UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

GERAÇÃO DE B-SPLINES VIA FPGA

LUIZ MARCELO CHIESSE DA SILVA

**Tese apresentada à Escola de Engenharia
de São Carlos, como parte dos requisitos
para a obtenção do título de Doutor em
Ciências, Programa de Engenharia
Elétrica, na Área de Processamento de
Sinais e Instrumentação.**

Orientadora: Prof^a. Dr^a. Maria Stela Veludo de Paiva

**São Carlos
2012**

Trata-se da versão corrigida da tese. A versão original se encontra disponível na EESC/USP que aloja o Programa de Pós-Graduação em Engenharia Elétrica.

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Silva, Luiz Marcelo Chiesse da
S586g Geração de *B-splines* via *FPGA*. / Luiz Marcelo Chiesse da Silva ;
orientadora Maria Stela Veludo de Paiva. — São Carlos, 2012.

Tese (Doutorado - Programa de Pós-Graduação em Engenharia Elétrica e Área de
Concentração em Processamento de Sinais e Instrumentação) — Escola de Engenharia de São
Carlos da Universidade de São Paulo, 2012.

1. Reconstrução de superfícies. 2. *B-spline*. 3. *FPGA*. 4. Lógica reconfigurável. I.
Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **LUIZ MARCELO CHIESSE DA SILVA**.

Título da tese: "Geração de B-Splines via FPGA".

Data da defesa: 10/08/2012

Comissão Julgadora:

Resultado:

Profª. Drª. **Maria Stela Veludo de Paiva (Orientadora)**
(Escola de Engenharia de São Carlos/EESC)

APROVADO

Dr. **Lúcio André de Castro Jorge**
(Empresa Brasileira de Pesquisa Agropecuária/EMBRAPA)

Aprovado

Prof. Dr. **José Eduardo Cogo Castanho**
(Universidade Estadual Paulista "Júlio de Mesquita Filho"/UNESP-Bauru)

Aprovado

Prof. Associado **Luis Gustavo Nonato**
(Instituto de Ciências Matemáticas e de Computação/USP)

Aprovado

Prof. Dr. **Ricardo Menotti**
(Universidade Federal de São Carlos/UFSCar)

Aprovado

Coordenador do Programa de Pós-Graduação em Engenharia Elétrica e Presidente da
Comissão de Pós-Graduação:

Prof. Titular **Denis Vinicius Coury**

Agradecimentos

À orientadora Professora Doutora Maria Stela Veludo de Paiva, pelo seu empenho e comportamento profissional nos trabalhos de pesquisa tanto nos programas de graduação quanto pós-graduação, pelo excelente relacionamento referente aos alunos contemplados com sua orientação, assim como pela sua experiência e conhecimento na área.

Aos Professores do Departamento de Engenharia Elétrica da Escola de Engenharia de São Carlos, Universidade de São Paulo, quer seja pelas disciplinas ministradas durante o curso de pós-graduação ou pelo apoio técnico vista ao esclarecimento de dúvidas e disponibilização de infraestrutura física.

Aos amigos e professores do campus de Cornélio Procopio da UTFPR, que me auxiliaram para tornar possível a conclusão deste trabalho, cujos resultados gerados possam auxiliá-los assim como a futuros profissionais no desenvolvimento de trabalhos na área.

À minha mãe Marina, que há tempo se foi, mas nos deixou as boas lembranças, ensinamentos, educação e saudades.

Ao meu irmão Luis Felipe, apesar da pouca convivência no passar dos anos de sua vida, pela saudade e exemplo de empenho, dedicação e esforço apesar da doença e dificuldades que se acometiam no seu tempo de vida.

A minha esposa Cristiane, sempre ao meu lado com a compreensão e paciência devida ao tempo dispendido a este trabalho, assim como pelo auxílio nos momentos mais difíceis, pelos bons frutos que colheremos futuramente deste e de outros desafios conjuntos.

"Sessenta anos atrás eu sabia tudo, hoje sei que nada sei. A educação é o descobrimento progressivo da nossa ignorância."

William James Durant

Resumo

Silva, L. M. C. (2012) *Geração de B-Splines Via FPGA*.

Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2012.

As *b-splines* são utilizadas em sistemas *CAD/CAM/CAE* para representar e definir curvas e superfícies complexas, sendo adotada pelos principais padrões da computação gráfica devido a características como representação matemática de forma compacta, flexibilidade e transformações afins. Em sistemas de aquisição de dados 3D e sistemas *CAM-CNC* integrados, a utilização da *b-spline* na transferência de informações geométricas e na reconstrução da superfície de objetos resulta em um significativo incremento na eficiência do processo, geralmente implementado em sistemas embarcados. Nestes sistemas embarcados, integrados no auxílio a máquinas de manufatura, a utilização de *FPGAs* é incipiente, sem circuitos para *b-splines* disponibilizados em lógica reconfigurável de circuito aberto (*open core*), razão pela qual este projeto propõe o desenvolvimento de um circuito de geração *b-spline* aberto, em um sistema embarcado *FPGA*, utilizando algoritmos adaptados para os circuitos, elaborados em linguagem *Verilog HDL*, padronizada para a síntese de circuitos em lógica reconfigurável. Os circuitos foram desenvolvidos, utilizando-se um barramento de dados padronizado em circuito aberto, nas seguintes implementações para processamento paralelo das *b-splines*: o *BFEA*, o método baseado em funções base fixas, ambos projetados para circuitos integrados, e o *fast Cox-de Boor*, desenvolvido para *FPGAs*. Foram comparados o tempo de execução e o consumo de recursos disponíveis no *FPGA* utilizado, entre cada implementação. Os resultados evidenciaram que os circuitos de funções base fixas apresentaram o processamento mais rápido para a geração de *b-splines* em um *FPGA*, com um tempo de execução em média 20% menor em relação às outras implementações. Os circuitos *BFEA* apresentaram a menor utilização de elementos lógicos, em média 50% menor em relação aos outros circuitos implementados. O circuito *fast Cox-de Boor* apresentou a melhor escalabilidade, devido à modularidade da implementação, com tempos de execução similares aos circuitos de funções base fixas.

Palavras chave: Reconstrução de superfícies, *B-Spline*, *FPGA*, Lógica Reconfigurável.

Abstract

Silva, L. M. C. (2012) *B-Spline Generation Via FPGA*.

Thesis (Philosophical Doctor) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2012.

The b-splines are used in CAD/CAM/CAE systems to represent and define complex curves and surfaces, being adopted by the main computer graphics standards due to features like compact mathematic representation, flexibility and affine transformations. In 3D acquisition systems and integrated CAM-CNC systems, the use of the b-spline in the geometric information data transfer and in the object surface reconstruction results in a increase in the process efficiency, generally implemented in embedded systems. In these embedded systems, integrated in the aid to manufacturing machines, the use of FPGAs is incipient, without available b-splines open core circuits in reconfigurable logic, the reason why this project propose the development of a b-spline generation open core circuit, in a FPGA embedded system, using adaptated algorithms for the circuits, made in Verilog HDL language, standardized for the circuit synthesis in reconfigurable logic. The circuits were developed, using an open core standardized data bus, in the following implementations of b-spline parallel processing: the BFEA, fixed basis functions based method, both designed for integrated circuits, and the fast Cox-de Boor, developed for FPGAs. The execution time and available resource consumption in the FPGA were compared, between each implementation. The results show that the fixed basis functions circuits presented the fastest processing for the b-splines generation in a FPGA, with a 20% mean execution time reduction in relation to the other implementations. The BFEA circuits presented the lowest logic elements use, in mean 50% fewer in relation to the other implemented circuits. The fast Cox-De Boor circuit presented the best scalability, due to the implementation modularity, with execution times similar to the fixed basis functions circuits.

Keywords: Surface Reconstruction, B-Spline, FPGA, Reconfigurable Logic.

Lista de Figuras

Figura 1 – Trajetórias de uma ferramenta em um sistema de manufatura sem continuidade e com continuidade (curva), com os vetores tangentes estimados em cada ponto de passagem P . A trajetória contínua se aproxima mais da posição dos vetores tangentes.	33
Figura 2 – Roteador CNC marca Suda, modelo SD5040, implementado com um controlador <i>FPGA</i> compatível com código G. Fonte: (GLOBAL SUDA TONGXING, 2012).	34
Figura 3 – Conjunto de pontos inicial (primeira coluna à esquerda) e superfície final aproximada (última coluna à direita) obtida via superfícies de subdivisão. A figura 3c consiste em uma superfície de subdivisão aproximada a uma superfície <i>NURBS</i> . Fonte: (HOPPE ET AL., 1994).	42
Figura 4 - <i>CPG – DAG</i> para o processamento das funções base de uma <i>b-spline</i> de grau 3. Adaptado de: (GOPI, 1997).	43
Figura 5 – <i>BFEA</i> para uma <i>b-spline</i> de grau 3. Adaptado de (GOPI, 1997).	44
Figura 6 – Implementação do <i>BFEA</i> para curvas (quadrado: elemento de atraso). Fonte: (GOPI, 1997).	45
Figura 7 – Resultados do método de Gopi e Manohar em relação ao método de Megson (1991) para curvas. Fonte: (GOPI, 1997).	46
Figura 8– Resultados do método de Gopi e Manohar em relação ao método de Megson (1991) para superfícies. Fonte: (GOPI, 1997).	46
Figura 9 – Fluxo de dados típico da engenharia reversa. Fonte: (BESL, 1998).	47
Figura 10 – Resultado da transformada <i>wavelet</i> da força resultante no eixo da ferramenta de uma máquina fresadora, obtida por três <i>DSPs</i> em um <i>FPGA</i> . Fonte: (JÉSUS ET AL., 1994).	56
Figura 11 – Os pontos gerados cujos parâmetros se encontram dentro do mesmo intervalo de nós utilizam os mesmos pontos de controle e os mesmos valores de nós. Na figura acima, de P_0 a P_2 e de u_0 a u_1 para uma curva de grau 2.	57
Figura 12 – Desempenho computacional em ciclos do sinal de <i>clock</i> do algoritmo Cox-de Boor em uma arquitetura computacional <i>PC</i> , arquitetura <i>VLSI</i> (GOPI, 1997) e o método <i>fast</i> Cox-de Boor (YAU, 2006), de acordo com o número de eixos (coordenadas Euclidianas) e grau da curva <i>NURBS</i> . Fonte: (YAU ET AL., 2006).	58
Figura 13 – Tempo de processamento do algoritmo <i>Virtual Digitizing</i> obtido por um computador pessoal (<i>PC</i>) com (linha cheia) e sem (linha pontilhada) a aceleração via <i>FPGA</i> . Fonte: (JIMENO ET AL., 2006; 2007).	59
Figura 14 – Arquitetura de <i>hardware</i> de CNC reconfigurável baseada em <i>FPGA</i> proposta por DongDong et al. Adaptado de: (DONGDONG ET AL., 2007).	60
Figura 15 – Sistema <i>dual CPU</i> . Fonte: (XU ET AL., 2008).	60
Figura 16 – Arquitetura básica de um controlador CNC. Adaptado de: (DONG ET AL., 2009).	62
Figura 17 – Estrutura de uma fresa CNC reconfigurável proposta por Dong et al. (2009). Adaptado de: (DONG ET AL., 2009)	62

Figura 18 – Arquitetura de um sistema CNC totalmente digital baseado em um <i>FPGA</i> (3), com interface para sistema CNC externo (1), <i>USB</i> (2) e sistema de controle de potência (4) para servomecanismos (5). Fonte: (LI ET AL., 2009).	63
Figura 19 – Estrutura simplificada do sistema CNC de controle de 4 eixos. Adaptado de: (WANG ET AL., 2011).	64
Figura 20 – Estrutura simplificada do sistema CNC de controle de 8 eixos obtido pelo <i>upgrade</i> do sistema CNC reconfigurável de controle de 4 eixos. Adaptado de: (WANG ET AL., 2011).	64
Figura 21 – Perfis de taxa de alimentação da máquina-ferramenta para o sistema protótipo de arquitetura reconfigurável de <i>pipeline NC</i> . Fonte: (DONG E WANG, 2012).	66
Figura 22 – <i>Pipeline</i> gráfico implementado por Styles e Luk. Fonte: (STYLES E LUK, 2000).	69
Figura 23 - <i>CPG – Curve Point Generator</i> para a implementação <i>VLSI</i> em arquitetura paralela. Fonte: (OU ET AL., 2002).	71
Figura 24 – Resultados do algoritmo de Ou em relação aos algoritmos de de Boor, Gopi, Ying e Shiu. Fonte: (OU ET AL., 2002).	71
Figura 25 – Primeira implementação do sistema <i>DSP-FPGA</i> proposto por Zemcik (2002). Adaptado de: (ZEMCIK, 2002).	73
Figura 26 – Segunda implementação do sistema <i>DSP-FPGA</i> proposto por Zemcik (2002). Adaptado de: (ZEMCIK, 2002).	73
Figura 27 – Tempo de processamento para a biblioteca de funções em ponto flutuante em relação à aceleração via <i>hardware</i> (em nanosegundos). Fonte: (OUELLETTE E CONNORS, 2005).	77
Figura 28 – Ciclos do sinal de <i>clock</i> para a execução do algoritmo de eliminação Gaussiana, com o tamanho da entrada representando o tamanho do <i>grid</i> de computação para a <i>GPU</i> , <i>CPU</i> com processamento de um único <i>thread</i> , 4 <i>threads</i> simultâneos, e para o <i>FPGA</i> . Fonte: (CHE ET AL., 2008).	80
Figura 29 - Ciclos do sinal de <i>clock</i> para a execução do algoritmo Needleman-Wunsch, com o tamanho da entrada representando o tamanho do <i>grid</i> de computação para a <i>GPU</i> , <i>CPU</i> com processamento de um único <i>thread</i> , 4 <i>threads</i> simultâneos, e para o <i>FPGA</i> . Fonte: (CHE ET AL., 2008).	80
Figura 30 – Tamanho utilizado pela tecnologia do processo de fabricação dos dispositivos processadores gráficos da nVidia, <i>FPGAs</i> da Xilinx e a tendência de evolução da tecnologia de fabricação destes dispositivos fornecida pelo ITRS - International Technology Roadmap for Semiconductors. Fonte: (COPPE, 2008).	82
Figura 31 – Classes de arquitetura para a utilização da lógica reconfigurável em uma Arquitetura de Elemento de Multi-Processamento Homogêneo (<i>HoMPE</i>). Fonte: (COPPE, 2008).	83
Figura 32 – Resultados em ciclos de <i>clock</i> do processamento por um computador pessoal das operações de produto, soma, diferença e rotação 3D, da álgebra de Clifford; com o <i>core</i> coprocessador Clifford (CliffoSor) e via gerador de biblioteca GAIGEN. Fonte: (FRANCHINI ET AL., 2009).	85

Figura 33 – Desempenho computacional gráfico obtido pela biblioteca Mobile-GL para os processadores StrongARM de 200MHZ e PAX255 de 400MHZ para o preenchimento de pixels (a) e vértices (b). Fonte: (WOO ET AL., 2010).	85
Figura 34 – Curva <i>b-spline</i> (linha pontilhada) de 100 pontos a partir de 5 pontos de controle <i>P</i> (linha cheia) e pesos unitários (<i>b-spline</i>).	90
Figura 35 – Superfície <i>b-spline</i> de 10.000 pontos (100x100) a partir de 16 pontos de controle <i>P</i> e pesos unitários (<i>b-spline</i>).	91
Figura 36 – Algoritmo recursivo de Boor para curvas <i>b-splines</i> de grau 2 (3 camadas).	92
Figura 37 – Pontos de controle resultantes do algoritmo recursivo de Boor para a geração do ponto de parâmetro 0,6 na curva <i>b-spline</i>	93
Figura 38 – Algoritmo recursivo de Boor para superfícies, de topologia em pontos longitudinais e isoparamétricos.	94
Figura 39 – Pontos de controle de uma superfície <i>b-spline</i> no sentido longitudinal e isoparamétrico, formando um poliedro de controle de 16 pontos.	94
Figura 40 – Curvas isoparamétricas geradas pelo algoritmo de de Boor.	95
Figura 41 – Superfície final gerada pelas curvas isoparamétricas.	95
Figura 42 – Superfície final.	96
Figura 43 - Diagrama geral das partes constituintes do <i>Soc</i> de geração e visualização de curvas e superfícies <i>b-splines</i>	99
Figura 44 – Circuitos contadores do <i>clock</i> para cada <i>core</i> e de um sistema geral.	99
Figura 45 – Curvas de grau 1 (pontos na reta) e 2 no formato inteiro 16 <i>bits</i> sem normalização.	100
Figura 46 – Curvas de graus 1, 2 e 3 (ordenadas de baixo para cima) no fomato inteiro de 32 <i>bits</i> sem normalização.	100
Figura 47 – Placa DE2-70 utilizada para a implementação em lógica reconfigurável. Fonte: (TERASIC TECHNOLOGIES INC., 2007).	102
Figura 48 – Acesso aos <i>chips</i> de memórias SDRAM.	104
Figura 49 – Pinagem para as memórias SDRAM 0 e SDRAM 1. Fonte: (TERASIC, 2007).	105
Figura 50 – Signal Tap II Logic Analyzer. Fonte: (ALTERA, 2011).	107
Figura 51 – Geração do segmento de reta para <i>P</i> circuitos processadores (<i>k</i> - índice do processador). Adaptada de: (WRIGHT, 1990).	109
Figura 52 – Fluxograma das etapas de processamento de um circuito para a geração de um segmento de reta <i>k</i>	110
Figura 53 – Circunferência <i>NURBS</i> (linha cheia) de grau 2, formado no mínimo por 7 pontos de controle (<i>P</i>) com vetor de nós [0, 0, 0, 0.25, 0.5, 0.5, 0.75, 1,1,1] e vetor de pesos [1, 0.5, 0.5, 1, 0.5, 0.5, 1].	110
Figura 54 – Divisão de um arco de circunferência em <i>P</i> processos para o algoritmo de Wright. Adaptado de: (WRIGHT, 1990).	111
Figura 55 – Cálculo recursivo das funções base e circuito <i>BFEA</i> . Adaptado de: (GOPI, 1997).	113

Figura 56 – Circuito simplificado de uma célula de processamento da <i>BFEA</i> implementado em <i>FPGA</i> . Adaptado de: (GOPI, 1997).	114
Figura 57 – Exemplo de um circuito para o triângulo Cox-de Boor do cálculo de um ponto em uma curva <i>b-spline</i> de grau 2 (3 camadas).	121
Figura 58 – Circuito fast Cox-de Boor para uma curva <i>b-spline</i> de grau 2 (3 camadas). Os valores u e P não sofrem alterações se t_n estiver no intervalo $[u_r, u_{r+p}]$, sendo i o índice do intervalo do vetor de nós, k a ordem da curva (grau+1) e j a camada no circuito <i>fast</i> Cox-de Boor.	122
Figura 59 – Diagrama geral dos <i>cores</i> para o processamento de uma curva <i>b-spline</i> de “ n ” pontos de controle.	123
Figura 60 – Circuitos equivalentes para a geração de uma superfície. Os pontos gerados para um parâmetro são utilizados como pontos de controle para a outra direção isoparamétrica.	124
Figura 61 – Comunicação entre circuitos mestre e escravo na metodologia Wishbone.	126
Figura 62 – Sinal CYC_O de sincronia dos <i>cores</i>	126
Figura 63 – Interconexão em fluxo de dados com os <i>cores</i> mestre e escravo, e a nomenclatura padrão Wishbone para as entradas e saídas.	127
Figura 64 – Sistema de interconexão de barramento compartilhado.	128
Figura 65 – Combinação de tipos de interconexão Wishbone.	128
Figura 66 – Retas <i>b-splines</i> de graus 1, 2 e 3 (esquerda para direita) e pontos de controle (quadrados).	135
Figura 67 – Arco de circunferência <i>NURBS</i> de 90^0	136
Figura 68 – Circunferência <i>NURBS</i> de 128 pontos a partir de 7 pontos de controle (quadrados).	138
Figura 69 – Circunferência <i>NURBS</i> de 128 pontos obtido pela junção de arcos de 90^0	138
Figura 70 – Curva <i>b-spline</i> de 128 pontos em laço aberto de graus 1, 2 e 3 (sentido de fora para dentro).	139
Figura 71 – Curva <i>b-spline</i> fechada de 128 pontos em laço duplo (formato 8), graus 1, 2 e 3 (sentido de fora para dentro).	140
Figura 72 – <i>NURBS</i> com seções de segmento de reta e arco de circunferência.	141
Figura 73 – Curva <i>NURBS</i> de grau 3, de 128 pontos, com amplitude decrescente.	143
Figura 74 – Plano de 1.024 pontos formado por 16 retas <i>NURBS</i> de 64 pontos cada (63a) e plano com o peso do ponto de controle médio modificado em uma reta (63b).	144
Figura 75 – Plano retorcido de 1.024 pontos formado por 16 retas <i>NURBS</i> de 64 pontos cada.	145
Figura 76 – Conjunto de 36 retas <i>b-splines</i> (108 pontos de controle) no formato de um paralelepípedode 4.608 pontos.	146
Figura 77 – 50 segmentos de retas <i>b-splines</i> de 25 pontos cada dispostos com os pontos inicial e final coincidentes com os pontos de uma circunferência <i>NURBS</i>	147
Figura 78 – 10 circunferências <i>NURBS</i> de 64 pontos cada em planos paralelos.	148

Figura 79 - Curva <i>b-spline</i> de 512 pontos em formato de paralelepípedo helicoidal.....	149
Figura 80 Cilindro fechado de 512 pontos, com extremidades curvas formado por 24 curvas <i>NURBS</i> de 64 pontos cada.....	150
Figura 81 – Nuvem de pontos original (esquerda) de 20 curvas de 483 pontos cada, e curvas reconstruídas (direita) com 1.000 pontos cada (total de 20.000 pontos).	151
Figura 82 – Superfície <i>b-spline</i> de um plano de 2.048 pontos obtidos a partir de 16 pontos de controle (pontos quadrados).....	152
Figura 83 – Superfície <i>NURBS</i> de 1.024 pontos de um plano, com o peso de um ponto de controle reduzido em 10%.	153
Figura 84 – Superfície <i>b-spline</i> de 1.024 pontos gerada a partir de 16 pontos de controle, em que os pontos de controle finais possuem ordem inversa em relação aos pontos iniciais. ..	154
Figura 85 – Superfície <i>b-spline</i> de 1.024 pontos (50x50) e grau 2x2, no formato de um paralelepípedo, gerada a partir de 17x16 pontos de controle (quadrados).....	155
Figura 86 – Cilindro <i>NURBS</i> de 512 pontos (64x8) em formato cilíndrico.	156
Figura 87 – Superfície <i>NURBS</i> de 2.048 pontos gerados a partir de 56 pontos de controle (quadrados) em 8 seções transversais de 7 pontos no formato quadrado, dispostas em circunferência.	157
Figura 88 – Superfície <i>b-spline</i> de grau 2 (a) composta por 32.800 (410x40x2) pontos gerados a partir de 3.600 (450x20x2) pontos de controle (b) dispostos em 20 seções transversais de 900 pontos cada.	158
Figura 89 - – Exemplo de circuito para a habilitação dos <i>cores</i> de processamento dos pontos de uma <i>b-spline</i>	163
Figura 90 – Relação entre o número de pontos de controle e o número de parâmetros (pontos) processados. O número de parâmetros processador pode ser habilitado por um vetor (habilita) de entrada em cada <i>core</i> para uma arquitetura fixa.	163
Figura 91 – Barramento específico para a conexão entre as coordenadas dos pontos de controle e os <i>cores</i> de processamento de cada parâmetro.	164
Figura A1 – Barra sujeita a esforços mecânicos transversais. Adaptado de: (ROGERS E ADAMS, 1990).	178
Figura A2 – Exemplos reais de dispositivos que atuam como “ <i>ducks</i> ”. Adaptado de: (ROGERS E ADAMS, 1990).	178
Figura A3 – Deflexão em uma barra. Adaptado de: (ROGERS E ADAMS, 1990).	178
Figura A4 - Junção de polinômios.....	180
Figura A5 – Pedacos de polinômio. Adaptado de: (ROGERS E ADAMS, 1990).....	181
Figura A6 – Vetores posição e tangente iniciais, finais e na junção de uma <i>spline</i> . Fonte: (ROGERS E ADAMS, 1990).	182
Figura A7 - Funções <i>spline</i> de mistura. Adaptado de: (ROGERS E ADAMS, 1990).	185
Figura A8 - Curva <i>spline</i> obtida das funções de mistura. Adaptado de: (ROGERS E ADAMS, 1990).	185

Figura A9 – Curva <i>Spline</i> fechada com continuidade no ponto inicial p_0 , dada pela repetição dos pontos finais p_5 e p_6 pelos pontos iniciais p_0 e p_1 . Adaptado de: (ROGERS E ADAMS, 1990).	186
Figura A10 - Influência dos valores dos parâmetros na curva final obtida. Fonte: (ROGERS E ADAMS, 1990).	187
Figura A11 - Pulso triangular $N_{0,2}(t)$. Adaptado de: (ROGERS E ADAMS, 1990).	189
Figura A12 – Função base triangular $N_{0,3}(t)$. Adaptado de: (ROGERS E ADAMS, 1990).	190
Figura A13 – Funções base para a <i>spline</i> cúbica. Adaptado de: (ROGERS E ADAMS, 1990).	191
Figura B1 – Circuito principal para 4 <i>cores fast</i> Cox-de Boor.	195
Figura B2 – Circuito das três camadas para um <i>core fast</i> Cox-de Boor de grau 2.	196
Figura B3 – Entrada dos pontos de controle para o <i>core fast</i> Cox-de Boor.	196
Figura B4 – Circuito da combinação convexa do <i>core fast</i> Cox-de Boor.	197
Figura B5 – Circuito para o cálculo de <i>alfa</i> do algoritmo Cox-de Boor.	197
Figura C1 – Alocação matricial dos dados para o processamento do circuito Cox-de Boor.	201
Figura C2 – Organização dos tipos de memória na placa <i>GPU</i> .	202
Figura F1 – Sinais e conexões para o circuito de interface de vídeo <i>VGA</i> . Fonte: (TERASIC, 2007).	210
Figura F2 – Pipeline 3D mínimo.	212
Figura F3 – Mapeamento da memória de vídeo.	212
Figura F4 – Operações de geometria com os pontos das primitivas geométricas no estágio de geometria de um pipeline gráfico.	213
Figura F5 – Operações de transformação de vista dos gráficos.	213

Lista de Tabelas

Tabela 1 – Incremento no desempenho dos algoritmos de matrizes sistólicas em relação ao processamento sequencial. Fonte: (MEGSON, 1991).....	40
Tabela 2 – Parâmetros e resultados para o método de Hoppe para superfícies de subdivisão. Fonte: (HOPPE ET AL., 1994).....	42
Tabela 3 – Resultados para o método do <i>loop</i> duplo de geração de trajetórias de ferramentas em sistemas <i>CAM</i> , em relação aos métodos <i>CNC</i>	50
Tabela 4 – Desempenho dos métodos de interpolação <i>NURBS</i> . Fonte: (YAU, 2006).	58
Tabela 5 – Resultados para a implementação dos algoritmos gráficos no <i>FPGA</i> Xilinx XC4000. Fonte: (SINGH E BELLEC, 1994).	67
Tabela 6 – Resultados da implementação dos algoritmos gráficos para o <i>FPGA</i> Altera Flex8000. Fonte: SINGH E BELLEC, 1994).	68
Tabela 7 – Exemplos de tarefas dos três níveis de hierarquia da visão computacional. Fonte: (RATHA E JAIN, 1999).	68
Tabela 8 – Tempos de processamento computacional obtidos por Ratta e Jain para tarefas dos três níveis hierárquicos de visão computacional, em implementação sequencial em um Sun SPARCStation 20 e na plataforma <i>FPGA</i> Splash 2. Fonte: (RATHA E JAIN, 1999).	69
Tabela 9 – Tempo de processamento de taxa de <i>frames</i> por segundo para as plataformas de implementação. Fonte: (STYLES E LUK, 2000).	70
Tabela 10 – Resultados do multiplicador paralelo de matrizes implementado pelas funções das bibliotecas de ponto fixo (Celoxica)e CoreGen (Xilinx). Fonte: (BENSAALI ET AL., 2003).	74
Tabela 11 – Área, recursos utilizados e latência para as operações de multiplicação, divisão, adição e subtração nos formatos de ponto flutuante (PF) e números logarítmicos (NL), em simples e dupla precisão. Fonte: (HASELMAN ET AL., 2005).	75
Tabela 12 – Área, recursos necessários do <i>FPGA</i> e latência para a conversão entre os formatos de ponto flutuante (PF) e números logarítmicos (NL). Fonte: (HASELMAN ET AL., 2005).	75
Tabela 13 – Utilização de recursos de <i>hardware</i> para a <i>FSL – FPU</i> . Fonte: (OUELLETTE E CONNORS, 2005).	78
Tabela 14 – Potência consumida dos <i>cores FSL</i> e <i>FPU</i> para a Transformada Cosseno Discreta. Fonte: (OUELLETTE E CONNORS, 2005).	78
Tabela 15 – Comparação do tempo de processamento da transformação de rotação 3D (de um objeto de 2154 vértices), em <i>FPGA</i> e em uma placa gráfica Radeon FSC 32 MB. Fonte: (BENSAALI E AMIRA, 2006)	79
Tabela 16 – Comparação entre as arquiteturas <i>FPGA</i> e <i>multicore</i> para três modelos de simulação de eventos discretos de dinâmica molecular, para até 4 <i>threads</i> simultâneos (4 <i>cores</i>). Fonte: (DEAN ET AL., 2008).	84
Tabela 17 – Frequências de Operação Permitidas. Fonte: (ISSI, 2007).	106
Tabela 18 – Resultados das implementações para a geração de um segmento de reta de 128 pontos.	135

Tabela 19 – Resultados das implementações para a geração de um segmento de circunferência de 90^0 , formado por 64 pontos.	137
Tabela 20 – Resultados das implementações para a geração de uma circunferência <i>NURBS</i> de 128 pontos.	139
Tabela 21 – Resultados das implementações para a geração de uma curva <i>b-spline</i> em laço aberto.	140
Tabela 22 – Resultados das implementações para a geração da curva <i>b-spline</i> fechada em formato “8”.	141
Tabela 23– Resultados das implementações para a curva <i>NURBS</i> de 64 pontos com extremidades em arcos de circunferência.	142
Tabela 24– Resultados das implementações para curvas <i>NURBS</i> com ondulações decrescentes.	143
Tabela 25 – Resultados das implementações para a geração de um plano formado por retas <i>NURBS</i>	144
Tabela 26 – Resultados das implementações para a superfície do plano retorcido formado a partir de retas <i>b-splines</i>	145
Tabela 27 – Resultados das implementações para retas <i>b-splines</i> formando um paralelepípedo.	146
Tabela 28 – Resultados das implementações para retas <i>NURBS</i> formando um cilindro.	147
Tabela 29 – Resultados das implementações de circunferências <i>NURBS</i> em planos paralelos.	148
Tabela 30 – Resultados das implementações para a curva <i>b-spline</i> em formato de paralelepípedo helicoidal.	149
Tabela 31 – Resultados das implementações para um cilindro fechado de extremidades curvas formado por curvas <i>NURBS</i>	150
Tabela 32– Resultados das implementações para curvas <i>b-splines</i> da segmentação de imagens médicas.	151
Tabela 33 – Resultados das implementações para a geração de uma superfície <i>b-spline</i> de um plano.	153
Tabela 34 – Resultados das implementações para a geração de uma superfície <i>NURBS</i> de um plano com o peso associado ao ponto de controle médio reduzido em 10%.	154
Tabela 35 – Resultados das implementações para a superfície <i>b-spline</i> de um plano em curva de 180^0	154
Tabela 36 – Resultados das implementações para a superfície <i>b-spline</i> no formato de um paralelepípedo.	155
Tabela 37 – Resultados das implementações para a superfície <i>NURBS</i> de um cilindro.	156
Tabela 38– Resultados das implementações de um toróide formado por superfície <i>NURBS</i>	157
Tabela 39– Resultados das implementações da reconstrução da superfície de um joelho a partir das imagens de ressonância magnética.	158
Tabela 40 – Resultados do tempo de execução de <i>b-splines</i> para a implementação em <i>GPU</i> de curvas (2D e 3D) e superfícies.	159

Tabela 41 – Resumo dos melhores resultados obtidos para as três implementações em <i>FPGA</i> para os <i>cores</i> implementados (FBF-Funções Base Fixas).....	160
Tabela C1 – Características dos tipos de memória na placa <i>GPU</i>	202
Tabela E1 – Frequências de operação e relações de latência. Fonte: (ISSI, 2007).....	205

Lista de Siglas e Abreviaturas

ACM	<i>Acumulador Multiplicador</i>
ANSI	<i>American National Standard Institute</i>
API	<i>Application Programming Interface</i>
AS	<i>Active Serial</i>
ASIC	<i>Application Specific Integrated Circuit</i>
BGA	<i>Ball Grid Array</i>
BFEA	<i>Basis Functions Evaluation Array</i>
B-Splines	<i>Base Splines ou Bézier Splines</i>
CA	<i>Corrente Alternada</i>
CAD	<i>Computer Aided Design</i>
CAM	<i>Computer Aided Manufacturing</i>
CC	<i>Corrente Contínua</i>
CNC	<i>Controle Numérico Computadorizado</i>
CORDIC	<i>Coordinate Rotation Digital Computer</i>
CRC	<i>Cyclic Redundancy Check</i>
CUDA	<i>Computer Unified Device Architecture</i>
DDR	<i>Double Data Rate</i>
DAC	<i>Digital-Analogic Converter</i>
DES	<i>Data Encryption Standard</i>
DES	<i>Discrete Event Simulation</i>
DSP	<i>Digital Signal Processor</i>
EP	<i>Elemento de Processamento</i>
fMRI	<i>functional Magnetic Resonance Imaging</i>
FPU	<i>Floating Point Unit</i>
FSL	<i>Fast Simplex Link</i>
FPGA	<i>Field Programmable Gate Array</i>
GP-GPU	<i>General Purpose GPU</i>
GPU	<i>Graphics Processing Unit</i>
IDE	<i>Integrated Development Environment</i>
IGES	<i>Initial Graphics Exchange Specification</i>
JTAG	<i>Join Test Action Group</i>
HoMPE	<i>Homogeneous Multi-Processing Element</i>
IP-core	<i>Intellectual property core</i>

IrDA	<i>Infrared Data Association</i>
ITRS	<i>International Technology Roadmap for Semiconductors</i>
HDL	<i>Hardware Description Language</i>
Le	<i>Logic element</i>
LUT	<i>Look-Up Table</i>
NURBS	<i>Non Uniform Rational B-Spline</i>
PCI	<i>Peripheral Component Interconnect</i>
PET	<i>Positron Emission Tomography</i>
PO	<i>Physical Optics</i>
Phigs	<i>Programmer's Hierarchical Interactive Graphics System</i>
Pixel	<i>Picture element</i>
PLL	<i>Phase Locked Loop</i>
PWM	<i>Pulse Width Modulation</i>
RAM	<i>Random Access Memory</i>
REDA	<i>Reconfigurable Engine for Data Access</i>
RGB	<i>Red-Green-Blue color system</i>
RTL	<i>Register Transfer Level</i>
SD Card	<i>Secure Digital Card</i>
SDRAM	<i>Synchronous Dynamic RAM</i>
Soc	<i>System on a Chip</i>
Spline	<i>S-polynomial line</i>
SSRAM	<i>Synchronous Static RAM</i>
STEP	<i>Standard for the Exchange of Product model data</i>
T&L	<i>Transform and Lighting</i>
TCD	<i>Transformada Cosseno Discreta</i>
TCL	<i>Tool Command Language</i>
ULA	<i>Unidade Lógica e Aritmética</i>
USB	<i>Universal Serial Bus</i>
VGA	<i>Video Graphics Array</i>
VHDL	<i>Very High Speed Integrated Circuits Hardware Description Language</i>
VLSI	<i>Very Large Scale Integration</i>

Sumário

1. Introdução	31
1.1. Considerações Iniciais	31
1.2. Justificativas	32
1.3. Objetivos.....	35
1.4. Contribuições.....	36
1.5. Estrutura do Trabalho	37
2. Trabalhos Relacionados.....	39
2.1. Considerações Iniciais	39
2.2. <i>B-Splines</i>	39
2.3. <i>FPGAS</i>	51
2.4. Considerações Finais	86
3. Fundamentação Teórica.....	89
3.1. Considerações Iniciais	89
3.2. Conceitos sobre <i>B-Splines</i>	89
3.3. Algoritmo Cox-de Boor.....	91
4. Materiais e Métodos	97
4.1. Considerações Iniciais	97
4.2. Metodologia.....	97
4.3. Materiais	101
4.4. Memória <i>SDRAM</i>	103
4.5. Comunicação de Dados	106
4.6. Primitivas Gráficas – Segmento de Reta	108
4.7. Primitivas Gráficas – Circunferência e Esfera	109
4.8. Circuitos <i>B-Splines</i>	112
4.9. Barramento Wishbone	124
4.10. Implementação <i>GPU</i>	129
5. Resultados e Discussões	131
5.1. Considerações Iniciais	131
5.2. Descrição dos Conjuntos de Pontos Teste.....	131
5.3. Curvas.....	134
5.4. Superfícies	152
5.5. Resultados - <i>GPU</i>	159
5.6. Considerações Finais	160
6. Conclusão	165
6.1. Artigos Publicados.....	167

6.2. Sugestões para Trabalhos Futuros.....	167
7. Referências Bibliográficas	169
APÊNDICE A – Introdução às <i>Splines</i>	177
A.1. <i>Splines</i>	177
A.2. <i>B-splines</i>	188
A.3. Superfícies NURBS	191
APÊNDICE B – Diagramas RTL	195
APÊNDICE C – Dados Técnicos e Implementação GPU	199
APÊNDICE D – Dados Técnicos – Sistema <i>FPGA</i>	203
D.1. Placa DE2-70 (TERASIC, 2007):	203
D.2. <i>FPGA</i> Altera Cyclone II EP2C70F896C6N (ALTERA, 2008):	204
D.3. <i>SDRAM</i> (ALTERA, 2008):	204
D.4. <i>VGA DAC</i> (ADV123, 2008):	204
APÊNDICE E – Operação da <i>SDRAM</i>	205
APÊNDICE F – Sistema de Visualização.....	209
F.1. Interface de Vídeo.....	209
F.2. Sistema de Visualização	211

1. Introdução

1.1. Considerações Iniciais

As *b-splines*, consistem em um tipo particular de *splines*, funções matemáticas polinomiais parametrizáveis, utilizadas principalmente para representar formas geométricas genéricas de curvas e superfícies suaves. Ao utilizar uma família de funções para a ponderação dos termos desta curva polinomial, denominada funções base, originam-se as *b-splines*, que apresentam, em relação às *splines*, melhor controle local e possibilidade da mudança no grau da curva sem a modificação no número de pontos de controle.

