação como essa, o grande número de elementos que possivelmente têm de ser introduzidos para especificar as mudanças necessárias e que, em seguida, precisam ser processados pelo sistema gráfico, pode tornar inadequada a representação de formas via exclusivamente elementos lineares. Formas de se especificar uma figura de maneira mais compacta têm então que ser usadas [PER89].

2.1.1 OBJETIVOS DO USO DE CURVAS E SUPERFÍCIES

O projetista que obtém, ao final do processo interativo descrito acima, uma forma compatível com alguma idealização sua faz o que se chama de síntese de formas. Em contraposição, pode-se efetuar um processo de análise onde, dado um objeto, deve-se descrever sua forma e dimensões por intermédio de uma representação, a qual deve ser a mais compacta possível que permita um grau de precisão desejado e também possibilite que avaliações de grandezas, tais como, por exemplo, o volume do objeto, sejam obtidas com facilidade [PER89].

Dentro dessa linha está o problema de aproximar um corpo tridimensional pela união de cubos ou cilindros visando obter uma estimação de seu volume. Outro exemplo é o

problema de se representar uma curva por uma poligonal mantida dentro de uma faixa de tolerância como apresentado na Figura 1 a seguir.

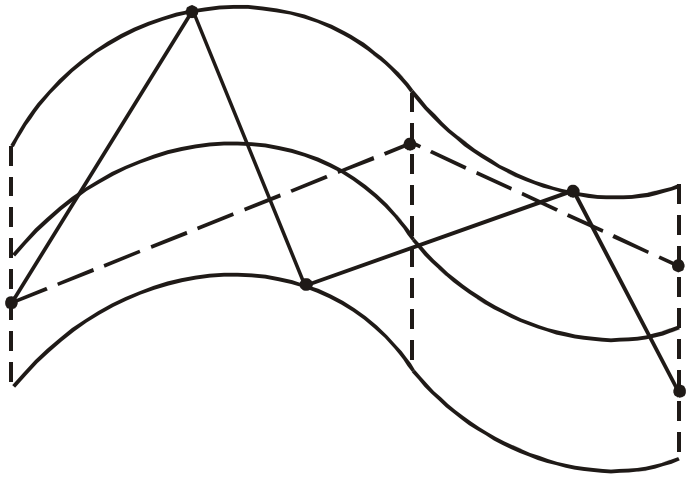


Figura 1 : Representação Econômica

Atendo-se à síntese de formas, em especial com relação a esse objetivo, há todo um conjunto de propriedades que devem ser satisfeitas por uma boa representação de figuras de duas ou três dimensões. Essas propriedades são as seguintes [PER 89]:

a) independência do sistema de coordenadas: a imagem de uma representação não deve mudar quando se muda o sistema de referência;
b) possibilidade de se efetuar um controle local da curva ou superfície: no problema de síntese de formas é essencial que o projetista possa fazer alterações restritas a trechos pequenos de uma curva ou superfície. Isso lhe possibilita, por exemplo, fazer ajustes finos depois de ter considerado satisfatório o aspecto geral da curva ou superfície;
c) versatilidade: o conjunto de formas de curvas e superfícies passíveis de serem geradas deve ser amplo. Mais especificamente, o que deve ser requerido é que uma quantidade satisfatória de formas, incluindo aquelas mais comumente empregadas, possam ser obtidas introduzindo-se um número não muito grande de parâmetros. Com isso, quer-se dizer que se um sistema gráfico é capaz de gerar apenas poligonais, muito embora se possa aproximar por poligonais qualquer curva contínua do plano, não significa que esse sistema seja exatamente versátil. Isso porque o número de pontos que devem ser dados para aproximar por uma poligonal, uma curva elementar como um arco de círculo, pode ser muito grande, dependendo da precisão requerida;

d) continuidade de curva gerada e existência de suas derivadas até um determinado grau: uma curva ou superfície contínua mais complexa, em geral, é modelada juntando-se curvas ou superfícies contínuas, de formulação simples, por extremidades comuns. Na maior parte das aplicações não se necessita mais do que obter curvas e superfícies duas vezes continuamente diferenciáveis. Isso assegura, no caso de representação de curvas, que a reta tangente e a curvatura estejam bem definidas em todo ponto e que elas não variem bruscamente;
e) facilidade de introdução de parâmetros de controle: voltando-se ao processo em que um projetista realiza a síntese de uma curva que suponha-se ser plana, com o auxílio de um sistema gráfico interativo, como os parâmetros de uma representação são todos pontos do plano, eles podem ser introduzidos usando um dispositivo localizador cujo eco pode ser, por exemplo, a exibição da poligonal $P_1, P_2,...,P_n$.

Caso o ajuste da curva possua as propriedades acima citadas, pode-se atingir uma boa representação da superfície, a qual se quer modelar.

2.2 CONCEITO GERAL DE SPLINES

Splines se utilizam de curvas paramétricas que constituem o meio mais utilizado para representação de objetos em modelagem geométrica [MOR 85]. Superfícies bicúbicas são as formas mais freqüentemente utilizadas, devido ao fato de que superfícies de menor grau apresentam pouca flexibilidade sobre o controle de suas formas, enquanto que superfícies de maior grau podem introduzir torções indesejáveis, além de requisitarem maior esforço computacional [FOL 90].

2.3 TIPOS DE SPLINES

Na literatura, hoje existem diferentes tipos de algoritmos de *splines*, entre eles podem ser citados os mais usados como: *Bezier*, *Casteljau*, *B-Spline*, *Hermite*, *Spline, Beta-Spline e Catmull* [THA 85], [BAR 79].

Os tipos propostos ao estudo e avaliação, constituem-se de *Bezier*, *Casteljau* e *B-Spline*.

3 DETALHAMENTO DE SPLINES

Os algoritmos de *Splines*, são, na maioria dos casos, complexos e com uma particularidade em cada um dos tipos. As equações polinomiais mudam completamente de um algoritmo para o outro, estabelecendo diferenças entre os métodos de desenvolvimento das curvas [PRE 88].

Neste Capítulo será apresentado de forma detalhada, as principais características e fundamentos dos algoritmos propostos (*Bezier*, *Casteljau* e *B-Spline*).

3.1 BEZIER

Esta palavra teve origem através do nome *Berntein Bézier*, o qual desenvolveu este algoritmo. *Bezier* é no momento um dos algoritmos mais utilizados na área de *splines*. Caracteriza-se através de pesos estipulados dependendo do número de pontos a serem usados na curva [BAR 87].

3.1.1 CONCEITOS E FUNDAMENTOS

Esse algoritmo foi proposto por *B. Bézier* da firma francesa *Régie Renault* que o empregou em seu sistema UNISURF aplicado desde 1972. As funções peso F_i empregadas por *Bézier* são exatamente os termos da fatoração binominal de $(a + b)^n$ quando a = t e b = 1-t, também conhecidas por *funções de Berntein* [PER 89]. Explicitamente.

$$F_i : [0,1] \rightarrow R$$

$$F_i(t) = C_n^i t^i (1-t)^{n-i}; i = o,...,n$$

onde

C_n^i

é o número de combinações de *n* elementos *i* a *i*, expresso por *n!/i!(n – 1)!*. Utilizando essas F_i na equação acima obtem-se as curvas de *Bézier* de grau *n* que representa-se por B_n.

Verificou-se imediatamente que como

$$\sum F_i(t) = \sum C_n^i t^i (1-t)^{n-1} = (t + (1-t))^n = 1$$

as curvas de *Bézier* são indiferentes a mudanças no sistema de coordenadas utilizado. Além disso, elas caem dentro do fecho convexo de seus pontos de controle já que as F_i são também não negativas, o que as impede de oscilar demais [PER 89].

Com respeito aos demais atributos, entretanto, algumas considerações devem ser efetuadas. Observemos inicialmente que

$$B_n(0) = P_o$$

e

$$B_n(1) = p_{n'}$$

ou seja, a curva de *Bézier* passa pelos pontos de controle[1] extremos. Além disso,

$$B_n^{'}(0) = n(p_1 - p_0)$$

e por simetria,

$$B_n^{'}(1) = n(p_{n-1} - p_n).$$

Considere, então, duas curvas de *Bézier* tais que o último ponto da primeira e o primeiro da segunda sejam coincidentes. Conforme viu-se acima, verificou-se que essas curvas podem ser emendadas de forma que a curva união seja suave, bastando que o penúltimo ponto da primeira, o ponto comum às duas e o segundo ponto da outra sejam colineares. Nessas condições tem-se que a derivada em 1 da primeira curva e a derivada em 0 da segunda possuem a mesma direção. Isso é o bastante para garantir que no ponto da emenda haverá uma única direção tangente à curva união. Não há, portanto, necessidade de fazer com que essas derivadas se igualem [PER 89].

Assim, dada uma curva de *Bézier*, emendar nela uma outra, de modo que a curva resultante continue suave, é uma simples questão de introduzir, nas posições devidas, pontos de controle que obriguem a condição de colinearidade acima a ser cumprida. Deve ser lembrado, entretanto, que, ao fazer isso, modificou-se a expressão de ambas as curvas, mesmo que elas já tenham, originalmente, uma extremidade comum e que os pontos acrescentados estejam bem próximos dessa extremidade. Isso se dá porque o controle local de uma curva de

---

[1] Pontos de Controle: Pontos que controlam e influenciam a geração da curva de spline.

Bézier não é completo, uma vez que todas as F_i são não nulas no intervalo [0,1]. Deve-se observar, porém, que o valor máximo de cada F_i se dá para $t = i/n$ e em torno desse ponto, no intervalo $[i/(n + 1), (i + 1)/(n + 1)]$, o valor de F_i predomina sobre os demais. Dessa forma o intervalo [0,1] fica particionado em $(n + 1)$ intervalos iguais, em cada um deles preponderando a influência de um p_i diferente. Isso é mostrado na Figura 2 para o caso em que n = 3 [PER 89].

Uma alternativa para melhorar o controle local é ao invés de trabalhar com uma única curva de *Bézier*, utilizar várias delas de grau menor emendadas em extremidades comuns. Observa-se, então, que se for exigido apenas que a curva composta seja suave, pode-se fazer essas emendas de modo simples, apesar da ressalva que fizeram quanto a isso. Se, entretanto, for preciso garantir que essa curva possua um grau maior de diferenciabilidade, fazer as emendas da maneira devida pode ficar mais confuso [PER 89].

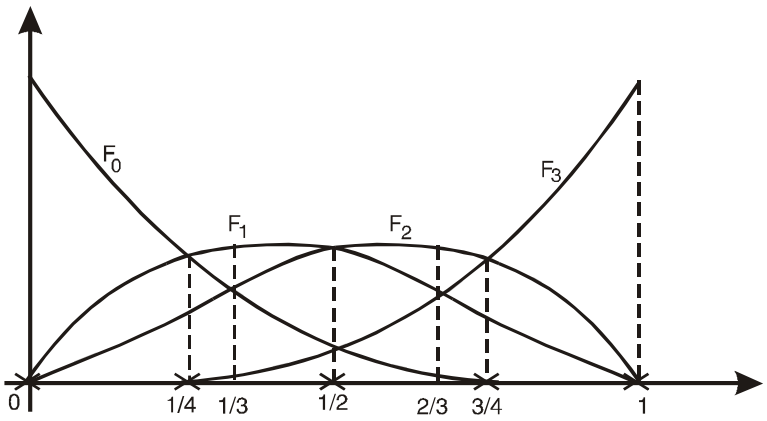


Figura 2: Curvas de *Bézier* para n=3

Outro inconveniente das curvas de *Bézier* é que o grau e a própria definição das $F_i^{'}$ dependem do número de pontos de controle. Isso significa que o projetista, ao acrescentar mais um ponto para, digamos, tentar acertar a forma de um determinado trecho da curva torna mais complicada a expressão de toda a curva, mesmo onde a influência desse ponto é pequena.

Finalmente, deve-se observar que a formulação de *Bezier* para curvas se estende com facilidade para superfícies.

3.1.2 ALGORITMO

Curvas *Bezier* paramétricas $C_B(v)$ (Figura 3) apresentam a seguinte forma matricial [MOR 85]:

$$\mathbf{C_B(v) = [\ (1\text{-}v)^3 \quad 3v(1\text{-}v)^2 \quad 3v^2(1\text{-}v) \quad v^3\ ] \quad [\ P_1 \ P_2 \ P_3 \ P_4\ ]^T}$$

Com $v \in [0,1]$ e P_i representando o i-ésimo coeficiente geométrico (ponto de controle) da curva $C_B(v)$.

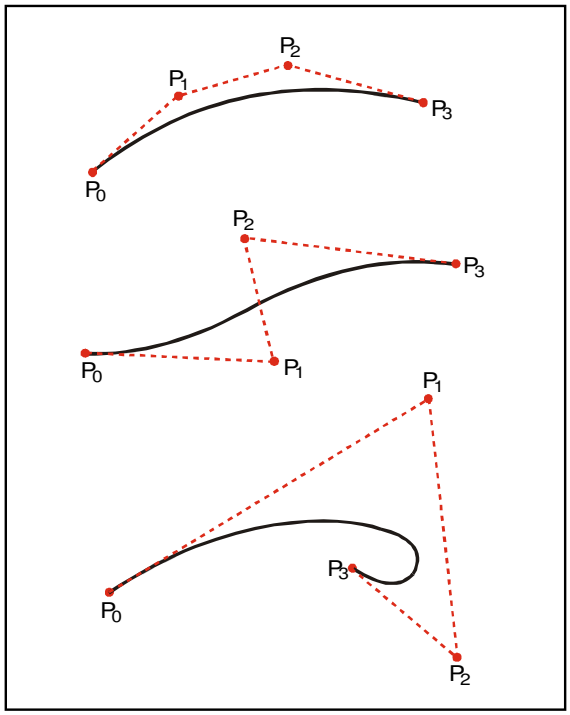


Figura 3: Curvas de Bezier

Por sua vez, uma superfície *Bézier* (Figura 3) é expressa por [MOR 85]:

$$SB(v,u) = \begin{bmatrix}(1-v)^3 & 3v(1-v)^2 & 3v^2(1-v) & v^3\end{bmatrix} P \begin{bmatrix}(1-u)^3 \\ 3u(1-u)^2 \\ 3u^2(1-u) \\ u^3\end{bmatrix}$$

Com v e $u \in [0,1]$ e

$$P = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{bmatrix}$$

sendo P a matriz dos coeficientes geométricos da superfície.

Pode-se observar o exemplo abaixo (Figura 4).

$$P(v) = (1-v)^3 P_0 + 3v(1-v)^2 P_1 + 3v^2(1-v) P_2 + v^3 P_3$$

Exemplo:
Nº de Segmentos: 5
v = 1 / 5
Intervalo entre pesos = 0,20
Pesos: 0,0 / 0,2 / 0,4 / 0,6 / 0,8 / 1,0

Pontos de Controle:
P0= (10,10)
P1= (10,100)
P2= (100,100)
P3= (100,10)

PONTOS DE CONTROLE / V	x,y (10,10) P0	x,y (10,100) P1	x,y (100,100) P2	x,y (100,10) P3	PONTOS DA CURVA X	PONTOS DA CURVA Y
0,0	1	0	0	0	10	10
0,2	0,512	0,384	0,096	0,008	19,36	53,2
0,4	0,216	0,432	0,288	0,064	41,68	74,8
0,6	0,064	0,288	0,432	0,216	68,32	74,8
0,8	0,008	0,096	0,384	0,512	90,64	53,2
1,0	0	0	0	1	100	10

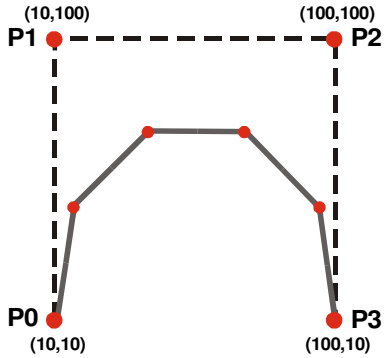


Figura 4: Exemplo da geração de uma curva pelo algoritmo de *Bezier*.

3.2 CASTELJAU

Casteljau, é um algoritmo de implementação recursiva, e uma das suas características principais é que a geração de curva de *splines* é baseada em níveis, o qual determina o número de pontos.

3.2.1 CONCEITOS E FUNDAMENTOS

O algoritmo de *Casteljau* é uma construção geométrica, que se baseia em interpolações lineares (bi-lineares) dos coeficientes que definem as curvas (superfícies) [FAR 90 *apud* OLI 92]. Para o caso de curvas, o algoritmo consiste em construir recursivamente, através de interpolações lineares, um polinômio de mesmo grau que a curva em questão [OLI 92].

O desenvolvimento deste método não é baseado em número de pontos na curva e sim no número de níveis que possui a curva, ou seja, os níveis são etapas seguidas pelo algoritmo, onde quanto mais etapas executadas, mais pontos terá a curva. O processo evolutivo do número de pontos baseado no nível, acontece na proporção de $\mathbf{2^{Número\ da\ Etapa} + 1}$, ou seja:

1 etapa = 3 pontos, 2 etapas = 5 pontos, 3 etapas = 9 pontos, 4 etapas = 17 pontos, 5 etapas = 33, 6 etapas = 65 pontos e assim progressivamente.