A representação fiel de curvas na forma de seções cônicas, como círculos e elipses, necessita de uma função matemática racional, as *b-splines* racionais, as quais podem representar todas as formas geradas pelas *b-splines*, incluindo também as seções cônicas (PIEGL, 1991). As *NURBS-Non Uniform Rational B-Splines*, são *b-splines* racionais em que os nós de uma curva ou superfície podem ser distribuídos, concentrando-se de maneira a se adequar ao nível de curvatura local, característica denominada de não uniformidade (na distribuição dos nós).

A flexibilidade das *b-splines* em representar quase todos os tipos de curva ou superfície, principalmente os formatos representados por outras primitivas geométricas (arcos de circunferência e retas), as tornam ideais para a modelagem geométrica computacional de superfícies complexas, como a superfície de tecidos, órgãos humanos (UNSER, 2002), estruturas geográficas (superfícies de terrenos rochosos ou florestas) e entes biológicos, como plantas (ANDERSON E HINES, 2000). Devido a estas características, as *b-splines racionais* estão presentes como padrão em sistemas de modelagem geométrica computacional, em sistemas que necessitam da troca de dados geométricos de modelos computacionais, utilizando muitas vezes sistemas embarcados programáveis, principalmente na área de *CAM-Computer Aided Manufacturing* e *CNC-Controle Numérico Computadorizado* (TSAI ET AL., 2003), onde o processamento de dados utilizando lógica configurável via *FPGAs* mostrou-se promissor (YAU ET AL., 2006).

Os circuitos sintetizados nos *FPGAs* apresentam um tempo de execução superior em relação a *ASICs – Application Specific Integrated Circuits* para a geração de *b-splines*, mas

oferecem uma grande flexibilidade na criação da arquitetura de sistemas de computação (HAUCK E DEHON, 2008), possibilitando o desenvolvimento de circuitos de processamento distintos no mesmo *FPGA*, como a sintetização de circuitos para a geração de dados geométricos de superfícies, e circuitos para o controle de uma máquina-ferramenta em um mesmo *chip*, constituindo um *Soc – System on a Chip* (CHANG ET AL., 2002).

1.2. Justificativas

Na troca de informações digitais entre sistemas gráficos, as *b-splines* são utilizadas como formato padrão na representação de curvas e superfícies de formato genérico, com propriedades que facilitam a modificação de seus dados geométricos de acordo com a aplicação. No caso de mudança de escala em sistemas que trabalham com *buffer* de memória de vídeo de tamanho fixo, as *b-splines* podem proporcionar, em relação a outras representações de curvas e superfícies, o aumento ou redução na resolução gráfica de acordo com o sistema, possibilitando o ajuste a *displays* de equipamentos portáteis, como telefones celulares (WOO ET AL., 2010), até equipamentos de visualização no padrão *high definition*. Além da computação gráfica, os circuitos de controle de equipamentos de manufatura da indústria de moldes para injeção e extrusão, incorporam as *b-splines* em sistemas de manufatura integrados (ORENSTEIN, 2002; TSAI ET AL., 2003; YANG ET AL., 2010), como *CAM-CNC*-máquinas de manufatura, comprovando-se uma redução significativa no tempo do ciclo de produção de moldes com formato composto por superfícies complexas, fornecendo melhor precisão no acabamento da superfície final. Em comparação com sistemas *CNC* que utilizam a interpolação linear convencional, as *b-splines* possibilitam a utilização de um número menor de blocos de código de controle numérico para uma trajetória mais precisa da máquina-ferramenta, além de movimentos em trajetórias de curvas suaves, necessárias para a manufatura de superfícies com maior precisão. As superfícies *b-splines* possibilitam gerar uma aceleração mais suave em cada eixo da máquina, tornando o movimento mais suave em relação a um movimento linear. Como resultado, a dinâmica da máquina e a suavização da superfície de corte podem ser significativamente incrementadas (YANG ET AL., 2010). Como exemplo, a Figura 1 ilustra os vetores tangentes estimados aos pontos interpolados P_n da trajetória de uma máquina-ferramenta via *b-splines* e via interpolação linear.

Figura 1 – Trajetórias de uma ferramenta em um sistema de manufatura sem continuidade e com continuidade (curva), com os vetores tangentes estimados em cada ponto de passagem P . A trajetória contínua se aproxima mais da posição dos vetores tangentes.

Na área de computação gráfica, os fabricantes de *hardware* desenvolvem circuitos dedicados a executar primitivas gráficas, sendo que muitos destes sistemas possuem circuitos especializados em funções gráficas 3D (COPPE, 2008), geralmente integrado ao circuito de processamento, principalmente em equipamentos portáteis destinados ao consumidor final como *palmtops*, *smartphones* e *DVD players*. Em relação ao desenvolvimento de aplicações de computação gráfica em *FPGAs*, os trabalhos são incipientes (KNUTSSON, 2005), excetuando-se algumas tentativas de comparação em relação a outras tecnologias (SINGH E BELLEC, 1994). Porém, muitos sistemas embarcados, como os utilizados em sistemas *CAM*, não possuem *hardware* específico que atenda às especificações das *b-splines* na geração e troca de dados de superfícies com outros sistemas (WANG ET AL., 2008). Nos sistemas de manufatura, os dados relativos às informações geométricas para protótipos de peças ou moldes devem ser tratados com prioridade na transferência de dados entre os subsistemas, tornando importante sua formatação e o tempo de processamento em trajetórias para a ferramenta de manufatura (HERWECK, 2009).

Os *FPGAs* constituem-se na tecnologia base para sistemas embarcados, que permite a montagem de uma arquitetura de processamento definida pelo projetista para uma determinada aplicação, possibilitando a criação dos sistemas CNC reconfiguráveis (WANG ET AL., 2011), aplicados em equipamentos e sistemas comerciais, como fresas. Aplicações, como o controle digital da aceleração de motores (DONGDONG ET AL., 2007), que resulta

em maior precisão do processo; a determinação da quebra de ferramenta pela estimativa da força resultante aplicada, medida indiretamente pela corrente fornecida pelos *drives* dos motores (JESÚS ET AL., 2004); ou a reamostragem de pontos da trajetória da ferramenta (DONG ET AL., 2009), são exemplos importantes do aumento da eficiência em processos de manufatura controlados computacionalmente via *FPGA*. Na Figura 2 é exibido o roteador CNC de três eixos, modelo SD 5040, da Global SUDA Tongxing Development Ltda. (2012), fabricante de máquinas de manufatura que utilizam somente controladores *FPGA* (controlador remoto, no modelo da Figura 2).

Figura 2 – Roteador CNC marca Suda, modelo SD5040, implementado com um controlador *FPGA* compatível com código G. Fonte: (GLOBAL SUDA TONGXING, 2012).

Os *FPGAs*, pela sua constituição básica em elementos lógicos e circuitos de conexão, permitem a montagem de diversos sistemas digitais em um único *chip*, apresentando atualmente o maior nível de integração entre os circuitos eletrônicos integrados atuais, com um baixo consumo de energia e ferramentas integradas e padronizadas de projeto de circuitos. Sistemas baseados em *FPGAs* são altamente escaláveis, permitindo a criação de sistemas com *arrays* de *FPGAs*, sua utilização como aceleradores em sistemas de computação ou em conjunto com outras tecnologias em computação heterogênea. A implementação de um *Soc – System on a chip* permite que sistemas digitais de processamento, de aplicações distintas, possam ser sintetizados em um único *chip*.

Pelas razões mencionadas anteriormente, o presente trabalho contribui com a área de sistemas embarcados baseados em *FPGA*, propondo um sistema implementado em lógica

configurável em *FPGA*, para a geração de curvas e superfícies *b-splines*, via sintetização de circuitos baseados em algoritmos e *ASICs b-splines* modificados para a lógica reconfigurável.

Além da área de sistemas de manufatura reconfiguráveis, o presente trabalho também pode ser aplicado na área de computação gráfica, onde é comum o uso de matrizes *FPGA* (KUON, 2007), *GPUs* (NVIDIA, 2009; STOKES, 2007), ou processadores específicos (SINGH E BELLEC, 1994) em substituição aos sistemas de processamento paralelo envolvendo diversos microprocessadores (MEGSON, 1991). As tendências atuais consistem na utilização de circuitos de tecnologias integradas, envolvendo lógica configurável, núcleos matemáticos de *DSP* e multiprocessamento (XU ET AL., 2008; ZEMCIK, 2002), interligados por barramentos lógicos de alta capacidade.

A lógica reconfigurável possui duas linguagens padronizadas para a descrição de hardware, a *Verilog* e a *VHDL* (IEEE, 2009), além de ferramentas de síntese, depuração e simulação, que estão em constante evolução. A descrição de *hardware* através destas linguagens possibilita a disponibilização dos circuitos desenvolvidos na forma de *open cores* (circuitos abertos), similar aos códigos *open source* (código aberto), em que o circuito descrito pela linguagem pode ser utilizado, sem custo, por desenvolvedores de outros sistemas.

1.3. Objetivos

Este trabalho descreve o projeto de um *Soc* para a geração de *b-splines*, em um sistema embarcado baseado em uma plataforma de lógica reconfigurável em um *FPGA*, utilizando o barramento *Wishbone* (OPENCORES ORGANIZATION, 2010) para a interligação dos *cores*. Este sistema inclui a comparação entre os três principais métodos para a geração de curvas e superfícies *b-splines* 2D e 3D em sistemas de processamento paralelo: o circuito para a geração de curvas e superfícies *b-splines*, baseado no circuito de Gopi e Manohar (1997), denominado neste trabalho de circuito *BFEA* (*Basis Functions Evaluation Array*), circuito baseado nas combinações fixas de funções *b-splines*, proposto por Ou et al. (2002), denominado neste projeto de circuito das funções base fixas, e o circuito baseado no método *fast Cox-de Boor*, proposto por Yau et al. (2006).

Os dois primeiros circuitos, *BFEA* e funções base fixas, são adaptados de projetos em *ASICs* para a lógica reconfigurável. O circuito *fast Cox-de Boor*, apesar de ser projetado para *FPGAs*, não foi disponibilizado originalmente na forma de circuito aberto e não possui um

barramento de dados específico para sua implementação. Neste trabalho, o circuito *fast* Cox-de Boor é projetado na forma modular, utilizando o barramento Wishbone para a interconexão entre os módulos.

Devido à falta de precisão na representação de seções cônicas e à falta de eficiência computacional na representação de segmentos de reta, o objetivo adicional deste projeto é a implementação de circuitos para a geração de segmentos de retas e circunferências baseados nos algoritmos paralelos de Wright (1990). O circuito para o barramento de dados e sinais de controle dos *cores* é baseado na metodologia Wishbone para barramentos, que possui uma topologia aberta padrão, disponível pelo *opencores.org*.

1.4. Contribuições

Uma das contribuições do presente projeto é a implementação dos métodos *BFEA* e de funções base fixas em *FPGA*, com os dados no fomato de ponto fixo de 32 *bits*.

O método *BFEA* (GOPI E MANOHAR, 1997) foi desenvolvido para *ASICs* – Circuitos Integrados de Aplicação Específica, utilizando uma estrutura de matrizes sistólicas que processam os pontos de controle de uma *b-spline* de forma sequencial. Neste trabalho, esta implementação foi adaptada para um sistema de lógica reconfigurável, utilizando um barramento padronizado para os dados.

O método de Ou et al. (2002) se baseia no processamento dos valores fixos das funções base, para a geração de uma *NURBS* em arquitetura *VLSI*, utilizando o Visual-C++ e OpenGL para testes, com os dados no formato ponto fixo de 16 *bits*. O projeto deste trabalho inclui o método de Ou, implementado em um *FPGA*, utilizando o formato dos dados em ponto fixo de 32 *bits* e o barramento padrão Wishbone.

Este trabalho também contribui com a implementação do método *fast* Cox-de Boor (2006), cujo circuito é implementado de forma modular, incluindo o barramento Wishbone. No trabalho original não é mencionada a utilização de um barramento de dados, assim como detalhes de constituição do circuito.

As seguintes contribuições adicionais podem ser mencionadas:

- implementação de um circuito adaptado para acesso à uma memória externa;
- a implementação de dois *cores* para a geração de segmento de retas e arcos de circunferências, baseados nos algoritmos paralelos de Wright (1990);

- a disponibilização dos códigos dos circuitos em Verilog *HDL* na forma de *open cores*, em um repositório no *opencores.org*, um domínio da internet específico para a publicação de projetos de circuitos disponíveis e reutilizáveis gratuitamente, baseado na *GPL –General Public License*, disponibilizada para *softwares*;
- resultados obtidos com a avaliação entre os tempos de execução e o número de elementos lógicos utilizados de cada implementação;
- implementação do método *fast Cox-de Boor* em uma *GPU*.

1.5. Estrutura do Trabalho

Este trabalho de tese encontra-se dividido em 6 capítulos e 6 apêndices. Os capítulos estão descritos resumidamente abaixo, assim como os apêndices e anexos.

O capítulo 1 trata de uma breve introdução sobre o trabalho, sobre as *b-splines* e *FPGAs*, assim como os objetivos, a justificativa, e a estrutura deste trabalho.

O capítulo 2 trata dos trabalhos relacionados sobre as *b-splines*, *FPGAs*, os métodos e projetos existentes sobre sua utilização em sistemas de manufatura, na computação gráfica, e em aplicações gerais, apresentando os métodos mais usuais de implementação em processamento paralelo e em lógica reconfigurável.

O capítulo 3 apresenta uma breve fundamentação matemática e algorítmica das *b-splines*, os dados necessários à sua geração, assim como a influência destes dados nas características e propriedades das curvas e superfícies geradas.

O capítulo 4 descreve os materiais e métodos utilizados na implementação das partes componentes do sistema em lógica reconfigurável, a implementação dos métodos comparados, e a inclusão do barramento lógico padrão nos circuitos desenvolvidos.

O capítulo 5 exhibe os resultados e discussões obtidos das implementações efetuadas, em termos do número de recursos de *hardware* utilizado e do tempo de execução do sistema.

O capítulo 6 apresenta as conclusões finais com base nos resultados obtidos, trabalhos em desenvolvimento baseados neste projeto, propostas de melhoria, e trabalhos futuros.

Os apêndices estão divididos em 6 partes: introdução às *splines*, diagramas *RTL* dos circuitos sintetizados para os métodos comparados, dados técnicos e metodologia da implementação de geração de *b-splines* em uma *GPU*, dados técnicos do material utilizado no sistema *FPGA*, operação da memória *SDRAM* utilizada no sistema e o sistema básico de visualização desenvolvido.

2. Trabalhos Relacionados

2.1. Considerações Iniciais

Neste capítulo são apresentados trabalhos relevantes aos temas deste projeto de pesquisa, de acordo com a área ao qual se propõe: *b-splines*, *FPGAs* (lógica reconfigurável) e aplicações envolvendo tecnologias em *splines*, dispositivos de lógica reconfigurável, ou tecnologias mistas, como processadores de propósitos gerais, *DSPs*, *GPUs* e *FPGAs*.

As *b-splines* são importantes na área da computação gráfica, com muitas técnicas e procedimentos que podem ser utilizados também na área de manufatura. Trabalhos de pesquisa que utilizam dispositivos de lógica reconfigurável, como opção para aceleradores de processamento com outras tecnologias, podem ser implementados em sistemas embarcados nas áreas de computação gráfica e sistemas de manufatura.

2.2. B-Splines

A formulação das *b-splines* foi estabelecida por Boor em 1972, mas as curvas de Bézier, segundo Yamaguchi (1988), podem ser consideradas as antecessoras das *splines*; as primeiras (curvas de Bézier) foram originadas como um modelo matemático das curvas provenientes da curva francesa (FARIN, 1997), e a segunda pela modelagem de sistemas físicos.

A base matemática das *b-splines* mantém-se praticamente inalterada desde 1983, sendo provenientes da modelagem da deformação de barras longitudinais de madeira ou plástico reforçado (ROGERS et al., 1990) por funções polinomiais. Além da representação de curvas, um tipo específico de *b-splines*, as *NURBS – Non Uniform Rational B-Splines*, podem representar com precisão segmentos de retas e arcos de circunferência, constituindo assim um formato único de representação matemática de primitivas geométricas, tanto em relação aos algoritmos convencionais como para algoritmos de sistemas de processamento paralelo na representação computacional destes tipos de primitivas geométricas.

Pang (1990) e Wright (1990) apresentaram métodos de geração das primitivas gráficas de segmentos de reta e circunferência em sistemas de processamento paralelo, desenvolvendo

algoritmos similares aos de Bresenham (1968) para a geração dos pontos de retas e de circunferências em sistemas gráficos digitais, em que a posição do próximo ponto gerado é dependente somente de uma variável. Os algoritmos descritos por estes sistemas podem ser adaptados para outros sistemas de processamento paralelo, como *multithreading* e sistemas de lógica reconfigurável.

O trabalho de Piegl (1991) lista detalhadamente as propriedades, vantagens e desvantagens características das *b-splines* e das *NURBS*, como a representação unificada de curvas, incluindo segmentos de reta e seções cônicas (circunferências) com precisão. Dentre as características das *NURBS*, além da representação unificada de formas geométricas, é possível citar a representação precisa de formas analíticas e formas livres, manipulação da forma geométrica via pontos de controle e pesos, disponibilidade de ferramentas geométricas, invariância sobre mudança de escala, rotação, translação, corte, e nas projeções perspectiva e paralela e disponibilidade de biblioteca de algoritmos testados. Por estas razões as *NURBS* estão incluídas em padrões de troca de arquivos de informação geométrica.

Megson (1991) desenvolveu algoritmos de processamento paralelo em matrizes sistólicas para a construção das funções base de curvas e superfícies *b-splines*, com resultados até 7.000 vezes mais rápidos que o processamento sequencial equivalente. Os resultados deste algoritmo para a geração de um ponto de uma superfície *b-spline* são demonstrados na Tabela 1, assim como o ganho na velocidade de processamento em relação ao algoritmo sequencial em uma mesma plataforma computacional, para superfícies geradas a partir de uma malha de controle com números de pontos “m” e “n” nas direções longitudinal e transversal.

Tabela 1 – Incremento no desempenho dos algoritmos de matrizes sistólicas em relação ao processamento sequencial.
Fonte: (MEGSON, 1991).

$\begin{smallmatrix} m \\ n \end{smallmatrix}$	5	10	15	20	25	30
5	20,2	19,9	19,7	19,5	19,4	19,3
10	27,3	27,9	28,5	28,9	29,3	29,5
15	32,7	34,3	35,6	36,7	37,6	38,3
20	37,0	39,5	41,5	43,2	44,7	45,9
25	40,5	43,7	46,5	48,6	50,8	52,5
30	43,4	47,3	50,7	53,6	56,1	58,3

Mann et al. (1992) apresentaram uma análise de métodos de interpolação local dos vértices de um poliedro por uma superfície suave, via interpolantes paramétricos triangulares,

i.e., que constroem “retalhos” de superfície para cada face triangular de um poliedro, a partir de curvas de contorno ao redor do triângulo da face do poliedro. Os métodos de Chiyokura-Kimura, Herron, Jensen, Piper, Nielson, *triangular Gregory patches* e Herron são analisados, além de mencionar os métodos de Farin, Gregory-Charrot e Hagen-Pottman, não incluídos na análise por apresentarem características que geram não uniformidades na superfície final. Esta análise leva em conta somente o aspecto visual em relação aos conjuntos de pontos utilizados, efetuada por imagens sombreadas e curvaturas Gaussianas nos interpoladores para a verificação de defeitos na interpolação dos pontos. Como resultado, todos estes métodos produzem superfícies similares, inclusive com as mesmas falhas no formato, sendo uma das razões a utilização de pontos de controle de um modo similar às *splines*, para a geração das curvas de contorno.

Hoppe et al. (1994), em seu artigo, propõem a utilização das superfícies de subdivisão como alternativa às *NURBS*, em aplicações gráficas e *CAD/CAM*, mesmo tratando-se de uma metodologia que envolve o refinamento sucessivo de uma malha triangular de controle inicial, apresentando um tratamento matemático diferenciado simplificado para a geração de superfícies suaves. Porém, o método proposto apresenta a mesma desvantagem de outros métodos poligonais em relação às *NURBS*, a não compactação dos dados e o procedimento recursivo da determinação de novos pontos para a divisão dos segmentos de reta. As superfícies finais obtidas por este método e os respectivos conjuntos de ponto utilizados, com variações na nuvem de pontos, são exibidas na Figura 3, com os respectivos resultados de desempenho na Tabela 2, sendo os algoritmos executados em uma estação de trabalho DEC Alpha. A Figura 3c (bule para chá) mostra uma superfície de subdivisão aproximada a uma superfície *NURBS*.

Na área de reconstrução de superfícies, o trabalho de Krishnamurthy e Levoy (1996) ilustra o exemplo de uma aplicação para a modelagem geométrica de peças utilizando um *scanner* 3D. Os dados dos pontos obtidos pela digitalização são organizados em seções ou curvas características, e esta informação é utilizada para criar superfícies *NURBS*. Este processo é utilizado normalmente em estruturas *CAD*, onde as *NURBS* são descritas e utilizadas como padrão para transferência de informação de gráficos em arquivos de sistemas de computação e nas etapas do processo de digitalização e geração do modelo geométrico.

Figura 3 – Conjunto de pontos inicial (primeira coluna à esquerda) e superfície final aproximada (última coluna à direita) obtida via superfícies de subdivisão. A figura 3c consiste em uma superfície de subdivisão aproximada a uma superfície *NURBS*. Fonte: (HOPPE ET AL., 1994).

Tabela 2 – Parâmetros e resultados para o método de Hoppe para superfícies de subdivisão. Fonte: (HOPPE ET AL., 1994).

Conjunto de pontos	Número de pontos	Parâmetro de precisão para as fases de refinamento da malha	Tempo (horas)
a	16.475	10^{-5}	2.2
b	10.000	10^{-5}	2.2
c	26.103	10^{-5}	5.3

O *IGES* (1996) é um padrão *ANSI* que inclui as *NURBS*, devido às características vantajosas como transformações afins, assim como os algoritmos e métodos numéricos utilizados para as *NURBS*, em diversos sistemas de processamento, sequencial ou paralelo. O *IGES* é o padrão utilizado em *softwares CAD* e sistemas *CAM* para o formato de informações sobre a geometria de modelos computacionais criados nestes sistemas, para a transferência destas informações em arquivos de dados.

Gopi e Manohar (1997) desenvolveram o algoritmo para arquitetura paralela, um dos mais referenciados para *b-splines*, baseado no *BFEA*, sendo possível sua utilização em todos os tipos de curvas e superfícies *b-splines*. Este método consiste no cálculo das funções base

em matrizes ou vetores para cada parâmetro de formação da *b-spline* para as curvas, repetindo-se o processo no caso de superfícies. O cálculo das funções base é efetuado conforme o *DAG – Directed Acyclic Graph*, como mostra a Figura 4. O algoritmo de Boor é utilizado para o cálculo recursivo dos valores das funções base, de acordo com este grafo.

Figura 4 - CPG – DAG para o processamento das funções base de uma *b-spline* de grau 3. Adaptado de: (GOPI, 1997).

Os valores das funções base *b-splines* são dados pelas seguintes equações recorrentes:

$$N_{i,0}(t) = \begin{cases} 1, & \text{se } u_i \leq t < u_{i+1} \\ 0, & \text{caso contrário} \end{cases} \quad (1a)$$

$$N_{i,p}(t) = \frac{t - u_i}{u_{i+p-1} - u_i} \cdot N_{i-1,p-1}(t) + \frac{u_{i+p} - t}{u_{i+p} - u_{i+1}} \cdot N_{i,p-1}(t) \quad (1b)$$

Sendo t o valor do parâmetro, u os valores dos nós, N as funções base *b-splines*, i o índice da função base em cada “camada” (ou nível) do DAG, e p o índice da camada do DAG. A matriz para o cálculo das funções base, a *BFEA*, possui um controlador e p elementos de processamento (p é o grau da curva), como mostra a Figura 5 (EP – Elemento de Processamento). Cada um destes elementos é responsável pelo cálculo do respectivo nível do DAG, composto por células de processamento. Como o primeiro elemento $N_{i,0}$ não necessita

de processamento, de acordo com a equação 1a, somente os três níveis restantes são processados pelos EPs.

Figura 5 – BFEA para uma *b-spline* de grau 3. Adaptado de (GOPI, 1997).

De acordo com as equações 1a a 1b:

$$N_{i-1,p} = \frac{t - u_{i-1}}{u_{i+p-2} - u_{i-1}} N_{i-2,p-1}(t) + \frac{u_{i+p-1} - t}{u_{i+p-1} - u_i} N_{i-1,p-1}(t) \quad (2a)$$

$$N_{i,p} = \frac{t - u_i}{u_{i+p-1} - u_i} N_{i-1,p-1}(t) + \frac{u_{i+p} - t}{u_{i+p} - u_{i+1}} N_{i,p-1}(t) \quad (2b)$$

$$N_{i+1,p} = \frac{t - u_{i+1}}{u_{i+p} - u_{i+1}} N_{i,p-1}(t) + \frac{u_{i+p+1} - t}{u_{i+p+1} - u_{i+2}} N_{i+1,p-1}(t) \quad (2c)$$

Desenvolvendo o primeiro termo da equação 2b:

$$\frac{t - u_i}{u_{i+p-1} - u_i} N_{i-1,p-1}(t) = \frac{(u_{i+p-1} - u_i) - (u_{i+p-1} - t)}{u_{i+p-1} - u_i} N_{i-1,p-1}(t) \quad (3a)$$

$$\frac{t - u_i}{u_{i+p-1} - u_i} N_{i-1,p-1}(t) = N_{i,p-1}(u) - \frac{u_{i+p-1} - t}{u_{i+p-1} - u_i} N_{i-1,p-1}(t) \quad (3b)$$

Observa-se que o segundo termo da equação 3b é o segundo termo da equação 2a, não sendo necessário seu cálculo novamente. Deste modo, é possível configurar o circuito de cada EP que compõe a BFEA de acordo com a Figura 6.

Figura 6 – Implementação do *BFEA* para curvas (quadrado: elemento de atraso). Fonte: (GOPI, 1997).

Conforme a Figura 6, as operações de soma e subtração indicadas no quarto ciclo do cálculo da função base $N_{i,j}(u)$ utilizam o segundo termo da soma da função base da camada anterior, reduzindo o número de operações. Os resultados obtidos por Gopi, para curvas e superfícies, foram comparados com o método para sistemas paralelos, proposto por Megson (1991), e são apresentados nas Figura 7 e Figura 8. Destes gráficos conclui-se que o método proposto por Gopi e Manohar possui um desempenho computacional melhor em relação ao método de Megson, já que o número de ciclos de *clock* necessários para o processamento é linearmente proporcional ao número de pontos de controle utilizados.

Carr et al. (1997, 2001) em seus trabalhos, descreveram a utilização das funções de base radial para a interpolação de um conjunto de pontos por uma superfície suave, e utilizaram métodos numéricos modificados para obter um desempenho superior aos algoritmos para *b-splines*. O tempo de processamento para a reconstrução de superfícies através de funções de base radial é significativamente reduzido pelos métodos numéricos atuais, mas apresentam desvantagens em relação a representações paramétricas, como as *b-splines*, por não apresentarem propriedades como as transformações afins, por exemplo.

Em seu trabalho, Besl (1998) demonstrou o incremento na eficiência e precisão na manufatura de peças utilizando a interpolação *NURBS*, em relação aos dados “ponto-a-ponto”, devido à menor dimensão dos segmentos lineares obtidos por taxas de interpolação maiores, resultando em uma peça com um acabamento final mais preciso, reduzindo em média de 100.000 para 50.000 comandos de deslocamento enviados ao controlador de posicionamento. Além disto, o tamanho do programa é reduzido de 1/10 a 1/100 em relação a programas de

interpolação linear, com movimentos em velocidades mais altas, independente da complexidade do molde.

Figura 7 – Resultados do método de Gopi e Manohar em relação ao método de Megson (1991) para curvas. Fonte: (GOPI, 1997).

Figura 8– Resultados do método de Gopi e Manohar em relação ao método de Megson (1991) para superfícies. Fonte: (GOPI, 1997).

Como ilustrado no diagrama da Figura 9, na prototipagem ou na engenharia reversa, este grupo de pontos é utilizado para controlar uma máquina-ferramenta CN – Controle Numérico, através de um material rígido, criando uma fôrma ou molde, e o tempo de

interpolação de uma curva *NURBS* para formar a trajetória desta ferramenta, e consequentemente o formato da superfície a ser manufaturada, reflete-se no erro cordal desta curva. Conforme Makino (1999), a precisão de superfícies complexas modifica-se com o quadrado do tempo de interpolação. Reduzindo o tempo de interpolação de 4 para 1 milissegundo, há um incremento na precisão em 16 vezes.

Figura 9 – Fluxo de dados típico da engenharia reversa. Fonte: (BESL, 1998).

Anderson e Hines (2000) apresentaram em seu trabalho o desenvolvimento de uma ferramenta computacional para converter modelos de estruturas orgânicas, fornecidos em malhas poligonais 3D, para superfícies *NURBS* que aproximam, com determinada precisão, os dados poligonais fornecidos por estas malhas. A justificativa para este trabalho são as aplicações como a telemedicina, em que a característica da representação compacta é prioritária para a transmissão dos dados em tempo real. Outra justificativa é a utilização das *NURBS* por um grande número de sistemas gráficos, mas há poucos *softwares* disponíveis comercialmente que geram com precisão superfícies *NURBS*, a partir de dados de estruturas orgânicas. Inicialmente, foi demonstrado e implementado um algoritmo para otimizar uma

superfície *NURBS*, obtida pela aproximação do desenho manual de regiões de interesse do conjunto de dados de estruturas orgânicas (neste caso, o banco de dados do Humano Visível – *Visible Human*). Redes neurais artificiais foram utilizadas para reduzir o tempo necessário de otimização do preenchimento destes dados para a superfície *NURBS*. Os resultados demonstraram uma redução em até 98% no tamanho dos arquivos, utilizando as *NURBS*, em relação aos modelos poligonais. Como conclusão, os métodos poligonais necessitam de um número elevado de polígonos para se obter superfícies de formatos suaves, razão pela qual muitos pacotes de software *CAM* não aceitam modelos poligonais como entrada.

Boor (2001) e Rogers (2001) mencionaram em seus trabalhos que, além da representação de modelos geométricos, as *splines* são utilizadas extensivamente como funções matemáticas genéricas de interpolação ou aproximação, dando origem a outros tipos de funções, como a *cardinal spline* e a *taut spline*.

Unser (2002) apresentou a importância das *splines* no processamento de imagens, nas aplicações para a reconstrução de imagens, e na visão computacional, mencionando os métodos de interpolação, multiresolução e *wavelets splines*. Como resultados, as *splines* apresentaram o melhor equilíbrio entre custo computacional e desempenho entre os métodos de interpolação disponíveis. O desempenho computacional da interpolação via *splines* foi efetuado pela comparação da relação sinal/ruído, a partir de 15 rotações de 24 graus de imagens médicas teste, com os seguintes algoritmos: interpolação do vizinho mais próximo (*spline* de grau 0), interpolação bilinear (*spline* de grau 1), interpolante quadrático de Dodgson, diversas versões do interpolante cúbico de Keys, interpolante cúbico de Shaum, interpolante cúbico de Lagrange, interpolante síncrono de janela de Hamming e interpolante polinomial *spline* de alta ordem com filtragem recursiva. As propriedades da multiresolução para as *wavelets splines* são comentadas pelas aplicações como compressão de imagens médicas, reconstrução tomográfica, melhoramento de imagens em mamografias, redução de ruídos em imagens médicas, como ressonância magnética, e a análise estatística de imagens funcionais do cérebro (*PET – Positron Emission Tomography* e *fMRI – functional Magnetic Resonance Imaging*), citando como exemplo as *wavelets* ortogonais de *Battle-Lemarié* e as *wavelets b-splines*.

Os trabalhos de Orenstein (2002) e Herweck (2009) descrevem o uso de funções de otimização da taxa de velocidade de objetos móveis, como máquinas-ferramenta, em que os movimentos em trajetórias constituídas por segmentos retilíneos, com mudanças bruscas de direção (como trajetórias em “cotovelo”), são substituídos por trajetórias na forma de curvas *splines*, reduzindo o tempo de usinagem em mais de 50%, mesmo em máquinas CNC comuns.

Tsai et al. (2003) mencionaram o uso de superfícies paramétricas em máquinas CNC para otimizar o controle da velocidade em três eixos, importando ou enviando modelos 3-D prontos para um sistema *CAM*, sendo programados em CNC e manufaturados via usinagem em alta velocidade, de acordo com o padrão *IGES - STEP – Standard for the Exchange of Product Model Data* (ISO 10303, 2002; SCRA, 2006), para sistemas *CAD* e *CAM*.

Sefi e Oppelstrup (2004) apresentaram um exemplo de aplicação das *NURBS* com sistemas de subdivisão de superfícies para o cálculo da integral *PO – Physical Optics*, pela geração gráfica da geometria complexa de padrões de radiação eletromagnética. O integrando *PO* exige o cálculo de um número elevado de pontos por comprimento de onda, devido à alta frequência apresentada. Nas superfícies 3D representadas por *NURBS*, uma subdivisão de superfícies foi realizada utilizando os pontos de controle como base para a triangulação inicial. O cálculo da integral *PO*, que fornece a direção das linhas dos campos das ondas eletromagnéticas, foi efetuado com base na renderização dos triângulos obtidos a partir da subdivisão da superfície *NURBS* inicial. Este método foi implementado na geração da superfície de geometria complexa de um veículo aéreo não tripulado denominado Eikon, descrevendo a viabilidade das *b-splines* para a modelagem computacional de um sistema matemático, além da utilização na computação gráfica.

Wen et al. (2005) mencionaram em seu trabalho projetos e prototipagem de produtos envolvendo superfícies complexas, que foram abandonados em razão da impossibilidade de manufatura dos modelos e pelo tempo necessário para serem finalizados. Esses projetos se tornaram viáveis com o desenvolvimento dos sistemas *CAD/CAM*. Como exemplos, ambientes computacionais específicos para o *projeto* de cascos de navios e perfis de asas de aviões, que incluem ferramentas especializadas para a criação e modificação de superfícies *splines* necessárias para os usuários destes tipos de aplicativos.

Yang et al. (2010) desenvolveram um algoritmo para manufatura de precisão, com o objetivo de gerar um grupo de curvas para trajetórias de corte de um centro de ferramentas, descrevendo também técnicas de processamento numérico para sistemas concorrentes, e novas tecnologias de *hardware* para o processamento de superfícies compatíveis com programas *CAM*. Foi proposta uma técnica de preenchimento de curvas, denominada de *loop* duplo, para a geração de uma superfície *NURBS*, utilizando o número mínimo de pontos de controle. Os resultados para o método de *loop* duplo foram obtidos pelo preenchimento de uma superfície teste, através de pontos de passagem da ferramenta (*CLs - Cutter Locations*), os quais são utilizados em métodos convencionais para a interpolação linear via CNC. Os resultados são exibidos na Tabela 3, indicando o erro no preenchimento da superfície, para

números de pontos de controle fixos, e o número de pontos de interpolação (método convencional) e de pontos de controle (método do *loop* duplo), necessários para se obter a tolerância de preenchimento. Os resultados demonstram que o método do *loop* duplo produz uma superfície mais suave, sendo adequado para converter trajetórias CNC convencionais, em trajetórias *NURBS*.

Tabela 3 – Resultados para o método do *loop* duplo de geração de trajetórias de ferramentas em sistemas CAM, em relação aos métodos CNC.

Número de <i>CLs (cutter locations)</i>	Erro no preenchimento (mm)			Número de pontos de controle		
	Número de pontos de controle	Método convencional	Método do <i>loop</i> duplo	Tolerância de preenchimento (mm)	Método convencional	Método proposto
85	16	0,0195	0,0080	0,0030	39	23
84	16	0,0190	0,0081	0,0030	39	23
82	16	0,0181	0,0074	0,0030	36	21
78	15	0,0191	0,0072	0,0030	36	21
77	15	0,0190	0,0072	0,0030	35	19
76	15	0,0201	0,0080	0,0030	36	19
74	15	0,0194	0,0073	0,0030	36	18
70	14	0,0180	0,0069	0,0030	35	17
66	13	0,0150	0,0055	0,0020	37	16
64	13	0,0126	0,0035	0,0020	36	15

O trabalho de Scherzer (2011) descreve os diversos métodos matemáticos computacionais aplicados em imagens médicas digitais para problemas inversos, análise e restauração de imagens. As *thin plate splines* são utilizadas para a minimização de energia em funções de base radial ou no aprendizado supervisionado de Máquinas de Vetores de Suporte (*SVM – Support Vector Machines*). A principal utilização das *b-splines* no processamento de imagens são como funções base *wavelet* em sistemas de multiresolução, por apresentar *b-*

splines poliharmônicas, complexas, fracionais, *splines* hexagonais e análise da imagem pelas *b-splines* de Schoenberg.

2.3. *FPGAS*

A evolução no desenvolvimento da lógica configurável incrementou a viabilidade de sistemas com *FPGAs* e *CPLDs*, assim como o número de aplicações e projetos utilizando estes tipos de dispositivos. A flexibilidade proporcionada pelos elementos lógicos permite a constituição de qualquer tipo de sistema digital, a partir de portas lógicas básicas até sistemas programáveis complexos. Deste modo, é necessária uma introdução ao *FPGA* e às ferramentas computacionais utilizadas para a sintetização, simulação e depuração dos dispositivos de lógica configurável. Estas ferramentas proporcionam a otimização de aplicações para sistemas de lógica reconfigurável e a criação de técnicas e métodos cuja implementação não era possível em sistemas programáveis de arquitetura computacional não reconfigurável, como a implementação de sistemas paralelos e distintos em um mesmo sistema embarcado.

As aplicações de lógica reconfigurável, descritas nesta seção, são divididas em duas subseções, sendo uma de manufatura e outra de computação gráfica e aplicações gerais. Na manufatura, a utilização de dispositivos de lógica configurável em relação a outros dispositivos de *hardware* programáveis em sistemas embarcados, proporciona maior poder de processamento, originando a área de sistemas de manufatura reconfiguráveis. Na computação gráfica, há dispositivos de *hardware* específicos para o processamento gráfico (*GPUs*), com a consequente redução no número de aplicações para dispositivos de lógica reconfigurável. As aplicações gerais em lógica reconfigurável descrevem técnicas e trabalhos que podem ser aplicados na maior parte dos sistemas *FPGAs*.

2.3.1. Introdução

Como nos circuitos integrados, os *FPGAs* são formados por unidades básicas, os elementos lógicos, cujo número máximo disponível varia entre os diversos modelos. O número de elementos lógicos ativos e sua frequência de processamento (ROSE ET AL., 1990) determinam o desempenho do circuito reconfigurável. A arquitetura do circuito integrado de

um *FPGA* (ROSE ET AL., 1993) mantém-se basicamente a mesma com o passar dos anos, e o projetista de um circuito configura estes elementos de modo a formar um ou mais *cores*: circuitos digitais formados pela combinação dos elementos lógicos, equivalentes às funções e classes das linguagens de programação convencionais.

Conforme Gupta e Zorian (1997): “Um *core* de circuito integrado é um bloco de circuito projetado e pré-verificado, normalmente possuindo no mínimo 5.000 portas lógicas, e que pode ser utilizado na construção de aplicações maiores ou mais complexas”. Os *cores* podem ser classificados em:

- *soft core*: consiste em uma descrição em linguagem *HDL* de sintetização de circuitos;
- *firm core*: além da descrição do circuito, contém uma estrutura adicional, como um *netlist*, para a disposição e roteamento de componentes em um *chip*;
- *hard core*: inclui *layout* do circuito integrado e informações de temporização dependente da tecnologia, para estar pronto para um sistema *ASIC*.

De acordo com Meyer-Baese (2001), as arquiteturas de diferentes fabricantes de *FPGAs* diferem principalmente nos blocos construtivos que agrupam os elementos lógicos, e a reconfiguração espacial destes elementos no *chip* permite mais de um subsistema digital, ou *core*, operar ao mesmo tempo, até o limite máximo de elementos lógicos, habilitando a implementação de *Socs – Systems on a chip*, destinando áreas do *chip* para determinados sistemas (CHANG ET AL., 2002). Como exemplo, um sistema de comunicação via rede de dados pode ser implementado em conjunto com um sistema de processamento de imagens, interconectando diversas interfaces de *cores* diferentes.

Deschamps et al. (2006) descrevem em seu trabalho as bases matemáticas para representação numérica, aritmética computacional e processamento de funções matemáticas, apresentando as respectivas bases de numeração e os métodos mais eficientes para circuitos em *FPGA*. Estes circuitos incluem as operações aritméticas de soma, subtração, multiplicação e adição com números no formato inteiro, ponto fixo ou flutuante. Além dos métodos computacionais para as operações aritméticas básicas, são analisados procedimentos de conversão entre sistemas de representação numérica, métodos de aproximação de funções trigonométricas, exponenciais e logarítmicas, como as séries de Taylor-MacLaurin e a aproximação polinomial. São observados com maior atenção os algoritmos *CORDIC*, suas aplicações em funções trigonométricas, e os algoritmos de cálculo da raiz quadrada baseados em recorrência de dígitos. Diferentes métodos de convergência são comparados, enfatizando

uma característica comum na implementação de muitos algoritmos nos *FPGAs*: a utilização das *look-up tables (LUTs)*, cuja utilização acompanha a evolução da tecnologia de memória *ROM*, direcionada a menor dimensão e menor custo, resultando no aumento da eficiência computacional e redução do custo dos procedimentos em geral que utilizam as técnicas para *FPGAs* baseadas em *LUT*.

Conforme mencionado no trabalho de Kuon et al. (2007), os elementos lógicos, também denominados blocos lógicos (BL) ou células lógicas (CL), possibilitam a reconfiguração do circuito, e se encontram presentes em parte dos principais tipos de microprocessadores (HENNESSY E PATTERSON, 2007) para reduzir o custo de produção, evitando a produção de novas máscaras de prototipagem, caso ocorra uma mudança no circuito.

Howell (2007) apresentou as vantagens das características funcionais dos modelos de *FPGA*, de permitir a atualização em campo, e até mesmo em operação, e de oferecer suporte à migração de densidade: *FPGAs* de capacidades lógicas diferentes mantêm o mesmo tamanho do invólucro, sem afetar o *layout* da *PCI*, resultando em redução de custos no desenvolvimento, produção, serviços e logística. Os *FPGAs* possuem um preço maior por unidade em relação aos *ASICs*, mas o desenvolvimento destes é caro e, se o projeto for modificado para se adequar a novos padrões, este custo é efetuado novamente para a modificação ou para novos projetos em *ASICs*.

Hauck e Dehon (2008) descrevem o *FPGA* como sendo constituído por elementos lógicos organizados em blocos que possuem conexões com outros blocos vizinhos, de forma a permitir que o *FPGA* seja eletricamente programável através da mudança na configuração destas conexões e blocos, originando qualquer subsistema digital. Esta possibilidade de reconfiguração resulta em duas formas de se executar o processamento computacional: via *hardware* ou *software*. O *hardware*, como os circuitos de aplicação específica (*ASICs* – *Application Specific Integrated Circuits*) fornecem recursos para tarefas de desempenho crítico, mas possuem uma configuração permanente, destinada a uma determinada aplicação para a qual foi fabricado, a um custo elevado, envolvendo o projeto e processo de fabricação. O *software* fornece a flexibilidade para a modificação das aplicações e executam diversas tarefas distintas sendo, no entanto, inferiores aos *ASICs* em relação ao desempenho, utilização da área útil do *chip* e na potência.

Os *FPGAs* incluem vantagens de *hardware* e *software*, implementando circuitos com melhor utilização da área do *chip*, da potência e do desempenho em relação ao *software*, sendo possível sua reprogramação (reconfiguração) diversas vezes para atender a uma

variedade de tarefas. Nos *FPGAs* é possível implementar o processamento computacional espacialmente, efetuando simultaneamente milhares de operações, via recursos distribuídos através de seu *chip*, incrementando seu desempenho em relação aos dispositivos que executam *softwares*.

Ainda de acordo com Hauck e Dehon (2008), nas duas décadas desde a introdução dos *FPGAs* no mercado de dispositivos eletrônicos, a forma como a lógica digital é fornecida e projetada foi modificada, unindo o alto desempenho computacional dos circuitos integrados de aplicação específica (*ASICs*) e a flexibilidade dos microprocessadores, substituindo sistemas com *ASICs* e *DSPs* e tornando possível o surgimento de novas aplicações.

De acordo com o fabricante de *FPGAs* Altera (2009), os circuitos projetados em lógica configurável são produtos de tecnologia, *software* e *know-how* sujeitos a patentes e direitos autorais, ou seja, um bloco de *core* representa uma propriedade intelectual que seu projetista licencia para o usuário deste *core*. Para alcançar um número maior de usuários, os fabricantes disponibilizam gratuitamente alguns dos *cores* mais comuns para suas plataformas e muitos projetistas de circuitos disponibilizam *cores* abertos gratuitamente, com direitos de utilização semelhantes aos criados para *software*.

2.3.2. Ferramentas de Automação do Projeto Eletrônico

Segundo Thomas e Moorby (2002), os *cores* são projetados a partir de linguagens de descrição de *hardware*, como a linguagem *Verilog*, que possibilita a sintetização de circuitos como uma linguagem de programação. A *Verilog* é padronizada com especificações para ferramentas de síntese lógica e comportamental, sendo originada em 1983 na *Gateway Design Automation*, desenvolvida posteriormente para a *DARPA*, e aprovada como padrão *IEEE 1364* (IEEE, 2006) em 1995.

Conforme o fabricante Altera (2004), a avaliação do desempenho de um circuito específico em um *FPGA* leva em conta as medidas de desempenho (*benchmarking*) na frequência máxima de operação.

De acordo com Chen et al. (2006), as medidas de desempenho, assim como a interface de sintetização, simulação, teste e verificação de plataformas de lógica reconfigurável, são executadas por ferramentas de *EDA* – *Electronic Design Automation*. Essas ferramentas proporcionam a sintetização de circuitos em nível de transferência entre registradores (*RTL* –

Register Transfer Level) através da utilização de linguagens padronizadas: o *Verilog HDL* – *Hardware Description Language* e o *VHDL* – *VHSIC* – *Very High Speed Integrated Circuit HDL*, que permitem uma programação em um nível mais alto, além do projeto via diagramas esquemáticos. A linguagem de sintetização *VHDL* surgiu em torno de 1985 no Departamento de Defesa dos Estados Unidos (*DARPA*) e teve a primeira versão de padronização em 1987 (IEEE, 2009).

2.3.3. Aplicações

As aplicações das *b-splines* relacionadas à lógica reconfigurável abrangem principalmente sua implementação em sistemas embarcados, com um número significativo de trabalhos relacionados a sistemas *CAM* e de manufatura. O desenvolvimento constante, pelos fabricantes de *hardware*, de dispositivos programáveis específicos para esta área, como placas aceleradoras de vídeo, circuitos integrados gráficos, além das respectivas ferramentas computacionais, resultando em um número reduzido de trabalhos envolvendo lógica reconfigurável na computação gráfica.

A importância dos circuitos digitais tornou necessária a utilização do controle digital embarcado em sistemas automatizados de manufatura (YAU ET AL., 2006; LI ET AL., 2009) e no *upgrade* de sistemas convencionais (XU ET AL., 2008), originando novas áreas de pesquisa, como os sistemas *CNC* reconfiguráveis (DONGDONG ET AL., 2007; DONG ET AL., 2009; WANG ET AL., 2011; DONG E WANG, 2012), que possibilitam a aplicação da lógica reconfigurável em métodos de manufatura avançados.

Deste modo, as aplicações da computação gráfica podem ser incluídas junto às aplicações gerais que utilizam a lógica reconfigurável como acelerador, como um sistema para análise comparativa em relação a sistemas com outras tecnologias de processamento computacional, e técnicas utilizadas para a lógica reconfigurável. As técnicas utilizadas nestas aplicações gerais podem ser implementadas em diversas áreas, dos sistemas embarcados reconfiguráveis até a simulação e a computação heterogênea.

2.3.3.1. Manufatura e Sistemas CAM

Jesús et al. (2004) abordaram um dos principais problemas em sistemas CNC: a detecção da quebra da ferramenta. O trabalho dos autores utilizou a medida das forças de corte atuantes em um eixo de movimento da máquina-ferramenta, para calcular a transformada *wavelet* discreta da força resultante, através de técnicas de processamento de sinais em *hardware* (PSH), que consiste em um *Soc* implementado em um *FPGA* e uma memória *ROM* para armazenamento dos valores de coeficientes da transformada. Esta aproximação garante a detecção *on-line* da quebra da ferramenta a um baixo custo, utilizando a medida indireta das forças de corte pela corrente elétrica dos *drivers* dos motores (*sensorless*). Os resultados foram obtidos por meio da implementação deste sistema em uma fresa CNC, com uma estimativa ponderada da detecção *on-line* executada por três circuitos *DSPs* específicos a esta tarefa, sintetizados em um *FPGA*, fornecendo o valor da transformada 38,9 μ s após o eixo sujeito às medições completar uma revolução. Os resultados da transformada *wavelet* da força resultante estimada, fornecidos pelos circuitos *DSP*, em relação ao número de rotações do eixo atuante de uma fresa, desde o início da medição até a quebra da ferramenta de corte, são exibidos pelo gráfico da Figura 10, com a indicação do período de operação normal da ferramenta de corte, o período de desgaste desta ferramenta e a indicação de quebra como o valor máximo no gráfico.

Figura 10 – Resultado da transformada *wavelet* da força resultante no eixo da ferramenta de uma máquina fresadora, obtida por três *DSPs* em um *FPGA*. Fonte: (JÉSUS ET AL., 1994).

Yau et al. (2006) apresentaram, em seu trabalho, o desenvolvimento do circuito sintetizado *fast* Cox-de Boor, que consiste basicamente no processamento concorrente dos parâmetros para a formação de cada ponto de uma curva ou superfície *NURBS*. Neste trabalho, um circuito para o posicionamento de dois servomotores e um filtro digital, para eliminação de ruídos, são adicionados ao *Soc*, atuando como um controlador PID para uma mesa de posicionamento XY, com todos os circuitos sintetizados em um mesmo *FPGA*. A contribuição relevante do circuito *fast* Cox-de Boor é o processamento paralelo dos pontos a serem gerados de uma curva ou superfície *NURBS*, via utilização de um circuito, ou “triângulo” Cox-de Boor para cada ponto a ser gerado. Os pontos cujos parâmetros se situam dentro de uma mesma faixa de valores de nós utilizam os mesmos valores de nós e de pontos de controle no algoritmo Cox-de Boor, de acordo com a Figura 11. Os resultados foram comparados com a implementação em um sistema computacional *PC* (*Personal Computer*) e com a implementação em *hardware VLSI* do método de Gopi (1997), fornecidos na Tabela 4 e Figura 12, para o número de ciclos do sinal de *clock* exigidos para cada método, o número de operações aritméticas fundamentais necessárias para cada método, e o número de chamadas às rotinas. Observa-se que o desempenho computacional para o método *fast* Cox-de Boor é proporcional ao grau p da curva *NURBS*. Para o computador pessoal, o desempenho computacional também é dependente do número de pontos gerados (variável N).

Figura 11 – Os pontos gerados cujos parâmetros se encontram dentro do mesmo intervalo de nós utilizam os mesmos pontos de controle e os mesmos valores de nós. Na figura acima, de P_0 a P_2 e de u_0 a u_1 para uma curva de grau 2.

Tabela 4 – Desempenho dos métodos de interpolação *NURBS*. Fonte: (YAU, 2006).

Arquitetura	Método de interpolação <i>NURBS</i>	Número total de chamadas à rotina	Numero de operações aritméticas	Número de ciclos totais de <i>clock</i>
PC	Cox-de Boor	$\frac{p(p+1)}{2}$	1+, 4-, 2x, 2÷	$89N \frac{p(p+1)}{2}$
VLSI	<i>BFEA</i> de Gopi	$p-1$	2+, 1-, 1x, 1÷	$7p-1$
FPGA	<i>Fast Cox-deBoor</i>	p	1+, 1-, 1x, 1÷	$4p$

Figura 12 – Desempenho computacional em ciclos do sinal de *clock* do algoritmo Cox-de Boor em uma arquitetura computacional *PC*, arquitetura *VLSI* (GOPI, 1997) e o método *fast Cox-de Boor* (YAU, 2006), de acordo com o número de eixos (coordenadas Euclidianas) e grau da curva *NURBS*. Fonte: (YAU ET AL., 2006).

Jimeno et al. (2006, 2007) apresentaram uma arquitetura de *hardware/software* em *FPGA* para aumentar a eficiência computacional do algoritmo *Virtual Digitizing*, desenvolvido para a geração de *tool paths* em sistemas *CAD/CAM*. Este algoritmo exige um alto custo computacional de processamento, sendo desenvolvido especificamente para as seguintes tarefas: gerar o modelo discreto, simular a trajetória da ferramenta de manufatura e digitalizar virtualmente a superfície de calçados, sendo implementado somente em sistemas que seguem os padrões *CAD/CAM* tradicionais. Deste modo, os pontos do *grid* para gerar o modelo discreto da superfície são obtidos a partir de uma superfície *NURBS*, cujo formato é padronizado para sistemas *CAD/CAM*. Uma implementação da geração de pontos de uma superfície *NURBS* em *FPGA* pode complementar este sistema embarcado, constituindo um *Soc* mais completo. O resultado geral dos tempos de processamento deste sistema, atuando ou não como acelerador da aplicação, operando em conjunto com um computador pessoal, são exibidos na Figura 13, onde o tempo ideal é o tempo de processamento obtido pela geração de

uma trajetória circular da ferramenta, o formato de trajetória mais simples e com o menor tempo de processamento.

Figura 13 – Tempo de processamento do algoritmo *Virtual Digitizing* obtido por um computador pessoal (PC) com (linha cheia) e sem (linha pontilhada) a aceleração via *FPGA*. Fonte: (JIMENO ET AL., 2006; 2007)

O trabalho de DongDong et al. (2007) descreve o sistema CNC reconfigurável como um sistema CNC que utiliza um *FPGA* como o centro de controle principal, em substituição às placas de controle tradicionais baseadas em microcontroladores. O trabalho de DongDong et al. apresenta a opção da reconfiguração lógica aplicada aos sistemas de controle numérico programáveis, e inclui os projetos de um gerador de sinais programável, o controle de velocidade de motores de passo e a comunicação entre o sistema CNC-*FPGA* e um computador pessoal. O gerador de pulsos programável consiste em um *IP core PWM – Pulse Width Modulation*, fundamental para sinais de controle de drive de motores CC ou AC, e foi implementado pela contagem dos pulsos de *clock* de referência do *core* como base de tempo para determinar a largura do pulso. Pelo *FPGA*, foi demonstrada a geração simultânea de mais de um sinal *PWM*, com o envio dos sinais ao mesmo tempo para mais de um drive de controle de velocidade. A reconfiguração lógica permite o funcionamento deste sistema com servomotores, motores CC e motores de passo, com os respectivos drives. A arquitetura proposta de um CNC reconfigurável é exibida na Figura 14, e os resultados demonstram que o número de componentes no *hardware* de controle é menor em relação à arquitetura CNC convencional (não reconfigurável).

Figura 14 – Arquitetura de *hardware* de CNC reconfigurável baseada em *FPGA* proposta por DongDong et al. Adaptado de: (DONGDONG ET AL., 2007).

Xu et al. (2008) desenvolveram em seu trabalho um sistema CNC baseado em processadores embarcados com operação em tempo real, composto por um modelo de projeto *dual CPU* (ARM e *DSP*), com um módulo *FPGA* executando reconfiguração e manufatura flexível. Este sistema foi implementado em um CNC reconfigurável, com possibilidade de controle de quatro eixos e um eixo principal simultaneamente, com testes de interpolação ciclo-plana, interpolação helicoidal, interpolação *CIP-Circle Interpolation Point* e em método de controle de alimentação adaptativa, suportando altas taxas de precisão à alta velocidade de manufatura, eliminando as desvantagens da transformação da curva ou superfície em linhas e círculos.

Figura 15 – Sistema *dual CPU*. Fonte: (XU ET AL., 2008).

Dong et al. (2009) apresentaram uma nova arquitetura para sistemas CNC reconfiguráveis, transformando módulos CNC com processamento baseado em microcontroladores, em sistemas reconfiguráveis e incluindo um módulo para tarefas de reamostragem. A utilização do controle e dos módulos em lógica reconfigurável possibilita direcionar o controle principal a uma tarefa específica, como o controle de um número determinado de eixos de uma máquina-ferramenta. A arquitetura básica de um sistema controlador CNC, fornecida por Dong et al., é exibida na Figura 16, e a arquitetura proposta em lógica reconfigurável para uma fresa na Figura 17. O protótipo de testes baseado neste sistema é dividido em duas partes: tempo real e tempo não real. O subsistema responsável pelas tarefas que não exigem tempo real utiliza um microprocessador de 32 bits, uma memória *flash*, uma *SDRAM*, portas de comunicação e uma unidade de depuração, executando uma versão do sistema operacional Linux, e é responsável pela interface do usuário, interpretador, interpolador, lógica sequencial, comunicação, gerenciamento dos arquivos de controle numérico, e outras funções gerais. O *hardware* do subsistema proposto, responsável pelas tarefas de tempo real, é composto por um microcontrolador de 8 *bits*, uma memória *RAM dual port*, um *chip FPGA*, um conversor D/A e os circuitos para conversão do sinal e circuitos de isolamento (acopladores óticos). A memória *dual port* armazena os comandos de tempo real que são organizados em pacotes e enviados ao *FPGA* e conversor D/A do eixo da máquina-ferramenta. Este subsistema de tempo real baseado no *FPGA* foi utilizado em uma fresa e depois reconfigurado para a utilização em um torno, para os mesmos corpos de prova manufaturados em sistemas não reconfiguráveis, que utilizam sistemas de controle distintos de acordo com o tipo de máquina (fresa ou torno). Os resultados demonstraram que a arquitetura proposta em lógica reconfigurável pode ser facilmente modificada de acordo com a aplicação da máquina-ferramenta.

Li et al. (2009), em seu trabalho, descreveram o projeto de um sistema de controle numérico totalmente digital baseado em *FPGA*, com os controles integrados de servo mecanismos, comunicação *USB*, sinais de *encoder* e de interface de expansão analógico-digital. Nos sistemas CNC tradicionais, o sistema de servo mecanismos necessita de sinais analógicos para o controle, e o sistema de controle numérico processa e fornece os dados na forma digital, sendo necessário pelo menos uma conversão digital-analógico compatível entre os dois sistemas (a resolução e a faixa de operação devem ser compatíveis). Deste modo, as interfaces entre os sistemas de cálculo numérico e de controle de servo mecanismos afetam significativamente o desempenho do sistema.

Figura 16 – Arquitetura básica de um controlador CNC. Adaptado de: (DONG ET AL., 2009).

Figura 17 – Estrutura de uma fresa CNC reconfigurável proposta por Dong et al. (2009). Adaptado de: (DONG ET AL., 2009)

Este sistema de controle numérico totalmente digital baseado em *FPGA*, com a arquitetura das partes componentes exibida na Figura 18, utiliza a interface *USB* entre os sistemas de cálculo numérico e de controle dos servomecanismos, simplificando e incrementando o desempenho computacional do sistema CNC, eliminando a conversão de sinais analógicos.

Figura 18 – Arquitetura de um sistema CNC totalmente digital baseado em um *FPGA* (3), com interface para sistema CNC externo (1), *USB* (2) e sistema de controle de potência (4) para servomecanismos (5). Fonte: (LI ET AL., 2009).

Wang et al. (2011) apresentaram em seu trabalho um sistema CNC de reconfiguração tripla, efetuando a divisão da tecnologia de reconfiguração de sistemas CNCs em três níveis interativos, que constituem os principais tópicos de pesquisa na área de CNC reconfigurável:

1. Reconfiguração do sistema de *hardware* ou reconfiguração de *chip*, em que os circuitos do *hardware* propriamente dito são modificados para satisfazer a reconfiguração do sistema CNC inteiro, via lógica reconfigurável, e sintetização de novos circuitos em *FPGAs* ou *CPLDs*. Como exemplo, é equivalente a troca de um microcontrolador ou circuitos periféricos do sistema de controle de um CNC tradicional por modelos mais modernos (*upgrading*), e não somente modificar ou complementar a programação destes circuitos ;
2. Reconfiguração do sistema de *software*, que pode ser modificada ou inicializada do zero (*reset*) para satisfazer os requisitos de novas aplicações, como a inclusão da geração de pontos de curvas suaves, além das primitivas geométricas de segmentos de reta e arcos de circunferências, nas trajetórias de ferramentas;

3. Reconfiguração de módulo, no qual a lógica reconfigurável pode integrar diferentes módulos (*drivers*, motores, interface homem-máquina, CLPs,...) de diferentes fabricantes, definindo uma única interface padrão de comunicação para todo o sistema CNC, facilitando também a inclusão ou exclusão de um módulo.

Esse sistema utiliza a implementação de arquivos distintos de configurações lógicas para a reconfiguração do CNC, como reconfiguração de *hardware*, e o projeto, modificação, seleção e configuração de diferentes bases de dados para uma plataforma de desenvolvimento de reconfiguração de um *software* guia (reconfiguração de *software*). Para a reconfiguração de módulos, foi implementada a comunicação PROFIBUS-DP (padrão para dispositivos controladores) entre os módulos de funções do sistema CNC proposto. Para validação e testes, este sistema de reconfiguração tripla foi implementado em um sistema CNC com controle de até oito eixos-passo (Figura 19), incluindo a interpolação de cinco destes eixos, a partir da reconfiguração do controle de outro sistema CNC, que controla até quatro eixos-passo (Figura 20), com interpolação de até três desses quatro eixos.

Figura 19 – Estrutura simplificada do sistema CNC de controle de 4 eixos. Adaptado de: (WANG ET AL., 2011).

Figura 20 – Estrutura simplificada do sistema CNC de controle de 8 eixos obtido pelo *upgrade* do sistema CNC reconfigurável de controle de 4 eixos. Adaptado de: (WANG ET AL., 2011).

Os resultados obtidos demonstraram que os dois sistemas podem funcionar à mesma taxa de transferência de dados para o PROFIBUS-DP, sem perder a eficiência do processo no sistema atualizado de controle de 8 eixos. Um sistema CNC reconfigurável mais avançado pode ser obtido a partir de um sistema mais simples, e a reconfiguração tripla facilita este processo de *upgrading*, classificando o melhoramento das funções de um sistema para outro em três tipos:

- Homologia: uma mesma função é utilizada nos dois sistemas, sem modificações ou melhoramentos, como as funções tradicionais de interpolação de linha 2D e 3D;
- Expansão: uma função é expandida para incrementar os eixos de controle da máquina-ferramenta, como no caso estudado em que o número de eixos controlados passou de quatro para oito;
- Incremento: funções adicionadas no sistema atualizado, que não existem no sistema anterior.

Dong e Wang (2012) apresentaram uma arquitetura reconfigurável baseada em *pipeline* para controladores CNC, que consiste em um controlador principal com um *software* e um gerenciador *multithreading*, em comunicação com um processador de microcódigo de controle numérico (*NCMP – NC Microcode Processor*), via *pipeline* de *threads*. O *pipeline* de *threads* no controlador principal converte o programa de manufatura, no microcódigo de controle numérico (*NCM – NC Microcode*), otimizado para a execução em tempo real no *NCMP*. Um algoritmo de planejamento da taxa de alimentação da máquina-ferramenta foi desenvolvido para esta arquitetura, apresentando uma estrutura reconfigurável e a característica de previsão da taxa de alimentação para manufatura em alta velocidade. O planejamento da taxa de alimentação altera a produtividade, precisão e vibração da ferramenta, no processo de manufatura. O método de previsão da taxa de alimentação é utilizado para misturar o movimento de alimentação de linhas contínuas, ao invés de tratá-lo separadamente, armazenando segmentos de reta (de pequenas dimensões) provenientes dos programas CN, que compõem a trajetória da ferramenta (gerada na maioria das vezes por *softwares CAD/CAM*), gerando um perfil de taxa de alimentação otimizado no tempo, com controle de aceleração para movimentos suaves. No algoritmo desenvolvido, os segmentos de reta fornecidos pelo programa de controle numérico são armazenados em uma memória *FIFO-First In First Out*, sendo processadas a leitura dos dados geométricos e das variáveis cinemáticas. Os resultados foram obtidos de dois sistemas protótipos: o primeiro com o controlador principal como um computador pessoal (*PC*), executando o sistema operacional

Microsoft Windows, no qual foi desenvolvido o *pipeline* de *thread NC* e os algoritmos *NCM* implementados em *software*. No segundo sistema protótipo, o controlador principal consiste em um microcontrolador ARM executando um *kernel* Linux, os algoritmos *NCM* processados por *hardware FPGA* no *NCMP*, e o *pipeline* de *thread NC* do primeiro sistema protótipo (executado em Microsoft Windows), reconfigurado e transferido para o sistema Linux. Deste modo, foram criadas duas versões para os algoritmos de execução *NCM*: a versão em linguagem C para a implementação em *software* do primeiro sistema protótipo, e uma versão em *Verilog Hardware Description Language (HDL)* para a implementação em *hardware* do segundo sistema protótipo. Para os resultados obtidos exibidos na Figura 21, foram implementados os algoritmos de aceleração trapezoidal, aceleração exponencial e aceleração em curva-S para o teste do algoritmo de previsão da taxa de alimentação, sendo a variável *N* o número de segmentos de reta utilizados para interpolar os dados geométricos adjacentes (pontos) da trajetória da ferramenta. Estes resultados apresentam uma redução significativa do tempo de manufatura e a reconfiguração de acordo com a mudança nos esquemas de previsão da taxa de alimentação de avanço e retrocesso.

Figura 21 – Perfis de taxa de alimentação da máquina-ferramenta para o sistema protótipo de arquitetura reconfigurável de *pipeline NC*. Fonte: (DONG E WANG, 2012).

2.3.3.2. Computação Gráfica e Aplicações Gerais

Singh e Bellec (1994) realizaram a primeira análise de um sistema reconfigurável na computação gráfica, de um sistema de hardware gráfico, cujos circuitos foram sintetizados para a geração de primitivas gráficas e renderização de imagens com sombreado, em dois modelos de *FPGA*. Os resultados foram comparados com aos *chips* gráficos customizados e com processadores gráficos de propósitos gerais. Foram escolhidos três algoritmos gráficos para esta análise, para primitivas gráficas similares, mas que diferem no processamento

computacional exigido: o algoritmo do ponto médio para a geração dos pontos de uma circunferência e de um círculo, e o algoritmo *fast spheres* para a renderização de esferas sombreadas. Estes algoritmos foram implementados na linguagem de programação Haskell, uma variante da linguagem algébrica Ruby de descrição de *hardware*, e posteriormente adaptados para ferramentas de simulação e compilação *VHDL* Alliance V1.9 para fins de teste, e para o compilador *VHDL* Compass para gerar os *netlists*, alocação, roteamento, programação dos *FPGAs* e testes. Foram utilizados dois *FPGAs*, o Xilinx XC4000 e o Altera Flex8000, com os resultados apresentados na Tabela 5 e na Tabela 6, que mostram o número de elementos lógicos utilizados, tempo médio de síntese, otimização, alocação, roteamento e os tempos de processamento para os algoritmos gráficos do ponto médio e *fast spheres* (para circunferência, círculo e esfera de raio igual a 100 unidades). Para fins de comparação, o processador gráfico TMS34020 da Texas Instruments foi utilizado como o circuito integrado específico para aplicações gráficas, possuindo uma taxa máxima de processamento de 2 milhões de pixels/s., enquanto a implementação em *FPGA* gera um processamento de 12 milhões de pixels/s. Porém, os autores observam o problema da utilização de *FPGAs* como co-processadores gráficos, pelo conflito de acesso às memórias compartilhadas com o processador mestre e sincronia na transferência de dados.

Tabela 5 – Resultados para a implementação dos algoritmos gráficos no *FPGA* Xilinx XC4000. Fonte: (SINGH E BELLEC, 1994).

Xilinx XC4000	Circunferência	Círculo	Esfera (<i>Fast Sphere</i>)
Síntese	30 s.	50 s.	22 min.
Otimização	7 min.	15 min.	20 min.
Dispositivo	4003APC84	4004APC84	4010CB196
Utilização de <i>CLBs</i>	61/100(P) 78/100(O)	88/144(P) 107/144 (O)	347/400(P) 379/400(O)
Caminho crítico	84 ns.	92,3 ns.	135,1 ns
Tempo de processamento, r=100	47,518 ms.	2,89 ms.	4,23 ms

Tabela 6 – Resultados da implementação dos algoritmos gráficos para o FPGA Altera Flex8000. Fonte: SINGH E BELLEC, 1994).

Altera Flex8000	Circunferência	Círculo	Esfera (<i>Fast Sphere</i>)
Síntese	30 s.	50 s.	-
Otimização	7 min.	15 min.	-
Dispositivo	EPF828LC84	EPF828LC84	-
Utilização de <i>CLBs</i>	134/204 64%	185/208 89%	-
Caminho crítico	92,4 ns.	85,4 ns.	-
Tempo de processamento, r=100	52,3 ms.	2,64 ms.	-

Ratha e Jain (1999) desenvolveram um trabalho na área de visão computacional sobre a customização de um sistema em tempo real, com a disposição dos *cores* dentro do *chip FPGA* para uma implementação em circuitos reconfiguráveis de métodos voltados ao processamento paralelo de algoritmos de visão computacional, os quais exigem computação de alta performance para atuar em tempo real. Os resultados foram demonstrados para três aplicações comumente utilizadas em visão computacional, classificadas de acordo com as características computacionais e de comunicação em níveis baixo, intermediário e alto, fornecidas na Tabela 7.

Tabela 7 – Exemplos de tarefas dos três níveis de hierarquia da visão computacional. Fonte: (RATHA E JAIN, 1999).

Nível da tarefa	Característica computacional	Exemplos
Baixo	Acesso reduzido a dados de elementos vizinhos, operações simples, grande quantidade de dados;	Deteção de bordas, filtragem, morfologia de imagens
Intermediário	Vizinhança pequena, operações mais complexas;	Transformada de Hough, identificação de componentes conectados, relaxamento
Alto	Acesso a dados não locais, operações complexas e não polinomiais.	Combinação de pontos, árvores de combinações, grafo de combinações.

Três algoritmos, correspondentes a cada nível de tarefa, foram desenvolvidos em um sistema computacional de lógica reconfigurável denominado *Splash 2*, desenvolvido pelo Instituto do Centro de Pesquisa de Supercomputação para Análise da Defesa, baseado em um FPGA Xilinx 4010. Os tempos de processamento foram comparados com uma implementação

sequencial em uma plataforma computacional Sun SPARCStation 20. Os resultados obtidos por Ratha e Jain estão descritos na Tabela 8.

Tabela 8 – Tempos de processamento computacional obtidos por Ratha e Jain para tarefas dos três níveis hierárquicos de visão computacional, em implementação sequencial em um Sun SPARCStation 20 e na plataforma *FPGA* Splash 2. Fonte: (RATHA E JAIN, 1999).

Nível	Tarefa	SPARCStation	Splash 2
Baixo	Convolução 2D	3.600 ms	13,89 ms
Intermediário	Segmentação de Imagem	250 s	3,8 s
Alto	Combinação de padrão de pontos	15,38 ms	0,00909 ms

Styles e Luk (2000) mencionaram em seu trabalho um estudo sobre arquiteturas customizáveis para aplicações gráficas, como os *pipelines* de renderização via lógica reconfigurável, explorando formatos de dados e tamanhos de barramentos de dados customizados, otimizando operações gráficas como mapeamento de textura e remoção de superfícies ocultas. Foram implementadas pelos autores, técnicas para balanceamento do *pipeline* gráfico, exibido na Figura 22, e para configuração lógica em tempo real foram implementadas arquiteturas customizadas projetadas em Handel-C (linguagem de programação similar à linguagem C com suporte ao paralelismo e tamanho de dados flexível) e compiladas para *FPGAs* Xilinx 4000 e Virtex, além de uma aplicação para interface de programação baseada no OpenGL, voltada à aceleração automática de aplicações gráficas. Os resultados para as plataformas computacionais são dados na Tabela 9, para duas aplicações: simulação de imagens em infravermelho e visualização geométrica.

Figura 22 – Pipeline gráfico implementado por Styles e Luk. Fonte: (STYLES E LUK, 2000).

Tabela 9 – Tempo de processamento de taxa de *frames* por segundo para as plataformas de implementação. Fonte: (STYLES E LUK, 2000).

Aplicação	Plataforma de implementação	Taxa de <i>clock</i> (MHz)	<i>Frame rate</i> (fps)
Simulação de imagens em infravermelho	PC (<i>software</i>)	400	96
	Xilinx XC40150	20	167
	Xilinx XCV1000	40	330
	SGI Onyx2 Reality	180	2750 (estimado)
Visualização geométrica	PC (<i>software</i>)	400	24
	Xilinx XC40150	25	26
	Xilinx XCV1000	40	41
	Nvidia TNT2 Ultra	170	55

Ou et al. (2002) em seu trabalho efetuaram testes em linguagem C++ e *OpenGL* para arquitetura paralela, em uma implementação *VLSI* direta do algoritmo para funções *b-splines* dado pelas equações 1a a 1b (GOPI E MANOHAR (1997), seção 2.2 – *B-Splines*), utilizando combinações fixas dos valores dos nós para as funções base, de acordo com o grau desejado de uma curva *NURBS*. Como exemplo, uma curva de grau 2 possui somente as seguintes combinações dos valores do parâmetro para gerar as funções base:

$$\left. \begin{array}{l} u^2 - 2u + 1, 2(u - u^2), u^2 \\ u^2 - 2u + 1, 2u - \frac{3u^2}{2}, \frac{u^2}{2} \end{array} \right\} p/0 \leq u < 1 \quad (4a)$$

$$\left. \begin{array}{l} \frac{u^2}{2} - 2u + 2, 4u - \frac{3u^2}{2} - 2, u^2 - 2u + 1 \\ \frac{u^2}{2} - 2u + 2, 3u - u^2 - \frac{3}{2}, \frac{u^2 - 2u + 1}{2} \end{array} \right\} p/1 \leq u < 2 \quad (4b)$$

Os pontos da curva são calculados pelo *CPG – Curve Point Generator*, mostrado na Figura 23 para a *NURBS* de grau 2. A flecha tripla indica que para cada ponto gerado, é necessária a soma de somente três funções base nas equações (1) e (2).

Figura 23 - CPG – Curve Point Generator para a implementação VLSI em arquitetura paralela. Fonte: (OU ET AL., 2002).