Sendo assim, pode-se considerar que no método de *Casteljau*, a evolução da curva acontece por número de níveis, não podendo, por exemplo, traçar uma curva com 12 pontos, mas sim com 9 ou 17 pontos correspondente a uma curva com nível 3 ou 4, respectivamente.

3.2.2 ALGORITMO

Sejam P_1, P_2, P_3 e P_4 os coeficientes geométricos de uma curva *Bézier*. Tem-se:

$$P_{11}(v) = (1 - v)P_1 + vP_2$$
$$P_{12}(v) = (1 - v)P_2 + vP_3$$
$$P_{13}(v) = (1 - v)P_3 + vP_4$$
$$P_{21}(v) = (1 - v)P_{11} + vP_{12}$$
$$P_{22}(v) = (1 - v)P_{12} + vP_{13}$$
$$P_{31}(v) = (1 - v)P_{21} + vP_{22}$$

Substituindo P_{11} , P_{12} e P_{13} em P_{21} e P_{22} e estes por sua vez em P_{31}, obtém-se a expressão de uma curva *Bézier*:

$$P_{31}(v) = (1 - v)^3P_1 + 3v(1 - v)^2P_2 + 3v^2(1 - v)P_3 + v^3P_4$$

O algoritmo preserva os coeficientes entermediários $P_{ki}(v)$, os quais podem convenientemente serem rescritos em forma triangular – o esquema de *Casteljau*. (Figura 5), onde *a* corresponde aos coeficientes originais da curva; *b*, aos coeficientes que definem o segmento compreendido entre $P_1(v)$ e P_4(v) (Figura 6) [OLI 92].

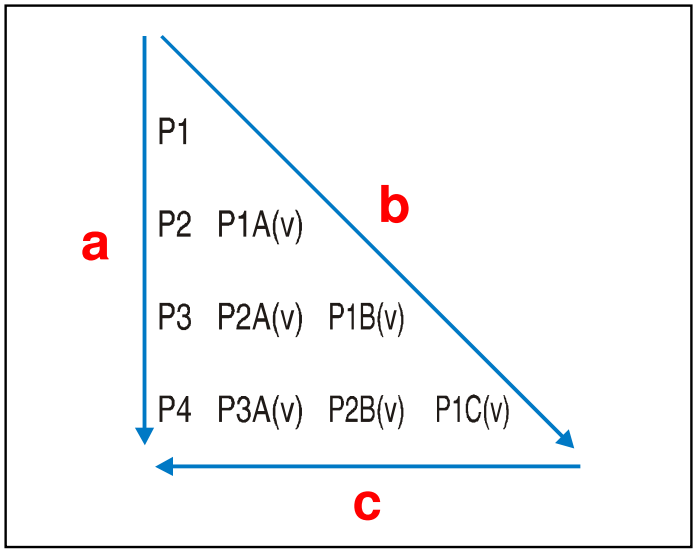


Figura 5: Esquema de *Casteljau*

Graficamente o processo do algoritmo de *Casteljau* se desenvolve da seguinte maneira

Baseando-se em 4 pontos distintos (P1,P2,P3 e P4), são calculados pontos médio P1A, P2A e P3. Após isso é calculado um ponto médio entre P1A e P2A chamado de P1B e um ponto médio entre P2A e P3A chamado de P2B. Finalmente o ponto esperado onde passará a curva é encontrado calculando o ponto médio entre P1B e P2B. Isso para encontrar um ponto, no nível 1 que possui 3 pontos, onde 2 deles são respectivamente a origem e o destino (P1 e P4).

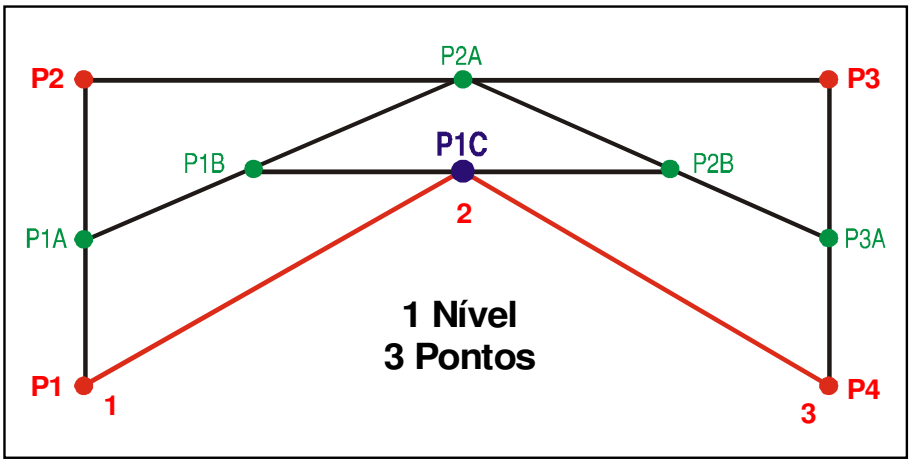


Figura 6: Algoritmo de *Casteljau* nível = 1

Para encontrar os pontos com nível = 2, procede-se da mesma maneira até encontrar o nível 1, sendo que após o nível = 1 encontrar-se-ão mais quatro pontos à esquerda e mais quatro pontos a direita (em vermelho na figura 7). Procede-se da mesma forma anterior para encontrar um novo ponto à esquerda e um novo ponto a direita (Figura 7).

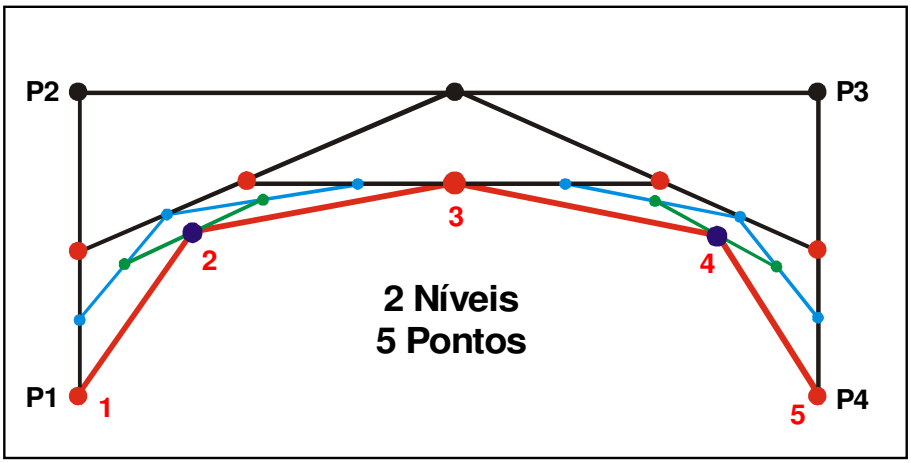


Figura 7: Algoritmo de *Casteljau* nível = 2

Os próximos pontos dos decorrentes níveis, são obtidos progressivamente seguindo o mesmo método acima, com *n* níveis que estabelece *n* pontos, gerando uma melhor qualidade na definição da curva com o aumento de níveis (Figura 8) [OLI 92].

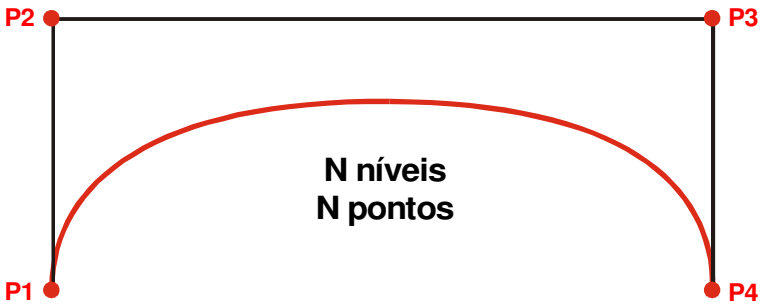


Figura 8: Algoritmo de *Casteljau* nível = n

3.3 B-SPLINE

Caracteriza-se por ser genérico, onde não importa o número de pontos de controle. Tem-se 2 tipos, um periódico e outro não periódico [PER 89].

3.3.1 CONCEITOS E FUNDAMENTOS

Uma definição clássica de *spline* é a seguinte: dada uma seqüência de números reais não repetidos, chamados nós, $t_0 < t_1 <...< t_{n+1}$, uma *spline* de grau m é uma função de domínio real com as seguintes propriedades [PER 89]:

a) Nos intervalos(t) abertos

$$(-\infty, t_0), (t_0, t_1), \ldots (t_{n-1}, t_n), (t_n, \infty)$$

a função é dada por um polinômio de grau *m* ou menor;

b) As derivadas da função até ordem $(m - 1)$ existem e são contínuas, inclusive nos pontos t_i.

Certas famílias de *Splines* recebem denominações especiais. Entre elas a das *splines* naturais, que são aquelas de grau ímpar $(2k - 1)$ que, nos intervalos ilimitados

$$(-\infty, t_0)$$

e

$$(t_n, \infty)$$

se reduzem a polinômios de grau menor que k. Uma *spline* natural de grau $2k-1$ dá o menor valor possível para a integral, entre todas as funções k vezes diferenciáveis, que com ela coincidem nos pontos f_i .

$$\int_a^b \left(f^k(t)\right)^2 \, dt, \, a \le t_0, b > t_n$$

É, no sentido de minimizar essa integral para k = 2, que a *spline* natural cúbica que coincide em seus nós (t_i) com uma função f, é dita ser a função mais suave que interpola f em (t_i). Sob esse ponto de vista, elas seriam bastante adequadas para o papel das F_i*'s* do nosso modelo de síntese de curvas. Entretanto, para atender outras condições igualmente desejáveis, essa otimalidade pode ter que ser "sacrificada" [PER 89].

Uma generalização do conceito de *spline* dado acima permite que os t_i sejam repetidos. Relaxa-se então a condição que obriga a função a ser $(m - 1)$ vezes continuamente diferenciável em t_i, para apenas $(m - r_i)$ vezes, sendo r_i o número de aparições de t_i na seqüência de nós. As funções que se enquadram nessa conceituação mais geral são chamadas *splines* com nós múltiplos. *Splines* com nós múltiplos são menos diferenciáveis mas seu uso pode permitir que sejam atendidas condições tais como fazer com que

$$\sum F_i(t) = 1, \, \forall t$$

ou obrigar a curva gerada a passar pelos pontos de controle extremos p_o e p_n. É para atender a esses requisitos que, por exemplo, um dos tipos de *spline* mais usado na modelagem de curvas, as chamadas *B-splines* não periódicas, possuem nós repetidos nos extremos da seqüência (t_i) não se encaixando, por conseguinte, na definição inicialmente dada.

Mas, que uma *spline* não nula de grau m tenha ou não nós repetidos, o seu suporte sempre conterá um intervalo da forma $[t_k, t_{k+m+1}]$ onde (t_i) é a seqüência de seus nós. Para ilustrar esse fato considere (Figura 9), uma *spline* S_2 não nula de grau 2, sem nós repetidos e que, portanto, deve ser uma função continuamente diferenciável. Observa-se então que o ponto mais à esquerda do suporte de uma *spline* deve ser um de seus nós, dado que fora deles a *spline* é localmente um polinômio. Assim seja t_k o ponto de mínimo do suporte de S_2 . A partir de t_k o gráfico de S_2 segue então uma parábola até pelo menos t_{k+1}, quando pode tomar outra parábola de concavidade oposta à da primeira. Em t_{k+2} a concavidade da parábola pode

mudar novamente de forma que seja possível chegar a t_{k+3} com tangente horizontal. De t_{k+3} em diante, a função pode voltar a ser identicamente nula [PER 89].

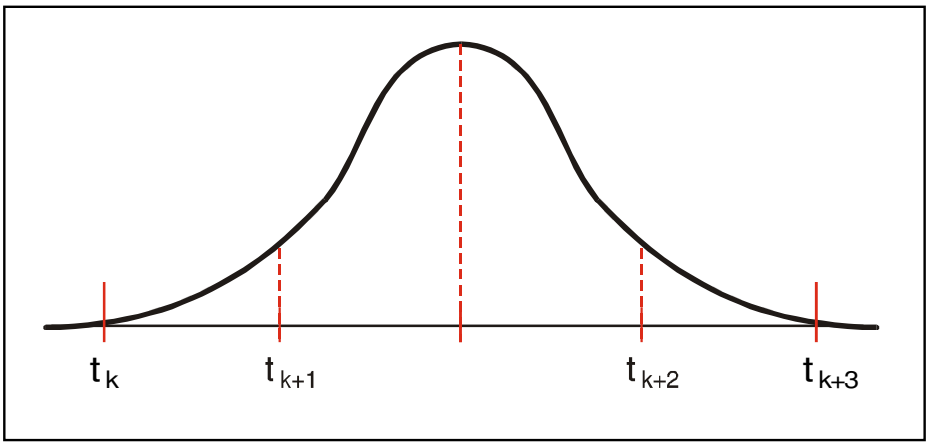


Figura 9: Curvas de *B-Splines*

Dada agora uma seqüência não decrescente de pontos.

(t_i), $i = 1,..., n + m + 1$,

uma família de *B-splines* de grau m ($B_{m,i}$; $i = 1 , ... , n$) pode ser definida recursivamente usando as seguintes relações (Figuras 10, 11, 12 e 13):

a) $B_{0i}(t) = 1$ se $t_i < t < t_i+1$

$B_{0i}(t) = 0$ em caso contrário

$i = 1 , ... , n + m$

b)

$$B_{k,i}(t) = \frac{(t - t_i)B_{k-1,i}(t)}{t_{i+k} - t_i} + \frac{(t_{i+k+1} - t)B_{k-1,i+1}(t)}{t_{i+k+1} - t_{i+1}} \qquad i = 1,\ldots,n+m-k$$

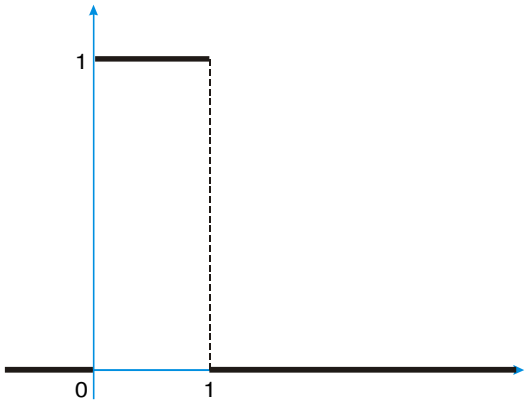


Figura 10: Curvas de *B-Splines* – (B 0,0)

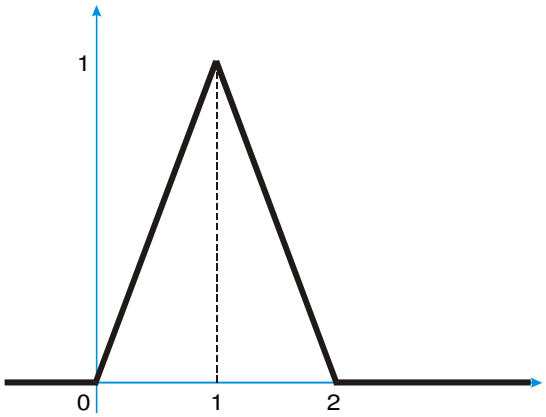


Figura 11: Curvas de *B-Splines* – (B 1,0)

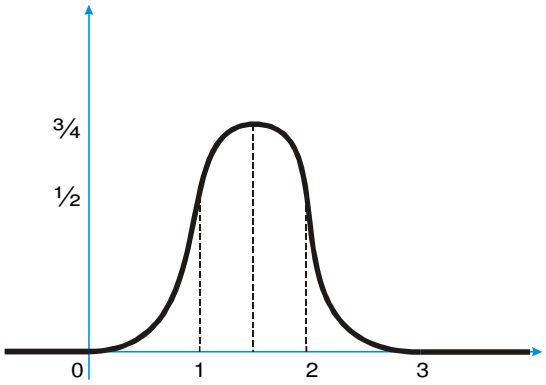


Figura 12: Curvas de *B-Splines* - (B 2,0)

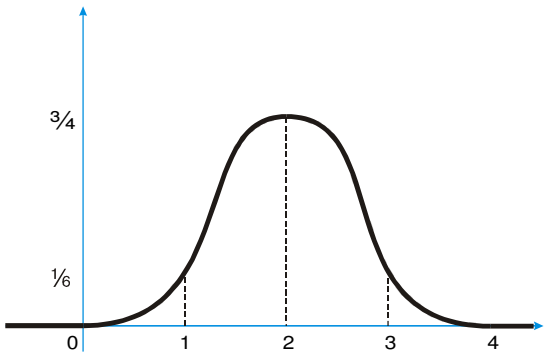


Figura 13: Curvas de *B-Splines* - (B 3,0)

Evidentemente, pode-se ter zero no denominador de um ou de ambos os termos do lado direito da expressão. Se o denominador de um desses termos for zero, entretanto o numerador também o será, qualquer que seja o valor de *t*. Resolveremos a indeterminação resultante fazendo: 0 / 0 = 0. Em particular, se $t_i = t_{i+1} = \ldots = t_{i+k+1} = t_{i+k}$, teremos, por essa regra, que:

$$B_{k,i}(t) = 0 \qquad \forall \quad t.$$

Comprovar que as $B_{m,j}$ atendem, de fato, à definição de *spline* dada no início da seção não é difícil. Mais forte que isso, qualquer *spline* pode ser obtida como combinação linear das *B-splines*, definidas por (a) e (b) anteriormente citadas [PER 89].