Porém, as equações 4a e 4b são válidas somente para um vetor de nós uniforme para uma curva de grau 2. Para vetores de nós não uniformes, estas equações não são válidas, e para curvas de graus diferentes, o número de equações aumenta exponencialmente (16 equações para o grau 3), e apresenta maior eficiência somente para o cálculo de um valor de parâmetro, com o valor da função base respectiva dado no CPG. Os resultados obtidos por Ou et al. estão exibidos graficamente na Figura 24, em comparação ao algoritmo Cox-de Boor para arquiteturas PC (*Personal Computer*) e à arquitetura BFEA proposta por Gopi e Manohar (1997).

Figura 24 – Resultados do algoritmo de Ou em relação aos algoritmos de de Boor, Gopi, Ying e Shiu. Fonte: (OU ET AL., 2002).

O trabalho de Sanford (2002) na AMD – *Advanced Micro Devices* menciona a incorporação de *cores* de processamento gráfico em CPUs, que são circuitos destinados às aplicações gráficas os quais tendem a ser anexados em circuitos integrados. O realismo visual satisfatório ao usuário, no processamento computacional necessário às aplicações gráficas 3D, é obtido pelo aumento no poder de processamento dos dispositivos gráficos, pelo aumento da

quantidade e taxa de transferência da memória disponível ao processador vídeo, e por outros meios, como concentrar e transferir tarefas gráficas para um processador de vídeo específico (como as *GPUs*), onde a eficiência computacional depende também da transferência dos dados pelo barramento do sistema computacional. Como exemplo, Sanford cita a inclusão de *cores* de unidades de ponto flutuante em *CPUs* para possibilitar um número maior de operações executadas por ciclo do *clock*, disponibilizando a Transformação e Iluminação (*T&L - Transform and Lighting*), uma tecnologia de vídeo criada originalmente para transferir da *CPU* para uma *GPU* toda a informação 3D para gerar e controlar a transformação de coordenadas e a iluminação, o que exige um conjunto de cálculos em ponto flutuante, necessários para qualquer ambiente gráfico dinâmico 3D.

Zemcik (2002) apresenta em seu trabalho a integração de circuitos de *FPGAs* com *DSPs*, para o processamento computacional de algoritmos de computação gráfica e processamento de imagens, como uma alternativa ao *hardware* existente, como placas aceleradoras de vídeo, em que o poder de processamento está concentrado principalmente na renderização de polígonos e/ou triângulos planos. Os resultados obtidos demonstraram que mesmo algoritmos e arquiteturas que não eram suportados por este *hardware* existente, foram possíveis de serem implementados em dispositivos lógicos programáveis, sendo muitas vezes razoável combinar o projeto no *FPGA* com uma unidade controladora capaz de executar tarefas complexas (mas não críticas), como os *DSPs*. Os testes foram efetuados em uma implementação inicial em que o *FPGA* atuou como um módulo de entrada/saída para um *DSP* e como uma unidade de processamento de dados em lote, e devido a limitações de memória e velocidade de processamento, este módulo é utilizado somente para aplicações de processamento de imagens e gráficos 2D e em *wireframe*. O acesso à memória é efetuado somente via controladores *DMA* (inclusos no *DSP*), em um sistema dado pelo diagrama da Figura 25, onde a unidade de processamento (ou grupos de unidades) no *FPGA* implementou algoritmos como filtro de convolução 3x3 com coeficientes fixos, filtros morfológicos e funções em pixels. Um exemplo desta primeira implementação é a aceleração no processamento da filtragem de convolução 3x3 de até 5 milhões de pixels por segundo, com um incremento médio na velocidade de processamento, em relação a uma implementação tradicional em um processador de propósitos gerais, em 20 vezes, utilizando funções com janelas de dimensões 3x3. Na segunda implementação, o projeto foi atualizado para um sistema capaz de acelerar gráficos 3D e o processamento de imagem, com as seguintes alterações: uso de um *DSP* com velocidade de processamento próxima à dos computadores pessoais, uso de um modelo mais avançado de *FPGA* com memória local (pequena capacidade), memórias de maior capacidade e suporte à reconfiguração dinâmica em tempo real. O diagrama em blocos deste segundo sistema é exibido na Figura 26, para a aplicação de renderização de volume baseado na ordem de objeto, com

resultados de desempenho superiores à primeira implementação, porém, os algoritmos que utilizam dados de volumes 3D apresentaram gargalos de processamento em relação à taxa de transferência da memória externa. A primeira e a segunda implementação de Zemcik (2002) mostram a viabilidade e as vantagens da reconfiguração dinâmica em aplicações de processamento de imagens e renderização, assim como a inclusão de sistemas como circuitos adicionais para a visualização de dados, e etapas de um *pipeline* de visualização 3D.

Figura 25 – Primeira implementação do sistema *DSP-FPGA* proposto por Zemcik (2002). Adaptado de: (ZEMCIK, 2002).

Figura 26 – Segunda implementação do sistema *DSP-FPGA* proposto por Zemcik (2002). Adaptado de: (ZEMCIK, 2002).

O trabalho de Bensaali et al. (2003) foi o primeiro a descrever os resultados do cálculo matricial para transformações afins 3D implementado em *FPGA*, visando a utilização dos *FPGAs* como aceleradores de baixo custo com performance computacional entre os sistemas com processadores de uso geral e os *ASICs*. Os circuitos foram implementados de modo a

utilizar matrizes de grandes dimensões através da Handel-C, uma linguagem de programação similar à linguagem C que suporta paralelismo e tamanho dos dados flexível, facilitando a avaliação do projeto e sendo mais adequada ao projeto de *IP cores*. A arquitetura proposta baseia-se em elementos de processamento (EPs), cujo número é um múltiplo de quatro, e cada EP consiste em um ACM-Acumulador Multiplicador (*MAC-Multiplier Accumulator*) e um registrador para armazenamento do resultado obtido. Foram utilizadas duas aproximações: ACM baseado na biblioteca ponto fixo da Celoxica e baseado no CoreGen da Xilinx, sendo que a primeira é uma biblioteca independente do *hardware* e possibilita ao projetista definir o tamanho da parte inteira e fracionária dos dados e macros para executar as operações aritméticas, e a segunda possui *designs* prontos para serem utilizados, reduzindo o tempo de projeto e de elaboração da interface para a integração dos blocos CoreGen com Handel-C. Os resultados das duas aproximações são exibidos na Tabela 10.

Tabela 10 – Resultados do multiplicador paralelo de matrizes implementado pelas funções das bibliotecas de ponto fixo (Celoxica) e CoreGen (Xilinx). Fonte: (BENSAALI ET AL., 2003).

ACM (Acumulador Multiplicador)	Número de EPs (elementos de processamento)	Área (% da área máxima disponível)	Velocidade (MHz)
Biblioteca ponto-fixo Celoxica	4	17	22
	8	35	20
	16	75	17
CoreGen Xilinx	4	10	35
	8	20	30
	16	42	24

Haselman et al. (2005) desenvolveram uma biblioteca de números logarítmicos parametrizados como alternativa às bibliotecas de ponto flutuante, efetuando uma comparação entre bibliotecas dos dois tipos. Os números logarítmicos oferecem uma simplificação nas operações aritméticas computacionais, exceto na adição e subtração, e necessita do processo de conversão entre formatos. Os resultados obtidos consideram as operações de multiplicação, divisão, adição, subtração e conversão de formatos, determinando quando um formato deve ser utilizado em relação a outro assim como a vantagem da mudança entre formatos durante o cálculo computacional. A Tabela 11 exibe os resultados da área ocupada (normalizada) e latência para as quatro operações fundamentais com dados nas precisões simples e dupla nos

dois formatos, e a Tabela 12 mostra os resultados para a conversão entre os formatos ponto flutuante e número logarítmico. O formato logarítmico permite uma análise comparativa de desempenho computacional, com os formatos de ponto fixo e ponto flutuante, na conversão e no processamento dos dados, de acordo com a aplicação.

Tabela 11 – Área, recursos utilizados e latência para as operações de multiplicação, divisão, adição e subtração nos formatos de ponto flutuante (PF) e números logarítmicos (NL), em simples e dupla precisão. Fonte: (HASELMAN ET AL., 2005).

Precisão	Multiplicação				Divisão				Adição				Subtração			
	Simples		Dupla		Simples		Dupla		Simples		Dupla		Simples		Dupla	
Formato	PF	NL	PF	NL	PF	NL	PF	NL	PF	NL	PF	NL	PF	NL	PF	NL
Fatias	297	20	820	36	910	20	3376	36	424	750	836	1944	424	838	836	2172
Multiplicadores	4	0	9	0	0	0	0	0	0	24	0	72	0	24	0	72
18K BRAM	0	0	0	0	0	0	0	0	0	4	0	8	0	8	0	16
Unidades por FPGA	14	537	6,2	298	12	537	3,2	298	25	2,33	12,9	0,78	25	2,33	12	0,78
Área normalizada	368	20	981	36	910	20	3376	36	424	1291	836	3456	424	1491	836	3907
Latência (ns)	65	10	83	12	150	10	350	12,7	45	91	67	107	45	95	67	110

Tabela 12 – Área, recursos necessários do FPGA e latência para a conversão entre os formatos de ponto flutuante (PF) e números logarítmicos (NL). Fonte: (HASELMAN ET AL., 2005).

Precisão	Conversão PF-NL		Conversão NL-PF	
	Simples	Dupla	Simples	Dupla
Fatias	163	7631	236	3152
Multiplicadores	0	0	20	226
18K BRAM	24	0	4	14
Unidades por FPGA	2,333	1,4	2,8	0,25
Área normalizada	832,6	7712	8,66	9713
Latência (ns)	76	380	72	84

A utilização do sistema logarítmico em relação ao sistema de ponto fixo e de ponto flutuante fica a cargo do projetista do sistema, com o número de operações da aplicação a ser desenvolvida de acordo com os resultados anteriores para sua otimização.

Holten-Lund (2005) mostra em seu trabalho a utilização de *FPGAs* como aceleradores gráficos 3D em sistemas embarcados, e a implementação de uma *CPU* capaz de executar o sistema operacional μ Linux, específico para sistemas embarcados. Este sistema efetua o processamento de tarefas normalmente executadas via *software*, para as quais não existem *ASICs* específicos. O sistema de aceleração de gráficos utiliza o modelo da arquitetura de gráficos Hybris, que apresenta a possibilidade de implementação mínima em sistemas embarcados até implementações em sistemas de processamento paralelo de alto desempenho computacional, utilizando a divisão do quadro de uma cena em janelas retangulares para a renderização. O número de janelas e sua dimensão podem ser ajustáveis de acordo com a capacidade de processamento de cada sistema, possibilitando a renderização simultânea de mais de uma janela em paralelo. Os testes do *core* de renderização foram efetuados por simulação e por circuitos sintetizados no *FPGA*, com uma *engine* de visualização utilizando janelas retangulares de 32x32 *pixels*, capaz de renderizar um triângulo básico (triângulo de lados de um *pixel*) por ciclo do sinal de *clock*, ocupando 2.107 fatias e 3 *RAMB16* (Blocos de memória *RAM* de 16 *bits* sintetizados em *FPGAs* Xilinx), com possibilidade de operar até 70MHz (taxa de processamento de 70 milhões de triângulos básicos/s.) na simulação. Na prática, este *core* foi limitado pela taxa de transferência máxima do barramento com a memória externa. Foi proposta pelos autores a inclusão de unidades de processamento vetoriais em *cores* separados para o balanceamento do sistema de gráficos, para gerar primitivas como os triângulos e pontos de curvas *splines*, assim como renderizá-los à mesma taxa de processamento.

Ouellette e Connors (2005) depreciam em seu trabalho uma análise da aceleração via *hardware*, comparando duas plataformas, sendo uma constituída por processadores de propósitos gerais (*CPUs*) implementados em lógica configurável, e outra constituída por circuitos de *hardware* configurável, destinados a uma aplicação específica. Esta análise foi efetuada em aplicações de operações básicas de comunicação de dados e de ponto flutuante, considerando o custo adicional de recursos de *hardware*, potência consumida e área utilizada, para transferir rotinas de uma biblioteca em *software* para blocos de *hardware* em um sistema embarcado de lógica configurável baseado no processador Microblaze. São citados exemplos de plataformas de lógica configurável como a Xilinx XC-2VP7, que possui 11.088 células lógicas, 792kbit de *RAM on-chip*, 44 macrocélulas para multiplicação, e um procesador embarcado PowerPC 405 de 400Mhz; a placa de processamento em lógica configurável Xilinx ML300-1, com o *FPGA* XC-2VP7, oito interfaces *gigabit* 1000BaseT ou conexões *Infiniband*. Outros modelos da série Xilinx Virtex-II Pro possuem mais de quatro *CPUs*

PowerPC 405 e 24 *ports* de entrada/saída *gigabit* Rocket, capazes de formar um ambiente computacional de desenvolvimento quando integrados com dispositivos de interface externa como teclados, monitores, interfaces *Ethernet* e controladores *IDE*, executando o sistema operacional Linux nos processadores PowerPC. A Figura 27 exibe o tempo de processamento de uma biblioteca para as operações básicas, em ponto flutuante, emulada por *software* no processador Microblaze, em relação às mesmas operações aceleradas em *hardware*, resultando em um tempo em média 30% menor nesta implementação via *hardware*. A arquitetura do processador Microblaze oferece a *FSL – Fast Simplex Link*, uma interface de comunicação ponto a ponto de alta velocidade, como opção para a conexão entre a unidade de *hardware* de ponto flutuante e os outros circuitos componentes deste processador. A Tabela 13 exibe os valores obtidos para os recursos alocados no sistema configurável, incluindo ou não a interface *FSL*, que resulta na utilização de mais recursos de *hardware*. As métricas utilizadas para avaliar o sistema foram o número de portas lógicas básicas usadas em todo o *design* e o número de fatias de elementos lógicos utilizados no *FPGA*, com o fator de ponderação sendo a razão entre os recursos de hardware utilizados com o *FSL* efetuando a comunicação com a *FPU*, e sem o *FSL*. A Tabela 14 exibe os valores de potência consumida, para duas configurações de um circuito para o cálculo da transformada cosseno discreta; a primeira configuração consiste no cálculo sequencial com a comunicação serial do *FSL*, e a segunda configuração implementa operações em grupos de vetores de forma paralela. “*Software* puro” é a denominação dos autores para a implementação da aplicação somente via *software* (comandos) do processador Microblaze; e como esperado, o consumo na configuração de operações em paralelo exige mais potência pelo maior número de circuitos em funcionamento em determinado instante.

Figura 27 – Tempo de processamento para a biblioteca de funções em ponto flutuante em relação à aceleração via *hardware* (em nanosegundos). Fonte: (OUELLETTE E CONNORS, 2005).

Tabela 13 – Utilização de recursos de *hardware* para a *FSL – FPU*. Fonte: (OUELLETTE E CONNORS, 2005).

Tipo de lógica	Sem <i>FSL/FPU</i>	Com <i>FSL/FPU</i>	Incremento	Fator de ponderação
Contagem de portas lógicas	2158761	2191646	1,5%	0,98
Fatias	744	1519	104,16%	0,48

Tabela 14 – Potência consumida dos *cores FSL e FPU* para a Transformada Cosseno Discreta. Fonte: (OUELLETTE E CONNORS, 2005).

Tipo de lógica	<i>Software</i> puro	<i>FSL FPU</i>	1ª TCD	2ª TCD
Potência (mW)	1406,22	1441,78	1441,78	1622,71
Incremento em relação ao <i>software</i> puro	n.a.	2,53%	2,53%	15,39%

O trabalho de Bensaali e Amira (2006) descreve os resultados obtidos com a implementação das transformações afins 3D de matrizes densas em *FPGAs* em ponto fixo, utilizando duas bibliotecas (ponto fixo Celoxica e CoreGen Xilinx), e a implementação em *pipeline* utilizando a biblioteca de dados de ponto flutuante, da Celoxica. Os resultados do tempo de processamento dessas três implementações foram comparados, em duas plataformas de lógica reconfigurável baseadas em *FPGAs*: Virtex-II, com 168 blocos multiplicadores, e Virtex-II Pro, com 556 blocos multiplicadores e com uma frequência de processamento maior em relação ao Virtex-II. Foram incluídos os resultados de uma implementação similar em uma placa aceleradora gráfica Radeon FSC de 32 MB, conforme exibido pela Tabela 15.

Che et al. (2008) apresentam um estudo do desempenho computacional das aplicações de eliminação Gaussiana, *DES - Data Encryption Standard* e o Needleman-Wunsch, um algoritmo para alinhamento global em duas sequências de dados, comumente utilizado em bioinformática para alinhar sequências de proteínas ou nucleotídeos. Estes algoritmos foram implementados em um *FPGA*, uma *GPU* e uma *CPU* multicore, sendo efetuada a comparação do comportamento destas aplicações nestes aceleradores. Foi considerado o desempenho computacional e a complexidade do código, para determinar o mapeamento e o particionamento dessas aplicações em fases, que podem ser executadas nos aceleradores disponíveis (além da *CPU* de propósitos gerais) da maneira mais eficiente e econômica em termos de custos. Este trabalho forneceu como resultados as vantagens e desvantagens das plataformas de *hardware FPGA* e modelos de programação comparados com plataformas *GPU*.

Tabela 15 – Comparação do tempo de processamento da transformação de rotação 3D (de um objeto de 2154 vértices), em *FPGA* e em uma placa gráfica Radeon FSC 32 MB. Fonte: (BENSAALI E AMIRA, 2006)

ACM (Acumulador Multiplicador)	Número de EPs (elementos de processamento)	Frequência máxima (MHz)		Tempo de processamento (μ s)	
		V-II	V-II Pro	V-II	V-II Pro
Biblioteca ponto-fixo Celoxica	4	36	52	134,74	93,28
	8	30	45	89,88	59,92
	16	26	38	36,33	24,85
CoreGen Xilinx	4	83	114	58,43	42,55
	8	78	110	34,57	24,51
	16	61	102	13,30	9,26
Biblioteca ponto flutuante em pipeline Celoxica	4	108	119	27,92	25,34
Placa gráfica Radeon FSC 32 MB	---	---	---	---	14,81

Adicionalmente, foi incluída uma análise das três aplicações, tendo como aspecto principal as diversas características de desempenho que permitem a aceleração computacional, e uma proposta de mapeamento dessas características da aplicação para uma plataforma específica, levando em conta os requisitos de métrica, custo de programação e desempenho. A *CPU multicore* consiste na plataforma principal de teste, operando tanto em *single thread* (processamento sequencial) e em quatro *threads* simultâneos em *OpenMP*, tornando esta implementação mais independente da arquitetura *multicore*. Para o processamento de um bloco de 64 *bits* do algoritmo *DES*, o *FPGA* exige somente 83 ciclos do sinal de *clock*, enquanto a *GPU* necessita de $5,8 \times 10^5$ ciclos, devido à falta de suporte para operações necessárias a esta aplicação. Estas operações, devido à falta de suporte da *GPU*, utilizam uma característica do *hardware* da *GPU*: o *design* de ocultação da latência. Os *FPGAs* podem executar um cálculo *bit a bit* em um ciclo do sinal de *clock*, e dados intermediários de 64 *bits* podem ser armazenados em *flip-flops*, enquanto a *GPU* utiliza tempos de acesso (para carregar *look up tables* necessárias aos algoritmos) e sincronização com a memória externa, de 300 a 400 ciclos. Esta diferença no tempo de execução entre o *FPGA* e a *GPU* tende a diminuir com o aumento do tamanho dos dados, segundo os autores, pois requer acesso do *FPGA* a uma memória externa. Os resultados para os algoritmos de eliminação Gaussiana são exibidos na Figura 28 e para o algoritmo Needleman-Wunsch, na

Figura 29. Estes resultados demonstram a viabilidade do *FPGA* como acelerador, como alternativa à *GPU*, oferecendo a flexibilidade na sintetização de circuitos a partir de portas lógicas. As *GPUs* possibilitam uma programação em alto nível, com um custo e tempo de processamento menores, mas estão vinculadas à arquitetura *multithreading* e à dificuldade de implementação em sistemas embarcados.

Figura 28 – Ciclos do sinal de *clock* para a execução do algoritmo de eliminação Gaussiana, com o tamanho da entrada representando o tamanho do *grid* de computação para a *GPU*, *CPU* com processamento de um único *thread*, 4 *threads* simultâneos, e para o *FPGA*. Fonte: (CHE ET AL., 2008).

Figura 29 - Ciclos do sinal de *clock* para a execução do algoritmo Needleman-Wunsch, com o tamanho da entrada representando o tamanho do *grid* de computação para a *GPU*, *CPU* com processamento de um único *thread*, 4 *threads* simultâneos, e para o *FPGA*. Fonte: (CHE ET AL., 2008).

Os trabalhos de Cope (COPE ET AL., 2005; 2008) analisam a utilização colaborativa e substitutiva de duas alternativas para a implementação de técnicas de processamento digital de vídeo: os processadores gráficos e a lógica reconfigurável. Conforme exibido pela Figura 30, até o ano de 2010, a previsão da tecnologia empregada na fabricação dos dispositivos *FPGAs* sempre apresenta uma evolução em relação à tecnologia utilizada para as *GPUs*. Isto se deve ao fato de que os *chips* processadores gráficos possuem uma arquitetura mais complexa em relação aos *FPGAs*, exigindo um projeto mais complexo na ampliação de seus circuitos, enquanto os *FPGAs* utilizam os mesmos elementos construtivos básicos para a ampliação de sua capacidade (elementos lógicos ou blocos lógicos), incrementando somente o número destes elementos pela replicação de seus respectivos circuitos. Além disto, os algoritmos de processamento digital de vídeo geralmente não utilizam todas as características presentes em processadores gráficos. No trabalho de Coppe foi apresentada uma estrutura aproximada para a comparação entre arquiteturas para processadores de vídeo digital, com um esquema proposto dos “Três Eixos da Caracterização de Algoritmo”, e uma formulação para *drivers* de desempenho. Esta aproximação determinou os fatores vantajosos dos processadores gráficos e da lógica reconfigurável, e as condições para determinar qual das duas tecnologias é superior no processamento de vídeo digital. Os resultados desta comparação sugeriram opções de customização para a arquitetura do processador gráfico, considerado como parte de um tipo de arquiteturas denominadas arquiteturas de elemento de multi-processamento homogêneo (*HoMPE* - *Homogeneous Multi-Processing Element*). Nesta classe de arquitetura, o elemento de multi-processamento pode ser um processador, uma função fixa ou um barramento reconfigurável, com modelos de computação de processamento paralelo. O autor descreveu uma nova ferramenta de exploração, adequada à verificação das opções de customização das arquiteturas *HoMPE*. Como resultado dessa verificação, foi proposta uma *engine* reconfigurável para acesso de dados (*REDA* – *Reconfigurable Engine for Data Access*), com o objetivo de otimizar o desempenho do processador gráfico, de acordo com padrões de acesso à memória específicos para o processamento de vídeo digital. Esta *engine REDA* demonstrou a viabilidade da utilização da lógica reconfigurável como interconexão lógica colaborativa na arquitetura do processador gráfico.

Os resultados obtidos por Coppe demonstraram que o impacto da representação numérica adotada e da área ocupada no dispositivo de lógica reconfigurável necessitaram de análise adicional, em modos diferentes de implementação. O processador gráfico apresentou uma tecnologia superior para algoritmos com heurística de reutilização de dados variáveis,

como a interpolação, ou os que exigem muitos cálculos aritméticos, como correção de cores primárias. A lógica reconfigurável apresentou uma tecnologia superior para algoritmos com alto grau de dependência de dados, em particular nos casos em que há restrição na iteração entre níveis de paralelismo, como a estimativa do vetor de movimento, mas é inferior se for necessário um grande número de acessos à memória por núcleo de cálculo, como na convolução 2D. O autor observou que o processador gráfico apresenta um grau de desempenho computacional muito similar aos dispositivos de lógica reconfigurável, com uma diferença entre as tecnologias, de uma ordem de magnitude. Em relação ao processador gráfico, o *engine* para acesso de dados (*REDA*) incrementou até em uma ordem de magnitude os algoritmos com baixo desempenho computacional, ocupando uma área estimada de 2% da área equivalente ocupada no processador gráfico. As propostas apresentadas para classes de arquiteturas, que incluem a lógica reconfigurável e um processador gráfico na arquitetura *HoMPE*, são mostradas na Figura 31.

Figura 30 – Tamanho utilizado pela tecnologia do processo de fabricação dos dispositivos processadores gráficos da nVidia, FPGAs da Xilinx e a tendência de evolução da tecnologia de fabricação destes dispositivos fornecida pelo ITRS - International Technology Roadmap for Semiconductors. Fonte: (COPPE, 2008).

Figura 31 – Classes de arquitetura para a utilização da lógica reconfigurável em uma Arquitetura de Elemento de Multi-Processamento Homogêneo (HoMPE). Fonte: (COPPE, 2008).

Dean et al. (2008) compararam em seu trabalho o processamento da simulação da dinâmica molecular utilizando as arquiteturas *multithreading* das *GPUs*, com o mesmo processamento em circuitos *FPGAs*, em um estudo de caso para uma aplicação com um nível difícil de paralelismo, explorando os recursos de *FPGAs*, em que o mesmo modelo pode ser utilizado em processadores *multicore*. Essa simulação baseou-se em eventos moleculares discretos, dados por parâmetros aplicáveis a cada molécula em um determinado sistema, sendo que mais de um evento pode ser simulado simultaneamente em sistema *multithreading*. Nesta implementação, foi obtida uma taxa de 50M eventos/s na simulação de eventos discretos (*DES – Discrete Event Simulation*), com os resultados obtidos exibidos na Tabela 16 para três modelos de simulação de um gás ideal uniforme com 131.000 partículas, cada modelo de 1 a 3 é sobrecarregado respectivamente com o dobro de parâmetros para cada evento, *BL* consiste no *baseline code* para cada evento indicando o tempo de simulação necessário. Os testes foram efetuados para simulações utilizando até quatro *threads* simultâneos, em uma tentativa de se aproximar do processamento de uma *CPU quad core*.

Tabela 16 – Comparação entre as arquiteturas *FPGA* e *multicore* para três modelos de simulação de eventos discretos de dinâmica molecular, para até 4 *threads* simultâneos (4 *cores*). Fonte: (DEAN ET AL., 2008).

<i>Threads</i>	Modelo 1	Modelo 2	Modelo 3
BL	10,0 μ s	57,6 μ s	523 μ s
1	0,81x	1,00x	1,00x
2	0,79x	1,64x	1,92x
3	0,47x	2,20x	2,80x
4	0,23x	2,39x	3,65x

Franchini et al. (2009) implementaram um *IP core* de coprocessamento paralelo embarcado que oferece suporte direto a *hardware* para os operadores da Álgebra de Clifford, denominado coprocessador Clifford. A álgebra geométrica, ou Álgebra de Clifford, fornece um modo natural de modelar objetos e cenas, além de constituir uma poderosa ferramenta para cálculo simbólico. A arquitetura proposta para este coprocessador foi implementada em uma placa *FPGA* baseada em um interface para barramento *PCI*, visando aplicações em sistemas embarcados de visão computacional e computação gráfica, como parte de plataformas robóticas ou placas gráficas dedicadas, com suporte a operandos homogêneos 4D (escalares, vetores, bivectores, trivetores e pseudoescalares) e suas respectivas operações (produtos geométricos, produto vetorial, contrações à esquerda e direita, somas e diferenças), como na maioria das bibliotecas gráficas disponíveis. Os resultados são exibidos na Figura 32 para quatro operações da álgebra de Clifford, em um computador pessoal sem o coprocessador Clifford, executando o gerador de biblioteca de álgebra geométrica padrão GAIGEN, e com o coprocessador (CliffoSor).

O trabalho de Woo et al. (2010) descreve as implementações comerciais de circuitos de processamento de *pipelines* gráficos 3D em sistemas embarcados, visando aplicações em dispositivos móveis, que exigem um projeto voltado à eficiência com baixo consumo de energia. Algumas técnicas apresentadas para o processamento gráfico, podem ser utilizadas em outras aplicações, como o formato de números logarítmicos, que fornece um desempenho computacional superior para a geração de gráficos sem perdas na qualidade dos gráficos. Para a computação gráfica móvel, foram apresentadas as APIs KAIST MobileGL, Khronos OpenGL-ES (KHRONOS OPENCL WORKING GROUP, 2011) e a Microsoft's Direct3D-Mobile, com as implementações em *Soc* KAIST RAMP Architecture IV a VII e as arquiteturas proprietárias nVidia Mobile GPU – SC10 e Tegra, o Sony PSP e a Imagination Technology MBX/SGX.

Figura 32 – Resultados em ciclos de *clock* do processamento por um computador pessoal das operações de produto, soma, diferença e rotação 3D, da álgebra de Clifford; com o *core* coprocessador Clifford (CliffoSor) e via gerador de biblioteca GAIGEN. Fonte: (FRANCHINI ET AL., 2009).

O desempenho computacional de *pipelines* gráficos 3D em sistemas portáteis foi verificado com a biblioteca gráfica móvel embarcada Mobile-GL, otimizada para aritmética em ponto fixo em *CPUs* embarcadas de dispositivos portáteis em que a aritmética em ponto flutuante não está implementada. A Figura 33 exibe a taxa de preenchimento de pixel com esta biblioteca sendo executada em um processador PXA-255 de 400MHz, demonstrando que a renderização é o principal estágio em um *pipeline* gráfico 3D que deve ser acelerado em *hardware*, pelo fato de exigir maior desempenho computacional. A geração de curvas e superfícies *b-splines* podem auxiliar e complementar estes modelos de *Soc* para a mudança de escala, preparando os pontos de uma superfície para a poligonação.

Figura 33 – Desempenho computacional gráfico obtido pela biblioteca Mobile-GL para os processadores StrongARM de 200MHz e PAX255 de 400MHz para o preenchimento de pixels (a) e vértices (b). Fonte: (WOO ET AL., 2010).

2.4. Considerações Finais

Neste capítulo foram apresentados artigos relacionados a *b-splines* e *FPGA*. Os métodos matemáticos para *b-splines* estão bem fundamentados desde o trabalho de Farin (1997) com o algoritmo de Cox-de Boor, utilizado em sistemas computacionais como alternativa ao algoritmo de Casteljau.

O trabalho de Gopi e Manohar (1997) para *ASICs* torna a sua implementação única em relação ao processamento paralelo em *hardware*, com os melhores resultados em relação ao desempenho computacional. O método de Ou et al. (2002) apresenta uma alternativa melhor em relação ao método de Gopi e Manohar no aspecto de desempenho computacional, mas somente para casos particulares de configurações dos parâmetros das *b-splines* (valores uniformes dos nós) e às custas do quesito de *hardware*, i.e., área ocupada no circuito integrado ou dispositivo configurável, sendo dependente no grau da *spline*. Porém, estes métodos foram desenvolvidos com objetivo de implementação em circuitos integrados, não sendo direcionados a *FPGAs*. A implementação em lógica reconfigurável constitui-se em uma alternativa interessante a ser levada em conta em relação às implementações existentes, motivando a adaptação destes métodos à lógica reconfigurável, que é um dos objetivos do presente trabalho.

Um dos poucos projetos propostos para a implementação das *b-splines* em lógica reconfigurável é o circuito *fast Cox-de Boor* de Yau et al. (2006), o qual segue a lógica do processamento de diversos pontos de uma curva ou superfície simultaneamente. Porém, o autor destaca principalmente a importância do *Soc* em geral, sem apresentar detalhes referentes à implementação das *b-splines* em um circuito *FPGA*. No entanto, no único item avaliado pelos autores do circuito *fast Cox-de Boor*, o de desempenho computacional, este circuito apresentou um resultado melhor em relação ao método de Gopi e Manohar.

Considerando somente o número de ciclos do sinal de *clock* utilizado, os métodos de geração de pontos de uma *b-spline* em *FPGAs*, propostos por Ou et al. (1997) e o *fast Cox-de Boor* (YAU ET AL., 2006), demonstram um desempenho computacional significativamente melhor em relação aos computadores pessoais (baseados no *PC*). Os resultados apresentados no trabalho original do método *fast Cox-de Boor* somente levam em consideração o número de sinais de *clock* necessários para um circuito do triângulo Cox-de Boor, em relação ao circuito de um elemento de processamento do *BFEA* proposto por Gopi e Manohar. Estes

resultados motivaram uma análise mais detalhada no presente trabalho, de avaliar estes métodos em relação a um número diferente de pontos de controle (dados de entrada).

Como mencionado por Piegl (1991) e Tiller (PIEGL E TILLER, 1997), as *b-splines* não representam com precisão as seções cônicas, que exigem uma representação matemática racional. As *NURBS* possibilitam uma representação mais precisa das seções cônicas, mas para a circunferência *NURBS*, são necessários dados e ajustes adicionais em relação à representação paramétrica convencional da circunferência. A representação paramétrica do segmento de reta é similar à representação por *b-splines* de grau 1, que utilizam somente uma combinação convexa. Porém, as *splines* exigem como dados de entrada ao menos dois pontos de controle (pontos inicial e final da reta *b-spline*), os valores do parâmetro e os valores dos nós, enquanto a representação paramétrica convencional de um segmento de reta exige somente o ponto inicial, o ponto final, e os valores dos parâmetros. Por estas razões, no presente trabalho, foram implementados *cores* para a geração de segmentos de reta e de segmentos de circunferência, baseados nos trabalhos de Pang (1990) e Wright (1990), que são adaptações dos algoritmos de Bresenham do ponto médio para o processamento paralelo, comparando-se o seu tempo de execução e número de elementos lógicos necessários com a implementação das *b-splines* no *FPGA*.

Os trabalhos relacionados às *b-splines* e à lógica reconfigurável, aplicados em sistemas de manufatura, descrevem a importância do processamento deste tipo de primitiva geométrica, principalmente no cálculo da trajetória da ferramenta de um sistema de manufatura, e na compactação dos dados geométricos. A aplicação descrita por Dong e Wang (2012) exemplifica uma utilização dos sistemas reconfiguráveis na predição da trajetória da ferramenta de um sistema, pelo cálculo dos pontos de interpolação desta trajetória, que permitem o melhor controle da aceleração e desaceleração da ferramenta.

3. Fundamentação Teórica

3.1. Considerações Iniciais

Neste capítulo será apresentada a teoria utilizada no desenvolvimento do projeto desta tese, incluindo conceitos matemáticos das *b-splines*, os tipos de dados necessários, e o algoritmo Cox-de Boor para a geração de curvas e superfícies. O apêndice A apresenta uma introdução mais detalhada sobre a criação das *splines* partir do modelo matemático de deformações de barras e superfícies, a dedução das funções base (*b-splines*) e características, até o tipo específico de *b-splines*, as *NURBS*, incluindo as vantagens e desvantagens na sua representação.

3.2. Conceitos sobre *B-Splines*

Matematicamente, uma curva *b-spline* de grau p é uma sequência de segmentos polinomiais, conectados para formar uma curva contínua, definida como (ROGERS, 2001):

$$C(t) = \sum_{i=0}^n P_i N_{i,p}(t) \quad (5)$$

onde t é o valor do parâmetro adimensional relativo a cada ponto gerado, P_i são os pontos que formam o chamado polígono de controle, formado por $n+1$ pontos de controle, e $N_{i,p}(t)$, $i = 0, \dots, n-1$, são as funções base *b-spline* definidas sobre um vetor de nós U , fornecidas pelas fórmulas recursivas de Cox-de Boor (BOOR, 1972):

$$U = \{u_0, \dots, u_m\}, u_i \leq u_{i+1}, i = 0, \dots, m \quad (6a)$$

$$N_{i,0}(t) = \begin{cases} 1 & \text{se } u_i \leq t < u_{i+1} \\ 0 & \text{caso contrário} \end{cases} \quad (6b)$$

$$N_{i,p}(t) = \frac{t-u_i}{u_{i+p}-u_i} N_{i,p-1}(t) + \frac{u_{i+p+1}-t}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(t) \quad (6c)$$

O vetor de nós possui a seguinte forma:

$$U = \{ \underbrace{a, a, \dots, a}_{p+1}, u_{p+1}, \dots, u_{m-p-1}, \underbrace{b, b, \dots, b}_{p+1} \} \quad (7)$$

onde, em muitas aplicações práticas, $a = 0$ e $b = 1$. A Figura 34 mostra uma curva *b-spline* de grau 3, com 100 pontos gerados a partir de 5 pontos de controle.

Figura 34 – Curva *b-spline* (linha pontilhada) de 100 pontos a partir de 5 pontos de controle P (linha cheia) e pesos unitários (*b-spline*).

Similarmente, uma superfície *b-spline* de grau (p, q) é definida como:

$$S(u, v) = \sum_{i=0}^n \sum_{j=0}^m P_{i,j} N_{i,p}(u) N_{j,q}(v) \quad (8)$$

onde t e v são os valores dos parâmetros nas direções longitudinal e isoparamétrica da construção da superfície, $P_{i,j}$, $i = 0, \dots, n$; $j = 0, \dots, m$, onde o conjunto dos valores $P_{i,j}$ forma a rede, malha, ou poliedro de controle. As funções base $N_{i,p}(u)$, $i = 0, \dots, n$, e $N_{j,q}(v)$, $j = 0, \dots, m$, são definidas pelas equações (6a) a (6c) (a definição de $N_{j,q}(v)$ é similar), sobre os vetores de nós:

$$U = \{u_0, \dots, u_r\}, u_i \leq u_{i+1}, i = 0, \dots, r-1 \quad (9a)$$

$$V = \{v_0, \dots, v_s\}, v_j \leq v_{j+1}, j = 0, \dots, s-1 \quad (9b)$$

Cada vetor de nós corresponde a uma “direção” de formação da superfície, como pontos no sentido longitudinal e transversal. A Figura 35 mostra uma superfície *b-spline* de grau 3x3, com 1.000 pontos gerados a partir de 16 pontos de controle (16x16). A ordem dos pontos de controle causa uma torção na superfície final.

Figura 35 – Superfície *b-spline* de 10.000 pontos (100x100) a partir de 16 pontos de controle *P* e pesos unitários (*b-spline*).

3.3. Algoritmo Cox-de Boor

Um método alternativo para o cálculo dos pontos resultantes de uma curva ou superfície *b-spline* consiste nas combinações convexas dos pontos de controle ponderados

pela combinação dos valores dos parâmetros e dos nós atuantes, dado pelo algoritmo de de Boor (FARIN, 1997):

$$C_i^j(t) = \left(1 - \frac{t - u_i}{u_{i+k-j} - u_i}\right) P_{i-1}^{j-1} + \left(\frac{t - u_i}{u_{i+k-j} - u_i}\right) P_i^{j-1} \quad (10)$$

$C_i^j(t)$ são os pontos gerados para a curva *b-spline*, t é o parâmetro adimensional que corresponde a um ponto na curva (alterando o número de valores de t , altera-se a resolução da curva), i corresponde ao índice do vetor de nós onde se encontra o valor de t , j é a camada do triângulo de Boor, k a ordem da curva (grau + 1) e P_i os pontos de controle. De acordo com a Figura 36, que ilustra um diagrama esquemático do algoritmo de de Boor para uma curva *b-spline* de grau $p=2$, a equação (6) é aplicada aos pontos de controle (valores P_n^0 da camada $p-2$ do diagrama) para cada ponto (valor de t) a ser gerado na curva final. Os valores resultantes deste primeiro cálculo são novos pontos de controle na camada $p-1$, cada um gerado a partir de dois pontos de controle dados da camada anterior. A equação (6) é aplicada novamente, resultando em novas camadas de pontos de controle, até completarem-se p camadas. Os mesmos pontos de controle originais (camada inicial) são utilizados para o cálculo dos pontos da curva que possuem o valor do parâmetro t dentro do mesmo intervalo de valores do vetor de nós $[u_i, u_{i+1}]$.

Figura 36 – Algoritmo recursivo de Boor para curvas *b-splines* de grau 2 (3 camadas).

Cada ponto gerado equivale a uma série de combinações convexas recursivas que geram novos pontos de controle de curvas equivalentes de menor grau. O número de combinações convexas é dependente do grau da curva final, gerando novos pontos de controle até o ponto da curva final. A Figura 37 mostra o ponto de uma curva *b-spline*, equivalente ao cálculo do parâmetro de valor 0,6, gerado a partir das combinações convexas dos pontos de controle. O algoritmo de de Boor é aplicado para cada coordenada utilizada no sistema (2 para sistemas 2D, 3 para sistemas 3D), sendo que no caso das *b-splines* racionais, os pesos são considerados uma coordenada a mais, denominada coordenada homogênea (terceira coordenada em sistemas 2D, quarta coordenada em sistemas 3D).

Figura 37 – Pontos de controle resultantes do algoritmo recursivo de Boor para a geração do ponto de parâmetro 0,6 na curva *b-spline*.

Para superfícies, o algoritmo de Boor é executado uma vez, de acordo com a Figura 38, gerando camadas de pontos de controle com a orientação dada pelo vetor de nós no sentido longitudinal, i.e., gerando curvas isoparamétricas em um sentido (Figura 39 e Figura 40). Os pontos destas curvas isoparamétricas são os pontos de controle dados pelo vetor de nós no sentido isoparamétrico (Figura 41), e a respectiva aplicação do algoritmo de de Boor, com este vetor de nós, resulta nos pontos da superfície final (Figura 42).

Figura 38 – Algoritmo recursivo de Boor para superfícies, de topologia em pontos longitudinais e isoparamétricos.

Figura 39 – Pontos de controle de uma superfície *b-spline* no sentido longitudinal e isoparamétrico, formando um poliedro de controle de 16 pontos.

Figura 40 – Curvas isoparamétricas geradas pelo algoritmo de de Boor.

Figura 41 – Superfície final gerada pelas curvas isoparamétricas.

Figura 42 – Superfície final.

4. Materiais e Métodos

4.1. Considerações Iniciais

O capítulo apresentado a seguir fornece uma visão geral do projeto, tema deste trabalho de tese, com os módulos de sua composição geral, os métodos utilizados no desenvolvimento do projeto, detalhes sobre a implementação de alguns módulos, algoritmos ou circuitos em que cada módulo foi baseado, e o modo de implementação em lógica reconfigurável.

São apresentados também o material principal, que consiste na placa (TERASIC, 2007) com uma plataforma baseada em um *FPGA* Altera (ALTERA, 2008), seus circuitos periféricos e sua configuração, assim como os circuitos utilizados, seu funcionamento geral, e detalhes nas implementações que influenciam características do projeto, como tempo de execução ou capacidade de *hardware*.

Para a geração da função *b-spline* na mesma plataforma de lógica configurável mencionada anteriormente, foram selecionadas as três principais implementações para a geração dos pontos: o circuito desenvolvido por Gopi e Manohar (1997) para *ASICs*, baseado nos algoritmos de Megson (1991) para matrizes sistólicas, o trabalho de Ou et al. (2002), que apesar de testes em linguagem C++ e *OpenGL* apresenta a idéia da utilização de combinações fixas dos valores das funções base; e o circuito *fast* Cox-de Boor, criado por Yau et al. (2006) para *FPGAs*, que resulta na maior eficiência pelo processamento concorrente de circuitos do triângulo do algoritmo Cox-de Boor.

A título de ilustração, uma implementação geral em *software* para *GPUs* compatíveis com a arquitetura *CUDA* também é apresentada.

4.2. Metodologia

A principal característica da lógica reconfigurável é a possibilidade da criação de uma arquitetura de processamento otimizada de acordo com a aplicação. Deste modo, a metodologia usada para o desenvolvimento de circuitos sintetizados pela lógica reconfigurável foi a utilização ou modificação de *cores* disponíveis, e a criação e síntese

de circuitos baseados em algoritmos de processamento paralelo. As etapas a seguir foram elaboradas para o desenvolvimento do sistema em *FPGA*:

- circuito para acesso à memória externa *SDRAM*;
- circuitos para a geração de segmentos de retas e circunferências baseados nos algoritmos paralelos de Wright;
- circuito para a geração de curvas e superfícies *b-splines* 2D e 3D, baseado no circuito de Gopi e Manohar (1997), denominado neste projeto de circuito *BFEA* (*Basis Functions Evaluation Array*);
- circuito para a geração de curvas *b-splines* 2D e 3D, baseado nos circuitos de combinações fixas de funções *b-splines*, proposto por Ou et al. (2002), denominado neste projeto de circuito das funções base fixas ;
- circuito para a geração de curvas *b-splines* 2D e 3D, baseado no método *fast* Cox-de Boor, proposto por Yau et al. (2006) ;
- circuito de barramento de dados e de sinais de controle baseado no *Wishbone bus*.

Os *cores* para a geração de segmentos de reta e *cores* para a geração dos pontos de arcos de circunferência e esferas, são incluídos no sistema como alternativa à utilização das *b-spline*, devido à representação mais compacta dos dados de entrada necessários para gerar tais primitivas. Como exemplo, para a geração paramétrica dos pontos de uma circunferência, são necessários somente os dados das coordenadas do centro, o vetor normal à superfície que irá conter a circunferência, e o seu raio. No caso da geração de uma circunferência por uma *spline*, é necessário uma *b-spline* racional (para representar com mais precisão as seções cônicas), com 7 pontos de controle, 7 valores dos pesos para cada ponto de controle, e 10 valores para os nós distribuídos não uniformemente, resultando em um acréscimo de 17 dados geométricos em relação à representação paramétrica da circunferência. Nos resultados são comparados os tempos de execução de cada um dos três tipos de circuitos *b-spline*, assim como dos circuitos gerados para retas e circunferências, onde o tempo de execução é dado em termos do número de ciclos do sinal de *clock*.

Um diagrama geral dos circuitos que compõem o sistema em lógica configurável é exibido na Figura 43, onde o módulo para a geração de curvas e superfícies *b-splines* é modificado de acordo com o método em teste: *BFEA*, das funções base fixas, e o *fast* Cox-de Boor. A descrição detalhada de cada subsistema, assim como suas partes componentes, detalhes do método, *hardware* e implementação são descritos nas seções seguintes.

Figura 43 - Diagrama geral das partes constituintes do Soc de geração e visualização de curvas e superfícies *b-splines*.

Pela quantidade limitada de elementos lógicos (*les – logic elements*) presente em cada *FPGA*, a quantidade de circuitos sintetizados é limitada, mas possibilita várias configurações com os *cores* utilizados, respeitando-se este limite. A medida do tempo de execução para cada *core* é realizada via circuitos em sincronia com o respectivo *core*, pela contagem do número de sinais de *clock*. Deste modo, é possível obter uma medida do tempo de processamento em relação ao número de sinais de *clock* utilizados, tanto para um único *core*, como para uma sequência de *cores*. A Figura 44 ilustra um diagrama geral dos circuitos contadores de pulsos do sinal de *clock* geral do sistema utilizado como medida do tempo de execução do *FPGA*.

Figura 44 – Circuitos contadores do *clock* para cada *core* e de um sistema geral.

Os algoritmos utilizados foram inicialmente testados em computadores pessoais (FARIN, 1997), adaptados de algoritmos similares de sistemas de processamento paralelo (GOPI E MANOHAR, 1997; OU ET AL., 2002 ;YAU ET AL., 2006). Nos computadores pessoais, estes algoritmos adaptados foram implementados e testados em *scripts* de linguagem interpretada *metafile* do *Matlab R2009b* (PALM, 2005).

No *Matlab* foram implementados algoritmos para as *b-splines* em linguagem própria interpretada na forma de *scripts*, com funções criadas para as *b-splines*, devido à facilidade na depuração dos programas, a disponibilidade de ferramentas para visualização de dados

matemáticos, geração de gráficos, de formatação automática dos dados e de manipulação de matrizes na forma vetorial.

Para compatibilização com os formatos utilizados no sistema *FPGA*, os algoritmos foram testados com dados no formato de ponto flutuante normal de 32 bits de precisão simples, e em inteiros de 16 e 32 *bits*, através das funções diretas de conversão *uint16* e *uint32*. O sistema de visualização de gráficos foi utilizado para gerar algumas figuras exemplo para este trabalho, em *scripts Matlab*. A Figura 45 exibe duas curvas de grau 1 e 2 geradas via *Matlab* utilizando dados no formato inteiro de 16 *bits* não sinalizados e a Figura 46 mostra curvas de graus 1, 2 e 3 no formato inteiro de 32 *bits*, geradas em *scripts Matlab*.

Figura 45 – Curvas de grau 1 (pontos na reta) e 2 no formato inteiro 16 *bits* sem normalização.

Figura 46 – Curvas de graus 1, 2 e 3 (ordenadas de baixo para cima) no formato inteiro de 32 *bits* sem normalização.

Para a conversão no tipo de dado inteiro de ponto fixo, os dados dos pontos de controle ou conjunto de pontos são normalizados para que os dígitos mais significativos do valor numérico das respectivas coordenadas euclidiana ocupem os 16 *bits* mais significativos em cada cálculo. Os vetores de nós e de parâmetros são normalizados na faixa de 0 a 1.000, tanto para ponto fixo em 16 ou 32 *bits* (em ponto flutuante de dupla precisão são utilizados na faixa de 0 a 1), assim como os cálculos realizados. Para os *cores* de geração de segmentos de reta, circunferências e entre os três *cores splines*, existem normalizações distintas nos cálculos.

Os circuitos para a geração de segmentos de reta foram adaptados de algoritmos que utilizam processamento paralelo e funções para números inteiros (seção 4.6), a não ser na geração dos segmentos circulares para a geração dos arcos de circunferência, que envolve o cálculo trigonométrico pelo circuito adaptado do algoritmo *CORDIC* (WRIGHT, 1990). Para a geração das *b-splines*, os cálculos com números que apresentam resolução abaixo dos 32 *bits*, apresentam erros na aproximação e truncamento dos resultados, principalmente no cálculo da recursividade das combinações convexas da divisão das funções base, nos circuitos *BFEA* e *fast Cox-de Boor*.

Como ilustração, foram obtidos os tempos de execução de uma implementação similar ao método fast Cox-de Boor para a geração de *b-splines* em uma *GPGPU*, utilizando o *Nvidia CUDA* (NVIDIA CORPORATION, 2011), em linguagem C++ no ambiente de desenvolvimento *Microsoft Visual Studio 2008*, sem a utilização de rotinas e circuitos próprios para desenvolvimento das *b-splines*, como o *OpenGL* (ver APÊNDICE C – Dados Técnicos e Implementação GPU).

4.3. Materiais

O sistema em lógica reconfigurável foi implementado em uma placa de desenvolvimento de sistemas DE2-70, da *Terasic Technologies* (2007), composta por: um *FPGA* modelo *Cyclone II EP2C70F896C6N* (ALTERA, 2008); um dispositivo de configuração serial *USB EPCS16*; uma interface de vídeo *DAC* de alta velocidade de 10 *bits* triplo *ADV7123 240-MHz*, compatível com padrão *VGA*; e dois *chips* de memória *SDRAM* de 32 Mbytes total (ISSI, 2007). São fornecidos dois sinais de *clock* via circuitos externos ao *chip FPGA* nas frequências de 50 MHz e 28,63 MHz. Outros componentes auxiliares incluem 4 *push-buttons*, 18 chaves, 18 *leds* vermelhos, 9 *leds* verdes, 8 *displays* de 7 segmentos, 1

display LCD de 2 linhas por 16 colunas de caracteres, controlador *USB host/slave* com dois conectores (1 tipo A e 1 tipo B), controlador *ethernet* 10/100 e memória *SSRAM* de 2 Mbytes. O apêndice D lista todos os componentes periféricos ao *FPGA* nesta placa de desenvolvimento, com as configurações gerais dos principais componentes utilizados neste projeto e suas respectivas referências às folhas de dados dos fabricantes.

As principais características do *FPGA Cyclone II EP2C70F896C6N* são:

- 68.416 elementos lógicos disponíveis;
- 150 multiplicadores;
- 4 *PLLs*;
- 622 pinos de *I/O* disponíveis ao usuário;
- 1.152 Mbits de memória *RAM*;
- suporte ao processador embarcado *NIOS II* da *Altera Corp.*

Figura 47 – Placa DE2-70 utilizada para a implementação em lógica reconfigurável. Fonte: (TERASIC TECHNOLOGIES INC., 2007).

A Figura 47 mostra a placa DE2-70 e os componentes periféricos ao *FPGA*, mencionados anteriormente. O dispositivo de configuração serial *EPCS16* permite controle do usuário via *API* – interface de programação do usuário, e modos de programação *JTAG* ou *AS* – *Active Serial*. Os sinais de *clock* de 50 MHz e 28,86 MHz são gerados por dois osciladores independentes e conectados aos *clocks* dos *PLLs* internos ao *FPGA*, sendo o segundo sinal utilizado principalmente para conformidade com o sinal necessário para dois circuitos

decoders de TV. A utilização do circuito de interface de vídeo padrão VGA é descrito no Apêndice F, sendo esta a implementação relacionada à visualização, requerendo ajustes precisos na temporização do circuito de geração de sinais para o monitor de vídeo VGA.

Os circuitos utilizados foram sintetizados e testados via *software Quartus II IDE*, da Altera Corp. (ALTERA, 2011), fornecido como o ambiente de desenvolvimento para os sistemas e dispositivos programáveis deste fabricante, assim como as ferramentas para a elaboração e edição da linguagem de descrição de *hardware* em *Verilog HDL* (IEEE, 2006; PALNITKAR, 2003; THOMAS E MOORBY, 2002), e a partir desta linguagem a geração de diagramas esquemáticos de blocos em *RTL – Register Transfer Level*. A ferramenta *Altera Signal Tap II Logic Analyzer* foi utilizada para a análise e depuração do sistema em funcionamento.

4.4. Memória SDRAM

Quando os recursos internos de *hardware* disponíveis no *FPGA* não são suficientes para o armazenamento de informações, é necessário utilizar um circuito externo para armazenamento. Como o acesso aos circuitos externos ao *chip FPGA* requer uma transferência de dados limitada pelo tamanho do barramento de dados, a configuração do circuito de acesso à memória e os respectivos tempos de latência influenciam na velocidade máxima de transferência de dados.

O circuito de armazenamento de dados com maior capacidade de armazenamento, em relação à memória *on-chip* (de 1,152 Mb), consiste em dois chips de memória *SDRAM* de 32 MB (256 Mb) cada, totalizando 64 MB. Cada *chip* é organizado em 4 bancos de memória de 8.192 linhas por 512 colunas, de 16 *bits* cada célula, incluindo modo *auto refresh*, *power-saving* e *power-down*. O circuito de acesso a cada *chip* de memória *SDRAM* é baseado no projeto de referência com código fonte em *Verilog* proposto por Fan (2007), utilizado para ambos os *chips SDRAM*, tornando o barramento de acesso à memória formado pelo total de 32 *bits* (16 *bits* de cada *chip SDRAM*). A Figura 48 mostra o diagrama esquemático de interligação destes *chips SDRAM* ao *core* Intercon, próprio do barramento Wishbone.

Utilizando dados de 32 *bits*, a escrita e leitura da memória é executada com os 16 *bits* mais significativos no *chip SDRAM* 0 e os 16 *bits* restantes (menos significativos) no *chip SDRAM* 1. No caso do acesso em 16 *bits*, somente o *chip SDRAM* 0 é utilizado. Os sinais e o

circuito de controle de acesso para estas memórias seguem a padronização dos circuitos de memórias *SDRAM*, com um circuito controlador próprio para cada *chip*, sinais sincronizados na borda de subida do sinal do *clock*, e acessos de leitura e escrita orientados ao *burst access*, em que o acesso à memória é efetuado rapidamente em um conjunto em sequência de endereços pré-estabelecidos, e as posições destes endereços podem ser sequenciais ou aleatórias.

Figura 48 – Acesso aos *chips* de memórias SDRAM.

A operação básica de cada *chip SDRAM* consiste no registro de um comando (*ACTIVE*) para iniciar o acesso à memória, seguido por um comando de leitura (*READ*) ou escrita (*WRITE*). O comando inicial para acesso, em conjunto com os *bits* de endereçamento nas entradas correspondentes, são utilizados para selecionar o banco e linha da memória a serem acessados através de pinos para seleção do banco (BA0 e BA1) e seleção da linha (A0 a A12, Figura 49). Os *bits* de endereçamento (A0 a A8), coincidentes com o acionamento do comando de leitura (*READ*) ou escrita (*WRITE*), são utilizados para selecionar o endereço da coluna inicial para o *burst access* (acesso em lote), classificado nos tipos de 1, 2, 4, 8 endereços, ou *full page*. O *burst access* inicia o acesso à memória no primeiro endereço fornecido, com continuidade em um número pré-determinado de endereços fornecidos em uma tabela. Estas configurações da *SDRAM* influenciam no tempo médio de acesso à memória.

Figura 49 – Pinagem para as memórias SDRAM 0 e SDRAM 1. Fonte: (TERASIC, 2007).

Configurações da *SDRAM* tais como a seleção do comprimento e tipo do *burst access*, a latência *CAS* (*Colum Address Strobe*), modo de operação e modo de escrita *burst access*, são efetuadas pelo registrador de modo, que deve ser configurado quando todos os bancos da memória estão ociosos, e operações subsequentes devem ser efetuadas após um intervalo de tempo especificado (de acordo com o tipo de operação) para uma nova programação deste registrador, evitando-se modos de configuração reservados ou de testes após a inicialização de um *chip* de memória.

4.4.1. Comprimento do *Burst*

O comprimento do *burst* reduz os intervalos de tempo de latência da memória, determinando o número máximo de endereços de colunas da memória que podem ser acessados por um comando *READ* ou *WRITE*. Comprimentos de 1, 2, 4 ou 8 endereços são disponíveis nos tipos sequencial e intercalado. O *burst full page* é disponível somente para o tipo sequencial, sendo utilizado em conjunto com o comando *LOTE TERMINATE* para gerar comprimentos de *burst* arbitrários. Quando um comando *READ* ou *WRITE* é executado, um bloco de colunas igual ao comprimento do *burst* é selecionado e ativado automaticamente, e

todos os acessos à memória são efetuados neste bloco, sendo concluído ao alcançar um dos seus extremos. A ordem dos acessos dentro do *burst* é determinada pelo comprimento, o tipo e o endereço da coluna inicial do *burst*.

A latência *CAS* consiste no atraso, em ciclos de *clock*, entre o registro de um comando *READ* e a disponibilidade dos dados nas saídas, sendo configurada para dois ou três ciclos do sinal de *clock*. Se um comando *READ* é registrado no sinal de *clock* “n”, e a latência é igual a “m” sinais do *clock*, o dado se tornará disponível no sinal de clock “n + m”. Os sinais de dados, disponíveis pelos pinos DQs, assumem os valores do resultado um ciclo antes deste sinal de *clock* (em “n + m – 1”) e o dado estará disponível no sinal de clock “n + m” (ISSI, 2007). A Tabela 17 indica as frequências de operação nas quais cada configuração de latência *CAS* pode ser utilizada, em número de ciclos do sinal de *clock*.

Tabela 17 – Frequências de Operação Permitidas. Fonte: (ISSI, 2007).

Latência CAS (número de ciclos do sinal de <i>clock</i>)	Velocidade	
	2	3
-6	125 MHz	166 MHz
-7	100 MHz	143 MHz

Combinações de valores diferentes do modo de operação normal nos *bits* de configuração M7 e M8 (*bits* A7 e A8 com valores zero no registrador de modo) resultam em operação desconhecida ou incompatível, sendo reservadas para uso futuro e/ou para modos de teste, e o comprimento de *burst access* aplica-se a *bursts READ* e *WRITE*.

Detalhes adicionais sobre a operação da *SDRAM* são apresentados no APÊNDICE E – Operação da *SDRAM*.

4.5. Comunicação de Dados

Para possibilitar a visualização e verificação dos dados do sistema *FPGA*, foi realizada a aquisição desses dados por meio de uma ferramenta de análise de dados via *USB JTAG*. Esta seção apresenta a ferramenta utilizada para esta finalidade, assim como sua configuração.

A comunicação dos dados do sistema *FPGA*, com a interface gráfica do computador pessoal, é executada via ferramenta de *EDA Quartus II Signal Tap Analyzer*, que fornece dados de sistemas compatíveis com *FPGAs Altera* via interface *USB JTAG*, *ethernet* ou porta paralela. O *Signal Tap Analyzer* permite a visualização dos sinais de registradores internos, entradas e saídas, em gráficos de sinais lógicos. O início da aquisição dos sinais pode ser configurado de acordo com eventos pré-definidos ou em intervalos do sinal de *clock* específicos. Os valores de sinais de registradores e saídas selecionados podem ser armazenados internamente no *FPGA* e transferidos via *USB JTAG* para um arquivo texto no computador pessoal. Esta ferramenta substitui sondas externas ou modificações no *projeto* do circuito para a depuração do funcionamento pela aquisição do estado de nós e pinos de entrada e saída. Os sinais de dados são adquiridos e armazenados na memória interna do *FPGA* em funcionamento pleno.

A aquisição de dados dos registradores internos e pinos de I/O pode ser efetuada até 128.000 pulsos do sinal de *clock* de referência do projeto, ou até o limite dado pela capacidade máxima da memória interna do *FPGA*. Os dados são adquiridos e armazenados até um sinal de desabilitação da aquisição, previamente configurado, ser acionado, eliminando o excesso de dados adquiridos. A Figura 50 exibe a tela de interface do *software EDA Signal Tap II Logic Analyzer*, destacando-se as formas de onda dos níveis lógicos dos sinais, com o respectivo valor (em formato hexadecimal ou binário).

Figura 50 – Signal Tap II Logic Analyzer. Fonte: (ALTERA, 2011).

4.6. Primitivas Gráficas – Segmento de Reta

As *b-splines* de grau maior que 1 são ineficientes na representação dos segmentos de reta, e no caso das seções cônicas, não as representam com precisão (exceto no caso das *b-splines* do tipo racionais). Os tipos particulares de seções cônicas, tais como a circunferência, elipse, esfera e elipsóide, também não são representadas pelas *b-splines* com eficiência na compactação dos dados, pois possuem representações paramétricas mais compactas. Como exemplo, para a representação de um segmento de reta via *b-splines* são necessários no mínimo três pontos de controle, além dos nós e parâmetros, com as condições inicial e final do polinômio especificadas, enquanto que para o algoritmo convencional de geração de um segmento de reta são necessários somente o ponto inicial e o ponto final do segmento.

Os circuitos adicionais desenvolvidos neste projeto, para segmentos de reta e circunferências, baseiam-se nos algoritmos sugeridos por Wright (1990), para a paralelização dos algoritmos de Bresenham (1965) do ponto médio. O funcionamento do circuito para a geração do segmento de reta é apresentado a seguir, conforme dados da Figura 51:

- a ordenada $dx = (x_f - x_0)$ é dividida em P segmentos, sendo P o número de circuitos de processamento paralelo;

- deste modo, deve-se determinar o valor y_k do início do segmento de cada circuito, sendo k o índice do circuito no processo;

- de acordo com o algoritmo de Bresenham, o valor de y do próximo ponto a ser gerado na reta deve ser igual ou uma unidade maior que o valor atual. A indicação ou ausência deste incremento é dada pela variável p , eliminando cálculos com ponto flutuante;

- w é o comprimento do segmento designado ao processador k , sendo:

$$w = \frac{dx + (P-1)}{P} \quad (11)$$

- deste modo, $x(k) = kw$;

$$p(k) = 2dy - dx + 2dy(x(k)) - 2dx(y(k)) \quad (12)$$

- sendo $m = dy/dx$ a inclinação da reta, os cálculos podem ser reduzidos a (incluindo os cálculos de ponto flutuante):

$$y(k) = \text{round}(m \cdot x(k)) \quad (13a)$$

$$y(k) = \text{truncate}(m \cdot x(k) + 0,5) \quad (13b)$$

$$y(k) = \text{truncate} \left(\frac{2 \cdot dy(x(k)) + dx}{2dx} \right) \quad (13c)$$

$$y(k) = \frac{2 \cdot dy(x(k)) + dx}{2dx} \quad (13d)$$

$$p(k) = 2dy - dx + 2dy(x(k)) - 2dx(y(k)) \quad (14)$$

Figura 51 – Geração do segmento de reta para P circuitos processadores (k - índice do processador). Adaptada de: (WRIGHT, 1990).

A Figura 52 apresenta o diagrama esquemático simplificado de um circuito para a geração do segmento de reta $(x(k), y(k))$ a $(x(k+w), y(k+w))$, baseado no algoritmo descrito anteriormente. Este circuito se repete para k segmentos de reta, que juntos formarão a reta final, sendo P o número de circuitos utilizados para o processamento em paralelo, $(x0, y0)$ e (xf, yf) os pontos inicial e final respectivamente do segmento de reta. Para segmentos de reta em 3D, o processo é repetido incluindo a terceira coordenada z , mediante o mesmo processo, como a inclusão de novos valores para $z0, zf, dz, p$ e y .

4.7. Primitivas Gráficas – Circunferência e Esfera

Para a geração de uma circunferência via *NURBS*, conforme Piegl (1991), são necessários o mínimo de sete pontos de controle no formato de um quadrado, com os respectivos pesos modificados para produzir uma circunferência, enquanto que o algoritmo convencional com equações paramétricas para a geração de uma circunferência necessita

somente das coordenadas do centro, raio e vetor perpendicular à superfície que contém a circunferência (Figura 53).

Figura 52 – Fluxograma das etapas de processamento de um circuito para a geração de um segmento de reta k .

Foram implementados neste projeto os circuitos baseados no algoritmo paralelo proposto por Wright (1990) para os segmentos de reta e circunferências, e também os circuitos para as *NURBS*, possibilitando ao usuário a escolha do tipo de representação desejada para os segmentos de reta e circunferências.

Figura 53 – Circunferência *NURBS* (linha cheia) de grau 2, formado no mínimo por 7 pontos de controle (P) com vetor de nós $[0, 0, 0, 0, 25, 0.5, 0.5, 0.75, 1, 1, 1]$ e vetor de pesos $[1, 0.5, 0.5, 1, 0.5, 0.5, 1]$.

Os circuitos para arcos de circunferência baseiam-se no mesmo princípio para a geração de segmentos de reta, descrito anteriormente, em que uma variável condicional determina se o próximo ponto a ser gerado na circunferência será no mesmo valor ou um acréscimo do valor da ordenada. O ponto inicial é determinado como o ponto de maior ordenada da circunferência, estendendo o cálculo para um arco de circunferência de 45° , conforme a Figura 54. Os 7 arcos restantes da circunferência são formados por simetria com os eixos coordenados do centro da circunferência, com 7 *cores* respectivos no *FPGA* que efetuam a geração destes pontos.

Figura 54 – Divisão de um arco de circunferência em P processos para o algoritmo de Wright. Adaptado de: (WRIGHT, 1990).

Para o segmento de reta, a projeção no eixo x foi dividida pelo número de processadores. No caso do circunferência, o arco de 45° foi dividido em P (número de processadores) arcos. Os valores iniciais das coordenadas x e y para cada arco são:

$$x(k) = \frac{r.A+1}{W} \quad (15)$$

$$y(k) = \frac{r.B+1}{W} \quad (16)$$

$$A = 2.\text{round}\left(2^t.\text{sen}\left(\left(\frac{k}{P}\right).\left(\frac{\pi}{4}\right)\right)\right) \quad (17)$$

$$B = 2.\text{round}\left(2^t.\text{cos}\left(\left(\frac{k}{P}\right).\left(\frac{\pi}{4}\right)\right)\right) \quad (18)$$

$$W = 2^t \quad (19)$$

Nas equações 15 a 18, os valores de A e B, que envolvem funções trigonométricas, são previamente tabelados em números inteiros para um número P de processos concorrentes, para cada processador k . O valor inicial $y(k)$ da equação 16 pode assumir três valores possíveis: $y_1 = y(k)$, $y_2 = y(k)+1$, $y_3 = y(k)-1$. A alternativa escolhida é a que minimiza o erro com a fórmula da circunferência $r^2 - x^2 - y_i^2$, sendo $i = 1, 2$ ou 3 . Deste modo:

$$d_1 = r^2 - x^2 - y^2 = (r + x)(r - x) - y^2 \quad (20a)$$

$$d_2 = r^2 - x^2 - (y + 1)^2 = d_1 - 2y - 1 \quad (20b)$$

$$d_3 = r^2 - x^2 - (y - 1)^2 = d_1 + 2y - 1 \quad (20c)$$

O menor d_i indica qual valor y_i será escolhido como valor inicial $y(k)$ do segmento. A variável p , que determina se o próximo valor de y será incrementado ou permanecerá o mesmo, é fornecida de acordo com os respectivos valores de d_i :

$$p_1(k) = -2d_1 + 4x - 2y + 3 \quad p/ d_1 \quad (21a)$$

$$p_2(k) = -2d_1 + 4x + 2y + 3 \quad p/ d_2 \quad (21b)$$

$$p_3(k) = -2d_1 + 4x - 6y + 7 \quad p/ d_3 \quad (21c)$$

O cálculo procede desta maneira incrementando-se o valor de x em cada segmento até $x=x(k+1)$, que indica o início do próximo segmento.

4.8. Circuitos *B-Splines*

Os circuitos para a geração de curvas e superfícies *b-splines* basearam-se em três métodos, descritos nos trabalhos de Gopi e Manohar (1997) para circuitos integrados, de Ou et al. (2006) e de Yau (2006), um dos únicos a implementar as *b-splines* em um sistema de lógica reconfigurável, denominada de implementação *fast* Cox-de Boor (ilustrada na Figura 11). Estes circuitos são denominados neste trabalho, respectivamente, de *BFEA*, das funções base fixas e *fast* Cox-de Boor.

4.8.1. *BFEA – Basis Function Evaluation Array*

O cálculo recorrente e o número de funções base, dependente do grau da *b-spline*, geram um cálculo sequencial dado pelo grafo da Figura 55. A *Basis Function Evaluation*

Array – BFEA, proposta por Gopi e Manohar (1997), consiste na matriz do cálculo dos valores das funções base para os parâmetros dentro de um intervalo de nós, através de um circuito de processamento linear simplificado dado pelos Elementos de Processamento – EP, circuitos responsáveis pelo processamento de cada nível do grafo das funções base. A geração da *b-spline* é feita pela equação 5, resultando na soma do produto das funções base com os pontos de controle, efetuada por um acumulador.

Figura 55 – Cálculo recursivo das funções base e circuito BFEA. Adaptado de: (GOPI, 1997).

As funções base recorrentes podem ser dadas pelas equações:

$$N_{i,p} = \frac{t-u_i}{u_{p+m}-u_{p+1}} \cdot N_{i,p-1}(t) = \alpha \cdot N_{i,p-1}(t) \quad (22a)$$

$$N_{i-1,p} = \left(1 - \frac{t-u_i}{u_{p+m}-u_{p+1}}\right) \cdot N_{i,p-1}(t) = (1-\alpha) \cdot N_{i,p-1}(t) \quad (22b)$$

Desenvolvendo a equação geral (22a):

$$N_{i,p} = \frac{t-u_i}{u_{i+p-1}-u_i} N_{i,p-1}(t) + \frac{u_{i+p}-t}{u_{i+p}-u_{i+1}} N_{i+1,p-1}(t) \quad (23a)$$

$$N_{i,p} = \frac{(u_{i+p-1}-u_i)-(u_{i+p-1}-t)_i}{u_{i+p-1}-u_i} N_{i,p-1}(t) + \frac{u_{i+p}-t}{u_{i+p}-u_{i+1}} N_{i+1,p-1}(t) \quad (23b)$$

$$N_{i,p} = N_{i,p-1}(u) - \frac{u_{i+p-1}-t}{u_{i+p-1}-u_i} N_{i,p-1}(t) + \frac{u_{i+p}-t}{u_{i+p}-u_{i+1}} N_{i+1,p-1}(t) \quad (23c)$$

O primeiro termo da equação 23c é obtido previamente do cálculo das funções de um nível anterior, e o segundo termo da equação 23c é obtido do primeiro termo da equação 22b, da função base do mesmo nível. O resultado destes cálculos, obtidos de um nível recorrente anterior, eliminam o cálculo do primeiro termo da equação 23a, reduzindo quatro operações por função base a cada nível (exceto para a primeira e para a última função base de cada nível, em que o primeiro ou último termo não é necessário). Esta simplificação é apresentada no circuito da Figura 56, pelas operações de soma e subtração de $N_{i,j-1}(t)$. Os valores de u_{i+k} , u_i e t devem ser fornecidos, adicionando mais duas operações de subtração a cada célula de processamento. Para simplificação, os valores de $t-u_i$ e $u_{i+j}-t$ são fornecidos às entradas de cada elemento de processamento, reduzindo a três o número de entradas.

Figura 56 – Circuito simplificado de uma célula de processamento da BFEA implementado em FPGA. Adaptado de: (GOPI, 1997).

O tempo de processamento de uma *b-spline* utilizando o BFEA é independente do número de pontos de controle, mas proporcional ao número de pontos calculado vezes o grau da curva ou superfície a ser gerada. O circuito proposto por Gopi e Manohar limita o grau da *b-spline* a um valor máximo para limitar o tamanho dos circuitos de cada Elemento de

Processamento. Caso seja necessário utilizar um grau menor, o processo é interrompido anteriormente ao final da *BFEA*.

4.8.2. Valores Fixos das Funções Base

Caso o valor dos nós tenha uma relação fixa conhecida, como um vetor de nós uniforme, a relação entre os valores dos nós adjacentes é constante, exceto entre os nós iniciais e finais no vetor de nós uniforme fechado. Um vetor de nós U deste tipo pode ser generalizado para a seguinte forma:

$$U = \{\underbrace{a, a, \dots, a}_{p+1}, u_{p+1}, \dots, u_{m-p-1}, \underbrace{b, b, \dots, b}_{p+1}\}$$

$$U = \{\underbrace{a; a; \dots; a}_{p+1}; a+c; a+2c; a+3c; \dots; b-3c; b-2c; b-c; \underbrace{b; b; \dots; b}_{p+1}\} \quad (24)$$

O termo c é uma constante que estabelece os valores para os nós, exceto nos extremos do vetor de nós, onde ocorre a repetição dos valores para manter a curva *b-spline* coincidente com os pontos de controle inicial e final. De acordo com o grau p de uma *b-spline*, existem 2^p tipos de subconjuntos de combinações de valores dos nós que podem compor as funções base. Para o grau 2 são utilizados quatro tipos de subconjuntos para o vetor de nós:

$$\{x, x, x+c, x+c\} = \{0, 0, 1, 1\} \quad (25a)$$

$$\{x, x, x+c, x+2c\} = \{0, 0, 1, 2\} \quad (25b)$$

$$\{x, x+c, x+2c, x+2c\} = \{0, 1, 2, 2\} \quad (25c)$$

$$\{x, x+c, x+2c, x+3c\} = \{0, 1, 2, 3\} \quad (25d)$$

Os subconjuntos dados pelas equações 25a a 25d utilizam vetores de nós uniformes e com valores normalizados no intervalo de 0 a 2. Substituindo estes tipos de vetores de nós nas equações das funções base, obtém-se para uma *b-spline* de grau 2 as equações 4a e 4b. Efetuando-se o cálculo para o primeiro tipo de vetor de nós (equação 25a) e considerando $0 \leq t < 2$, obtém-se:

$$N_{0,1} = \frac{t-u_0}{u_{0+2-1}-u_0} N_{0,0}(t) + \frac{u_{0+2}-t}{u_{0+2}-u_{0+1}} N_{0+1,2-1}(t) = \frac{t-0}{1-0} \cdot 1 = t \quad (26a)$$

$$N_{1,1} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,0}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = \frac{t-1}{1-0} \cdot 1 = t-1 \quad (26b)$$

$$N_{0,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = 0 + \frac{t-0}{1-0} \cdot t = t^2 \quad (26c)$$

$$N_{1,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = 2 \cdot (t-t^2) \quad (26d)$$

$$N_{2,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = 0 + \frac{t-1}{1-0} \cdot (t-1) = (t-1)^2 \quad (26e)$$

Para o segundo tipo de vetor de nós, dado pela equação 25b e considerando $0 \leq t < 1$:

$$N_{0,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = 0 + \frac{1-t}{1-0} \cdot (1-t) = t^2 - 2t + 1 \quad (27a)$$

$$N_{1,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = \frac{t-0}{1-0} \cdot (1-t) + \frac{2-t}{2-0} \cdot t = 2t - \frac{3}{2}t^2 \quad (27b)$$

$$N_{2,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = \frac{t-0}{2-0} \cdot t + 0 = \frac{t^2}{2} \quad (27c)$$

Para o terceiro tipo de vetor de nós, dado pela equação 25c e considerando $1 \leq t < 2$:

$$N_{0,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = 0 + \frac{2-t}{2-0} \cdot (2-t) = \frac{t^2}{2} - 2t + 2 \quad (28a)$$

$$N_{1,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = \frac{t-0}{1-0} \cdot (1-t) + \frac{2-t}{2-0} \cdot t = 4t - \frac{3}{2}t^2 - 2 \quad (28b)$$

$$N_{2,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = \frac{t-0}{2-0} \cdot t + 0 = t^2 - 2t + 1 \quad (28c)$$

Para o quarto tipo de vetor de nós, dado pela equação 25d e considerando $1 \leq t < 2$:

$$N_{0,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = 0 + \frac{2-t}{2-0} \cdot (2-t) = \frac{t^2}{2} - 2t + 2 \quad (29a)$$

$$N_{1,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = \frac{t-0}{2-0} \cdot (2-t) + \frac{3-t}{3-1} \cdot (t-1) = 3t - t^2 - \frac{3}{2} \quad (29b)$$

$$N_{2,2} = \frac{t-u_1}{u_{0+2-1}-u_1} N_{0,1}(t) + \frac{u_{1+2}-t}{u_{1+2}-u_{0+1}} N_{0+1,2-1}(t) = \frac{t-1}{3-1} \cdot (t-1) + 0 = \frac{t^2 - 2t + 1}{2} \quad (29c)$$

O ponto gerado é resultante do somatório do produto das funções base pelos pontos de controle correspondentes, de acordo com a equação 5. Algumas observações importantes sobre este método de funções base fixas, que não são mencionadas no trabalho original de Ou et al. (2002), são descritas nesta seção. A parametrização dos valores de t é normalizada para o intervalo dado para cada conjunto de equações, sendo o número total de parâmetros dividido pelo número de intervalos dos pontos de controle.