Para os que preferem, ao invés de uma definição dada de forma recursiva, uma expressão analítica compacta, pode-se mostrar se teverem

$$t_i = i,\ i = 1,\ldots,n+m,$$

então

$$B_{m,i}(t) = (1/m!) * \sum_{j=0}^{m+1} (-1)^{m+1-j} * C_j^{m+1} * (j+i-t)_+^m$$

onde

$$(f(t))_+ = \begin{cases} f(t) & se\ f(t) > 0 \\ 0 & em\ caso\ contrário. \end{cases}$$

podem-se verificar, por indução finita em *k*, os seguintes fatos relativos às *B-splines*:

a) O suporte de uma *B-spline* de grau *m* é exatamente um intervalo da forma [t_i , t_{i+m+1}] sendo elas, portanto, *splines* de suporte minimal.

b)

$$\nabla\, t \in [t_m, t_{n+1}]$$

é verdade que:

1)

$$\sum_{i=0}^{m} B_{m,i}(t) = 1$$

2)

$$B_{m,i}(t) \geq 0.$$

Assim, ao fazer $F_{i} = B_{m,i}$,$i = 0, 1, ..., n$ tem-se, por **a**, que as $B_{m,i}$ são soluções "ótimas", pelo menos entre as *splines*, para o problema de se conseguir efetuar um controle local eficiente. Além disso, ao trabalhar-se no intervalo [t_m , t_{n+1}] por **b**.1 obtem-se a desejada independência do sistema de coordenadas. Juntando **b**.2 tem-se também que a curva obtida se conserva dentro do fecho convexo dos pontos de controle. Como ainda, escolhendo *m* devidamente, as $B_{m,i}$ podem ter o grau de diferenciabilidade que seja exigido, atinge-se de modo considerável os atributos desejados.

Em comparação com as curvas de *Bézier* convém destacar que o grau de uma *B-spline* não está vinculado ao número de seus nós. Assim é possível manter as F_is com um grau baixo fixo, aumentando, à vontade, o número de pontos de controle. A extensão para superfícies pode ser feita nos mesmos moldes da efetuada no caso das curvas de *Bézier* [PER 89].

Um último comentário diz respeito à interpolação onde o fato de em cada t_i apenas (m + 1) $B_{m,i}$ serem não nulas dá origem a sistemas lineares esparsos. Isso, naturalmente, se *m* for baixo.

3.3.1.1 B-SPLINES PERIÓDICAS E NÃO PERIÓDICAS

Quando um projetista se dispõe a modelar uma curva, ele certamente sabe se a curva que vai ser gerada deve ser fechada ou aberta. Naturalmente é desejável que ele possa passar esta informação ao sistema gráfico que escolhe, então, uma forma de sintetizar a curva de modo a garantir o seu fechamento ou, no caso de a curva a ser gerada ser aberta, fazer com que seus extremos coincidam com um par de pontos dados *s* e *t*.

Uma primeira idéia para obter essa última condição seria usar *B-splines* de grau *m* e fazer os (*m* + 1) primeiros pontos de controle iguais a *s* bem como os (*m* + 1) últimos iguais a *t*. Isto, entretanto, faria com que a curva obtida se aproximasse de *s* e *t* sobre a própria poligonal de controle. Com maior generalidade: isso aumentaria, de forma considerável, a influência dos pontos extremos na determinação do traçado dessa curva, numa vizinhança deles, que pode ser grande [PER 89].

Uma maneira alternativa de resolver esta questão, e que é bastante usada, consiste em proceder da seguinte maneira:

I) fazer $P_0 = s$ e $P_n = t$, agregando os extremos conhecidos da curva à seqüência de pontos de controle.

II) escolher a seqüência de nós (t_i), $i = 0, n + m + 1$ da seguinte forma.

$t_i = 0$ *se* $i < m$

$t_i = i - m$ *se* $m + 1 < i < n$

$t_i = n - m + 1$ *se* $n + 1 < i\ n + m + 1$.

III) considerar as $F_i = B_{m,i}$ definidas no intervalo [0, n – m + 1] .

As *B-splines* obtidas pela escolha de nós dada em II são as chamadas *B-splines* uniformes e não periódicas onde a repetição do nó inicial no começo da seqüência tem o efeito de fazer com que:

$B_{m,0}(0) = 1$

$B_{m,j}(0) = 0 \quad j=1, ..., n$

e assim,

$$\sum F_i(0) * p_0$$

obrigando a curva a começar exatamente em *s*. A repetição do nó final tem efeito análogo forçando que ela termine em *t*.

A repetição de nós faz com que a zona de influência de um extremo seja reduzida, pois o suporte das funções F_0 e F_n passam a ter tamanho um ficando, assim, bem menores que o de uma F_i intermediária. Nas proximidades de um extremo, entretanto, seu peso é preponderante na determinação da curva. Essa redução na zona de influência de um extremo pode ser observada na figura a seguir [PER 89].

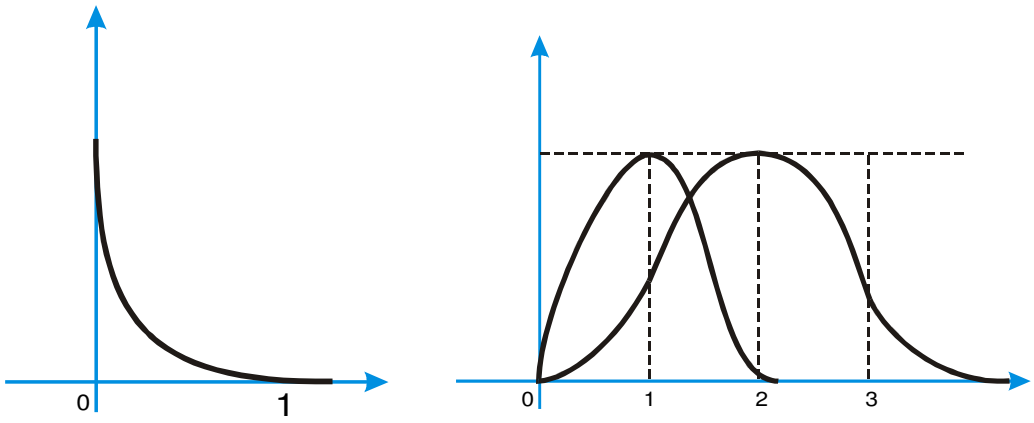


Figura 14: Grafico das *B-splines* uniformes não periódicas

Para completar este item descreve-se como é possível gerar curvas fechadas usando *B-splines*, que são chamadas *B-splines* uniformes periódicas [PER 89]. O que tem-se a fazer para gerar curvas fechadas é obrigar que

$$\sum F_i(a) * p_i = \sum F_i(b) * p_i$$

onde *a* e *b* são os extremos do intervalo de definição das F_i. No caso de ter-se $F_i = B_{mi}$ e de trabalhar com o intervalo [m, n + 1] essa condição se reduz a:

$$\sum_{i=0}^{m-1} B_{n,i}(m) * p_i = \sum_{j=n-m+1}^{n} B_{m,j}(n+1) * p_j$$

Seja então k = n – m + 1 o limite inferior do somatório da direita. A igualdade acima será logicamente atendida se, para i = 0, ..., m - 1, teve-se que:

a) $P_i = P_{k+i}$

b) $B_{m,i}(m) = B_{m,k+i}(n+1)$.

Sejam então q_0 ,..., q_N os pontos de controle dados. Para atender à condição (a) introduz-se, à frente dessa seqüência de pontos, cópias de seus m pontos finais. Geram, assim, uma nova seqüência de controle de cardinalidade $n = N + m$ dada por:

$$p_i = q_{N-m+1+i} \qquad i = 0,..., m-1$$
$$p_i = q_{i-m} \qquad i = m,..., n = 1 + m.$$

Para atingir à condição (b) basta fazer com que o gráfico de $B_{m,k+i}$ seja a translação do de $B_{m,i}$ por um deslocamento de k unidades para a direita. Precisamente:

$$B_{m,k+i}(t) = B_{m,i}(t - k)$$

satisfazendo, portanto, (b) quando $t = n + 1$.

A equação a cima pode ser obtida dispondo-se os t_is de modo uniforme ao longo da reta, como por exemplo fazendo:

$$t_i = i, \qquad i = 0, ..., n + m + 1.$$

Alcançando-se assim as duas condições requeridas para garantir o fechamento da curva. A mesma curva pode ser gerada mantendo-se a seqüência de pontos de controle originais e usando como pesos funções F_i s dadas por (Figura 15):

$$F_i(t) = B_{m,o}((t - i + n + 1) \bmod (n + 1)).$$

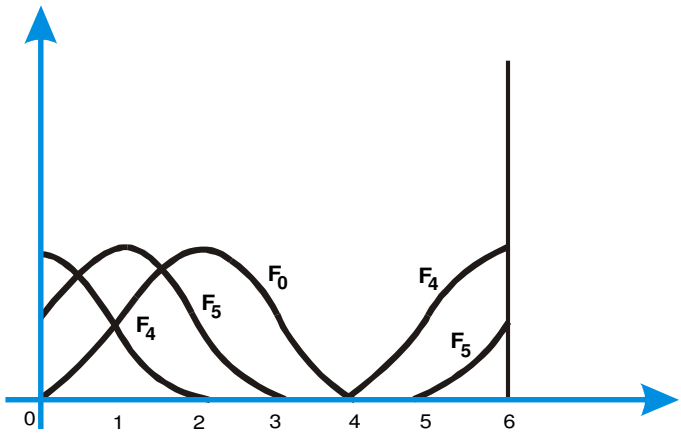


Figura 15: Gráfico com Funções F_is

Gráfico de algumas funções obtidas usando-se a expressão acima. Observar que as funções F_4 e F_5 não são *B-splines*, até porque seu suporte não é um intervalo. Se, entretanto, fossem emendadas as pontas do intervalo [0,6] elas se tornariam *B-splines*.

$B_{m,o}$ é obtido considerando-se a mesma seqüência de nós anterior [PER 89].

3.3.2 ALGORITMO

Apesar de existirem dois tipos de *B-Splines*, as periódicas e não periódicas ou uniformes e não uniformes, o tipo implementado e analisado é a uniforme.

É certo que a forma de matrizes é a maneira mais compacta para se representar uma curva paramétrica, e as operações e análises no algoritmo de *B-Spline* são simples manipulações de matrizes [MOR 85].

No *B-Spline*, tem-se três matrizes, que se relacionam entre si, formando o processo de geração da curva.

Tem-se

$P(u) = U * A, \qquad u \in [0,1]$

e para

$A = M * B$

gerando para p(u) a função genérica da curva paramétrica

$p(u) = U * M * B.$

Então a função p(u) *B-Spline*, para cada segmento i, seguindo os passos do algoritmo paramétrico cúbico, fica da seguinte forma:

$$p_i(u) = 1/6 \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} * \begin{vmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{vmatrix} * \begin{vmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{vmatrix}$$

onde i [1 : n – 2].

Em *B-Spline*, a maneira de encontrar os valores para a matriz B, é definida como

$B = [\ P_{i-1} \quad P_i \quad P_{i+1} \quad P_{i+2}\ )^T$

para cada segmento *i*, onde cada segmento é o intervalo entre dois nós [MOR 85], [BAR 79].

A matriz M é constante e a matriz U, são os pesos (*u*).

Pode-se analisar o exemplo (Figura 16).

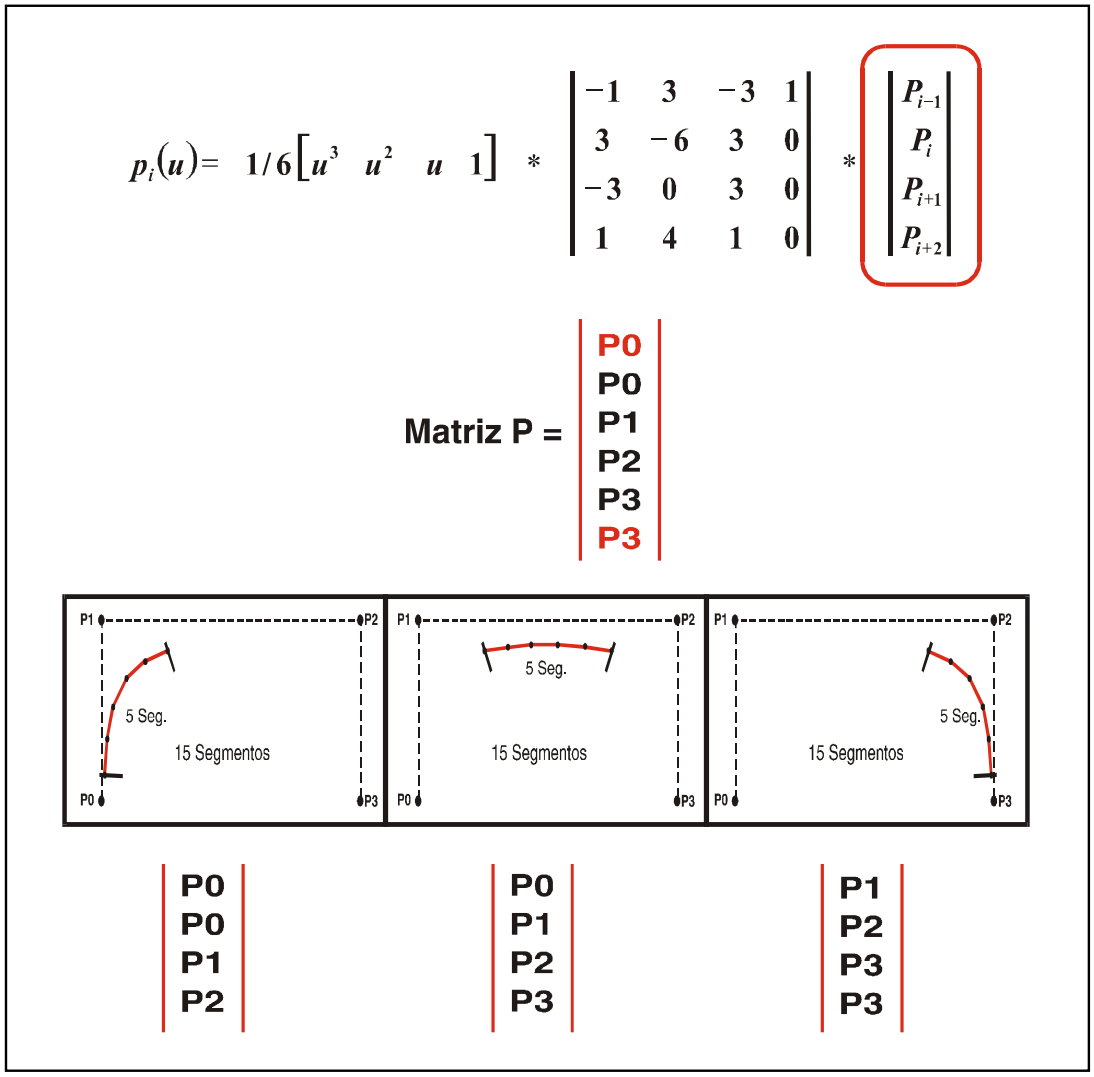


Figura 16: Exemplo geração de curva pelo algoritmo *B-Spline*.

4 PROJETO DO SOFTWARE E IMPLEMENTAÇÃO

Neste capítulo descrevem-se os princípios básicos de funcionamento e as principais características do protótipo de software, incluindo-se um fluxograma descrevendo seu funcionamento.

4.1 ABORDAGEM GERAL

O protótipo do software desenvolvido aborda o problema da escolha certa para determinadas aplicações de uso de *splines*, levando em conta a necessidade de cada aplicação. Quanto a memória ou tempo de execução por exemplo, gerando curvas através de três algoritmos diferentes, podendo ter-se uma comparação entre os diferentes algoritmos.

As *splines* são geradas através de equações matemáticas que compõem os algoritmos. Neste caso foi feito um estudo dessas equações, resultando em uma compreensão que é repassada para o software através de código de linguagem.

As situações que podem ser analisadas, são as mais diversas, porém, será usado neste protótipo 4 (quatro) pontos de controle. Estes servirão para a formação das curvas por diferentes métodos de cálculos.

Após a geração desses algoritmos, podem-se proceder comparativos estatísticos relativos a Ordem de Complexidade, Aspecto Visual, a Necessidade de Memória, o Fator de Precisão e a Facilidade de Implementação.

Objetiva-se com este protótipo, analisar estes comparativos, chegando a uma conclusão, ajudando no julgamento de um determinado algoritmo de *splines* para uma determinada ocasião.

4.2 ESPECIFICAÇÃO E IMPLEMENTAÇÃO

Na implementação deste protótipo foi utilizado a linguagem Delphi, versão 3.0. Visto que o Delphi oferece muitos recursos que permitem criar facilmente aplicações gráficas. O Windows 98 e o Windows NT oferecem alguns recursos para permitir que aplicações gráficas de alto desempenho utilizem os recursos do sistema eficientemente. O Delphi oferece um rico conjunto de componentes gráficos e métodos que ocultam do usuário muitos detalhes de

implementação do sistema operacional, abstraindo detalhes. Isto é extremamente útil para um programador [OSI 98].

O Delphi possui como componente gráfico, a biblioteca de Canvas. A classe *Canvas* permite que o código em Delphi manipule áreas gráficas durante a execução. Um recurso importante da classe *Canvas* é que ela consiste em propriedades e métodos que tornam o uso de gráficos no Delphi relativamente simples. Toda sobrecarga e contabilidade feita fica oculta e sobre a responsabilidade do objeto *Canvas* na implementação [DAM 79], [OSI 98]. Amplia-se, assim, o tempo disponível para a elaboração dos algoritmos de *splines*.

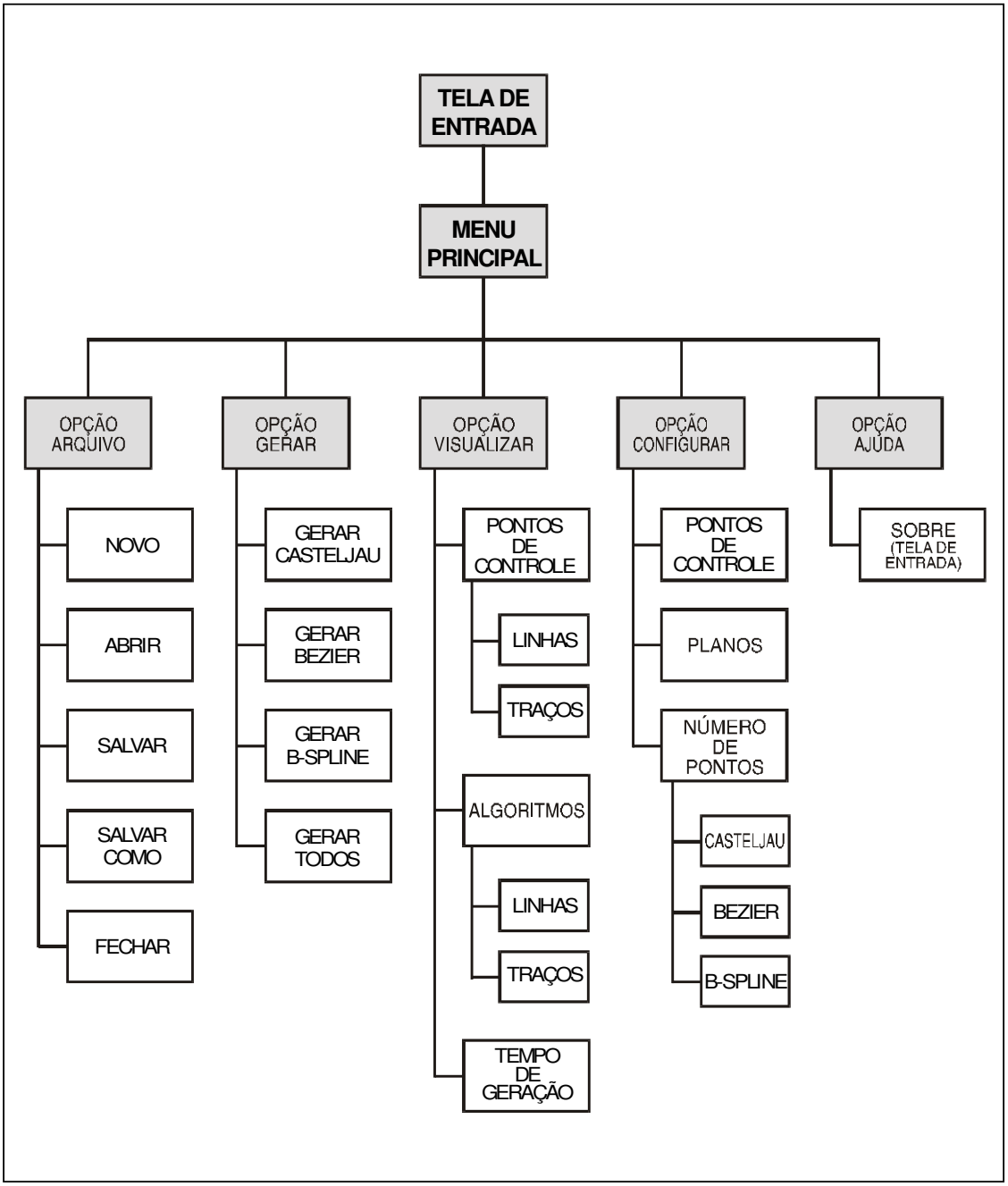
Por processar diretamente coordenadas de pontos armazenadas em arquivos textos ou digitadas quando solicitadas, como parâmetros, o protótipo do software não requer quaisquer bases de dados adicionais, pois o conhecimento necessário para a interpretação das mesmas está implementado em seus algoritmos.

No desenvolvimento deste protótipo foram implementados os algoritmos de *Bezier*, *Casteljau* e *B-Spline* separadamente, onde cada um destes processos possui suas próprias estruturas, não possuindo nenhuma rotina genérica, proporcionando uma análise personalizada de cada método [MEL 90].

Desta forma, o protótipo do software desenvolvido pode ser especificado a partir de um fluxograma de representação estrutural, ilustrado no Quadro 1.

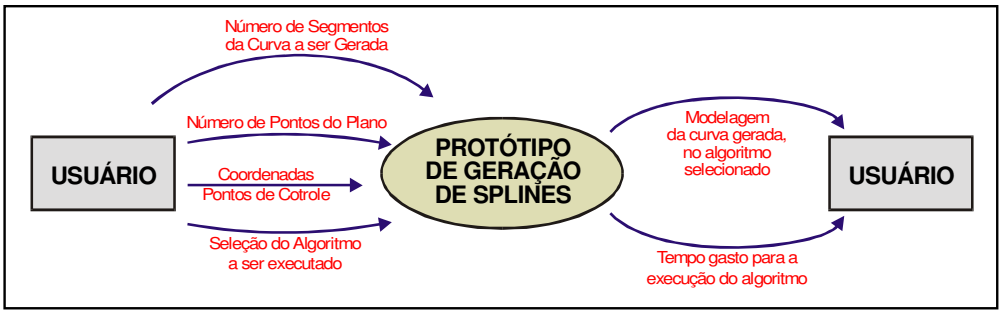
O protótipo foi feito com o intuito de permitir uma interação amigável através do uso de padrões de interface (*windows*), objetivando a utilização do protótipo pelo usuário.

Na implementação as principais rotinas, as quais fazem gerar as curvas de *splines* através de *Bezier, Casteljau* e *B-Spline* serão descritas a seguir.



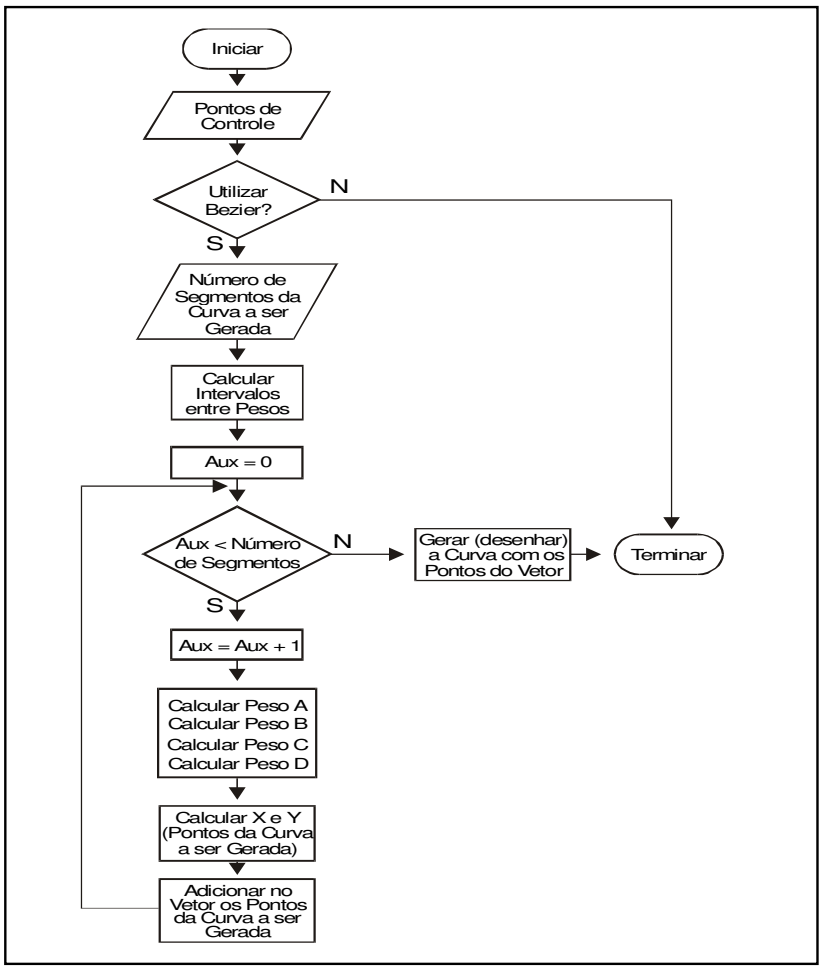
Quadro 1: Fluxograma da Representação estrutural do Protótipo (Menu).

Pode-se observar a estrutura do Quadro 2, verificando as entradas e saídas do protótipo.



Quadro 2: Estruturação de entrada e saída.

No algoritmo de *Bezier* pode-se observar o seguinte fluxograma (Quadro 3).



Quadro 3: Fluxograma do algoritmo de *Bezier*.

Em *Bezier* tem-se a rotina que processa a curva e retorna em um vetor, os pontos processados (Quadro 4).

```
//Processa Bezier e retorna em um vetor, os pontos processados
class function TBezier.Execute ( a4Pontos: T4Pontos; aNumberPontos: Integer ): TStringList;
var
  xBezier: TBezier;
begin
  xBezier := TBezier.Create( a4Pontos );
  try
    Result := TStringList.Create;
    try
      xBezier.LoadPontoControle(aNumberPontos); //Processa Bezier
      Result.AddStrings(xBezier.FVetor); //Adiciona em um vetor "FVetor"
    except
      Result.Destroy;
      raise;
    end;
  finally
    xBezier.Destroy;
  end;
end;
```

Quadro 4: Rotina Processa *Bezier*

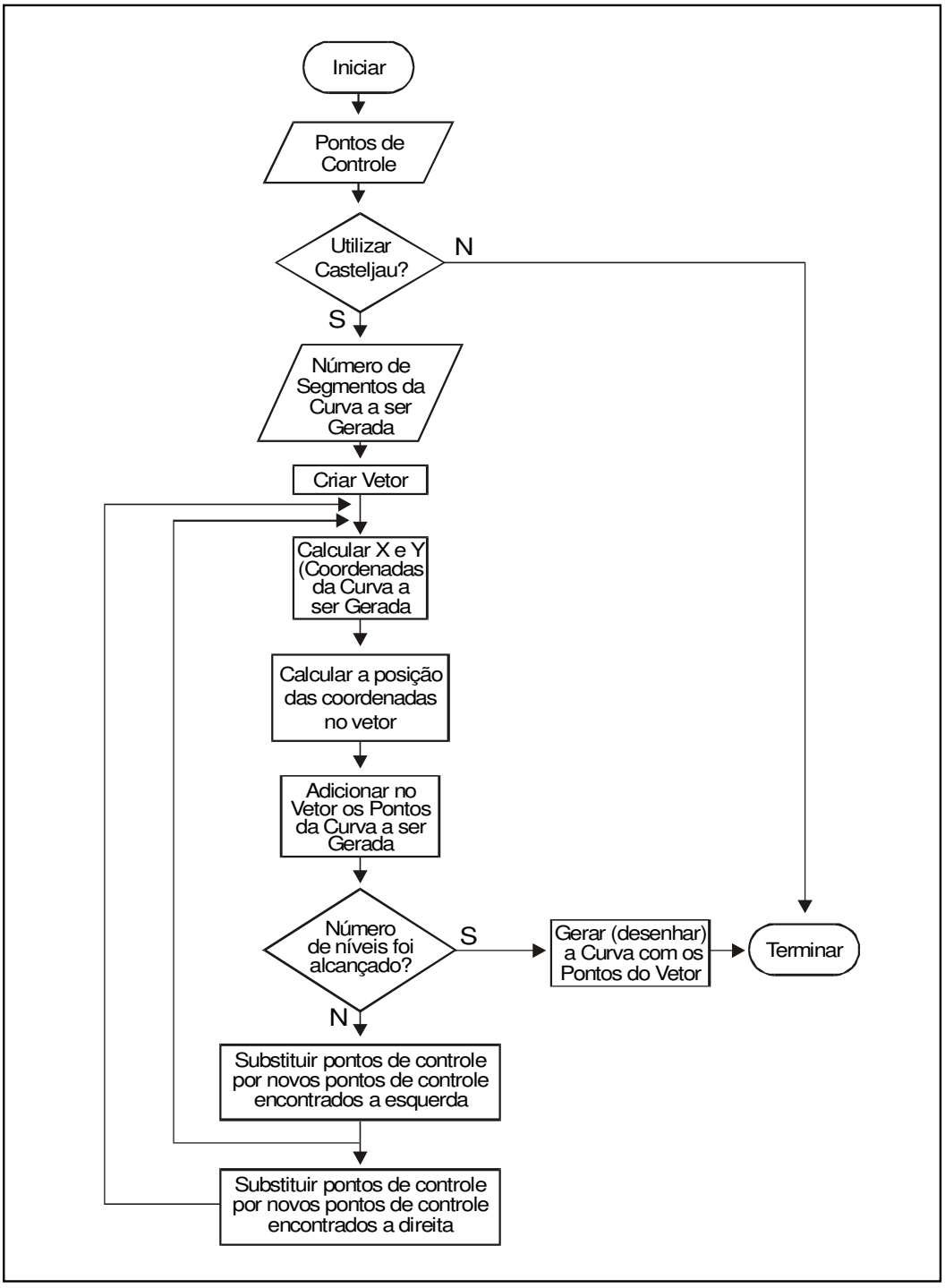
Esta rotina inicializa e finaliza o processo de geração da curva de *spline Bezier.* O item mais importante nesta rotina do Quadro 4, é a chamada para a rotina "*xBezier.LoadPontoControle(aNumberPontos)*" (Quadro 5). Esta *procedure* calcula o intervalo entre os pontos da curva gerada, executa um *for* de 0 até o número de pontos, onde calcula os pesos de cada Ponto de Controle, encontra as coordenadas dos pontos da curva a

ser gerada através da multiplicação dos pontos de controles com seus pesos encontrados e adiciona em um vetor.

```
//Processa Bezier
procedure TBezier.LoadPontoControle( _NumeroPontos: Integer );
var
  xBaseT,xAux: Real;
  xPA, xPB, xPC, xPD: Real;
  xX,xY: Cardinal;
  i: integer;
begin
  xBaseT := 1 / _NumeroPontos; // Encontra intervalo entre pontos da Curva

  for i:= 0 to _NumeroPontos do //De 0 ate o número de pontos
  begin

    // Calcula os pesos dos pontos de controle.
    xAux := i * xBaseT;
    xPA := GetPeso0(xAux);  //Função que calcula
    xPB := GetPeso1(xAux);
    xPC := GetPeso2(xAux);
    xPD := GetPeso3(xAux);

    //Calcula o X e o Y, dos pontos da curva a ser gerada
    xX := Trunc((FPA.X * xPA) + (FPB.X * xPB) +
                (FPC.X * xPC) + (FPD.X * xPD));
    xY := Trunc((FPA.Y * xPA) + (FPB.Y * xPB) +
                (FPC.Y * xPC) + (FPD.Y * xPD));

    AddToVetor(xX, xY); //Adiciona para o vetor
  end;
```

Quadro 5: Rotina que Calcula os pontos da Curva *Bezier*

No algoritmo de *Casteljau* tem-se o seguinte fluxograma (Quadro 6).



Quadro 6: Fluxograma do algoritmo de Casteljau.