É necessária a utilização de três valores dados pelas equações das funções base, para gerar um ponto de uma *b-spline* de grau 2. Assim, somente um dos quatro tipos de vetores de nós (equações 25a a 25d) é utilizado para o cálculo de cada parâmetro, ou seja, somente um dos conjuntos de três equações é utilizado para o cálculo de um ponto da *b-spline*. As equações 26c a 26e das funções base, relativas ao primeiro vetor de nós, são utilizadas somente para o caso em que a *b-spline* possui três pontos de controle (número mínimo de pontos de controle para gerar uma *b-spline* de grau 2). As equações 27a a 27c são utilizadas somente para os três primeiros pontos de controle, e as equações 28a a 28c somente para os três últimos pontos de controle. O quarto tipo de vetor de nós, $\{x, x+c, x+2c, x+3c\}$ e suas respectivas equações (29a a 29c) para as funções base, são utilizados para o cálculo do restante dos pontos (pontos intermediários).

De acordo com estas observações em relação aos tipos de vetores de nós, é necessário implementar somente uma vez os circuitos relativos às equações 27a a 28c, utilizando as funções base fixas. Os circuitos das equações 29a a 29c podem ocupar a área restante de *hardware* disponível. As equações das funções base do primeiro tipo de vetor de nós serão utilizadas somente no caso específico de três pontos de controle, em que o número de pontos da *b-spline* seja vantajoso para esta implementação.

A geração de superfícies, de acordo com a equação 8, resulta no cálculo do produto das funções das equações 26c a 29c, de acordo com os respectivos tipos de conjuntos de equações e variação da faixa de valores de t , resultando em um polinômio de grau 4.

De modo similar às *b-splines* de grau 2, um circuito de geração de *b-splines* de funções base fixas de grau 3, possui oito tipos de vetores de nós, e 32 equações para as funções bases (4 para cada tipo de vetor de nós).

$$\{x, x, x, x+c, x+c, x+c\} = \{0, 0, 0, 1, 1, 1\} \quad (30a)$$

$$\{x, x, x, x+c, x+2c, x+2c\} = \{0, 0, 0, 1, 2, 2\} \quad (30b)$$

$$\{x, x, x+c, x+2c, x+2c, x+2c\} = \{0, 0, 1, 2, 2, 2\} \quad (30c)$$

$$\{x, x, x, x+c, x+2c, x+3c\} = \{0, 0, 0, 1, 2, 3\} \quad (30d)$$

$$\{x, x, x+c, x+2c, x+3c, x+4c\} = \{0, 0, 1, 2, 3, 4\} \quad (30e)$$

$$\{x, x+c, x+2c, x+3c, x+4c, x+5c\} = \{0, 1, 2, 3, 4, 5\} \quad (30f)$$

$$\{x, x+c, x+2c, x+3c, x+4c, x+4c\} = \{0, 1, 2, 3, 4, 4\} \quad (30g)$$

$$\{x, x+c, x+2c, x+3c, x+3c, x+3c\} = \{0, 1, 2, 3, 3, 3\} \quad (30h)$$

As equações para as funções base fixas de uma *b-spline* de grau 3 são dadas a seguir, com os respectivos tipos de vetores de nós. Para o vetor de nós $\{0, 0, 0, 1, 1, 1\}$:

$$N_{0,3} = -t^3 + 3t^2 - 3t + 1 \quad (31c)$$

$$N_{1,3} = 3t^3 - 6t^2 + 3t \quad (31d)$$

$$N_{2,3} = -3t^3 + 3t^2 \quad (31e)$$

$$N_{3,3} = t^3 \quad (31e)$$

Para o vetor de nós $\{0, 0, 0, 1, 2, 2\}$:

$$N_{0,3} = -t^3 + 3t^2 - t + 1 \quad (32c)$$

$$N_{1,3} = \frac{7}{4}t^3 - \frac{9}{2}t^2 + 3t \quad (32d)$$

$$N_{2,3} = -t^3 + \frac{3}{2}t^2 \quad (32e)$$

$$N_{3,3} = \frac{t^3}{4} \quad (32e)$$

Para o vetor de nós $\{0, 0, 1, 2, 2, 2\}$:

$$N_{0,3} = -\frac{t^3}{4} + \frac{3}{2}t^2 - 3t + 2 \quad (33c)$$

$$N_{1,3} = -\frac{t^3}{2} - 6t^2 + 8t - 4 \quad (33d)$$

$$N_{2,3} = -\frac{7}{4}t^3 + 6t^2 - 6t + 2 \quad (33e)$$

$$N_{3,3} = t^3 - 3t^2 + 3t - 1 \quad (33e)$$

Para o vetor de nós $\{0, 0, 0, 1, 2, 3\}$:

$$N_{0,3} = -t^3 + 3t^2 - 3t + 1 \quad (34c)$$

$$N_{1,3} = \frac{5}{2}t^3 - \frac{11}{12}t^2 + 3t \quad (34d)$$

$$N_{2,3} = -\frac{11}{12}t^3 + \frac{4}{3}t^2 \quad (34e)$$

$$N_{3,3} = \frac{t^3}{6} \quad (34e)$$

Para o vetor de nós $\{0, 0, 1, 2, 3, 4\}$:

$$N_{0,3} = -\frac{t^3}{2} + 3t^2 - 6t + 4 \quad (35c)$$

$$N_{1,3} = \frac{t^3}{6} - \frac{5}{3}t^2 + \frac{9}{2}t - \frac{3}{2} \quad (35d)$$

$$N_{2,3} = -\frac{t^3}{2} + \frac{4}{3}t^2 - 2t + \frac{2}{3} \quad (35e)$$

$$N_{3,3} = \frac{t^3}{6} - \frac{t^2}{3} + \frac{t}{3} - \frac{1}{6} \quad (35e)$$

Para o vetor de nós $\{0, 1, 2, 3, 4, 5\}$:

$$N_{0,3} = -\frac{t^3}{6} + \frac{3t^2 - 9t + 9}{2} \quad (36c)$$

$$N_{1,3} = \frac{t^3}{2} - \frac{11}{3}t^2 + \frac{41}{3}t - \frac{22}{3} \quad (36d)$$

$$N_{2,3} = -\frac{t^3}{2} + \frac{7}{2}t^2 - \frac{15}{2}t + \frac{31}{6} \quad (36e)$$

$$N_{3,3} = \frac{t^3}{6} - t^2 + 2t - \frac{4}{3} \quad (36e)$$

Para o vetor de nós $\{0, 1, 2, 3, 4, 4\}$:

$$N_{0,3} = -\frac{t^3}{6} + \frac{3}{2}t^2 - \frac{9}{2}t + 9 \quad (37c)$$

$$N_{1,3} = \frac{t^3}{2} - 4t^2 + 10t \quad (37d)$$

$$N_{2,3} = -\frac{7}{6}t^3 + 8t^2 + 17t + \frac{35}{3} \quad (37e)$$

$$N_{3,3} = \frac{t^3}{4} - \frac{3}{2}t^2 + 3t - 2 \quad (37e)$$

Para o vetor de nós $\{0, 1, 2, 3, 3, 3\}$:

$$N_{0,3} = -\frac{t^3}{6} + \frac{3}{2}t^2 + \frac{9-9t}{2} \quad (38c)$$

$$N_{1,3} = \frac{11}{12}t^3 - \frac{27}{4}t^2 + 23t + 45 \quad (38d)$$

$$N_{2,3} = -\frac{7t^3 + 35t^2 - 93t + 63}{4} \quad (38e)$$

$$N_{3,3} = t^3 - 6t^2 + 12t - 8 \quad (38e)$$

Para *b-splines* de grau 3, são utilizadas 4 equações de funções base fixas para gerar um ponto.

4.8.3. Fast Cox-de Boor

No método *fast* Cox-de Boor, proposto por Yau et al. (2006), cada conjunto de camadas componentes para a geração de um ponto (processamento do valor de um parâmetro) é denominado triângulo de Boor, e os elementos de cada camada são processados paralelamente. Apesar do resultado apresentar um ciclo do sinal de *clock* mais rápido que o *BFEA* (GOPI, 1997), o trabalho de Yau et al. (2006) somente menciona o tamanho dos operandos e variáveis utilizados, de 32 *bits*, mas não apresenta as normalizações para efetuar o processamento em ponto fixo, bem como os resultados gráficos e numéricos para superfícies *b-splines*.

A implementação do método *fast* Cox-de Boor é executada por meio de circuitos internos que efetuam o processamento do triângulo de Boor, principalmente o cálculo das funções base, conforme mencionado na seção 3.2, dado por α representando o cálculo entre parênteses na equação (21a). As equações 39b e 39c simplificam a equação 10, substituindo a expressão entre parênteses, do segundo termo da equação 10, pela variável α .

$$C_i^j(t) = \left(1 - \frac{t - u_i}{u_{i+k-j} - u_i}\right) \cdot P_{i-1}^{j-1} + \left(\frac{t - u_i}{u_{i+k-j} - u_i}\right) \cdot P_i^{j-1} \quad (39a)$$

$$\alpha = \frac{t - u_i}{u_{i+k-j} - u_i} \quad ; \quad \alpha' = 1 - \frac{t - u_i}{u_{i+k-j} - u_i} = \frac{u_{i+k-j} - t}{u_{i+k-j} - u_i} \quad (39b)$$

$$C_i^j(t) = \alpha' \cdot P_{i-1}^{j-1} + \alpha \cdot P_i^{j-1} \quad (39c)$$

O circuito da Figura 57 exibe a implementação em diagrama de blocos da equação 39b. Este circuito é repetido para cada valor de parâmetro t . Para t dentro do intervalo de valores $[u_r, u_{r+1}]$, os valores das entradas u , α e P não se modificam, conforme a Figura 58.

Figura 57 – Exemplo de um circuito para o triângulo Cox-de Boor do cálculo de um ponto em uma curva *b-spline* de grau 2 (3 camadas).

Figura 58 – Circuito fast Cox-de Boor para uma curva *b-spline* de grau 2 (3 camadas). Os valores u e P não sofrem alterações se t_n estiver no intervalo $[u_r, u_{r+p}]$, sendo i o índice do intervalo do vetor de nós, k a ordem da curva (grau+1) e j a camada no circuito fast Cox-de Boor.

Cada intervalo de valores do vetor de nós U utiliza os respectivos intervalos de valores dos pontos de controle P (P_0 a P_n na Figura 59). Cada coordenada cartesiana do sistema necessita de um destes circuitos, inclusive para os pesos, considerados coordenadas homogêneas (terceira coordenada em sistemas 2D ou quarta coordenada em sistemas 3D). Caso os pesos possuam valor unitário, no caso de *b-splines*, não é necessário incluir o respectivo circuito, reduzindo-se a área ocupada no *FPGA*.

Para superfícies, as camadas do circuito se repetem de acordo com os graus designados para a superfície no sentido longitudinal e isoparamétrico. Os pontos gerados com o vetor de nós para uma direção são os pontos de controle para o vetor de nós na direção “oposta” (Figura 60). Deste modo, se a superfície possui o mesmo grau nas duas direções, o número de camadas dobra em relação ao circuito para curvas, causando problemas na aproximação para os cálculos em ponto fixo.

Figura 59 – Diagrama geral dos cores para o processamento de uma curva *b-spline* de “ n ” pontos de controle.

A principal operação que exige a normalização dos valores para ponto fixo, e para a aproximação dos valores finais nos cálculos, é o valor de α na equação 39c. Os pontos de controle são normalizados para ponto fixo de acordo com o número de *bits* utilizado (neste caso, 32 *bits*), enquanto que a fração para o cálculo do valor de α (equação 39a) é normalizada com um valor fixo em 10 vezes no numerador, pois os sistemas de computação gráfica utilizam processamento em ponto flutuante com os valores dos vetores de nós e dos parâmetros normalizados geralmente entre 0 e 1. Estes valores podem ser normalizados para a faixa entre 0 e 1000 no cálculo de α , tanto para 16 e 32 *bits*, mas a fração do valor de α , apresenta uma recorrência proporcional ao grau da curva, e o dobro do grau no caso de superfícies. As aproximações no cálculo com números inteiros de ponto fixo requerem um número de *bits* para as variáveis, igual ou maior aos exigidos para cálculo com ponto

flutuante, para se obter resultados razoáveis, ou uma normalização variável entre as camadas do circuito Cox-de Boor.

Figura 60 – Circuitos equivalentes para a geração de uma superfície. Os pontos gerados para um parâmetro são utilizados como pontos de controle para a outra direção isoparamétrica.

Cada *core* de um dos três circuitos *b-splines* apresentados equivale a um circuito para a geração de um ponto de uma *spline*. O Apêndice B mostra um exemplo de 4 *cores fast* Cox-de Boor de grau 2, com os circuitos equivalentes ao processamento das equações 39a a 39c.

4.9. Barramento Wishbone

Devido à necessidade da interconexão de *cores* em um sistema de lógica configurável, e a necessidade da sincronização entre estes circuitos, é aconselhável a adoção de um sistema de barramento padrão para a interligação e comunicação dos *cores*.

No presente projeto foi adotado o *Wishbone* (USSELMAN, 2001; OPENCORES ORGANIZATION, 2010), que além de um padrão para barramentos de dados e de controle entre *cores* de um *Soc*, consiste em uma metodologia de arquitetura aberta para a interconexão

de circuitos. Os objetivos desta metodologia, disponível em especificação aberta (como *open source*), são a integração rápida e com facilidade de *cores* através de uma interface comum entre os circuitos, de modo a promover a reutilização de circuitos em lógica reconfigurável, tanto em *soft core*, *firm core* ou *hard core*. A arquitetura *Wishbone* é semelhante ao barramento de um computador, especificando um protocolo de troca de dados padrão, em uma arquitetura mestre-escravo de comunicação e um *core* próprio para estabelecer esta interconexão (com denominação padrão de *Intercon*). Um resumo sobre a forma de comunicação básica entre dois *cores*, de acordo com o padrão *Wishbone*, estabelece basicamente o seguinte, de acordo com a Figura 61:

1. O *core* mestre envia dados para escrita ou recebe dados para leitura de um *core* escravo;
2. O *core mestre* envia um sinal pela saída identificada como CYC_O (*cycle out*), sinal padrão próprio da metodologia *Wishbone*, para a entrada CYC_I (*cycle in*) do circuito escravo, solicitando o envio ou recebimento de dados;
3. Se o circuito escravo estiver disponível, envia um sinal da sua saída ACK_O (*acknowledge out*) para a entrada ACK_I (*acknowledge in*) do circuito mestre solicitante da comunicação;
4. O circuito mestre inicia a transferência de dados com o circuito escravo, ativando sua saída STB_O (*strobe out*) para a entrada STB_I (*strobe in*) do circuito escravo;
5. Ao receber os dados, o circuito escravo desativa a saída ACK_O para o mestre, até terminar o processamento dos dados;
6. Caso exista mais alguma solicitação do circuito mestre, este mantém a saída CYC_O ativa para o escravo, aguardando novamente a resposta em sua entrada ACK_O.

Figura 61 – Comunicação entre circuitos mestre e escravo na metodologia Wishbone.

Figura 62 – Sinal CYC_O de sincronia dos cores.

Internamente a um *core*, quando um ou mais processos são executados por circuitos em paralelo, a sincronia dos circuitos pode ser feita pelo sinal CYC_O (Figura 62). Um *core*

pode ser mestre e escravo ao mesmo tempo, sendo as entradas e saídas mínimas obrigatórias para o circuito mestre as seguintes: CLK_I (*clock in*), RST_I (*reset in*), ACK_I, CYC_O e STB_O; e para o circuito escravo: CLK_I, RST_I, CYC_I, STB_I e ACK_O. As entradas e saídas de dados dos circuitos podem possuir denominação livre, recomendando-se as extensões “_O” e “_I” para indicar respectivamente os sinais de saída ou entrada (adotada no presente trabalho). Os sinais são sempre considerados ativos em nível lógico alto (1). A interconexão entre os *cores* pode ser efetuada sequencialmente, na qual as saídas de um *core* se conectam às entradas respectivas de outro único *core*, conforme exibido em um diagrama simplificado de blocos na Figura 63. Esta forma de interconexão é denominada interconexão de fluxo de dados.

Figura 63 – Interconexão em fluxo de dados com os *cores* mestre e escravo, e a nomenclatura padrão Wishbone para as entradas e saídas.

Quando um *core* recebe sinais de dois ou mais *cores*, um circuito denominado Intercon efetua a seleção e estabelecimento da comunicação, feito por um circuito árbitro interno. O Intercon é composto por multiplexadores e demultiplexadores com um número de entradas e saídas compatível com os *cores* interconectados. Os circuitos mestres enviam o sinal CYC_O para o circuito escravo selecionado, que envia o sinal ACK_O para a entrada ACK_I do respectivo circuito mestre, indicando o estabelecimento e comando de início da comunicação, ou seja, o sinal CYC_O é utilizado para a seleção do circuito na interconexão (Figura 64).

No sistema implementado neste trabalho, são utilizados os sistemas de interconexão de barramento compartilhado com um circuito árbitro tipo *FIFO – First In, First Out*, para a seleção do circuito mestre que efetuará a comunicação com o escravo. Neste caso, o primeiro circuito mestre que solicitar a interconexão ao barramento será o circuito a efetuar a comunicação. Circuitos subsequentes que solicitarem uma interconexão entram em uma fila, aguardando a finalização do processo dos circuitos anteriores. Um exemplo é demonstrado na Figura 65, na utilização da interconexão em barramento compartilhado e fluxo de dados.

Figura 64 – Sistema de interconexão de barramento compartilhado.

Figura 65 – Combinação de tipos de interconexão Wishbone.

4.10. Implementação *GPU*

No presente projeto, somente a título de ilustração, foi realizada uma implementação das *b-splines* em *GPU*, apresentando o tempo de execução da implementação do método *fast* Cox-deBoor em relação à *GPU*, para conjuntos de pontos 2D, 3D, superfície e *NURBS*. O modelo utilizado foi a *GeForce G 105M*, cujas especificações técnicas encontram-se no Apêndice C.

5. Resultados e Discussões

5.1. Considerações Iniciais

Este capítulo apresenta os resultados da implementação das *b-splines* em *FPGA*, com as comparações entre as implementações dos diferentes métodos: *BFEA*, funções base fixas e *fast* Cox-de Boor, incluindo as implementações para segmentos de reta e arcos de circunferências, quando o conjunto de pontos, utilizado como teste, possibilitar a representação por estas primitivas geométricas. Os resultados são analisados em relação ao tempo de execução, dado pelo tempo em nanosegundos obtido pelo número de ciclos do sinal de *clock* necessário para o processamento, para cada um dos métodos. A capacidade de *hardware* do *FPGA*, exigida por cada implementação, também é exibida nos resultados.

Os circuitos utilizados foram disponibilizados em sua última versão no *site opencores.org* (OPEN CORES ORGANIZATION, 2010), como *cores* do tipo aritmético, no projeto de título *B-Spline generation*.

5.2. Descrição dos Conjuntos de Pontos Teste

A implementação da geração de *b-splines* via lógica reconfigurável envolve diversos fatores que influenciam diretamente o resultado final do circuito sintetizado. Dentre os principais fatores, encontram-se a escolha das ferramentas de síntese dos circuitos, a configuração do *hardware* e os circuitos adicionais necessários. Para gerar os resultados, avaliar esses fatores, e determinar o tempo de execução do sistema *FPGA*, foram utilizados conjuntos de pontos teste, com diferentes características. Os tipos desses conjuntos de pontos diferem nos seguintes aspectos gerais:

- Curvas 2D:
 - tipos: geradas por computador e obtidas da digitalização de objetos reais;
 - formato: abertas, fechadas e *loops*;
 - tamanho do dado: 32 *bits*;
 - graus: 1, 2 até 3.
- Curvas 3D:

- tipos: geradas por computador e conjunto de curvas obtidas da digitalização de objetos reais;
- formato: abertas, fechadas, aproximando uma superfície;
- tamanho do dado: 32 *bits*;
- graus: 1, 2 até 3.
- Superfícies:
 - tipos: geradas por computador (plano, plano retorcido, paralelepípedo, cubo, esfera, toróide, *b-splines* e *NURBS*) e conjunto de pontos obtidos de superfícies reais digitalizadas.

Além dos aspectos gerais, cada conjunto de pontos teste apresenta características particulares que influenciam o processamento das *b-splines*, e que diferenciam um conjunto de pontos do outro. Um exemplo de uma destas características é a representação de seções cônicas, que exige uma *b-spline* do tipo racional para uma representação mais precisa. Cada conjunto de pontos teste, incluindo suas características particulares, foi descrito previamente à apresentação dos resultados numéricos.

Para cada conjunto de pontos teste foi gerado um *Soc*, de acordo com o número de pontos a serem gerados. Os *cores* que constituem o *Soc* de cada conjunto de pontos teste é repetido de acordo com o número de pontos a serem gerados, exceto os circuitos do método *BFEA*, que são replicados de acordo com o número de pontos de controle, conforme o método descrito originalmente. Os dados de entrada para os *Socs* de cada conjunto de pontos teste são: os vetores com as coordenadas dos pontos de controle, o vetor de nós, o número de pontos a serem gerados, e o tamanho dos dados (neste trabalho adotado como inteiro de 32 *bits*), sendo os dois últimos especificados em um arquivo de diretivas como parâmetros.

O tempo de execução, dado em nanosegundos e em número de ciclos do sinal do *clock*, para segmentos de retas, arcos de circunferências, planos e esferas, foi comparado com os respectivos tempos de execução dos *cores* para linhas e circunferências, nos conjuntos de pontos que envolvem a representação deste tipo de primitiva geométrica. Se o número de pontos de controle (dados iniciais), ou o número de pontos gerados, exceder a capacidade máxima de armazenamento dos registradores e da memória *on-chip* do *FPGA* (cuja capacidade é descrita na seção 4.3. Materiais), é incluído também o tempo de transferência de dados entre o *FPGA* e a memória externa *SDRAM*, responsável pelo armazenamento secundário dos dados.

Como o *FPGA* possibilita diversas configurações para uma mesma aplicação, o mesmo número de *cores*, necessários para gerar um ponto de uma *b-spline*, é utilizado para cada método testado, em cada conjunto de pontos teste. Por exemplo, para a geração de um ponto de uma *b-spline* de grau 2 pelo método *BFEA*, um *core* é considerado o conjunto dos cinco circuitos de elementos de processamento do grafo das funções base (dois para a camada 1 e três para a camada 2), para o método das funções base fixas, três funções base fixas, e para o *fast* Cox-de Boor, cinco funções base. O número de *cores* utilizado é dado na descrição da característica do conjunto de pontos teste.

Os pontos resultantes da *b-spline*, gerada para cada conjunto de pontos de controle teste, são visualizados graficamente. O número de dados de entrada, *hardware* utilizado (em número de elementos lógicos) e número de ciclos do sinal de *clock*, necessários para a geração dos pontos, são descritos nos resultados na forma tabular, para os diferentes métodos implementados.

Para a visualização dos gráficos dos resultados, gerados pelo sistema *FPGA*, os respectivos dados foram obtidos em arquivo texto (formato *.txt*) diretamente da memória *on chip* do *FPGA*, via ferramenta disponibilizada pelo *Quartus Signal Tap II Logic Analyzer* (seção 4.5. Comunicação de Dados). Estes dados foram transferidos do conteúdo da memória *on chip* do *FPGA*, via interface *JTAG*, na depuração do sistema em funcionamento, por esta ferramenta *EDA*. Neste processo, foi executado o *upload* da configuração do dispositivo de lógica reconfigurável, do computador pessoal para o *FPGA*. Posteriormente, com o sistema *FPGA* em funcionamento, os dados selecionados foram transferidos para a memória *on-chip* do *FPGA*, e transferidos via *USB JTAG* para o computador pessoal. Os dados transferidos podem ser visualizados pela interface gráfica da ferramenta *EDA*, mas para os resultados obtidos neste trabalho, foram gravados em um arquivo no formato texto simples (extensão *.txt*), formatados e convertidos para um arquivo de dados compatível com o *Matlab Mathworks*, que efetuou a geração dos gráficos correspondentes pelo seu sistema de visualização computacional.

Os requisitos de *hardware* do *FPGA*, exigidos para cada método implementado, foram obtidos via relatórios de compilação da ferramenta de automação de projetos eletrônicos *Quartus II 11.0* da Altera Corporation (ALTERA, 2011). Estes relatórios foram finalizados após os processos de compilação, análise, síntese e preenchimento do código fonte, apresentando, para o modelo de *FPGA* utilizado (ALTERA, 2008), o número total de elementos lógicos, dividido em funções combinacionais e registradores lógicos dedicados, para cada um dos métodos. Foi possível obter também o número total de registradores, pinos,

pinos virtuais, *bits* da memória *on chip*, multiplicadores embarcados e de *PLLs*, utilizados por cada método. Estes números não são descritos nos resultados por não apresentarem ou por apresentarem variações não significativas em seu valor (seção 4.3. Materiais e Apêndice D).

O processo de compilação e síntese de circuitos é dividido em análise, síntese, preenchimento e análise de tempo, exigindo alto desempenho computacional do sistema responsável pelo processamento da ferramenta *EDA*. Devido aos requisitos de desempenho computacional do ambiente de desenvolvimento, o número de *cores* ou de elementos lógicos a serem utilizados é limitado, para não exigir um tempo de processamento elevado a ponto de inviabilizar a implementação do circuito, principalmente na etapa de análise e síntese da linguagem *HDL*. Deste modo, de acordo com o sistema computacional *PC* utilizado pela ferramenta *EDA Quartus II IDE*, o número máximo de *cores* implementados em paralelo é de 1.024, baseado no *core* do método *fast Cox-de Boor* de grau 2 de duas variáveis, com o *Quartus II IDE* sendo executado em um sistema *PC* com um microprocessador de quatro núcleos e 8GB de memória principal.

Os exemplos dos conjuntos de pontos teste que fazem uso de mais de 1.024 *cores* gerados em *HDL* foram divididos em etapas, ou utilizam a memória externa *SDRAM* presente na placa de desenvolvimento (sendo indicado na descrição do conjunto de pontos teste).

Nas seções seguintes, serão apresentados os resultados referentes a cada tipo de conjunto de pontos teste utilizado, para cada método de geração de *b-splines*, segmentos de reta, e arcos de circunferências. Os *cores* do segmento de reta e circunferência são denominados de *line* e *circunference*, respectivamente.

5.3. Curvas

5.3.1. Curvas 2D

- Segmento de reta:

A Figura 66 mostra os segmentos de reta, de 128 pontos cada, gerados por *b-splines* no sistema *FPGA*. Cada segmento difere do outro pelo número de pontos de controle necessários, de acordo com o grau da *spline* gerada. Estes conjuntos de pontos demonstram que um segmento de reta pode ser representado por *b-splines* de graus diferentes. A representação mais simples compreende dois pontos de controle em uma camada de combinação convexa, gerando os pontos de um segmento de reta entre estes dois pontos de controle dados, de acordo com a variação dos valores do parâmetro. Esta representação utiliza

uma combinação convexa da *b-spline*, sendo o mesmo que a representação paramétrica de um segmento de reta.

No caso de um segmento de reta representado por uma *b-spline* de grau igual ou maior que 2, no mínimo 3 pontos de controle são necessários, e consequentemente, no mínimo duas combinações convexas para cada valor de parâmetro. Esse cálculo adicional (equação 10) deve ser levado em conta na geração de *splines* de grau maior que 1, que possuem pontos de controle desnecessários para representar primitivas gráficas como o segmento de reta.

Figura 66 – Retas *b-splines* de graus 1, 2 e 3 (esquerda para direita) e pontos de controle (quadrados).

De acordo com a Tabela 18, o número de elementos lógicos necessários para gerar o mesmo número de pontos na representação em *b-splines* de grau 1, para segmentos de retas, em qualquer um dos métodos *BFEA*, funções base fixas ou *fast Cox-de Boor*, é menor em relação à representação pelo *core line*, evidenciando a vantagem das *b-splines* para esta representação somente. O tempo de execução é menor somente para o método *fast Cox-de Boor* em relação ao *core line*.

O número de elementos lógicos aumenta com o grau da *b-spline* gerada, para os três métodos, sendo sempre maior para o método *fast Cox-de Boor*. No entanto, o tempo de processamento é sempre maior para o método *BFEA*, para todos os valores de grau.

Tabela 18 – Resultados das implementações para a geração de um segmento de reta de 128 pontos.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Line</i>	2	1	1.105	740	37
<i>BFEA</i>	2	1	93	480	24
	3	2	174	10.240	512
	4	3	604	14.480	724
Funções Base Fixas	2	1	93	480	24
	3	2	285	860	43
	4	3	631	1.020	51
<i>Fast Cox-de Boor</i>	2	1	93	480	24
	3	2	688	1.000	50
	4	3	1.024	1.200	60

- Arco de circunferência de 90°:

No caso de curvas que representam arcos de seções cônicas, a *b-spline* não apresenta uma representação fiel dos pontos gerados, porque as seções cônicas exigem uma representação racional. Deste modo, é necessário o uso de uma *b-spline* racional, na qual é incluída a coordenada homogênea (o peso). O vetor de nós também é modificado para um vetor não uniforme, para adequar os pontos o mais próximo possível dessas primitivas, como o arco de circunferência da Figura 67, constituído por 64 pontos. Deste modo, seções cônicas podem ser representadas pelas *NURBS*, como a circunferência, a qual é obtida por arcos de circunferência *NURBS* de 90°.

Figura 67 – Arco de circunferência *NURBS* de 90°.

Para gerar os resultados da Tabela 19, foram utilizados o seguinte vetor de nós e pesos:

Vetor de nós: $U = [0 \ 0 \ 0 \ 50 \ 100 \ 100 \ 100]$;

Pesos: $W = [100 \ 50 \ 100]$.

Quando os pesos são necessários, são incluídos os circuitos de mais uma coordenada (a coordenada homogênea), incrementando o número de elementos lógicos utilizados. No

caso de uma circunferência completa, é possível optar pela geração simultânea dos pontos dos quatro quadrantes, multiplicando por quatro o número de circuitos utilizados, ou pela geração de um quadrante e a replicação dos pontos nos eixos que passam pelo centro da circunferência.

Conforme observado na Tabela 19, todos os *cores b-spline* possuem um número de elementos lógicos menor que o *core circumference*, sendo que os *cores BFEA* e funções base fixas apresentam um número de elementos lógicos próximos. Os *cores b-spline* das funções base fixas e *fast Cox-de Boor* apresentam um tempo de execução menor do que o *core circumference*, e o *core BFEA* apresenta um tempo de execução maior.

Tabela 19 – Resultados das implementações para a geração de um segmento de circunferência de 90°, formado por 64 pontos.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos clock
<i>Circumference</i>	2	2	576	1.560	78
<i>BFEA</i>	4	2	205	10.240	512
Funções Base Fixas	4	2	214	960	48
<i>Fast Cox-de Boor</i>	4	2	333	1.000	50

- Circunferência:

Mesmo em uma circunferência representada por uma curva *NURBS*, o ajuste dos pesos e nós é sensível, principalmente em graus maiores que 2, devido à instabilidade numérica e ao número reduzido de pontos de controle, como ilustrado na Figura 68. Para essa Figura, são fornecidos a seguir os valores do vetor de nós e dos pesos (grau 2):

$$\text{Vetor de nós: } U = [0 \ 0 \ 0 \ 25 \ 50 \ 50 \ 75 \ 100 \ 100 \ 100];$$

$$\text{Pesos: } W = [100 \ 25 \ 75 \ 100 \ 75 \ 25 \ 100].$$

Na Figura 68 no segundo, terceiro e quarto (inferior esquerdo) quadrantes, são gerados arcos de circunferência não distorcidos, e somente o primeiro quadrante (superior direito) apresenta um arco de circunferência distorcido em relação aos demais. Tal fato se deve à presença da continuidade em cada nó, dada pelo cálculo recursivo das funções base racionais, de acordo com o grau da curva. Uma circunferência gerada pela replicação de arcos de circunferência *NURBS* é exibida na Figura 69, com um resultado visualmente melhor que a geração por sete pontos de controle com os pesos modificados.

Figura 68 – Círculo *NURBS* de 128 pontos a partir de 7 pontos de controle (quadrados).

Figura 69 – Círculo *NURBS* de 128 pontos obtido pela junção de arcos de 90° .

Os resultados da Tabela 20 são similares aos resultados obtidos para um arco de circunferência *NURBS* de 90° , exceto para o caso de geração da circunferência por 7 pontos de controle com os pesos modificados, caso em que ocorre um acréscimo no número de elementos lógicos. Quando os pesos são necessários, são incluídos os circuitos para mais uma coordenada, incrementando o número de elementos lógicos utilizados.

No caso de uma circunferência completa, é possível optar pela geração simultânea dos pontos dos quatro quadrantes, incrementando para quatro o número de circuitos utilizados, ou pela geração de um quadrante e replicação dos pontos nos eixos que passam pelo centro do círculo, do mesmo modo que no *core circumference*.

Conforme os resultados apresentados na Tabela 20, o *core fast* Cox-de Boor apresenta um tempo de execução menor em relação ao *core circumference*, porque o *core circumference* utiliza uma tabela de valores trigonométricos pré-fixados. O *core b-spline* de funções base fixas não apresenta o mesmo resultado visual que os outros *cores*, porque utiliza somente vetores de nós uniformes. Porém esse *core* apresenta o menor tempo de execução entre todos os *cores* testados, utilizando o menor número de elementos lógicos. O *core BFEA* apresenta um tempo de execução superior ao *core circumference*, mas utiliza um número menor de elementos lógicos.

Tabela 20 – Resultados das implementações para a geração de uma circunferência NURBS de 128 pontos.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Circunference</i>	3 (ponto inicial, final e raio)	-	1.307	2.540	127
<i>BFEA</i>	7	2	334	4.560	228
Funções Base Fixas	7	2	439	840	42
<i>Fast</i> Cox-de Boor	7	2	688	980	49

- Curva em laço aberto:

Curvas como a da Figura 70 são utilizadas como teste para verificar visualmente a influência da ordem dos pontos de controle, a propriedade do casco convexo das *b-splines*, a suavização da curva e a distribuição dos parâmetros. Nos pontos em que a direção da curva modifica-se constantemente (parte interna do laço), há maior concentração de pontos, enquanto nas seções que apresentam um comportamento mais linear, os pontos gerados são mais escassos.

Figura 70 – Curva *b-spline* de 128 pontos em laço aberto de graus 1, 2 e 3 (sentido de fora para dentro).

Os resultados dos tempos de execução da Tabela 21 são similares aos resultados obtidos para os segmentos de reta, com os mesmos valores de graus e número de pontos gerados. O número de elementos lógicos utilizados também é similar.

Tabela 21 – Resultados das implementações para a geração de uma curva *b-spline* em laço aberto.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>BFEA</i>	5	1	93	5.120	256
		2	174	1.020	512
		3	604	14.480	724
Funções Base Fixas	5	1	93	480	24
		2	285	860	43
		3	631	1.020	51
<i>Fast Cox-de Boor</i>	5	1	93	480	24
		2	688	980	49
		3	1.024	1.320	66

- Curva em formato 8:

A curva em formato “8” apresenta as mesmas características que uma curva em laço aberto, possuindo dois laços que se auto interseccionam, conforme a Figura 71. É possível verificar no ponto inicial, que coincide com o final, o efeito da repetição dos valores dos nós inicial e final, assim como a quebra na continuidade da curva nas suas extremidades.

Figura 71 – Curva *b-spline* fechada de 128 pontos em laço duplo (formato 8), graus 1, 2 e 3 (sentido de fora para dentro).

Os resultados da Tabela 22 são similares àqueles obtidos para a curva de laço aberto, considerando os mesmos graus e mesmo número de pontos gerados, observando-se a repetição do ponto de controle inicial como final, produzindo um contorno fechado.

Pode-se observar, pelos resultados obtidos nas Tabela 18, Tabela 21 e Tabela 22, que a representação *b-spline* para retas, curvas abertas e curvas fechadas, possui um tempo de execução e número de recursos do *FPGA* similares para os métodos utilizados, com o mesmo grau e número de pontos, apesar de produzirem formas geométricas diferentes.