Em *Casteljau* tem-se a rotina que processa a curva e retorna em um vetor, os pontos processados (Quadro 7).

```
//Chama o Processo Processar pontos Cateljau.
procedure TCastJ.Gerar(_Nivel: Cardinal);

var
  i: Integer;
  xCapacity: Integer;

begin
  if _Nivel <= 0 then  //Limita o Nível para que seja maior do que 0
    Raise Exception.Create('Nível inválido');

  FNivel := _Nivel;
  FCurNivel := 1;

  FVetor.Capacity := Trunc(Exp_(2,_Nivel) + 1); //Calcula o número de pontos

  for i := 0 to FVetor.Capacity - 1 do
   FVetor.Add('lixo');       // Joga lixo para criar o vetor no tamanho que deverá ser.
  AddToVetor(0, FPA);
  AddToVetor(FVetor.Count - 1, FPD);
  // Chama rotina CastJ, que calcula os pontos da curva e monta o vetor.
  CastJ(FPA,FPB,FPC,FPD,0,FVetor.Capacity - 1, 1);
end;
```

Quadro 7: Rotina que Processa *Casteljau*

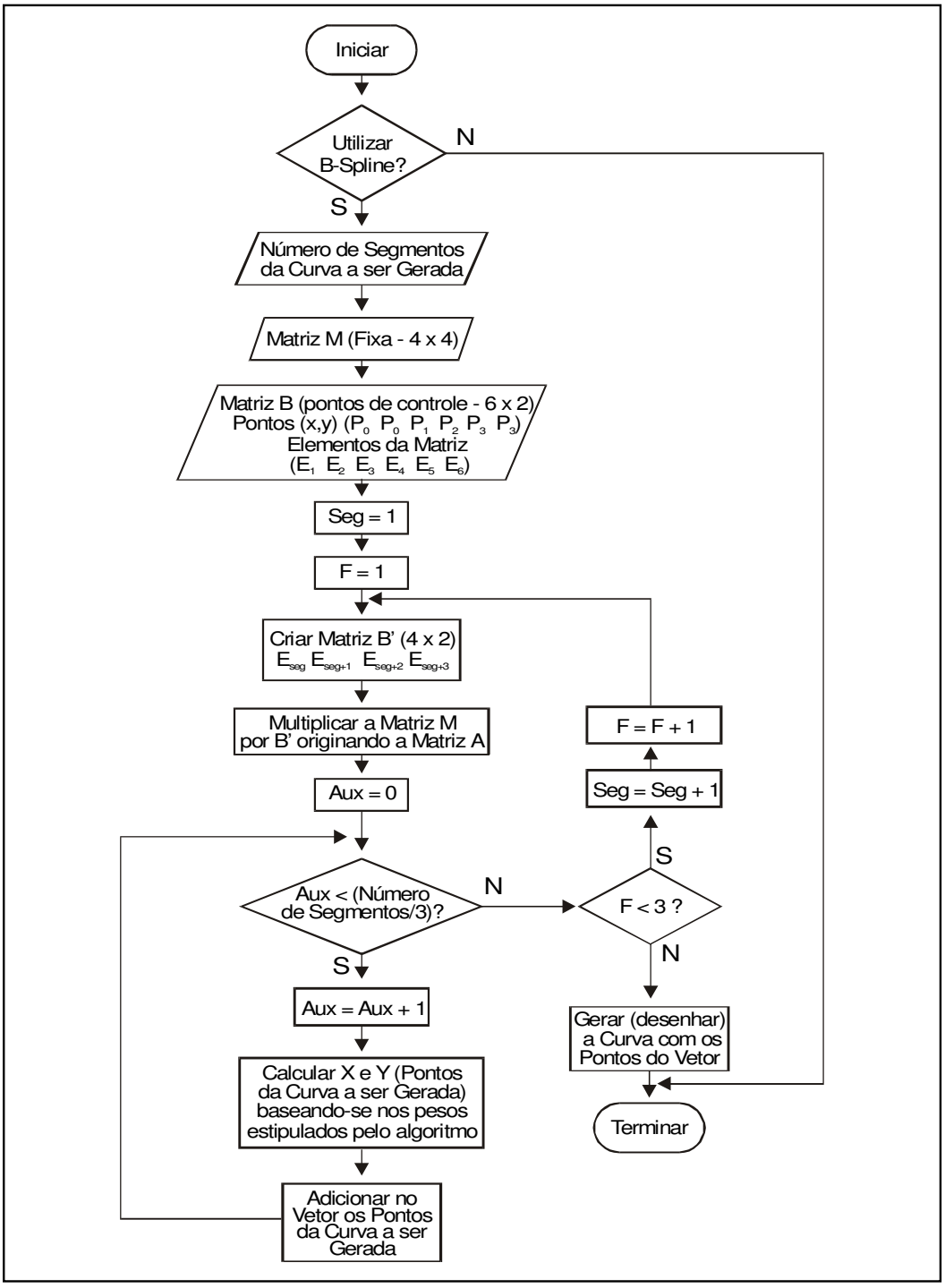
Esta rotina inicializa e finaliza o processo de geração da curva de *Casteljau*, o item mais importante nesta rotina do Quadro 7, é a chamada para a rotina "CastJ(FPA,FPB,FPC,FPD,0,FVetor.Capacity - 1, 1);" (Quadro 8). Esta *procedure* calcula os pontos médios até chegar a cada um dos pontos da curva a ser gerada, em um processo

recursivo. Encontra a posição no vetor onde serão adicionadas as coordenadas do ponto gerado devido ao algoritmo que não encontra esses pontos em seqüência. Os pontos no vetor devem ficar em seqüência para que depois seja traçada as curvas. Adiciona o ponto no vetor e faz o processo recursivo para a geração dos demais pontos da *spline* a ser desenhada.

```
//Principal de Casteljau, Processa os pontos (recursividade)
procedure TCastJ.CastJ(_A,_B,_C,_D: TPonto; IndIni, IndFim, aNivel: Integer);
var
  P1,P2,P3,P4,P5,PM: TPonto;
  xNewInd: Integer;

begin
  P1 := Ponto_Medio(_A, _B);    //Calcula Ponto Médio entre A e B
  P2 := Ponto_Medio(_B, _C);
  P3 := Ponto_Medio(_C, _D);
  P4 := Ponto_Medio(P1, P2);
  P5 := Ponto_Medio(P2, P3);
  PM := Ponto_Medio(P4, P5);

  // Encontra o posição do vetor para adicionar o resultado
  xNewInd := ((IndIni + IndFim) div 2);
   // Adiciona no vetor
  ADDToVetor(xNewInd,PM);
  if aNivel < FNivel then

  //Recursividade
  begin
    CastJ(PM,P5,P3,_D,xNewInd,IndFim, aNivel + 1); //Direita
    CastJ(_A,P1,P4,PM,IndIni,xNewInd, aNivel + 1); //Esquerda
  end;
end;
```

Quadro 8: Rotina que calcula os pontos da curva no algoritmo *Casteljau*

No algoritmo de *B-Spline* tem-se o seguinte fluxograma (Quadro 9).



Quadro 9: Fluxograma do algoritmo de *B-Spline*.

Em *B-Spline* tem-se a rotina que processa a curva e retorna em um vetor, os pontos processados (Quadro 10).

```
//Processa B-Spline e retorna em um vetor, os pontos processados
class function TBSpline.Execute ( a4Pontos: T4Pontos; aNumberPontos: Integer ):
                                                    TStringList;
var
  xBSpline: TBSpline;
begin
  xBSpline := TBSpline.Create( a4Pontos );
  try
    Result := TStringList.Create;
    try
      //Chama a rotina principal xBSpline.LoadPontoControle
      xBSpline.LoadPontoControle(aNumberPontos);
      Result.AddStrings(xBSpline.FVetor);
    except
      Result.Destroy;
      raise;
    end;
  finally
    xBSpline.Destroy;
  end;
end;
```

Quadro 10: Rotina que processa *B-Spline*

Esta rotina inicializa e finaliza o processo de geração da curva de *spline B-Spline*, o item mais importante nesta rotina do Quadro 10, é a chamada para a rotina "xBSpline.LoadPontoControle(aNumberPontos);" (Quadro 11). Esta *procedure* carrega a matriz M, multiplica essa matriz M com os Pontos de Controle que é a matriz B, multiplica a matriz que resultou dessa multiplicação anterior com os pesos que é a matriz U, gerando

assim as coordenadas dos pontos da curva a ser formada e finalmente adiciona essas coordenadas em um vetor.

```
//Adiciona os pontos na Matriz B
procedure TBSpline.LoadMatriz_BsM;
  begin
      FM_M[1,1] := -1; FM_M[1,2] :=  3; FM_M[1,3] := -3; FM_M[1,4] := 1;
      FM_M[2,1] :=  3; FM_M[2,2] := -6; FM_M[2,3] :=  3; FM_M[2,4] := 0;
      FM_M[3,1] := -3; FM_M[3,2] :=  0; FM_M[3,3] :=  3; FM_M[3,4] := 0;
      FM_M[4,1] :=  1; FM_M[4,2] :=  4; FM_M[4,3] :=  1; FM_M[4,4] := 0;
  end;

//Coloca os valores dos Pontos de Controle na matriz B
procedure TBSpline.LoadMatriz_BsXY( var aMatriz: TMatrizBS );
  begin
       aMatriz[1,1] := FPA.X; //X  -  Repete os pontos P0 x
       aMatriz[1,2] := FPA.Y; //y  -  Repete os pontos P0 y
       aMatriz[2,1] := FPA.X; //X
       aMatriz[2,2] := FPA.Y; //y
       aMatriz[3,1] := FPB.X; //X
       aMatriz[3,2] := FPB.Y; //y
       aMatriz[4,1] := FPC.X; //X
       aMatriz[4,2] := FPC.Y; //y
       aMatriz[5,1] := FPD.X; //X
       aMatriz[5,2] := FPD.Y; //y
       aMatriz[6,1] := FPD.X; //X – Repete os pontos P3 x
       aMatriz[6,2] := FPD.Y; //y – Repete os pontos P3 y
  end;

// Pega apenas 4 pontos da Matriz B, que é percorrida de 4 em 4 pontos
procedure TBSpline.Calcula_B( aSeg: Integer; var aMatrix: TMatrizAS );
  begin
        aMatrix[1,1] := FM_XY[aSeg ,      1];
        aMatrix[1,2] := FM_XY[aSeg ,      2];
        aMatrix[2,1] := FM_XY[aSeg + 1, 1];
        aMatrix[2,2] := FM_XY[aSeg + 1, 2];
        aMatrix[3,1] := FM_XY[aSeg + 2, 1];
        aMatrix[3,2] := FM_XY[aSeg + 2, 2];
        aMatrix[4,1] := FM_XY[aSeg + 3, 1];
        aMatrix[4,2] := FM_XY[aSeg + 3, 2];
  end;
```

Quadro 11: Rotina que calcula os pontos da Curva B-Spline

```
procedure TBSpline.Calcula_A( aMatriz: TMatrizAS ); //Matriz A = B * M
var  i1, i2: Integer;
begin
  FillChar(FM_A, SizeOf(FM_A),0);  // Zera a Variável
  for i1 := 1 to 4 do
  begin
     for i2 := 1 to 4 do
     begin
           FM_A[i1,1]:= FM_A[i1,1] + (FM_M[i1,i2] * aMatriz[i2,1]);
           FM_A[i1,2]:= FM_A[i1,2] + (FM_M[i1,i2] * aMatriz[i2,2]);
    end;  end;   end;

// Calcula os pontos da curva – Coordenadas x e y
procedure TBSpline.Calcula_Ponto( aNumeroPontos: Integer );
var
  xX, xY: Integer;      xAux, xBaseT: Real;      i: Integer;
begin
  xBaseT := 1 / aNumeroPontos;  //Aqui aNumeroPontos já foi dividido por 3
  for i := 0 to (aNumeroPontos) do
  begin
      xAux := i * xBaseT;

      xX := Round((Power(xAux,3) * FM_A[1,1] + Power(xAux,2) * FM_A[2,1] +
                                   xAux    * FM_A[3,1] + FM_A[4,1]) / 6);
      xY := Round((Power(xAux,3) * FM_A[1,2] + Power(xAux,2) * FM_A[2,2] +
                                   xAux    * FM_A[3,2] + FM_A[4,2]) / 6);

      AddToVetor(xX, xY); //Adiciona no vetor as coordenadas da curva
  end;  end;

//Procedure principal, gera as coordenadas da curva B-Spline
procedure TBSpline.LoadPontoControle( _NumeroPontos: Integer );
var   xBaseT,xAux: Real;  xMatrizB: TMatrizAS;  i: integer;
begin
  LoadMatriz_BsM; // Carrega a Matriz M
  _NumeroPontos := (_NumeroPontos div 3);
  xBaseT := 1 / _NumeroPontos; //Encontra o intervalo entre pontos
  LoadMatriz_BsXY(FM_XY);
     for i:= 1 to 3 do  // Laço com 3 repetições
         begin
            Calcula_B(i, xMatrizB); //Recalcula a Matriz B
            Calcula_A(xMatrizB); //Recalcula a Matriz A = B * M
            Calcula_Ponto(_NumeroPontos); // Calcula os pontos da Curva
         end;
    end;
```

Quadro 12: Rotina que calcula os pontos da Curva B-Spline (Cont.)

Essas rotinas vistas anteriormente formam 3 vetores independentes, um para cada algoritmo (*Bezier, Casteljau* e *B-Spline*). Pega-se estes vetores e através do propriedade *Canvas.LineTo* a curva é gerada traçando-se uma linha de ponto a ponto contido em cada um dos vetores, formando uma curva para cada vetor.

4.3 FUNCIONAMENTO

Ao ser inicializado, o protótipo apresenta uma tela de entrada com informações descritivas do protótipo (Figura 17).

Figura 17: Tela de entrada

Após a tela de abertura, vem o menu principal, onde estão todas as opções necessárias para o funcionamento do protótipo (Figura 18). As opções são básicas e claras, assemelhando-se às barras de ferramentas de alguns softwares disponíveis no mercado.

Além das opções e barras de ferramentas, apresenta também 4 planos distintos, onde o plano A serve para a visualização gráfica e configuração dos pontos de controle, o plano B para a representação gráfica do algoritmo de *Casteljau*, o plano C para a representação gráfica do algoritmo de Bezier e o plano D para o *B-Spline*.

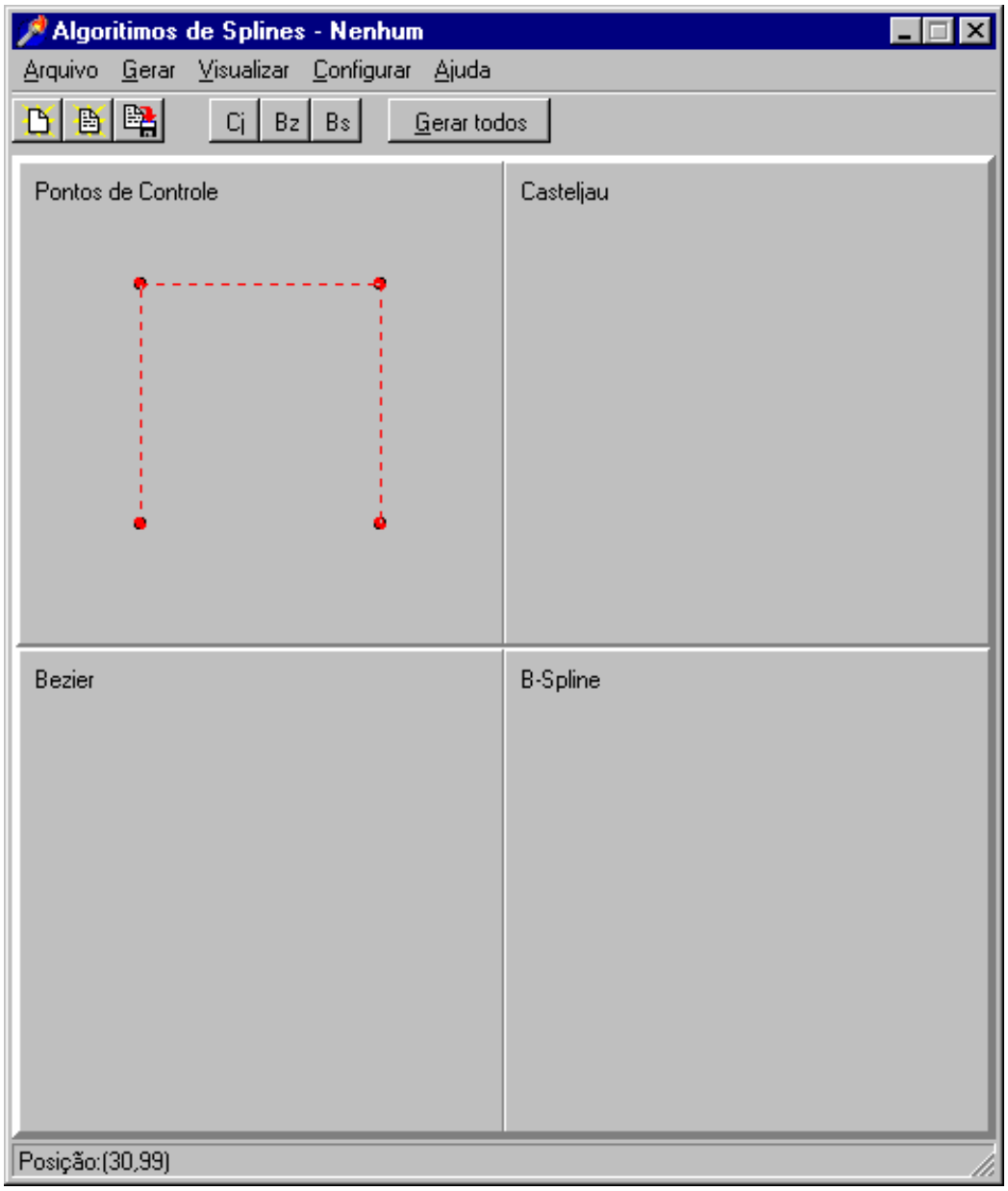


Figura 18: Menu Principal

4.3.1 OPÇÃO ARQUIVO

Na opção Arquivo tem-se: **Novo, Abrir, Salvar, Salvar Como, Imprimir e Fechar** (Figura 19). Estas opções são padrões adotados na maioria dos softwares disponíveis no mercado.