Tabela 22 – Resultados das implementações para a geração da curva *b-spline* fechada em formato “8”.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>BFEA</i>	6	1	93	5.120	256
		2	174	10.240	512
		3	604	14.480	724
Funções Base Fixas	6	1	93	480	24
		2	285	860	43
		3	631	1.020	51
<i>Fast Cox-de Boor</i>	6	1	93	740	37
		2	688	900	45
		3	1.024	1.260	63

- Curva com seção reta e extremidades com arco de circunferência:

A curva de 64 pontos, ilustrada pela Figura 72, apresenta pesos de valores diferentes para as curvas nas extremidades, e pode ser representada também pela junção de arcos de circunferência e segmento de reta. A representação via *NURBS* é única para todos os pontos desta curva.

Figura 72 – *NURBS* com seções de segmento de reta e arco de circunferência.

A Tabela 23 apresenta os resultados dos *cores b-spline* com os pesos adicionados como mais uma coordenada, com a representação feita em grau 2, para se adequar aos arcos de circunferência das extremidades. Como na representação da reta, curva aberta e fechada, é necessário somente uma *spline* para representar o segmento de reta e os dois arcos de circunferência da Figura 72.

Dos resultados, pode-se concluir que os *cores* de funções base fixas e *fast Cox-de Boor* apresentam o menor tempo de execução, sendo o menor número de elementos lógicos utilizado pelo *core BFEA*. A representação por arcos de circunferência e segmento de reta utiliza um número de elementos lógicos superior aos *cores b-spline*.

Tabela 23– Resultados das implementações para a curva *NURBS* de 64 pontos com extremidades em arcos de circunferência.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Line/ Circunference</i>	4	1/2	840	2.540	127
<i>BFEA</i>	7	2	131	10.240	512
Funções Base Fixas	7	2	214	960	48
<i>Fast Cox-de Boor</i>	7	2	333	1.000	50

- Curva de amplitude decrescente:

Uma curva *NURBS* ondulada pode ser gerada pela alternância nos valores dos pesos a cada dois pontos de controle ao longo da curva. No caso de variação no grau da curva, os pesos devem ser modificados de acordo com a variação (uma curva mais “aberta” ou “fechada”). Na curva ilustrada pela Figura 73, as ondulações são obtidas pelas aproximações com arcos de circunferência, variando-se o peso dos pontos de controle à medida que a ondulação (raio de curvatura) dos arcos de circunferência reduzem seu valor. Os pesos nas extremidades dos arcos possuem valor unitário, enquanto os pesos associados aos outros pontos de controle variam de acordo com a redução no raio de curvatura, necessitando de ajuste individual. Este ajuste nos valores dos pesos causa alteração na curva em outros pontos de controle associados a este peso, mesmo para uma curva de grau menor (segundo grau).

Figura 73 – Curva *NURBS* de grau 3, de 128 pontos, com amplitude decrescente.

Os resultados da Tabela 24 são aproximados aos resultados obtidos para outras curvas ou segmentos de reta de mesmo número de pontos e mesmo grau. Com graus maiores para uma curva que apresenta variações, os pesos são necessários para tentar manter ou evitar o ajuste da curva às condições de continuidade, aumentando o número de elementos lógicos necessários no *FPGA* e adicionando mais um parâmetro de ajuste (o peso).

Para esta implementação, o *BFEA* utilizou o menor número de elementos lógicos que os demais *cores* para os graus utilizados, no entanto, os menores tempos de processamento foram obtidos pelos *cores* de funções base fixas e *fast Cox-de Boor*.

Tabela 24– Resultados das implementações para curvas *NURBS* com ondulações decrescentes.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>BFEA</i>	28	1	93	5.320	266
		2	174	10.480	524
		3	604	14.760	738
Funções Base Fixas	28	1	93	560	28
		2	285	980	49
		3	631	1.240	62
<i>Fast Cox-de Boor</i>	28	1	93	560	28
		2	688	1.160	58
		3	1.024	1.380	69

5.3.2. Curvas 3D

- Plano:

De acordo com a Figura 74a, o plano é constituído por meio de um conjunto de segmentos de reta. Deste modo não há a influência da mudança do valor de um ponto de controle nos pontos vizinhos, como no caso de superfícies *splines*. A alteração do peso associado a um ponto de controle só influencia a curva (no caso uma reta) gerada por este ponto de controle, como ilustrado na Figura 74b.

Figura 74 – Plano de 1.024 pontos formado por 16 retas *NURBS* de 64 pontos cada (63a) e plano com o peso do ponto de controle médio modificado em uma reta (63b).

Na Tabela 25, os resultados exibidos para um plano formado por retas pelo *core line* e pelo *core spline* evidenciam que o tempo de execução do *core spline* é menor que o do *core line*, como no caso das curvas 2D. Entre os *cores spline*, o *BFEA* apresentou o menor número de elementos lógicos utilizados, sendo o menor tempo de processamento apresentado pelo método de funções base fixas, porém, similar ao *core fast Cox-deBoor*.

Tabela 25 – Resultados das implementações para a geração de um plano formado por retas *NURBS*.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Line</i>	40 (20x2)	1	17.680	740	37
<i>BFEA</i>	120 (40x3)	1	1.488	5.120	256
		2	2.784	10.240	512
Funções Base Fixas	120 (40x3)	1	1.488	540	27
		2	4.560	960	48
<i>Fast Cox-de Boor</i>	120 (40x3)	1	1.488	540	27
		2	16.384	1.000	50

- Plano retorcido:

De acordo com a Figura 75, o plano retorcido consiste em um conjunto de retas *b-spline* de grau 2 formando um plano retorcido no espaço, a partir de 48 pontos de controle. Retas *b-spline* de grau 3 somente aumentam o número de pontos de controle, produzindo o mesmo resultado. Na figura 64, a forma do plano foi obtida pelo posicionamento e ordem dos pontos de controle, que fornecem a topologia da superfície final. Para uma superfície *b-spline* seriam necessários somente 24 pontos de controle invertendo-se a ordem destes pontos na direção transversal (isoparamétrica), reduzindo o número de dados de entrada em relação à geração de segmentos de reta.

Figura 75 – Plano retorcido de 1.024 pontos formado por 16 retas *NURBS* de 64 pontos cada.

Os resultados dos tempos de execução apresentados na Tabela 26 são similares aos resultados obtidos para o conjunto de retas formando um plano (figura 63), porque esses conjuntos de pontos teste demonstram que o posicionamento dos pontos de controle para a geração das retas não influencia o resultado final, para o mesmo número de pontos.

Tabela 26 – Resultados das implementações para a superfície do plano retorcido formado a partir de retas *b-splines*.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Line</i>	40 (20x2)	1	17.680	740	37
<i>BFEA</i>	120 (20x3)	1	1.488	5.120	256
		2	2.784	10.240	512
Funções Base Fixas	120 (20x3)	1	1.488	540	27
		2	4.560	960	48
<i>Fast Cox-de Boor</i>	120 (20x3)	1	1.488	140	27
		2	16.384	1000	50

- Paralelepípedo:

A Figura 76 mostra um conjunto de retas formando uma superfície 3D na forma de um paralelepípedo. A falta da continuidade e da interpolação de pontos no sentido transversal (isoparamétrico) não permite uma forma “suave” em relação a uma superfície *spline*.

Figura 76 – Conjunto de 36 retas *b-splines* (108 pontos de controle) no formato de um paralelepípedode 4.608 pontos.

Os resultados da Tabela 27, referentes ao conjunto de pontos da Figura 76, foram obtidos em duas etapas de processamento dos *cores line* e *b-splines*, sendo o número de elementos lógicos das duas etapas somados. Esses resultados demonstram que os *cores* mantêm a mesma eficiência no processamento e na utilização dos elementos lógicos disponíveis.

Tabela 27 – Resultados das implementações para retas *b-splines* formando um paralelepípedo.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Line</i>	72 (36x2)	1	39.780	3.000	150
<i>BFEA</i>	72 (36x2)	1	2.961	5.120	256
	108 (36x3)	2	5.531	10.240	512
Funções Base Fixas	72 (36x2)	1	2.961	540	27
	108 (36x3)	2	9.062	960	48
<i>Fast Cox-de Boor</i>	72 (36x2)	1	2.961	540	27
	108 (36x3)	2	32.544	1.000	50

- Cilindro:

Conforme ilustrado pela Figura 77, um cilindro pode ser formado por um conjunto de segmentos de retas *b-splines*, dispostos paralelamente no espaço 3D de acordo com os pontos de uma circunferência *NURBS*. Esta forma de geração de um cilindro é mais simples e eficiente em relação às superfícies, mas não possui características como a continuidade.

Figura 77 – 50 segmentos de retas *b-splines* de 25 pontos cada dispostos com os pontos inicial e final coincidentes com os pontos de uma circunferência *NURBS*.

Os resultados da Tabela 28 apresentam o tempo de execução para a geração dos pontos das retas *spline*. A localização de cada reta exige a geração de uma circunferência *NURBS*, e três pontos de controle para cada reta. O *core line* apresenta um número de elementos lógicos próximo ao *core fast* Cox-de Boor, que apresenta o menor tempo de execução. O *core BFEA* apresenta o menor número de elementos lógicos.

Tabela 28 – Resultados das implementações para retas *NURBS* formando um cilindro.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Line</i>	128 (64x2)	-	10.791	2.960	148
<i>BFEA</i>	21 (7x3)	2	2.023	2.480	124
Funções Base Fixas	21 (7x3)	2	3.313	1.240	62
<i>Fast</i> Cox-de Boor	21 (7x3)	2	11.904	1.180	59

- Circunferências paralelas:

A Figura 78 apresenta um conjunto de circunferências *NURBS* paralelas, de raio variável, de modo a aproximar-se de uma esfera, sem a continuidade proporcionada por uma superfície *spline*.

Figura 78 – 10 circunferências *NURBS* de 64 pontos cada em planos paralelos.

De acordo com os resultados da Tabela 29, quando os pesos são necessários para o *core spline*, são incluídos os circuitos para mais uma coordenada no *FPGA*, incrementando o número de elementos lógicos utilizados. Para aproximar os pontos gerados a uma circunferência, são necessários os ajustes precisos dos pesos e do posicionamento, exigindo a intervenção do usuário. Os resultados são aproximados para circunferências 2D com o aumento proporcional no tempo de execução em relação ao acréscimo de mais uma coordenada. Para a geração de circunferências *NURBS*, a falta de precisão na representação

O *core circumference*, apesar de apresentar o maior número de elementos lógicos, apresenta o menor tempo de processamento, enquanto o *core BFEA* apresenta o menor número de elementos lógicos, mas o maior tempo de execução.

Tabela 29 – Resultados das implementações de circunferências *NURBS* em planos paralelos.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Circunference</i>	30 (10 x raio x centro)	2	9.129	1.040	52
<i>BFEA</i>	70 (7x10)	2	889	2.480	124
Funções Base Fixas	70 (7x10)	2	1.425	1.240	62
<i>Fast Cox-de Boor</i>	70 (7x10)	2	5.120	1.180	59

- Paralelepípedo helicoidal:

A Figura 79 exibe uma curva *b-spline* em que os pontos de controle estão dispostos na forma de um paralelepípedo, mas ordenados sequencialmente, determinando o formato de curvas em seções transversais conectadas. Neste caso se trata de uma curva contínua, e não de um conjunto de curvas no espaço.

Figura 79 - Curva *b-spline* de 512 pontos em formato de paralelepípedo helicoidal.

Os resultados do tempo de execução da forma da Figura 79 são exibidos na Tabela 30. Somente os circuitos *spline* foram utilizados para esta curva contínua, com o *BFEA* apresentando o menor número de elementos lógicos utilizados, mas um tempo de processamento significativamente superior aos outros *cores*, e os *cores* funções base fixas e *fast* cox-de Boor com tempos de processamento similares.

Tabela 30 – Resultados das implementações para a curva *b-spline* em formato de paralelepípedo helicoidal.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>BFEA</i>	125	2	1.392	10.240	512
Funções Base Fixas	125	2	2.296	960	48
<i>Fast</i> Cox-de Boor	125	2	8.192	1.000	50

- Cápsula:

Como exibido pela Figura 80, a “cápsula” consiste em um conjunto de pontos no formato de um cilindro fechado com as extremidades curvas, formado por uma série de curvas *b-splines* de seções retas e arcos de circunferência nas extremidades, com os pontos final e

inicial de cada curva coincidentes. O peso dos pontos de controle nas junções das retas com os segmentos de circunferência são modificados para apresentarem a curvatura.

Figura 80 Cilindro fechado de 512 pontos, com extremidades curvas formado por 24 curvas *NURBS* de 64 pontos cada.

Conforme os resultados da Tabela 31, no *FPGA*, quando os pesos são necessários, são incluídos os circuitos para mais uma coordenada, incrementando o número de elementos lógicos utilizados. O *fast Cox* de Boor apresenta o maior número de elementos lógicos entre os *cores b-splines*, e o menor tempo de execução, e o *core BFEA* utiliza o menor número de elementos lógicos.

Tabela 31 – Resultados das implementações para um cilindro fechado de extremidades curvas formado por curvas *NURBS*.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Line/Circunference</i>	144(48/96)	1/2	17.448	2.540	127
<i>BFEA</i>	168 (24x7)	2	3.172	2.480	124
Funções Base Fixas	168 (24x7)	2	5.138	1.240	62
<i>Fast Cox</i> -de Boor	168 (24x7)	2	7.077	1.180	59

- Curvas de imagens médicas de seções transversais:

A Figura 81 apresenta curvas obtidas de pontos segmentados de imagens médicas de ressonância magnética (de um joelho), interpolados por curvas *b-splines* dispostas no espaço, de acordo com as imagens adquiridas. Estas imagens foram segmentadas e as bordas

extraídas. Nas curvas reconstruídas, é possível observar um erro nesta aproximação, causado pela interpolação de retas nos pontos iniciais das curvas, devido à segmentação de pontos inexistentes nas bordas das imagens.

Figura 81 – Nuvem de pontos original (esquerda) de 20 curvas de 483 pontos cada, e curvas reconstruídas (direita) com 1.000 pontos cada (total de 20.000 pontos).

O processo de obtenção dos resultados da Tabela 32 foi efetuado em quatro etapas para o processamento e armazenamento dos pontos gerados. Esses resultados apresentam uma diferença significativa em relação aos resultados de outros conjuntos de pontos 3D. Isto se deva ao fato do elevado número de pontos gerados, sendo necessário o acesso à memória externa *SDRAM* (Seção 4.4 e Apêndice E). O cálculo do número de elementos lógicos utilizados foi feito pela soma aproximada dos elementos utilizados em cada etapa.

Para a geração dos pontos da *b-spline* a partir das imagens médicas, o método das funções base fixas apresentou o melhor desempenho em termos de tempo de processamento. O *core BFEA* comprovou a utilização do menor número de elementos lógicos em todas as implementações.

Tabela 32– Resultados das implementações para curvas *b-splines* da segmentação de imagens médicas.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de acesso à <i>SDRAM</i> (ns)	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>BFEA</i>	9.660 (20x483)	1	7.018	$5,35 \times 10^3$	5.000	250
		2	13.131	12×10^3	10.800	540
Funções Base Fixas	9.660 (20x483)	1	7.018	$0,335 \times 10^3$	940	47
		2	21.508	$6,24 \times 10^3$	1.400	70
<i>Fast Cox-de Boor</i>	9.660 (20x483)	1	7.018	$4,6 \times 10^3$	5.000	250
		2	77.280	$10,2 \times 10^{37}$	6.600	330

5.4. Superfícies

- Plano:

A Figura 82 apresenta a superfície *b-spline* de um plano reto, em que o número de pontos de controle necessários é reduzido em relação à representação por retas *splines*. A modificação de um ponto de controle influencia os pontos vizinhos, de acordo com o grau da superfície nas duas direções (longitudinal e isoparamétrica).

Figura 82 – Superfície *b-spline* de um plano de 2.048 pontos obtidos a partir de 16 pontos de controle (pontos quadrados).

Os resultados da Tabela 33 apresentam os tempos de execução para a geração dos pontos de uma superfície *b-spline* formando um plano reto, em que é necessário repetir os cálculos para a geração das curvas em duas direções. Deste modo, o tempo de execução aumenta se os graus das curvas possuem o mesmo valor nas duas dimensões, mas o número de pontos de controle utilizados é reduzido, produzindo maior compactação dos dados com um maior tempo de execução. Como o conjunto apresenta um número de pontos gerados maior que 1.024, é executado o acesso à memória externa, elevando o tempo de execução dos circuitos.

Deste modo, o *BFEA* apresenta a menor área, e o circuito de funções base fixas apresenta o menor tempo estimado de execução, similar ao *fast Cox-de Boor*.

Tabela 33 – Resultados das implementações para a geração de uma superfície *b-spline* de um plano.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de acesso à <i>SDRAM</i> (ns)	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>BFEA</i>	16 (4x4)	1	1.817	$30,6 \times 10^3$	8.320	416
		2	8.352	$31,4 \times 10^3$	20.480	1.024
Funções Base Fixas	16 (4x4)	1	1.817	$30,6 \times 10^3$	240	12
		2	18.240	$30,6 \times 10^3$	800	40
<i>Fast Cox-de Boor</i>	16 (4x4)	1	1.817	$30,6 \times 10^3$	360	18
		2	53.594	$30,7 \times 10^3$	1.040	72

- Plano com “cotovelo”:

A modificação do valor do peso de um ponto de controle na superfície *NURBS* de um plano gera uma modificação no formato do plano, conforme mostra a Figura 83, com o peso de um ponto reduzido em 10% (1 para 0,9). O efeito dos pesos em uma superfície é proporcional ao grau da superfície nas duas direções de formação.

Figura 83 – Superfície *NURBS* de 1.024 pontos de um plano, com o peso de um ponto de controle reduzido em 10%.

Os resultados do tempo de processamento para a superfície plana são exibidos na Tabela 34. Como nas curvas 3D, o circuito das funções base fixas apresentam o menor tempo de execução, seguido do circuito fast Cox-de Boor. O *BFEA* apresenta o menor número de elementos lógicos utilizado.

Tabela 34 – Resultados das implementações para a geração de uma superfície *NURBS* de um plano com o peso associado ao ponto de controle médio reduzido em 10%.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>BFEA</i>	16 (4x4)	1	2.969	980	49
		2	1.090	22.280	1.114
Funções Base Fixas	16 (4x4)	1	2.969	890	49
		2	5.680	2.160	108
<i>Fast Cox-de Boor</i>	16 (4x4)	1	30.625	980	49
		2	71.460	2.400	120

- Plano retorcido em 180°:

A superfície curva, conforme a Figura 84, formada pela inversão da ordem dos pontos de controle na seção média de um plano, causa uma torção nesta superfície, demonstrando a influência não somente da posição, mas da ordem dos pontos de controle.

Figura 84 – Superfície *b-spline* de 1.024 pontos gerada a partir de 16 pontos de controle, em que os pontos de controle finais possuem ordem inversa em relação aos pontos iniciais.

A Tabela 35 mostra os resultados para a superfície com uma torção de 180°, com o core *BFEA* apresentando o menor número de elementos lógicos utilizados, e o core de funções base fixas com o menor tempo de execução, como demonstrado anteriormente.

Tabela 35 – Resultados das implementações para a superfície *b-spline* de um plano em curva de 180°.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Line</i>	50 (25x2)	1	9.300	2.700	135
<i>BFEA</i>	16 (4x4)	1	2.969	1.100	55
		2	4.197	21.160	1.058
Funções Base Fixas	16 (4x4)	1	2.969	1.100	55
		2	6.909	1.940	97
<i>Fast Cox-de Boor</i>	16 (4x4)	1	2.969	1.100	55
		2	53.594	2.400	120

- Paralelepípedo:

A Figura 85 apresenta a superfície *b-spline* final de um paralelepípedo, em que o número de pontos de controle e de nós, nas direções longitudinal e isoparamétrica, possuem valores diferentes.

Figura 85 – Superfície *b-spline* de 1.024 pontos (50x50) e grau 2x2, no formato de um paralelepípedo, gerada a partir de 17x16 pontos de controle (quadrados).

Os resultados para o paralelepípedo são exibidos na Tabela 36, similares aos resultados obtidos para o conjunto do plano, descrevendo o *BFEA* como o circuito que utiliza menor número de elementos lógicos, e o circuito de funções base fixas com o menor tempo de execução.

Tabela 36 – Resultados das implementações para a superfície *b-spline* no formato de um paralelepípedo.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Line</i>	2	1	9.300	2.900	145
<i>BFEA</i>	272 (17x16)	1	2.969	620	31
		2	4.197	24.280	1.214
Funções Base Fixas	272 (17x16)	1	2.969	620	31
		2	6.909	1.820	91
<i>Fast Cox-de Boor</i>	272 (17x16)	1	2.969	620	31
		2	71.458	2.440	122

- Cilindro:

A Figura 86 mostra um cilindro, formado por uma superfície *NURBS*, com pontos de controle dispostos em forma de um paralelepípedo, similar à implementação de circunferências no espaço.

Figura 86 – Cilindro *NURBS* de 512 pontos (64x8) em formato cilíndrico.

Os resultados para a superfície cilíndrica *NURBS* são dados na Tabela 37, assim como os resultados para o cilindro formado por circunferências (*core circumference*). Apesar do *core circumference* apresentar menor número de dados de entrada, o número de circuitos utilizados para o processamento concorrente vezes o número de elementos lógico utilizado, torna este *core* o menos eficiente. Além do tempo de processamento e do número de elementos lógicos utilizados, não possui a característica da continuidade.

O *core BFEA* apresenta o menor número de elementos lógicos utilizados, inclusive em relação ao *core circumference*. O circuito *fast Cox-de Boor* apresenta o menor tempo de execução, devido à representação *NURBS*.

Tabela 37 – Resultados das implementações para a superfície *NURBS* de um cilindro.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº de elementos lógicos	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>Circumference</i>	16 (2x8)	1	10.921	1.560	78
<i>BFEA</i>	56 (7x8)	2	2.907	24.240	1.212
Funções Base Fixas	56 (7x8)	2	4.546	1.880	94
<i>Fast Cox-de Boor</i>	56 (7x8)	2	21.952	1.000	50

- Toróide:

O toróide apresentado na Figura 87 é formado por uma superfície *NURBS*, a partir de pontos de controle dispostos em seções transversais no formato de quadrados. A disposição entre as seções possuem o formato de circunferência, formando um toróide.

Figura 87 – Superfície *NURBS* de 2.048 pontos gerados a partir de 56 pontos de controle (quadrados) em 8 seções transversais de 7 pontos no formato quadrado, dispostas em circunferência.

De acordo com a Tabela 38, o conjunto de pontos de controle para constituir o toróide utiliza os pesos para cada seção transversal, i.e., os pontos de controle de cada curva isoparamétrica possuem mais uma coordenada (homogênea), incrementando o número de elementos lógicos. O *BFEA* apresenta o menor número de elementos lógicos utilizados, significativamente abaixo do número de elementos utilizado pelo circuito *fast Cox-de Boor*. Os tempos de execução são similares, devido ao acesso à memória externa *SDRAM*, sendo o mais rápido o circuito de funções base fixas.

Tabela 38– Resultados das implementações de um toróide formado por superfície *NURBS*.

<i>Cores</i>	Pontos iniciais necessários	Grau	Nº elementos lógicos	Tempo de acesso à <i>SDRAM</i> (ns)	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>BFEA</i>	56	2	11.329	$32,4 \times 10^3$	20.160	1.008
Funções Base Fixas	56	2	18.227	$30,8 \times 10^3$	520	26
<i>Fast Cox-de Boor</i>	56	2	58.539	$30,9 \times 10^3$	720	36

- Reconstrução aproximada da superfície de um joelho:

A Figura 88a exibe uma superfície da reconstrução de um conjunto de pontos, obtidos pelo processamento digital, das imagens de ressonância magnética de seções transversais de um joelho. Os pontos da superfície *b-spline* final interpolam os pontos das imagens, dispostos no espaço 3D, com o conjunto dos pontos de controle dado pela Figura 88b.

Figura 88 – Superfície *b-spline* de grau 2 (a) composta por 32.800 (410x40x2) pontos gerados a partir de 3.600 (450x20x2) pontos de controle (b) dispostos em 20 seções transversais de 900 pontos cada.

A Tabela 39 mostra os resultados para a superfície gerada com o grau 2, nas duas direções. Valores do grau maior que 2 geram instabilidade numérica, resultando em laços fechados para pontos de controle vizinhos, devido à condição de continuidade. O *BFEA* apresentou 50% elementos lógicos utilizados a menos, ao custo do tempo de execução, sendo o circuito mais lento. O circuito das funções base fixas apresentou o menor tempo de execução para este tipo de superfície.

Tabela 39– Resultados das implementações da reconstrução da superfície de um joelho a partir das imagens de ressonância magnética.

<i>Cores</i>	Pontos iniciais necessários	Grav	Nº elementos lógicos	Tempo de acesso à <i>DRAM</i> (ns)	Tempo de execução (ns)	Nº ciclos <i>clock</i>
<i>BFEA</i>	3.600	2	133.824	$494,3 \times 10^3$	108×10^3	5.400
Funções Base Fixas	3.600	2	218.448	$253,5 \times 10^3$	8340	417
<i>Fast Cox-de Boor</i>	3.600	2	661.500	$493,7 \times 10^3$	21780	1.089

5.5. Resultados - GPU

Na Tabela 40 são apresentados os resultados obtidos para a implementação do método *fast* Cox-de Boor na *GPU*, considerando-se os seguintes conjuntos de pontos: curvas 2D, 3D, superfícies e *NURBS*. Como a *GPU* possui uma arquitetura *multithreading* programável (HALFHILL, 2008), não é possível uma comparação com um dispositivo de lógica reconfigurável, por possuírem arquiteturas diferentes. Porém, é possível observar que os *FPGAs* são mais adequados à implementação em sistemas embarcados, utilizados em sistemas de manufatura e com possibilidade da utilização em sistemas de computação gráfica, enquanto as *GPUs* geralmente exigem um computador pessoal para o funcionamento conjunto (HOWES ET AL., 2006). As *GPUs* também exigem sistemas de dissipação de potência maiores em relação aos *FPGAs* com capacidade de processamento equivalente.

Dessa forma, os resultados da Tabela 40 apresentam somente o tempo de execução de um modelo de *GPU*. A título de ilustração, foram utilizados os mesmos conjuntos de pontos para o método *fast* Cox-de Boor, implementado em *FPGA*, apesar da compatibilidade das *GPUs* com funções do *OpenGL* (SHREINER ET AL., 2008). Essas funções são representadas por métodos de bibliotecas de alto nível no *OpenGL*, como a *gluNurbsCurve* e *gluNurbsSurface*, que geram e renderizam curvas ou superfícies a partir dos pontos de controle (racionais ou não racionais), vetor de nós, e ordem das funções base polinomiais do objeto *NURBS*. Os trabalhos de Krishnamurthy et al. (2007; 2008) apresentam uma descrição mais completa sobre a geração e operações de modelagem das *NURBS* em *GPU*.

Tabela 40 – Resultados do tempo de execução de *b-splines* para a implementação em *GPU* de curvas (2D e 3D) e superfícies.

	Grau	Número de pontos	Tempo de execução (nanossegundos)
Circunferência	2	128	26
Curva formato oito fechada	1	128	13
	2		24
	3		35
Curva ondulações decrescentes	1	128	13
	2		36
	3		45
Plano reto	1	2.048	36
	2		51
Plano retorcido formado por	1	2.048	36

retas	2		51
Cubo formado por retas	1	4.068	48
	2		55
Cilindro <i>NURBS</i>	2	1.250	38
Paralelepípedo helicoidal	2	1.024	31
Cilindro fechado (cápsula formada por 24 curvas)	2	1.024	35
Curvas de imagens médicas de seções transversais	1	20.000	$1,24 \times 10^6$
	2	(20x1000)	$2,4 \times 10^6$
Superfície plana com peso modificado em 10%	1	2.500	32
	2		48
	3		75
Paralelepípedo	1	2.500	37
	2		46
	3		77
Cilindro	2	768	48
Toróide	2	2.048	274×10^3
Superfície de imagens médicas	2	72.000	$21,88 \times 10^3$

Os sistemas *FPGAs* são mais adequados para a implementação de sistemas embarcados em relação às *GPUs*, devido principalmente à dissipação de potência e à necessidade da maioria das *GPUs* de um equipamento *host*, normalmente um computador pessoal. Porém, os fabricantes de *hardware* gráfico desenvolvem constantemente *chips* processadores gráficos, dedicados para equipamentos portáteis.

5.6. Considerações Finais

A Tabela 41 exibe um resumo dos resultados obtidos dos conjuntos de pontos teste, indicando o menor número de elementos lógicos utilizados e o menor tempo de processamento, entre os três métodos implementados para as *b-splines*. Este resumo mostra que, para estes resultados, o método das funções base fixas apresenta, em geral, a melhor implementação levando-se em conta a área versus tempo de processamento. Porém, este método é limitado a vetores de nós uniformes.

Tabela 41 – Resumo dos melhores resultados obtidos para as três implementações em *FPGA* para os *cores* implementados (FBF-Funções Base Fixas).

	Grau	Menor número de elementos lógicos (área)		Menor tempo de execução	
Segmento de reta	1	<i>BFEA</i>	93	FBF	24
	2	<i>BFEA</i>	174	FBF	43
	3	<i>BFEA</i>	604	FBF	51

Arco de circunferência	2	<i>BFEA</i>	205	FBF	48
Circunferência	2	<i>BFEA</i>	334	FBF	42
Curva laço aberto	1	<i>BFEA</i>	93	FBF	24
	2	<i>BFEA</i>	174	FBF	43
	3	<i>BFEA</i>	604	FBF	51
Curva formato oito	1	<i>BFEA</i>	93	FBF	24
	2	<i>BFEA</i>	174	FBF	43
	3	<i>BFEA</i>	604	FBF	51
Curva extremidades arco	2	<i>BFEA</i>	131	FBF	48
Curva com amplitude decrescente	1	<i>BFEA</i>	93	FBF	28
	2	<i>BFEA</i>	174	FBF	49
	3	<i>BFEA</i>	604	FBF	62
Plano	1	<i>BFEA</i>	1.488	FBF	27
	2	<i>BFEA</i>	2.784	FBF	48
Plano retorcido	1	<i>BFEA</i>	1.488	FBF	27
	2	<i>BFEA</i>	2.784	FBF	48
Paralelepípedo	1	<i>BFEA</i>	2.961	FBF	27
	2	<i>BFEA</i>	5.531	FBF	48
Cilindro	2	<i>BFEA</i>	2.023	<i>Fast Cox-de Boor</i>	59
Circunferências paralelas	2	<i>BFEA</i>	889	<i>Fast Cox-de Boor</i>	59
Paralelepípedo helicoidal	2	<i>BFEA</i>	1.392	FBF	48
Cápsula	2	<i>BFEA</i>	3.172	<i>Fast Cox-de Boor</i>	59
Curvas de imagens médicas	1	<i>BFEA</i>	7.018	FBF	$0,335 \times 10^3$
	2	<i>BFEA</i>	13.131	FBF	$6,24 \times 10^3$
Superfície plana	1	<i>BFEA</i>	1.817	FBF	$30,6 \times 10^3$
	2	<i>BFEA</i>	8.352	FBF	$30,6 \times 10^3$
Superfície plana com peso modificado	1	<i>BFEA</i>	2.969	FBF	49
	2	<i>BFEA</i>	5.680	FBF	108
Superfície plano retorcido	1	<i>BFEA</i>	2.969	FBF	55
	2	<i>BFEA</i>	4.197	FBF	97
Superfície paralelepípedo	1	<i>BFEA</i>	2.969	FBF	31
	2	<i>BFEA</i>	4.197	FBF	91
Superfície cilindro	2	<i>BFEA</i>	2.907	<i>Fast Cox-de Boor</i>	50
Toróide	2	<i>BFEA</i>	11.329	FBF	$30,8 \times 10^3$
Superfície de imagens médicas	2	<i>BFEA</i>	133.824	FBF	$253,5 \times 10^3$

Excluindo-se a limitação da uniformidade no vetor de nós, o método *fast Cox-de Boor* apresenta tempos de execução similares aos do método das funções base fixas, mas utiliza o maior número de elementos lógicos. O método *BFEA* utiliza o menor número de elementos

lógicos e de dados de entrada, mas apresenta o maior tempo de processamento entre os *cores b-spline*.

5.6.1. Discussões - Configuração Fixa

Neste trabalho não foi utilizada uma configuração fixa para a geração de *b-splines* em um *FPGA*, em que o número de circuitos para o processamento paralelo de cada ponto (parâmetro) é fixo. Uma configuração fixa genérica não foi utilizada pois o *Soc* de cada conjunto de pontos teste foi gerado de acordo com o número de pontos de controle de entrada e o número de pontos gerados.

Uma configuração fixa permite a entrada de um número variável de parâmetros a serem processados, e de um número variável de pontos de controle. Um exemplo de uma implementação fixa é ilustrado na Figura 89 onde é habilitado somente o número de circuitos necessários de acordo com o número de pontos a serem gerados. Um circuito de habilitação, com um vetor binário adicional, pode ser inserido às entradas de *clock* ou à entrada *ACK_I* (*Wishbone*) dos *cores* de processamento de cada parâmetro, para permitir ou não o seu processamento.

Uma configuração fixa também deve possibilitar a flexibilidade sobre as interconexões entre os pontos de controle, utilizados como entradas, e os *cores* de processamento dos parâmetros (pontos gerados). Conforme a Figura 90, dois vetores de tamanho fixo, um para cada coordenada dos pontos de controle e um para os parâmetros, podem ser utilizados, fixando-se as conexões entre estes vetores.

A variação da combinação destas conexões resulta em um barramento complexo, pois deve permitir a interconexão entre todos os valores de coordenadas do vetor dado com todos os parâmetros (Figura 91). Vetores binários adicionais podem ser utilizados para habilitar cada *core* responsável pelo processamento de um parâmetro, assim como os pontos de controle utilizados. Caso o tamanho do vetor para a coordenada dos pontos de controle, ou o número de circuitos de processamento dos parâmetros não seja suficiente, deve-se armazenar os resultados dos processamentos, carregar os vetores das coordenadas dos pontos de controle com os novos valores, e efetuar novamente o processamento.

Figura 89 - - Exemplo de circuito para a habilitação dos *cores* de processamento dos pontos de uma *b-spline*.

Figura 90 – Relação entre o número de pontos de controle e o número de parâmetros (pontos) processados. O número de parâmetros processador pode ser habilitado por um vetor (habilita) de entrada em cada *core* para uma arquitetura fixa.

Figura 91 – Barramento específico para a conexão entre as coordenadas dos pontos de controle e os *cores* de processamento de cada parâmetro.

6. Conclusão

Este projeto apresentou os três principais métodos para a implementação de um sistema de geração de *b-splines* em um sistema *FPGA*: *BFEA*, funções base fixas e *fast Cox-de Boor*. Foram implementados *cores* para cada um destes métodos, além de um *core* para segmentos de retas e um para circunferências. Para avaliar o resultado desta implementação, diferentes tipos de conjuntos de pontos foram utilizados, sendo obtidos os pontos resultantes do processo e exibidos na forma de imagem, assim como foram adquiridos dados para a avaliação do tempo de execução do sistema, e dos recursos de *hardware* utilizados por cada *core*, dado em número de elementos lógicos. Os tempos de execução dos *cores*, assim como os recursos, foram comparados para cada conjunto de pontos.

Em relação aos resultados das imagens exibidas, a implementação *FPGA* apresentou resultados coerentes para os testes de retas e curvas *b-splines*, 2D e 3D. A inclusão das coordenadas homogêneas para as *b-splines* racionais, apresentou alterações no formato das curvas finais, quando o ajuste dos valores dos pesos não satisfaz as condições de continuidade. Os arcos de circunferência *NURBS* apresentaram melhor resultado para formar seções cônicas, em relação às *b-splines* não racionais.

A implementação de *cores* para segmentos de reta e circunferência tiveram influência no tempo de processamento dos conjuntos de pontos que envolvem essas primitivas. Para os casos em que a representação de um segmento de reta é componente de uma curva de grau maior ou igual a dois, o *core* para segmento de reta apresenta um tempo de execução menor em relação aos *cores b-spline*, utilizando um número menor de dados de entrada. No entanto, utiliza maior número de elementos lógicos em relação aos *cores b-spline*.

O *core* para circunferência exige funções trigonométricas, resultando em um tempo de execução maior em relação aos *cores* de funções base fixas e *fast Cox-de Boor*, e utiliza maior número de elementos lógicos que os *cores b-spline*. No entanto, não exige parâmetros adicionais de entrada em relação às *splines*, as quais requerem o ajuste de pesos, dos valores dos nós e um número maior de pontos de controle.

O método *BFEA*, de uma maneira geral, para todos os conjuntos de pontos, apresentou o maior tempo de execução. Esse método foi desenvolvido diretamente para aplicações em *ASICs*, sendo o mais antigo dos métodos implementados. Porém, o número de elementos lógicos apresentado é o menor entre os três métodos, devido à utilização do resultado de

funções calculadas em etapas prévias do processamento. A implementação realizada neste trabalho possibilita a mudança no número de elementos de processamento pela reconfiguração do *FPGA* (não disponível em um *ASIC*), e difere do método original, baseado em matrizes sistólicas. O algoritmo de Casteljou e a *BFEA* possibilitam o incremento no grau da curva adicionando-se uma camada de circuitos de funções base ao sistema, tornando este método mais escalável em relação ao método de funções base fixas.

O método das funções base fixas, de uma maneira geral, apresentou a melhor relação entre número de elementos lógicos e tempo de processamento. A implementação do método original foi feita somente por simulações e testes em ferramentas computacionais. Este método limita-se a vetores de nós uniformes, não sendo adequado para vetores de nós não uniformes e *b-splines* racionais (*NURBS*), além da mudança nos valores do parâmetro. Porém, não é necessária a utilização do valor dos nós nos cálculos, reduzindo o número de dados de entrada. Esse método pode ser adaptado para vetores de nós não uniformes, desde que sejam consideradas as diferenças entre os nós, utilizadas no cálculo das funções base. Os valores fixos das funções base exigem um conjunto diferente de equações polinomiais para cada grau da curva, e respectivamente de circuitos, tornando este método pouco escalável.