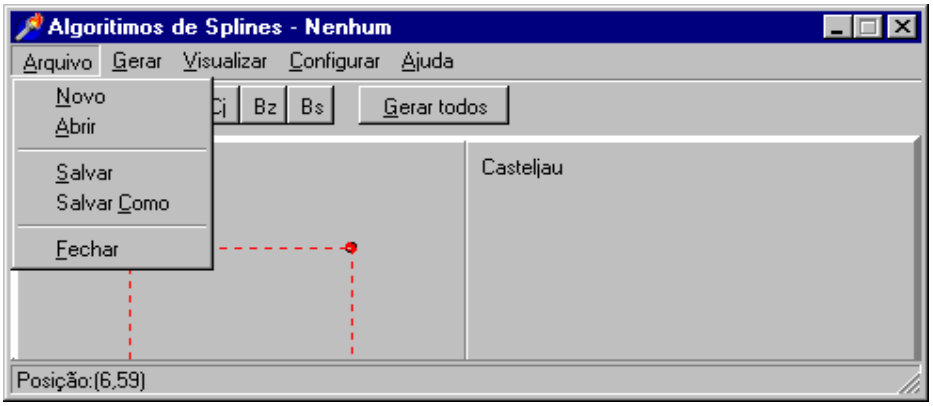


Figura 19: Menu Arquivo

O opção **Novo** estabelece um arquivo novo com configurações padrões, aplicando valores padrões no plano (100 x 100 unidades), na localização do pontos de controle, no número de níveis do algoritmo de *Casteljau* e no número de pontos dos algoritmos de *Bezier* e *B-Spline*.

A opção **Abrir**, serve para abrir um arquivo já existente, padronizado com a extensão .SPL de *splines* (Figura 20). O arquivo é do tipo Binário e a extensão SPL vem de Splines.

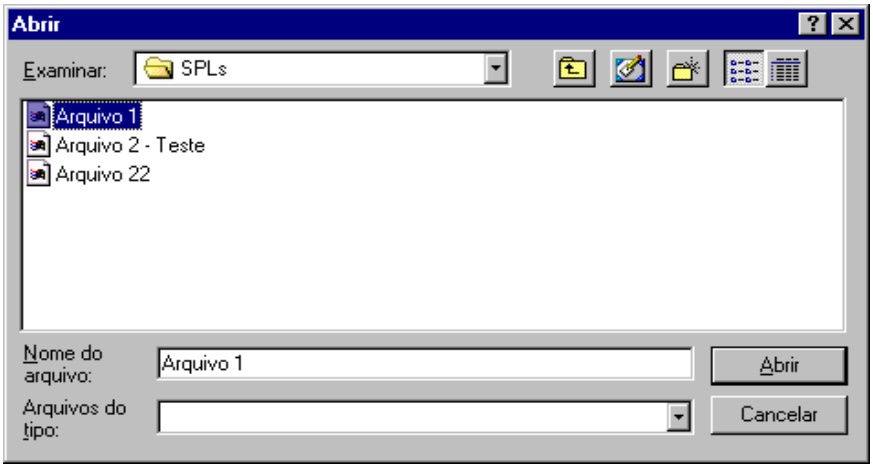


Figura 20: Menu Abrir

A opção **Salvar**, serve para salvar um arquivo com a terminação .SPL (tipo binário), onde são salvos as coordenadas dos pontos de controle, o plano, o número de níveis do algoritmo de *Casteljau* e o número de pontos que deve ter a curva gerada nos algoritmos de *Bezier* e *B-Spline* que estão na configuração atual.

O **Salvar Como**, salva um arquivo nas mesmas condições do **Salvar**, porém com outro nome. (Figura 21).

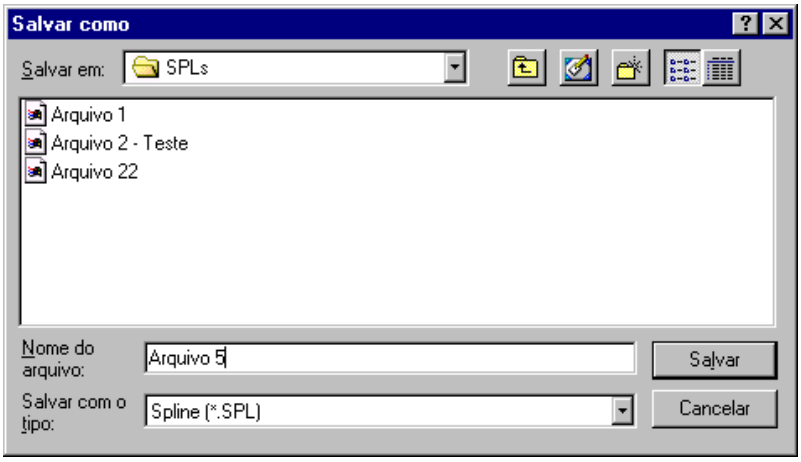


Figura 21: Menu Salvar Como

A opção **Fechar**, mostra uma mensagem perguntando se deseja salvar ou não o que foi feito. Dependendo da resposta, o arquivo é ou não salvo e o protótipo é finalizado.

4.3.2 OPÇÃO GERAR

Na opção Gerar tem-se: **Gerar *Casteljau*, Gerar *Bezier*, Gerar *B-Spline* e Gerar Todos** (Figura 22).

Figura 22: Menu Gerar

A opção **Gerar *Casteljau*** faz processar o algoritmo de *Casteljau* (Figura 22), e mostra o resultado graficamente no plano de *Casteljau* (Figura 23). A geração deste gráfico acontece levando em conta: a localização das coordenadas dos pontos de controle, a escala o número de níveis que estabelece o número de pontos que formará a curva de *Casteljau* e o algoritmo que processa os cálculos que determinam a localização dos pontos na curva de *Casteljau*.

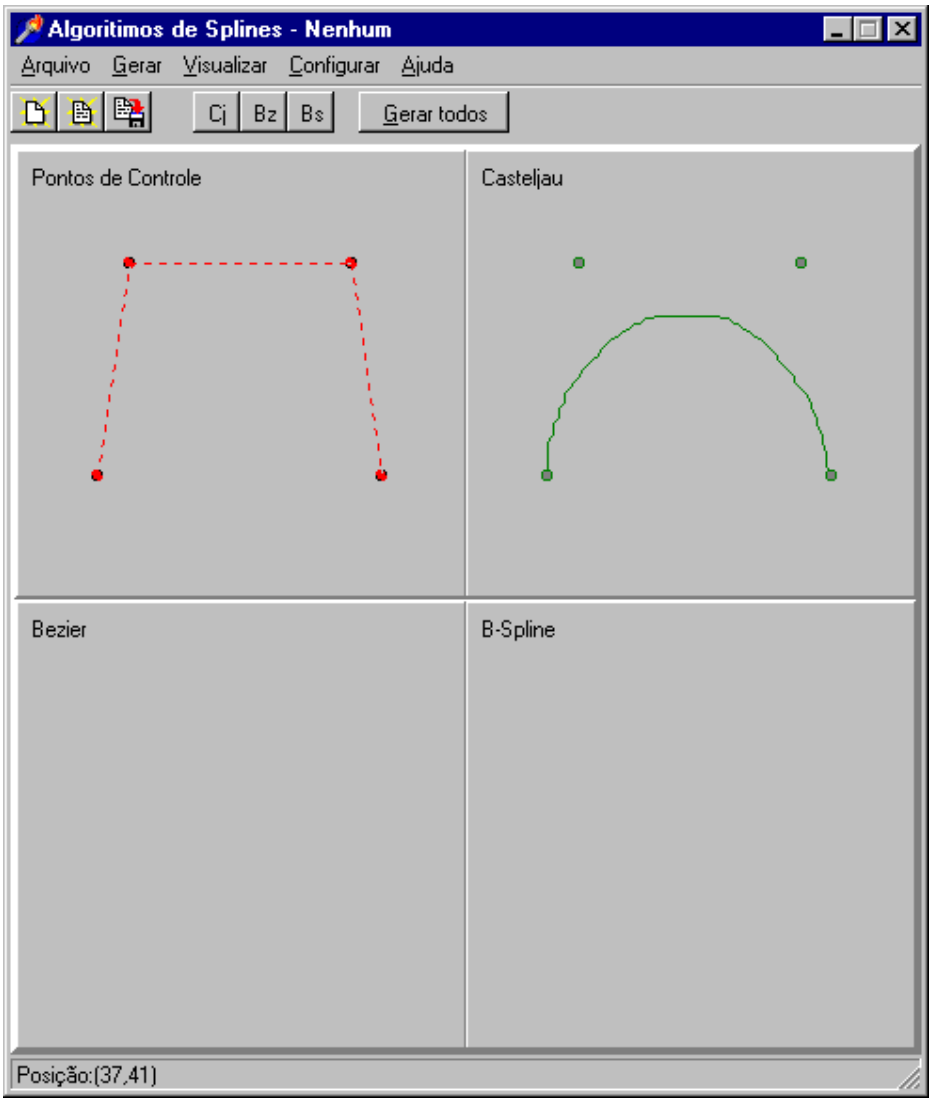


Figura 23: Gerar *Casteljau*

A opção **Gerar *Bezier*** faz processar o algoritmo de *Bezier*, e mostra o resultado graficamente no plano de Bezier (Figura 24). A geração deste gráfico também acontece levando em conta os ítens vistos no **Gerar *Casteljau*.**

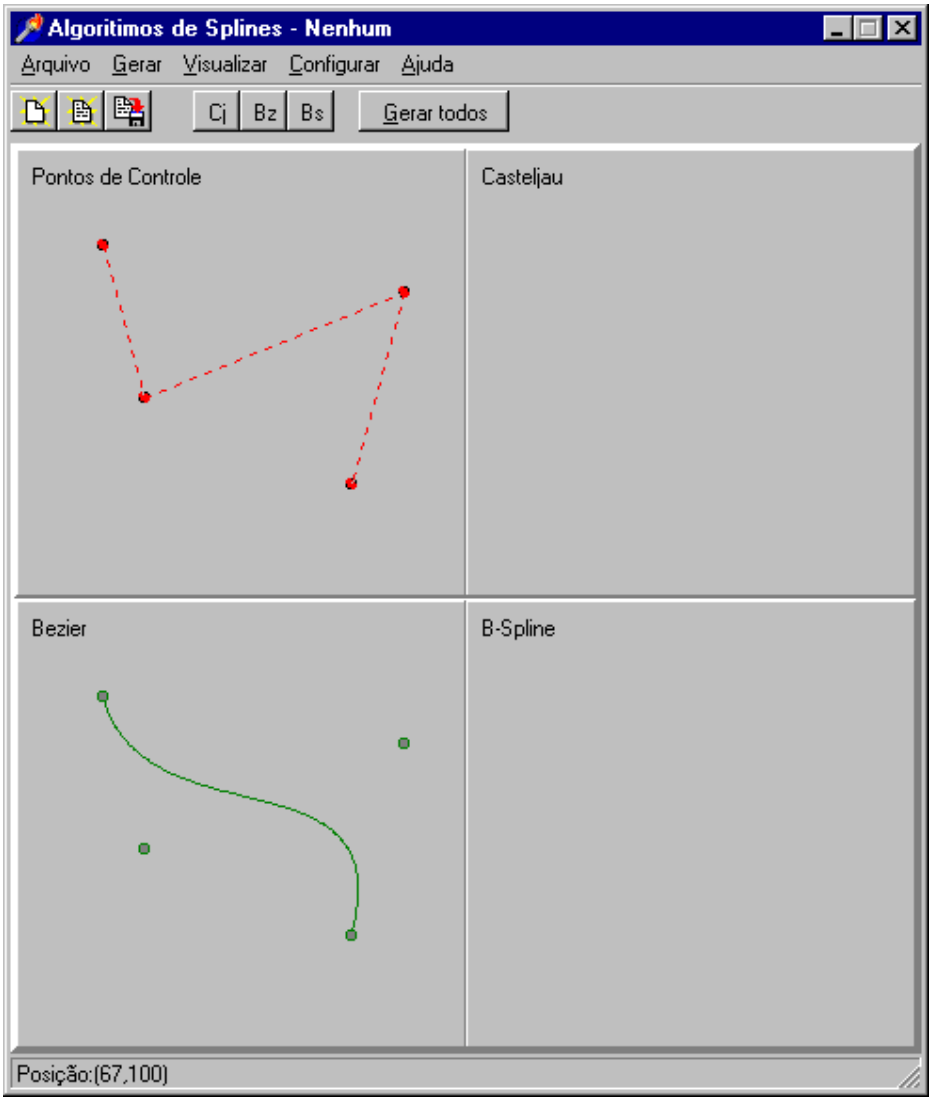


Figura 24: Gerar *Bezier*

A opção **Gerar *B-Spline*** faz processar o algoritmo de *B-Spline*, e também mostra o resultado graficamente no plano de *B-Spline* (Figura 25). A geração deste gráfico também acontece levando em conta os ítens vistos no **Gerar *Casteljau*.**

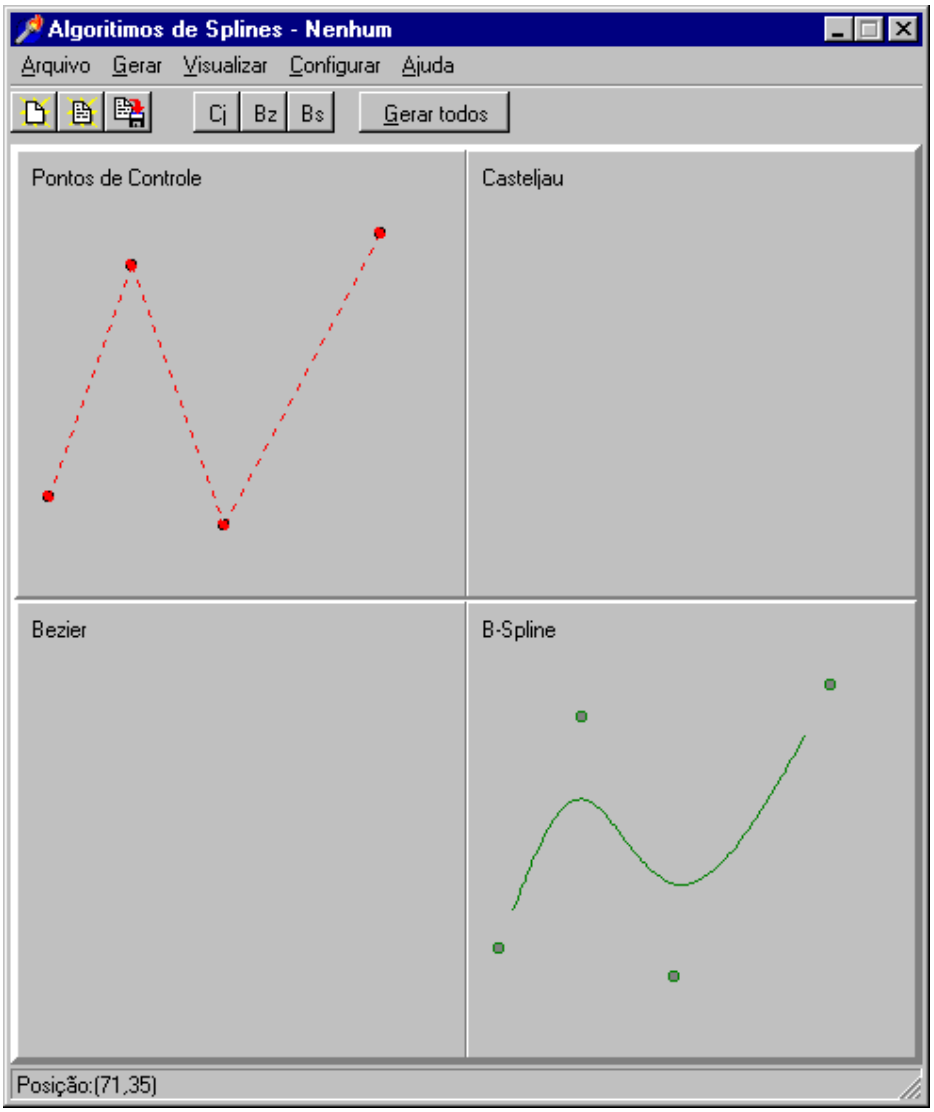


Figura 25: Gerar *B-Spline*

A opção **Gerar Todos**, faz processar os três algoritmos, *Bezier*, *Casteljau* e *B-Spline* e mostra o resultado gráfico dos três, cada um em seu plano.

4.3.3 OPÇÃO VISUALIZAR

O plano A se refere aos pontos de controle. Estes pontos são expostos de forma gráfica com uma representação de 4 pontos distintos e linhas tracejadas que juntam e dão seqüência a esses pontos (Poliedro de Controle).

Esta representação, permite a mudança das coordenadas dos pontos de controle através dos pontos desenhados no plano, basta somente clicar com o mouse em cima de um ponto de

controle e arrastar o mouse soltando este ponto onde desejar no plano. Após soltar o ponto as coordenadas X e Y desse ponto se alteram podendo ser visto a diferença nas curvas geradas, se optar em gerar novamente essas curvas.

A opção **Visualizar** divide-se em **Tempo de Geração**, **Pontos de Controle** e **Algoritmos** (Figura 26).

Figura 26: Menu Visualizar

A opção **Visualizar Tempo de Geração** (Figura 27) mostra a comparação de tempos em milessegundos, de processamento da curva gerada por cada algoritmo, permitindo a análise de complexidade do algoritmo de forma experimental. Com essa opção é possível concluir qual dos algoritmos é o mais veloz, ou que possui uma complexidade menor.

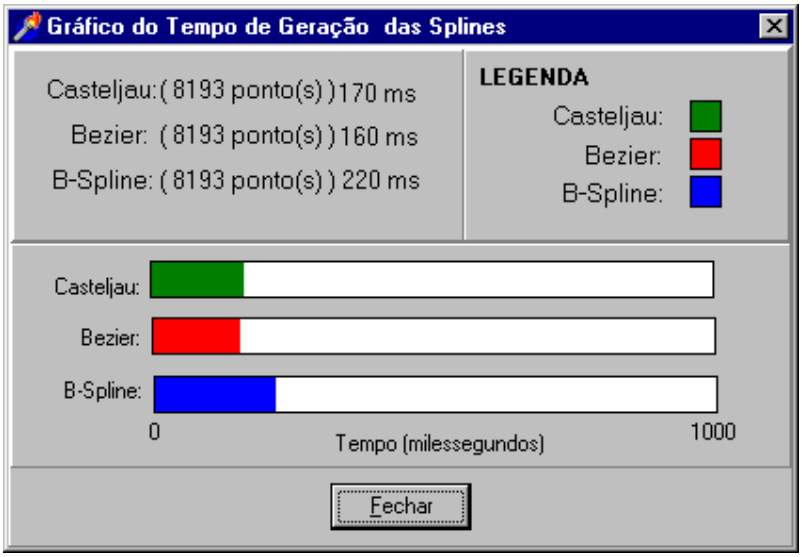


Figura 27: Gráfico do Tempo de Geração das *Splines*

A opção **Visualizar Pontos de Controle** permite a configuração da visualização ou não dos pontos de controle ou das linhas traçadas que ligam os pontos de controle (Figura 28).

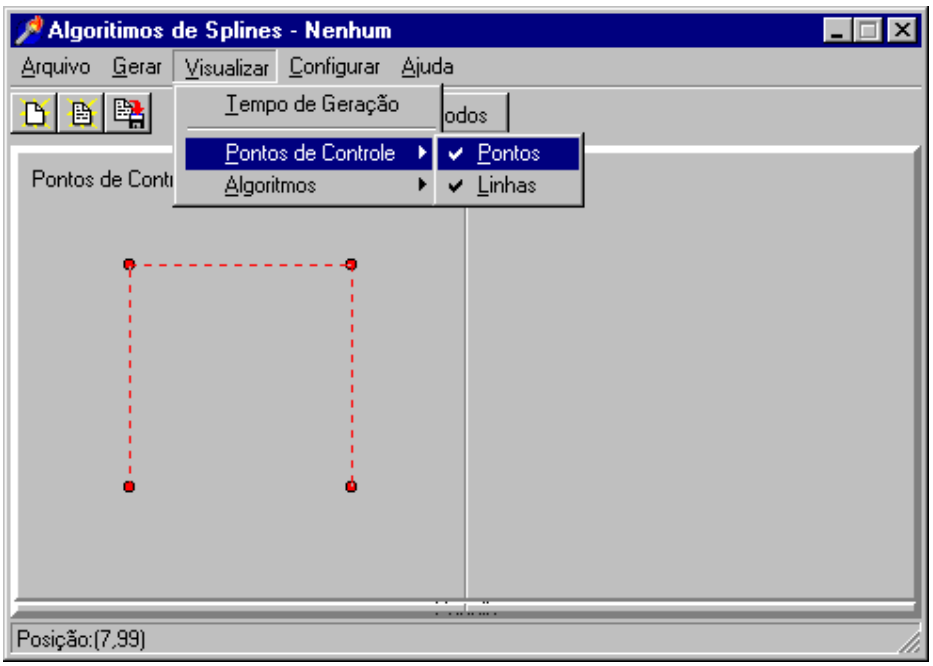


Figura 28: Visualizar Pontos de Controle

São oferecidas as opções de visualizar somente os pontos (pontos em vermelho) ou somente as linhas (linhas em vermelho tracejadas) ou nenhum dos dois (Figura 28).

A opção **Visualizar Algoritmo** permite a mesma configuração da opção Visualizar Pontos de Controle, porém se aplica aos planos de geração de curvas *Bezier, Casteljau* e *B-Spline,* juntamente com a curva gerada (Figura 29).

4.3.4 OPÇÃO CONFIGURAR

Esta opção de **Configurar** tem como opções secundárias **Ponto de Controle, Plano e Numero de Segmentos** (Figura 30), que tratam da configuração principal do sistema.

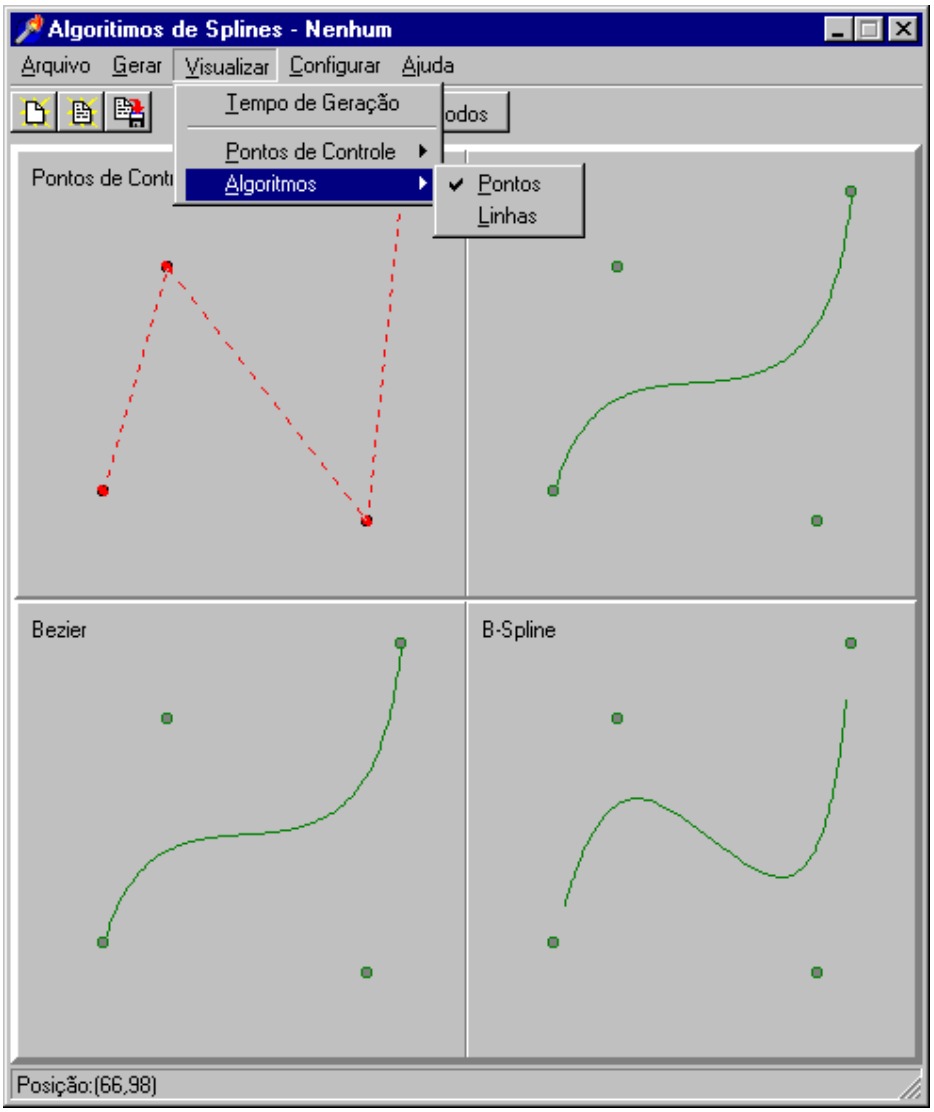


Figura 29: Visualizar Algoritmos

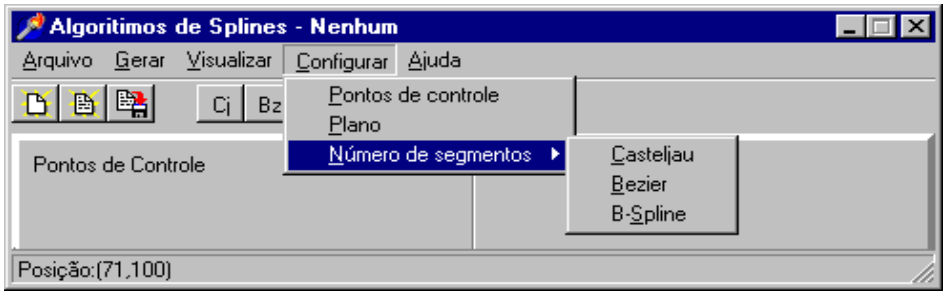


Figura 30: Menu Configurar

A opção **Configurar / Pontos de Controle**, abre um menu de Configuração dos Pontos de Controle (Figura 31) onde são configurados os pontos X e Y referentes as coordenadas dos Pontos de Controle. Uma vez que estes números do X e do Y são alterados, os pontos mudam de lugar, ocasionando uma mudança nas curvas geradas pelos algoritmos de *splines*. Tanto o número dado a X como Y são limitados ao plano, que tem como padrão de 100 x 100 pontos, sendo assim X ou Y vão de 0 a 100.

Assim como as mudanças feitas através deste menu refletem graficamente no plano dos Pontos de Controle, essas mudanças feitas através da movimentação dos Pontos de Controle através do mouse também se refletem em tempo real nos dados deste menu de Configuração dos Pontos de Controle.

Figura 31: Configuração dos Pontos de Controle

A opção **Configurar / Plano**, abre um menu de Configuração do Plano (Figura 32), que configura os limites dos pontos X e Y, ou seja, configura a quantidades de pontos que terá o plano onde a curva será formada. Estas quantidades são limitações em que a curva deve ser gerada.

A opção **Configurar / Número de Segmentos**, estabelece mais três opções(Figura 30) que consiste nos três algoritmos de geração de *splines*: ***Casteljau, Bezier e B-Spline***.

Figura 32: Configuração do Plano

Em **Configurar / Número de Segmentos / *Casteljau*,** abre um diálogo, onde será configurado o número de níveis, que estabelecerá um número de pontos a serem utilizados para a geração da curva pelo algoritmo de *Casteljau*. Os pontos aumentam proporcionalmente em relação a quantidade de níveis. Conforme é mudado o nível, automaticamente é atualizado ao lado o número de pontos que será usado(Figura 33).

Foi utilizado um limite de 14 níveis, chegando a 16384 pontos.

Figura 33: Configuração *Casteljau*

Em **Configurar / Número de Segmentos / *Bezier***, abre um diálogo (Figura 34), onde será configurado o número de pontos a serem utilizados para a geração da curva pelo algoritmo de *Bezier*. Foi utilizado um limite de 16000 pontos.

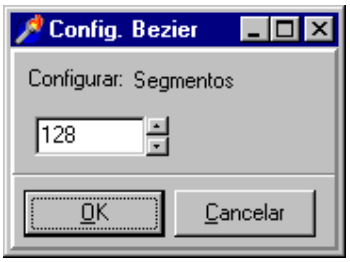


Figura 34: Configuração *Bezier*

Em **Configurar / Número de Segmentos / *B-Spline***, abre um diálogo (Figura 35), onde será configurado o número de pontos a serem utilizados para a geração da curva pelo algoritmo de *B-Spline*. Foi utilizado um limite de 16000 pontos, como no *B-Spline*.

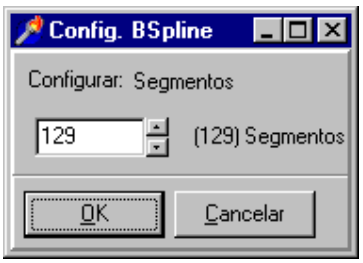


Figura 35: Configuração *B-Spline*

5 COMPARATIVOS

Para permitir a análise e comparação destes algoritmos(*Bezier*, *Casteljau* e *B-Spline*), pode-se adotar como fatores de avaliação e comparação: a ordem de complexidade, a necessidade de memória, a precisão, aspectos visuais e a facilidade de implementação.

5.1 ORDEM DE COMPLEXIDADE

A Ordem de Complexidade é um fator que está relacionado diretamente com a velocidade de execução do algoritmo, verificando-se o seu comportamento em relação ao tempo gasto para diferentes situações.

A análise dos algoritmos podem ser teórica ou experimental. A abordagem teórica, segundo Azeredo [AZE 96], permite uma análise independente da implementação, estimando-se analiticamente as quantidades de operações que o algoritmo deve efetuar diante de diferentes volumes e valores de entrada. O resultado da análise é uma função *f*, real, não-negativa, da variável $n \geq 0$ que representa a quantidade dos valores de entrada [AZE 96].

Já na análise experimental, segundo Azeredo [AZE 96], submete a implementação dos algoritmos a um variado e representativo conjunto de valores para permitir o monitoramento da sua execução. O objetivo é obter tabelas ou curvas que indicam a variação do seu comportamento em função de diferentes tipos de entrada [AZE 96]. Nesta análise deve-se levar em consideração a configuração do equipamento utilizado nos experimentos e não ter-se a execução de nenhuma aplicação em *background* (evitar problemas de compartilhamento de tempo).

Um algoritmo pode ser analisado levando-se em consideração o melhor caso, o pior caso e o caso médio. Na análise teórica, geralmente opta-se pelo estudo do pior caso levando-se em conta interesses práticos a fim de verificar o desempenho máximo exigido por um algoritmo, pois, mesmo que raro na prática, este caso ainda pode ocorrer [AZE 96]. Já na análise experimental, geralmente utilizam-se valores de entrada a fim de ter-se a média do custo de cada grupo. A fim de quantificar o crescimento das funções de ordem de complexidade dos algoritmos pode-se, por exemplo, utilizar conjuntos de 10, 100, 1000, 10.000 e 16.000 pontos como valores de entrada para a geração da curva.

5.1.1 ORDEM DE COMPLEXIDADE EXPERIMENTAL

Na ordem de complexidade experimental, utilizou-se como padrão para os três algoritmos, 4 pontos de controle com as mesmas coordenadas, com 9, 129, 1.025, 8.193 e 16.385 pontos e calculou-se o tempo para o processamento das curvas em cada algoritmo através do protótipo, repetindo-se o processo 20 vezes e retirando a média dos resultados.

O número de pontos usado na análise não são arredondados devido ao algoritmo de *Casteljau* que depende do número de níveis para gerar o número de pontos.

A análise de geração de curvas de *splines* forneceu os seguintes tempos (em ms.) mostrados na Tabela 1.

Tabela 1 – Ordem de Complexidade Experimental – Tempo em milissegundos.

Algoritmos	9 ptos	129 ptos	1.025 ptos.	8.193 ptos	16.385 ptos
1 – *Bezier*	0 ms	0 ms	60 ms	270 ms	490 ms
2 – *Casteljau*	0 ms	0 ms	50 ms	190 ms	330 ms
3 – *B-Spline*	0 ms	0 msl	60 ms	250 ms	440 ms

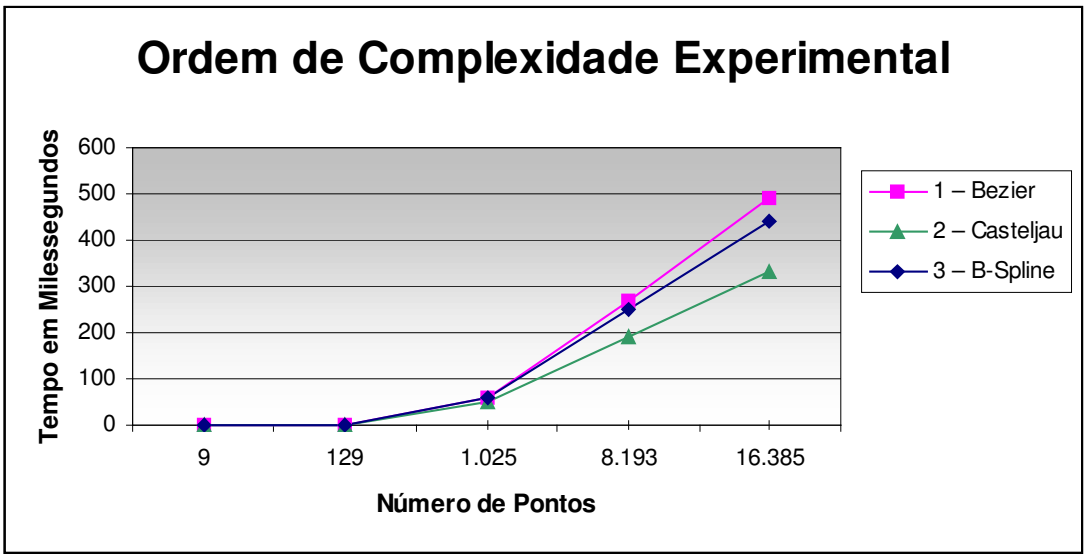


Figura 36: Ordem de Complexidade Experimental

Pode-se constatar pelo teste experimental realizado que até 129 pontos, os algoritmos não ultrapassam o tempo de 1 milissegundo para gerar a curva.

Constata-se também que o algoritmo *Bezier* é o que gasta um maior tempo, consequentemente é de uma ordem de complexidade maior.

O algoritmo de *B-Spline* gasta um tempo inferior ao algoritmo de *Bezier* mas maior que o *Casteljau*.

O algoritmo de *Casteljau* é o que gastou menos tempo para a realização do processamento da curva, dando a entender que é o melhor no sentido de tempo com a menor ordem de complexidade, segundo o teste experimental realizado.

5.2 NECESSIDADE DE MEMÓRIA

Nesta avaliação é verificado o espaço de memória exigido pela execução de um determinado procedimento, onde o tamanho e a quantidade de variáveis são os fatores a serem considerados. Em relação ao esforço computacional, a necessidade de memória não é fator crucial, a menos que esta necessidade seja elevada, pois dependendo da plataforma e/ou aplicação pode-se tornar inviável. Ao quantificar estes valores, não se leva em consideração variáveis comuns entre os algoritmos e tamanhos dos programas, mas sim os valores de entrada.

5.2.1 ANÁLISE TEÓRICA DA NECESSIDADE DE MEMÓRIA

A análise de memória necessária em relação ao aspecto teórico, considerou os espaços alocados em relação ao número de pontos dispersos. Neste estudo não foi considerada a memória adicional para gerenciar as alocações dinâmicas das listas utilizadas como a principal forma de armazenamento, já que todos os algoritmos possuem a mesma forma de gerenciamento.

Os custos verificados em relação à necessidade de memória na geração de curvas de *splines* encontram-se na tabela 2, onde consideram-se conjuntos de 0, 3, 5 e 9 pontos, para permitir quantificar as funções de custo da necessidade de memória.

O resultado encontrada na tabela 2, foi encontrado a partir de um estudo com as variáveis dos algoritmos, onde foi analisado as variáveis não comum aos três algoritmos e visto a necessidade de memória (em bytes) que necessita cada uma dessas variáveis incomuns no processo de execução dos algoritmos.

Tabela 2 – Necessidade de Memória – em bytes

Algoritmos	Função *f*(n)	0 ptos	3 ptos.	5 ptos	9 ptos
Bezier	42	42	42	42	42
Casteljau	8 + (pto*26)	8	86	138	242
B-Spline	184	184	184	184	184

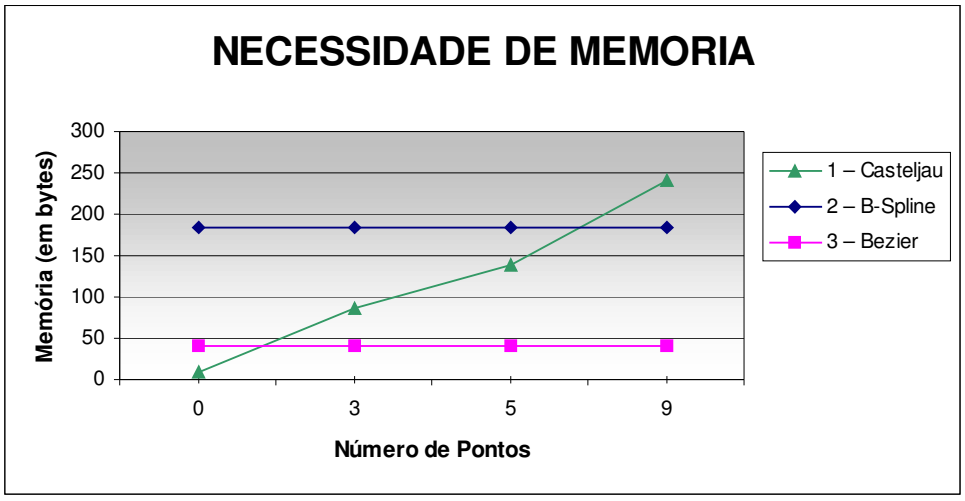


Figura 37: Gráfico de Necessidade de Memória.

Pode-se constatar, pela análise teórica da necessidade de memória, que os algoritmos *Bezier* e *B-Spline*, não variam sua necessidade de memória independente do número de pontos. Enquanto que o *Casteljau*, possui uma progressão, devido a existência de recursividade no algoritmo.

O algoritmo de *Bezier* manteve um gasto constante de 42 bytes, sem considerar o número de pontos existentes na curva gerada, e é o algoritmo que necessitou de menos memória para o processamento da curva.

O algoritmo de *B-Spline* também manteve-se constante, com 184 bytes de memória, independente do número de pontos, o qual consumiu um pouco mais que o *Bezier*.

Já o algoritmo de *Casteljau*, possui uma progressão dependendo do número de pontos da *spline*, tendo uma alta necessidade de memória para o processamento da curva, devido ao algoritmo que possui uma recursividade, acarretando a formação de novas variáveis a cada recursividade.

5.3 FATOR DE PRECISÃO

Considera-se quão mais próximo os valores de entrada podem demonstrar os resultados os quais estes valores propõem-se a representar. Em muitos casos não se conhece a função $z = f (x , y)$ que representa o comportamento destes valores e, dependendo do procedimento de coleta dos pontos, pode-se acumular uma grande quantidade de erros. Decorrente disto, uma prática bastante usual é a avaliação numérica das superfícies geradas por funções conhecidas (Fugura 27) [FEL 87] .

Neste caso, os algoritmos *Bezier e Casteljau,* apesar de serem diferentes nas suas definições matemáticas e sua implementação obtidos em relação aos algoritmos, mostram um resultado igual, ou seja, os mesmos pontos na *spline* (intervalos iguais), quando os dados de entrada são os mesmos. O algoritmo de *B-Spline* apresenta uma diferença na suavidade da curva, tendo os pontos mais próximos aos gerados pelos algoritmos *Bezier* e *Casteljau* à medida que se aproxima aos extremos desta *spline* (*Splines* Concavas).

Nos algoritmos estudados o fator de precisão foi pouco avaliado, justificando assim um estudo mais aprimorado em relação a esse aspecto.

5.4 ASPECTO VISUAL

Este fator é diretamente visível e é relacionado com a precisão da curva, ou seja, é analisado a suavidade da curva, que deve possuir um acabamento minucioso. Essa suavidade na geração da curva deve ser relacionado aos algoritmos vistos.

O protótipo provou que as curvas devem ser exatamente as mesmas, com as coordenadas dos pontos de formação da curva iguais na geração dos algoritmos de *Bezier e*

Casteljau, quando o número de pontos da curva forem iguais para os três intervalos e as coordenadas dos pontos de controle forem as mesmas.

O algoritmo de *B-Spline* mostrou uma suavidade boa, no entanto, a curva gerada, não é a mesma gerada por *Bezier* ou *Casteljau*, ou seja, a curva é menos acentuada, além de gerar uma curva, onde não teve como ponto de origem o primeiro ponto de controle assim como também não teve seu término no último ponto de controle, não passando pelos pontos extremos como aconteceu nos outros dois algoritmos (*Bezier*, *Casteljau*).

Inicialmente constatou-se uma diferença com a geração da curva pelo algoritmo de *Casteljau*, no entanto ao verificar que as curvas geradas pelos algoritmos *Bezier* e *Casteljau*, teoricamente deveriam ser exatamente iguais, verificou-se um erro no algoritmo. O algoritmo foi revisto e corrigido. Sem a constatação do fato de gerarem curvas iguais, não seria possível perceber o erro que foi descoberto visualmente no protótipo.

5.5 FACILIDADE DE IMPLEMENTAÇÃO

Este fator é julgado “altamente” subjetivo, podendo depender da familiaridade do implementador com algumas estruturas exigidas pelo método, da filosofia de implementação e do ambiente físico disponível para o mesmo. Mesmo tratando-se de características difíceis de serem analisadas, pode-se avaliar a facilidade de implementação considerando-se a complexidade das idéias envolvidas no método e a quantidade de código necessário para implementá-las [FEL 87].

Particularmente achou-se o algoritmo de *Casteljau* mais fácil de implementar, por se tratar de equações paramétricas apresentando um conteúdo menos abstrato. *Bezier*, no entanto foi trabalhado com pesos, onde foi exigido mais atenção e um maior tempo para a realização do mesmo. *B-Spline*, foi o que encontrou-se uma maior dificuldade, pois a bibliografia era pouca e menos clara, onde foi perdido um tempo muito grande para o estudo do algoritmo.

6 CONCLUSÕES

Este trabalho alcançou de forma satisfatória os objetivos propostos. O estudo foi feito, e com isso possibilitou a conversão de conhecimentos obtido, para o trabalho e para um programa de computador (protótipo), podendo assim representar o resultado dos algoritmos de *splines*.

A seguir apresentam-se as conclusões finais, resultados alcançados, dificuldades encontradas no decorrer do trabalho e sugestões para trabalhos futuros.

6.1 RESULTADOS ALCANÇADOS

O protótipo de software para o estudo dos métodos e algoritmos de *splines* cumpriu com os objetivos propostos e abordou as características especificadas, com o estudo dos algoritmos de *Bezier, Casteljau* e *B-Spline,* com a avaliação de sua utilização e geração de curvas de *splines* vistas com cada um destes três algoritmos separadamente, através do protótipo especificado e implementado.

O estudo bibliográfico, apesar da maioria estar no idioma inglês, juntamente com dados obtidos por periódicos e demais materiais utilizados, proporcionaram um aprendizado importante na área da Computação Gráfica, oportunizando um aprofundamento na área de geração de curvas. E o estudo da linguagem e ambiente de programação *Delphi*, proporcionou um aprendizado na área de programação enriquecendo o conhecimento na área computacional.

Através do protótipo conseguiu-se provar, comparar e analisar alguns aspectos de importância como: a ordem de complexidade dos algoritmos, a necessidade de memória para a geração das curvas, a precisão das curvas geradas, os aspectos visuais e a facilidade de implementação do processo.

Desses estudos feitos com o protótipo pode-se ver que:

A Ordem de Complexidade é um fator que está relacionado diretamente com a velocidade de execução do algoritmo, verificando-se o seu comportamento em relação ao tempo gasto para diferentes situações. No entanto, pode-se constatar pelo teste experimental

realizado que até 129 pontos, os algoritmos não ultrapassam o tempo de 1 milissegundo para gerar a curva, que o algoritmo *Bezier* é o que gasta um maior tempo, consequentemente é de uma ordem de complexidade maior. E que o algoritmo de *B-Spline* que gasta um tempo inferior ao algoritmo de *Bezier* mas maior que o *Casteljau*. Sendo que o algoritmo de *Casteljau*, é o que gastou menos tempo para a realização do processamento da curva, dando a entender que é o melhor no sentido de tempo com a menor ordem de complexidade, segundo o teste experimental realizado.

Na análise teórica da necessidade de memória, pode-se constatar que os algoritmos *Bezier* e *B-Spline*, não variam a necessidade de memória independendo do número de pontos. E o *Casteljau*, possui uma progressão, devida a existência de recursividade no algoritmo. O algoritmo de *Bezier* manteve um gasto constante de 42 bytes, sem considerar o número de pontos existente na curva gerada, e é o algoritmo que necessitou de menos memória para o processamento da curva. Já o algoritmo de *Casteljau*, possui uma progressão dependendo do número de pontos da *spline*, tendo uma alta necessidade de memória para o processamento da curva, devido ao algoritmo que possui uma recursividade, acarretando a formação de novas variáveis a cada recursividade.

No Fator de Precisão os algoritmos *Bezier e Casteljau*, apesar de serem diferentes nas suas definições matemáticas e sua implementação obtidos em relação aos algoritmos, mostram um resultado igual, ou seja, os mesmos pontos na *spline* (intervalos iguais), quando os dados de entrada são os mesmos e no *B-Spline* a curva muda, não chegando o ponto de origem ao primeiro ponto de controle e nem o ponto final ao ultimo ponto de controle.

Nos aspectos visuais o protótipo provou que as curvas devem ser exatamente as mesmas, com as coordenadas dos pontos de formação da curva iguais na geração dos algoritmos de *Bezier e Casteljau*, quando o número de pontos da curva forem iguais para os dois e as coordenadas dos pontos de controle forem as mesmas. O *B-Spline* se mostrou diferente dos outros dois algoritmos, mantendo curvas menos acentuadas.

Na implementação, o fator da dificuldade é julgado altamente subjetivo, podendo depender da familiaridade do implementador com algumas estruturas exigidas pelo método, da filosofia de implementação e do ambiente físico disponível para o mesmo. No entanto particularmente achou-se o algoritmo de *Casteljau*, mais fácil de implementar, por se tratar de

uma equações paramétricas apresentando um conteúdo menos abstrato, *Bezier*, no entanto foi trabalhado com pesos, onde foi exigido mais atenção e um maior tempo para a realização do mesmo e *B-Spline*, foi o que encontrou-se uma maior dificuldade, pois a bibliografia era pouca e menos clara, onde foi gasto um tempo muito grande para o estudo do algoritmo.

O objetivo do estudo de comparar os algoritmos foi alcançado, mostrando que nas diferenças de características entre o processamento das curvas pode-se encontrar o algoritmo mais apropriados a determinadas aplicação, considerando o uso de memória, o tempo de geração da curva, ou a facilidade de implementação, sabendo-se que a curva a ser gerada independe de algoritmo por apresentar resultados exatamente iguais nos algoritmos de Bezier e Casteljau, sendo que o B-Spline se alterou em relação aos outros dois, mostrando uma curva menos acentuada e um encurtamento na mesma.

6.2 DIFICULDADES ENCONTRADAS

Em relação a pesquisa bibliográfica, o tema abordado, os conceitos, as características e os algoritmos matemáticos de cada *spline* estudada foi encontrado em poucas fontes, atribuindo uma certa "barreira" na pesquisa, além do pouco encontrado ser de idioma inglês e utilizar de uma notação matemática, o qual dispendeu-se um tempo precioso para poder entendê-los.

A não familiaridade com a linguagem *Delphi* utilizada para a implementação dificultou um pouco no início, fazendo com que se gastasse um bom tempo de trabalho.

6.3 EXTENSÕES

Este trabalho estudou, analisou e mostrou a comparação entre três tipos de algoritmos de *spline*s, possibilitando melhorias e continuidade deste trabalho, podendo-se estender o assunto abordado em projetos futuros sugerindo-se:

a) estudo de outros algoritmos de *splines* (tais como *Hermite*, *Beta-Spline*, *Catmull*);

b) implementação com mais de 4 pontos de controle;

c) implementação com curvas tridimensionais;

d) estudo das *Splines* com valores baseados em superfícies reais podendo ser aprofundado o aspecto fator de precisão;

e) criação de um novo algoritmo;

f) confecção de um ambiente de edição de curvas e nós.

REFERÊNCIAS BIBLIOGRÁFICAS

[AZE96] AZEREDO, Paulo A**.. Métodos de Classificação de Dados e Análise de suas Complexidades**. Rio de Janeiro : Campus, 1996.

[BAR79] BARSKY, Brian A. **Computer Graphics and Geometric Modeling Using Beta-*splines***. Berlin-Alemanha : Springer-Verlag, 1992.

[BAR87] BARTELS, R.; BEATTY, B.; BARSKY, B. **An Introduction to *splines* for use in computer graphics & geometric modellig.** San Mateo : Morgan Kaufmann Publishers, 1987. 476p.

[DAM79] DAMASCENO JUNIOR, Americo. **Aprendendo Delphi Avançado**. São Paulo : Érica, 1995.

[FEL87] FELGUEIRAS, Carlos Alberto. **Desenvolvimento de um Sistema de modelagem Digital de Terreno para Microcomputadores**. São José dos Campos : INPE, 1987.

[FOL90] FOLEY, James D. **Computer graphics: principles and practice.** 2.ed. Washington : Wesley, 1990, 1175p.: il.

[MEL90] MELENDEZ, Rubem. **Prototipação de Sistemas de informações: fundamentos, técnicas e metodologias**. Rio de Janeiro : LTC, 1990.

[MOR85] MORTENSON, Michael. **Geometric modeling.** New York : John Wiley, 1985. 763p.

[OLI92] OLIVEIRA NETO, Manuel Menezes de. **Um algoritmo para interpolação de formas entre objetos modelados por superficies *spline*.** Porto Alegre : CPGCC da UFRGS, 1992. 136p.

[OSI98] OSIER, Dan. **Aprenda em 14 dias Delphi 3. Rio de Janeiro** : Campus, 1998.

[PER89] PERSIANO, Ronaldo Cesar Marinho. **Introdução a Computação Gráfica**. Rio de Janeiro : LTC-Livros Técnicos e Científicos Editora Ltda., 1989.

[PRE88] PREPARATA, Franco P. **Computational Geometry: na introduction; corrected and expanded second printing.** Nova York : Springer, 1988. 398p.: il.

[REI97] REIS, Dalton Solano. **Análise de Algoritmos de Triangularização para Geração de Grades Regulares Retangulares**. Dissertação. Porto Alegre-RS : UFRGS – CPGCC, 1997.

[THA85] THALMANN, Nadia; THALMANN, Daniel. **Computer Generated**. Tokyo : Springer-Verlag, 1985. 497p.