O método *fast* Cox-de Boor, que implementa o algoritmo Cox-de Boor para cada valor paramétrico de uma *b-spline*, apresentou maior número de elementos lógicos entre os três métodos, principalmente para graus maiores que 2. No entanto, os tempos de processamento foram próximos aos obtidos pelo método das funções base fixas. Este método foi desenvolvido originalmente para sistemas de lógica reconfigurável (YAU, 2006), utilizando dados no formato inteiro de 32 *bits*. Cada ponto em uma *b-spline* é gerado por um conjunto de *cores* fixo, que é repetido para cada grau da curva, tornando o método modular.

É possível destacar no presente trabalho, em relação aos métodos implementados originalmente, a utilização de um barramento, de arquitetura aberta e a possibilidade de armazenamento de dados em uma memória externa. Os resultados obtidos por Yau (2006) não utilizam circuitos para barramento de dados, tornando mais complexa a transferência de dados entre *cores* em um mesmo *Soc*. O barramento utilizado (WISHBONE, 2011) padroniza operações de sincronia entre *cores* para a troca de dados, característica que não é mencionada nos trabalhos originais dos demais métodos.

O presente trabalho contribui com um sistema de geração *b-spline* em *FPGA*, implementado em três métodos, fornecendo uma alternativa para sistemas que utilizam *as b-splines* em suas aplicações, como os sistemas CNC reconfiguráveis. Além disso, contribui

com o resultado da comparação desses três métodos e também da comparação do *core line* e *core* circunferência, destacando as vantagens e desvantagens de cada um.

O projetista do sistema pode optar pela otimização dos recursos disponíveis (número de elementos lógicos) ou pelo tempo de execução, caso em que se escolheria o método *BFEA* ou o método das funções base fixas, respectivamente. Porém, em termos de modularidade, o método *fast* Cox-de Boor pode ser implementado em *cores*, distintos, permitindo maior flexibilidade na geração de *b-splines* de graus diferentes, modificando-se o número de camadas do circuito Cox-de Boor. Cada método implementado permite variações em sua configuração, em relação à configuração original, proporcionando modificações no número de recursos utilizados, no tempo de execução e na modularidade.

6.1. Artigos Publicados

Relacionados ao tema do presente trabalho, foram apresentados e publicados os seguintes artigos para congressos:

SILVA, L. M. C., PAIVA, M. S. V. NURBS Toolbox based in FPGA, *Anais do 6^o Congresso Brasileiro de Engenharia de Fabricação*, 2011, Caxias do Sul - Brasil, 2011.

SILVA, L. M. C., PAIVA, M. S. V. Toolbox NURBS and Visualization System via FPGA, *Proceedings of the Designer Forum - VII Southern Programmable Logic Conference*, 2011, Córdoba – Argentina, Anais, p. 101-106.

SILVA, L. M. C., PAIVA, M. S. V. NURBS Interpolation with FPGA Acceleration, *Anais do VII Workshop de Visão Computacional*, 2011, Curitiba - Brasil, 2011, Anais, p. 341-346.

SILVA, L. M. C., PAIVA, M. S. V. B-Spline Generation via FPGA, *VIII Southern Programmable Logic Conference*, Bento Gonçalves – Brasil, 2012.

6.2. Sugestões para Trabalhos Futuros

Como sugestões para a melhoria do sistema implementado, sugere-se:

- Circuitos de ferramentas para a manipulação de *splines* e para o preenchimento de um conjunto de pontos (interpolação e aproximação);
- Algoritmos avançados para inversão de pontos, projeção para curvas e superfícies, reparametrização;

- Análise dos circuitos *spline* para dados em ponto flutuante e em plataformas de lógica com reconfiguração dinâmica;
- Implementações de outros tipos de aplicações que fazem uso das *splines*, como as funções *wavelets b-splines*;
- Implementação de *Socs* através de placas de *FPGAs* interconectadas por barramentos de alta velocidade.

A disponibilidade e viabilidade de *FPGAs* com um maior número de elementos lógicos, que possibilitem a implementação de cálculos de ponto fixo com um número maior de *bits*, de ponto flutuante de precisão simples e dupla, de núcleos de *ULA* com suporte a ponto flutuante; permite a criação de aplicações como o preenchimento de pontos por curvas ou superfícies *NURBS* (interpolação e aproximação). Este procedimento geralmente envolve a resolução do sistema de equações de matrizes dado pelo cálculo inverso da equação (1), gerando matrizes esparsas devido aos valores nulos das funções base racionais que não são utilizadas no cálculo para a geração de um ponto. Sistemas de equações com matrizes esparsas necessitam do processamento da pseudoinversa destas matrizes, o que exige um custo computacional alto, proporcional ao número de pontos a ser interpolado. Este procedimento permite, especificamente em relação às superfícies, a utilização de algumas técnicas de construção avançada, como *swing surface*, *surface skinning*, superfícies de varredura, superfícies de *Coons* e superfícies de interpolação de uma rede de curvas bidirecionais (PIEGL E TILLER, 1997). No caso do preenchimento de uma nuvem de pontos esparsos no espaço (conjunto de pontos que não são organizados na forma longitudinal e transversal), é necessário um processamento prévio para formar a topologia necessária para este conjunto de pontos inicial (SILVA, 2003).

7. Referências Bibliográficas

ALTERA CORPORATION, 2009. Altera Advanced Synthesis Cookbook. Disponível em <[http:// www.altera.com/literature/manual/stx_cookbook.pdf](http://www.altera.com/literature/manual/stx_cookbook.pdf)>. Acesso em: 19 de abr. de 2011.

ALTERA CORPORATION, 2008. Cyclone II Device Handbook, Volume 1. Disponível em <http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf>. Acesso em: 27 de mar. de 2009.

ALTERA CORPORATION, 2004. FPGA Performance Benchmarking Methodology. Disponível em <<http://www.altera.com/literature/wp/wpfpgapbm.pdf>>. Acesso em: 30 de ago. de 2007.

ALTERA CORPORATION, 2007. OpenCore Plus Evaluation of Megafunctions. Altera Application Note 320. Disponível em <<http://www.altera.com/literature/an/ an320.pdf>>. Acesso em: 05 de jan. de 2010.

ALTERA CORPORATION, 2011. Quartus II Handbook Version 11.0. Disponível em <http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf>. Acesso em: 30 de jun. de 2011.

ANALOG DEVICES, 2010. ADV7123. Disponível em <http://www.analog.com/static/imported-files/data_sheets/ADV7123.pdf>. Acesso em: 11 de abr. de 2010.

ANDERSON, C. W., S. C.-HINES (2000). Fast Generation of NURBS Surfaces from Polygonal Mesh Models of Human Anatomy, *Technical Report CS-99-101*, Computer Science Department, Colorado State University.

BESL, P. Hybrid Modeling for Manufacturing using NURBS, Polygons, and 3D Scanner Data. In *IEEE International Symposium on Circuits and Systems*, 1998, vol. 5, p. 484-487.

BENSAALI, F., A. AMIRA, I. S. UZUN, A. AHMEDSAID. An FPGA implementation of 3D affine transformations, *Proceedings of the 2003 - 10th IEEE International Conference on Electronics, Circuits and Systems*, vol. 2, Dec., 2003, p. 715-718.

BENSAALI, F., A. AMIRA. Rapid Prototyping - Field programmable gate array based parallel matrix multiplier for 3D affine transformations, *IEEE Proceedings in Vision, Image and Signal Processing*, vol. 153, issue 6, Dec., 2006, p. 739-746.

BOOR, C. DE. *A Practical Guide to Splines*. Revised ed. Springer-Verlag, New York, 2001, 366 p.

BOOR, C. DE. On Calculating with B-Splines. *Journal of Approximation Theory*, 1972, vol. 6, p. 50-62.

BRESENHAM, J. E. Algorithm for computer control of a digital plotter, *IBM Systems Journal*, vol. 4(1), 1965, p. 25-30.

CARR, J. C., R. K. BEATSON, W. R. FRIGHT. Surface Interpolation with Radial Basis Functions for Medical Imaging, *IEEE Transactions on Medical Imaging*, vol. 16, issue 1, Feb.,1997, p. 96–107.

CARR, J. C., R. K. BEATSON, J. B. CHERRIE, T. J. MITCHELL, W. R. FRIGHT, B. C. MCCALLUM, T. R. EVANS. Reconstruction and representation of 3D objects with radial basis functions, *Proc. 28th annual Conference on Computer Graphics and Interactive Techniques*, 2001, p. 67–76.

CHANG, H., L. COOKE, M. HUNT, G. MARTIN, A. MCNELLY, L. TODD. *Surviving the SOC Revolution – A Guide to Platform-Based Design*. Kluwer Academic Publishers, New York, NY, 2002, 235 p.

CHE S., J. LI, J. W. SHEAFFER, K. SKADRON, J. LACH (2008). Accelerating Compute-Intensive Applications with GPUs and FPGAs. In *Symposium on Application Specific Processors*, 2008, p. 101-107.

CHEN, D.S., R. C. JAIN, B. G. SCHUNCK. FPGA Design Automation: A Survey. *Foundations and Trends in Electronic Design Automation*, vol. 1, no 3, Nov., 2006, p. 195-330.

COPE, B. T. (2008). Video Processing Acceleration using Reconfigurable Logic and Graphics Processors. Tese Ph.D, Diploma de Membro do Imperial College London Circuits e Systems Group Department of Electrical and Electronic Engineering, Imperial College London, Londres, Inglaterra.

COPE, B., P. Y. K. CHEUNG, W. LUK, S. WITT. Have GPUs made FPGAs redundant in the field of Video Processing?. In *IEEE International Conference on Field-Programmable Technology*, 2005, p. 111-118.

DEAN, T., J. MODEL, MARTIN C. H. Multicore versus FPGA in the Acceleration of Discrete Molecular Dynamics. *Proceedings of High Performance Embedded Computing 2008 Workshop*, 2008.

DESCHAMPS, J.-P., G. J. A. BIOUL, G. D. SUTTER. *Synthesis of Arithmetic Circuits – FPGA, ASIC and Embedded Systems*. 2^a ed. Wiley Interscience, Hoboken, NJ, 2006, 556 p.

DONG, J., Z. YANG, Q. LIU, Z. WANG, T. WANG. Design and Implementation of CNC Controllers using Reconfigurable Hardware. In *IEEE International Conference on Control and Automation*, 2009, p. 1481-1486.

DONG, J. C., T. Y. WANG. A Pipeline Designed Reconfigurable CNC Architecture, *Materials Science Forum*, 2012, vols. 697-698, p. 288-291.

DONGDONG, L., W. L. ZHU, G. L. NANHUI. Design of Reconfigurable CNC System Based on FPGA, In *ICEMI '07 - 8th International Conference on Electronic Measurement and Instruments*, 2007, vol. 4, p. 857-861.

FAN, J., 2007. DE2_70 TV Box [CD-ROM]. Terasic Technologies, 2007.

FARIN, G. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. 4^a ed. Academic Press, San Diego, CA, 1997, 429 p.

FRANCHINI, S., A. GENTILE, F. SORBELLO, G. VASSALLO, S. VITABILE. An embedded, FPGA-Based Computer Graphics Coprocessor with Native Geometric Algebra Support, *The VLSI Journal of Integration*, 2009, vol. 42, núm. 3, p. 346-355.

GOPI, M., S. MANOHAR. A Unified Architecture for the Computation of B-Spline Curves and Surfaces, *IEEE Transactions on Parallel and Distributed Systems*, 1997, vol. 8, no. 12, Dec., p. 1275-1287.

GUPTA, R. K., ZORIAN, Y. Introducing Core-Based System Design, *IEEE Design & Test of Computers*, Oct.-Dec., 1997, p. 15-25.

HALFHILL, T. R. Parallel processing with CUDA Nvidia's high-performance computing platform uses massive multithreading, *Microprocessor Report*, vol. 22, no. 1, Jan. 2008.

HASELMAN, M., M. BEAUCHAMP, A. WOOD, S. HAUCK, K. UNDERWOOD, K. S. HEMMERT. A Comparison of Floating Point and Logarithmic Number Systems for FPGAs, *In Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2005, vol. 4, p. 181-190.

HAUCK, S., A. DEHON. *Reconfigurable Computing – The Theory and Practice of FPGA Based Computation*. The Morgan Kaufmann Series in Systems on Silicon, Morgan Kaufmann, Burlington, MA, 2008, 909 p.

HARRIS, M., 2007. Optimizing Parallel Reduction in CUDA. Disponível em <http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf>. Acesso em: 19 de abr. de 2011.

HENNESSY, J. L., D. A. PATTERSON. *Computer Architecture – A Quantitative Approach*. 4^a ed. Elsevier, San Francisco, 2007, 676 p.

HERWECK, P., 2009. Siemens SINUMERIK 840-D increases machine tool productivity by 50%. *Automation Technology*, 1996, 4th quarter, issue 10. Disponível em <<http://sea.siemens.com/aut/at/at4q96/840dsty1.htm>>. Acesso em: 02 de fev. de 2009.

HOLTEN-LUND, H. Embedded 3D Graphics Core for FPGA-based System-on-Chip Applications, *FPGA WorldConference* 2005, Sep., 2005, p. 8-13.

HOPPE, H., T. DEROSE, T. DUCHAMPY, M. HALSTEADZ, H. JIN, J. MCDONALD, J. SCHWEITZER, W. STUETZLEX. Piecewise Smooth Surface Reconstruction, *In '94 Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, 1994, p. 295-302.

HOWELL, K., 2007. FPGAs Control Graphics and Video in Embedded Systems. *FPGA and Structured ASIC Journal*. Disponível em <http://www.eejournal.com/archives/articles/20071113_lattice/>. Acesso em: 18 de mar. de 2009.

HOWES, L. W., P. PRICE, O. MENCER E O. BECKMANN. FPGAs, GPUs and the PS2 - A Single Programming Methodology, In *FCCM '06 - 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006, p. 313 – 314.

IEEE COMPUTER SOCIETY (2006). *IEEE Standard for Verilog Hardware Description Language*, Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board, Apr.

IEEE COMPUTER SOCIETY (2009). *IEEE Standard VHDL Language Reference Manual*, Accellera VHDL Technical Subcommittee and the IEEE VHDL Analysis and Screening Committee (VASG), Sep.

IGES/PDES ORGANIZATION (1996). *Initial Graphics Exchange Specification - IGES 5.3*, American National Standard Institute - ANSI, U. S. Product Data Association, Sep.

ISO 10303. *Industrial automation systems and integration – Product data representation and exchange, multipart standard (STEP)*, International Organization for Standardization – ISO, 2002.

ISSI – INTEGRATED SILICON SOLUTION INC., 2007. IS42S83200B/IS42S16160B 256-MBit Synchronous DRAM Datasheet. Disponível em < <http://www.issi.com/pdf/42S83200B-16160B.pdf> >. Acesso em: 02 de abr. de 2010.

JESÚS R.-T. R., H.-R. GILBERTO, T.-V. IVÁN, J.-C. J. CARLOS. FPGA Based On-line Tool Breakage Detection System for CNC Milling Machines, *Mechatronics*, 2004, May, vol. 14, issue 4, p. 439-454.

JIMENO-MORENILLA, A., A. MARTINEZ, S. CUENCA, J. L. SANCHEZ-ROMERO. Accelerating Tool Path Computing in CAD/CAM: A FPGA Architecture for Turning Lathe Machining, In *FPL 2007 - International Conference on Field Programmable Logic and Applications*, 2007, Aug., p. 702 – 705.

JIMENO, A., J. L. SÁNCHEZ, H. MORA, J. MORA, J.M. GARCÍA-CHAMIZO. FPGA-Based Tool Path Computation - An Application for Shoe Last Machining on CNC Lathes, *Computers in Industry*, 2006, number 57, p. 103 - 111.

KILTS, S. *Advanced FPGA Design – Architecture, Implementation and Optimization*. Wiley Interscience, Hoboken, NJ, 2007, 336 p.

KIRK, D. B., W.-M. W. HWU. *Programming Massively Parallel Processors – A Hands-on Approach*. Morgan Kaufman - Elsevier, Burlington, MA, 2010, 258 p.

KHRONOS OPENCL WORKING GROUP, 2011. The OpenCL Specification - Version 1.1. Disponível em < <http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf> >. Acesso em: 20 de mar. de 2011.

KNUTSSON, NIKLAS (2005). An FPGA-based 3D Graphics System. Dissertação de Mestrado em Sistemas Eletrônicos, Linköping Institute of Technology.

KRISHNAMURTHY, V., LEVOY, M. Fitting smooth surfaces to dense polygon meshes. In *23rd Annual conference on Computer Graphics and Interactive Techniques*, 1996, p. 313 - 324.

KRISHNAMURTHY, A., KHARDEKAR, R., MCMAIN, S. Direct Evaluation of NURBS Curves and Surfaces on the GPUs. *SPM '07 Proceedings of the 2007 ACM symposium on Solid and physical modeling*, p. 329-334, 2007.

KRISHNAMURTHY, A., R. KHARDEKAR, S. MCMAINS, K. HALLER, G. ELBER. Performing Efficient NURBS Modeling Operations on the GPU, *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*, 2008, p. 257-268.

KUON, I., R. TESSIER, J. ROSE. *FPGA Architecture: Survey and Challenges. Eletronic Design Automation*, vol. 2, no. 2, 2007, p. 135-253.

LI, F., J. YOU, W. TONG. A Design of Full-Digital CNC Interface Based on FPGA, In *ITCS Proceedings of the 2009 International Conference on Information Technology and Computer Science*, 2009, vol. 2, p.112-115.

MANN, S., M. LOUNSBERY, C. LOOP, D. MEYERS, J. PAINTER, T. DEROSE, K. SLOAN. A survey of parametric scattered data fitting using triangular interpolants, *Curve and Surface Design*, capítulo 8, SIAM Activity Group on Geometric Design, editado por Hans Hagen, Filadélfia, PA, USA, 1992, p. 01-33.

MAKINO. NURBS Interpolation. *Competitive Mold Maker*, 1999, vol. 3, number 2, p. 1-5.

MEGSON, G. M. Systolic Algorithms for B-Spline Patch Generation, *Journal of Parallel and Distributed Computing*, 1991, vol. 11, issue 3, Mar., p. 231-238.

MEYER-BAESE, U. *Digital Signal Processing with Field Programmable Gate Arrays*. 2^a ed. Springer Verlag, Berlin Heidelberg, 2001, 423 p.

NVIDIA CORPORATION, 2009. GeForce 256: The world's first GPU. Disponível em <<http://www.nvidia.com/page/geforce256.html>>. Acesso em: 02 de fev. de 2009.

NVIDIA CORPORATION, 2011. CUDA C Best Practices Guide - Version 4.0. Disponível em <http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf>. Acesso em: 19 de abr. de 2011.

NVIDIA CORPORATION, 2011. NVIDIA CUDA C Programming Guide - Version 4.0. Disponível em <http://developer.download.nvidia.com/compute/cuda/4_0_rc2/toolkit/docs/CUDA_C_Programming_Guide.pdf>. Acesso em: 19 de abr. de 2011.

NVIDIA CORPORATION, 2011. CUDA Toolkit 4.0 CUBLAS Library. Disponível em <http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CUBLAS_Library.pdf>. Acesso em: 25 de abr. de 2011.

OPENCORES ORGANIZATION, 2010. Wishbone B4 - Wishbone System-on-Chip Interconnection Architecture for Portable IP Cores. Disponível em <http://cdn.opencores.org/downloads/wbspec_b4.pdf>. Acesso em: 23 de jan. de 2011.

ORENSTEIN, P., 2002. High-Speed CAM of 3-D Sculpted Surfaces. *Time Compression Magazine*, 2002, Mar. Disponível em <<http://www.timecompression.com/articles/high-speed-cam-of-3-d-sculpted-surfaces>>. Acesso em: 03 de mar. de 2009.

OU, S.-C., L.-H. SHIU, S.-J. HSIAO, W.-T. SUNG. Accelerate the calculation of NURBS curves and surfaces based on parallel architecture, *Proceedings of Ninth International Conference on Parallel and Distributed Systems*, 2002. Dec., 2002, p. 245-250.

OUELLETTE, M., D. CONNORS. Analysis of Hardware Acceleration in Reconfigurable Embedded Systems, *Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium*, 2005. Apr., 2005, p. 168.

PALM, III, W. J. *Introduction to Matlab 7 for Engineers*. McGraw-Hill, Boston, 2005, 682 p.

PALNITKAR, S. *Verilog HDL: A Guide to Digital Design and Synthesis*. 2^a ed. Prentice Hall PTR, California, 2003, 496 p.

PANG, A. T. Line-Drawing Algorithms for Parallel Machines. *IEEE Computer Graphics & Applications*, 1990, vol. 10, p. 54–59.

PIEGL, L. On NURBS: A Survey, *IEEE Computer Graphics & Applications*, 1991, Jan., p. 55–71.

PIEGL, L., W. TILLER. *The NURBS Book*. 2^a ed. Springer, New York, 1997, 645 p.

RATHA, N.K.; A.K. JAIN. Computer Vision Algorithms on Reconfigurable Logic Arrays, *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, issue:1, Jan., 1999, p. 29–43.

ROGERS, D. F. *An Introduction to NURBS with Historical Perspective*, Academic Press, 2nd ed., San Diego, 2001, 343 p.

ROGERS, D. F., J. A. ADAMS. *Mathematical Elements for Computer Graphics*, McGraw-Hill, 2nd ed., New York, 1990.

ROSE, J., R. J. FRANCIS, D. LEWIS, P. CHOW. Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency, *IEEE Journal of solid State Circuits*, 1990, vol. 25, issue 5, Oct., p. 1217-1225.

ROSE, J., A. EL GAMAL, A. SANGIOVANNI-VINCENTELLI. Architecture of field-programmable gate arrays, *Proceedings of the IEEE*, Jul., 1993, p. 1013-1029.

SANFORD, B., 2002. Integrated Graphics Solutions for Graphics-Intensive Applications. *AMD White Paper*, Advanced Micro Devices Inc. Disponível em <http://www.amd.com/usen/assets/content_type/white_papers_and_tech_docs/Integrated_Graphics_Solutions_white_paper_rev61.pdf>. Acesso em 23 de nov. de 2008.

SCHERZER, O. *Handbook of Mathematical Methods on Imaging*. 1^a ed., Springer, Vienna, 2011, 1607 p.

SCRA INC, 2006. STEP Application Handbook – ISO 10303 – Version 3. Disponível em <http://www.uspro.org/documents/STEP_application_hdbk_63006_BF.pdf>. Acesso em 12 de dez. de 2010.

SEFI, S., J. OPPELSTRUP. Physical Optics and NURBS for RCS Calculations. In *EMB04 - Conference on Computational Electromagnetics - Methods and Applications*, 2004, Göteborg, Suíça, p. 90-97.

SHREINER, D., M. WOO, J. NEIDER, T. DAVIS. *The OpenGL Programming Guide: The Official Guide to Learning OpenGL - Version 2.1*, 2008, Addison-Wesley Professional, 6th ed., Boston, Massachusetts

SILVA, L. M. CHIESSE (2003). Reconstrução de Superfícies a partir de uma nuvem de pontos utilizando NURBS. Dissertação de Mestrado em Engenharia Industrial, Universidade Estadual Paulista, Escola de Engenharia de Bauru.

SINGH, S. E P. BELLEC (1994). Virtual hardware for graphics applications using FPGAs. In *IEEE Workshop on FPGAs for Custom Computing Machines*, 1994, p. 49-58.

STOKES, M. L. A Brief Look at FPGAs, GPUs and Cell Processors, *ITEA Journal*, 2007, Jun./Jul, p. 09-11.

STYLES, H. E W. LUK. Customising graphics applications: techniques and programming interface. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2000, p. 77-87.

GLOBAL SUDA TONGXING DEVELOPMENT TECHNOLOGY CO. LTDA., 2012. Suda Engraving Machine SD5040. Disponível em <<http://www.txgm.dpes.com.cn/product/2011/11/289.html>>. Acesso em: 19 de jun. de 2012.

TERASIC TECHNOLOGIES INC., 2007. DE2-70 User Manual [04/05/2009]. Disponível em <http://www.terasic.com.tw/cgi-in/page/archive_download.pl?Language=English&No=226&FID=e96456c80e4870fffc0f2869517fd21>. Acesso em: 04 de mai. De 2009.

THOMAS, D. E., P. R. MOORBY. *The Verilog Hardware Description Language*. 5^a ed. Luwer Academic Publishers, Dordrecht, 2002, 645 p.

TSAL, M. C., C. W. CHENG E M. Y. CHENG. A real-time NURBS surface interpolator for precision three-axis CNC machining, *International Journal of Machine Tools & Manufacture*, 2003, vol. 43, p. 1217-1227.

UNSER, M. Splines: A Perfect Fit for Medical Imaging. In *Part I of Proceedings of the SPIE International Symposium on Medical Imaging: Image Processing*, 2002, Feb., p. 225-236.

USSELMAN, R., 2001. OpenCores SoC Bus Review. Disponível em <http://cdn.opencores.org/downloads/soc_bus_comparison.pdf>. Acesso em: 12 de dez. de 2010.

WANG, T., L. WANG, Q. LIU. A three-ply reconfigurable CNC system based on FPGA and field-bus, *International Journal of Advanced Manufacturing Technology*, 2011, 20 apr., Online First, p. 1-12.

WANG, T., Y. CAO, Y. CHEN, H. WEI, B. WANG, Z. SHAO. A New Feedrate Adaptation Control NURBS Interpolation Based on de Boor Algorithm in CNC Systems. In *American Control Conference*, 2008, p. 4075-4080.

WEN, A. S., S. M. H. SHAMSUDDIN, Y. SAMIAN. Ship Hull Fitting Using NURBS. In *International Conference on Computer Graphics, Imaging and Visualization - CGIV'05*, 2005, 26-29 jul., p. 431-436.

WRIGHT, W. E. Parallelization of Bresenham's Line and Circle Algorithms, *IEEE Computer Graphics & Applications*, 1990, vol. 10, p. 60-67.

WOO, J.-H., J.-H. SOHN, B.-G. NAM, H.-J. YOO. *Mobile 3D Graphics SoC - From Algorithm to Chip*. John Wiley & Sons, Singapura, China, 2010, 342 p.

XU, Y., T. WANG, Y. ZHAO. Design of reconfigurable CNC system based on embedded technology. In *7th World Congress on Intelligent Control and Automation*, 2008, 25-27 jun., p. 6061-6064.

YAMAGUCHI, F. *Curves and surfaces in Computer Aided Geometric Design*. 2^a ed. Springer-Verlag, Berlim, 1988, 378 p.

YANG, X., G. SUN e Q. LI. A New NURBS Tool Path Generation Algorithm for Precise Sculptured Surface Machining, 2010, *Advanced Materials Research*, vol. 97-101, p. 2477-2480.

YAU, H. T., M. T. LIN E M. S. TSAI. Real-Time NURBS interpolation using FPGA for high speed motion control, 2006, *Computer-Aided Design*, no. 38, p. 1123-1133.

ZEMCIK, P. Hardware acceleration of graphics and imaging algorithms using FPGAs. In *18th Spring Conference on Computer Graphics*, 2002, p. 25-32.

APÊNDICE A – Introdução às *Splines*

Trecho de introdução às *NURBS*, extraído do trabalho de Silva (2003) baseado no conteúdo do trabalho de Rogers e Adams (1990).

Muitos trabalhos foram desenvolvidos com o objetivo da criação de superfícies por meio de vários tipos de polinômios com o propósito de suavização ou preenchimento de um conjunto de pontos fornecido, e de substituir esta informação gráfica com fórmulas e equações que permitirão a aplicação de técnicas analíticas, como análise estrutural. Mas estas técnicas matemáticas são aplicáveis somente quando a superfície já tem designado algum grau de integridade, fornecendo informação matemática suficiente. Esta estrutura de uma superfície, a ser descrita matematicamente, é planejada para implementar automaticamente o processo do projeto e torná-lo, do ponto de vista do projetista, extremamente natural e fácil. O projetista não necessita de conhecimento ou cuidado sobre estes detalhes matemáticos internos, não mais do que a necessidade de conhecer a composição específica da mecânica das *splines* com a qual desenha a superfície.

No projeto de um objeto tridimensional, o projetista necessita de um sistema que permita definir uma superfície com o mínimo de informação de entrada e modificar esta superfície, seja pela modificação das informações de entrada ou pela adição de componentes ao projeto do sistema.

A.1. *Splines*

Antigamente, a forma final aproximada ou completa de uma curva em um projeto, como o casco de um navio ou o perfil da fuselagem de um avião, podia ser obtida através de um processo chamado “*lofting*” (soteamento), que consistia na utilização de uma barra longa e estreita fabricada com um material resistente a esforços transversais e longitudinais, como madeira ou plástico. Esta barra era modelada no formato de uma curva desejada aplicando-se tensões ao longo da barra com pesos e suportes de condução denominados “*ducks*”(patos), conforme exibido pela Figura 1.

Figura A1 – Barra sujeita a esforços mecânicos transversais. Adaptado de: (ROGERS E ADAMS, 1990).

A Figura A2 mostra exemplos de dispositivos mecânicos que podem atuar como pesos de condução (“*ducks*”):

Figura A2 – Exemplos reais de dispositivos que atuam como “*ducks*”. Adaptado de: (ROGERS E ADAMS, 1990).

A curva formada pela barra modelada, pelos pesos de condução, é uma *spline* física, e sua forma é alterada pelo valor, posição e número destes pesos, o que possibilita uma “condução” da curva através de pontos determinados (Figura A3). Com o desenvolvimento da computação, houve um considerável avanço na utilização e no desenvolvimento matemático deste processo. Analiticamente, a forma da *spline* é determinada pela deflexão da barra em cada ponto onde são aplicados os pesos.

Figura A3 – Deflexão em uma barra. Adaptado de: (ROGERS E ADAMS, 1990).

O valor do momento fletor ao longo da barra da Figura A3 é dado pela equação de Euler:

$$M(x) = \frac{E.I}{R(x)} \quad (A1)$$

Onde $M(x)$ é o momento fletor, E é o módulo de Young (propriedade do material que compõe a barra), I é o momento de inércia, $R(x)$ é o raio da curvatura, e x é o comprimento da barra, considerando y a deformação na barra (o comprimento da deflexão na Figura A3). O material utilizado na barra origina pequenas deflexões em razão da resistência, ocasionando maior suavidade ao formato da curva. Deste modo, $y' \ll 1$, e o raio da curvatura é aproximado para a equação A2:

$$R(x) = \frac{\sqrt{(1 + y'^2)^3}}{y''} \approx \frac{1}{y''} \quad (A2)$$

Considerando que os pesos de condução atuam como simples suportes sem variação no esforço que exercem na barra, o momento fletor $M(x)$ varia linearmente entre estes pesos, ou seja:

$$M(x) = A.x + B \quad (A3)$$

$$A.x + B = \frac{E.I}{R(x)} \cong E.I.y'' \quad (A4)$$

$$y'' = \frac{A.x + B}{E.I} \quad (A5)$$

Considerando o termo “ $E.I$ ” constante nas equações A4 e A5, e integrando duas vezes:

$$y = A_I.x^3 + B_I.x^2 + C_I.x + D_I \quad (A6)$$

A equação (A6) é a fórmula matemática geral de um “pedaço” da *spline* física em função de “ x ”: a função *spline* cúbica. Matematicamente, esta função é um pedaço de polinômio (*piecewise polynomial*), que consiste na junção de polinômios limitados por determinados intervalos, tal que a continuidade da curva resultante desta função é satisfeita nos pontos de junção desses polinômios. Como exemplo a função $g(t)$, cujo gráfico é ilustrado na Figura A4, formada pelos polinômios $a(t)$, $b(t)$ e $c(t)$.

Figura A4 - Junção de polinômios.

Os valores do parâmetro “ t ” nos quais ocorre a junção dos pedaços de polinômio são chamados de “nós”, e são dados importantes na definição de uma *spline*.

Definição de uma Função *Spline*:

Uma *spline* de grau “ m ” ou de m -ésimo grau é um pedaço polinomial de grau “ m ” que possui continuidade de derivadas de ordem “ $m-1$ ” em cada nó.

A equação (A6) é um exemplo de uma função *spline* cúbica de y em função de x . Matematicamente, esta função também é um pedaço de polinômio, que consiste na junção de pedaços de outros polinômios, limitado em um determinado intervalo, tal que a continuidade da curva resultante desta função é satisfeita nos pontos de junção (nós) destes polinômios.

As próprias funções *splines* podem ser utilizadas para definir uma curva. As curvas *spline* são uma mistura de vetores utilizando pedaços de polinômios (Figura A5), sendo contínua nos seus nós, possuindo muita semelhança com as curvas de Bézier (também em relação às propriedades). As *splines* constituídas por polinômios de baixo grau são úteis para o preenchimento de curvas pela redução no tempo de processamento dos cálculos e da instabilidade numérica presente em curvas de alto grau. Como polinômios de baixo grau podem não conectar determinados pontos, quando se deseja que a curva passe por determinados pontos em uma interpolação, utiliza-se uma técnica, unindo uma série de segmentos de *spline* (geralmente *spline* cúbica), com cada segmento conectando somente dois pontos.

Figura A5 – Pedaços de polinômio. Adaptado de: (ROGERS E ADAMS, 1990).

Caso seja possível para as funções de mistura *spline* assumirem outras formas e havendo $L+1$ pontos de controle $p_0, p_1, \dots, p_L$; associando à cada um destes pontos uma função de mistura $R_k(t)$ a curva é dada por:

$$p(t) = \sum_{k=0}^L \mathbf{p}_k \cdot R_k(t) \quad (\text{A7})$$

Esta formulação pode ser utilizada para curvas de Bezier e curvas quadráticas *spline*. Para manter manuseável a família de formas das possíveis funções de mistura, cada função $R_k(t)$ deve ser um pedaço de polinômio contínuo sendo também contínuo em cada nó. Então a curva total $p(t)$ é uma soma de funções *spline* ponderada pelos pontos de controle, com a condição dos segmentos definirem esta curva como contínua. Tal curva é chamada de curva *spline*.

Além do terceiro grau, a função *spline* pode possuir outros graus, sendo a *spline* de grau “ n ” um pedaço de polinômio que tem continuidade da derivada de ordem “ $n-1$ ” em cada nó. Porém, pela similaridade com o comportamento físico dos materiais, as *splines* cúbicas são as mais utilizadas (grau 3). As Curvas *Spline* são uma mistura de vetores utilizando pedaços de polinômios, sendo contínua nos seus nós.

A equação para um segmento de *spline* cúbica é dada por:

$$P(t) = \sum_{i=0}^4 B_i \cdot t^{i-1} \quad (\text{A8})$$

Onde t_1 e t_2 são os valores do parâmetro “ t ” no início e fim dos segmentos, como mostrado na Figura A6.

Os coeficientes constantes “ B_i ” são determinados por quatro condições de contorno (*spline* cúbica):

$$P(t) = B_1 + B_2.t + B_3.t^2 + B_4.t^3 \quad t_1 \leq t \leq t_2 \quad (A9)$$

Figura A6 – Vetores posição e tangente iniciais, finais e na junção de uma *spline*. Fonte: (ROGERS E ADAMS, 1990).

Se P_1 e P_2 são os vetores posição inicial e final do segmento, respectivamente. Então:

$$P(0) = P_1 \text{ e } P(t_2) = P_2 \quad (A10)$$

$$P(0) = P_1 = B_1 + B_2.0 + B_3.0 + B_4.0 = B_1 \quad (A11)$$

$$P_1 = B_1 \quad (A12)$$

$$P(t_2) = B_1 + B_2.t_2 + B_3.t_2^2 + B_4.t_2^3 \quad (A13)$$

Considere também P'_1 e P'_2 os vetores tangentes iniciais e finais dos extremos do segmento *spline*:

$$P'(0) = P'_1 \quad (A14)$$

$$P'(t_2) = P'_2 \quad (A15)$$

$$P'(t) = B_2 + 2.B_3.t + 3.B_4.t^2 \quad (A16)$$

$$P'(0) = B_2 + 2.B_3.0 + 3.B_4.0 = B_2 = P'_1 \quad (A17)$$

$$P'(t_2) = B_2 + 2.B_3.t_2 + 3.B_4.t_2^2 \quad (A18)$$

Foram obtidos os valores de B_2 e B_1 a partir dos vetores extremos de posição e tangentes. Isolando-se B_3 e B_4 na equação A18:

$$P'(t_2) = P'_2 = B_2 + 2.B_3.t_2 + 3.B_4.t_2^2 = P'_1 + 2.B_3.t_2 + 3.B_4.t_2^2 \quad (A19)$$

$$P(t_2) = P_2 = B_1 + B_2.t_2 + B_3.t_2^2 + B_4.t_2^3 = P_1 + P'_1 + B_3.t_2^2 + B_4.t_2^3 \quad (A20)$$

Resolvendo-se as duas equações acima para B_3 e B_4 em relação aos vetores iniciais e finais de posição e tangentes:

$$B_3 = \frac{3.(P_2 - P_1)}{t_2^2} - \frac{2.P_1'}{t_2} - P_2 \quad (A21)$$

$$B_4 = \frac{2.(P_1 - P_2)}{t_2^3} + \frac{P_1'}{t_2^2} + \frac{P_2'}{t_2^2} \quad (A22)$$

Substituindo na equação inicial de $P(t)$:

$$P(t) = P_1 + P_1'.t + \left(\frac{3.(P_2 - P_1)}{t_2^2} - \frac{2.P_1'}{t_2} - \frac{P_2'}{t_2} \right) t^2 + \left(\frac{2.(P_1 - P_2)}{t_2^3} + \frac{P_1'}{t_2^2} + \frac{P_2'}{t_2^2} \right) t^3 \quad (A23)$$

Das equações A10 a A22, conclui-se que a forma do segmento de *spline* cúbica depende dos vetores posição e tangentes nos extremos do segmento, além do valor final do parâmetro “ t ”, que deve ser conhecido. Para a representação de uma curva completa, agrupam-se diversos segmentos como este.

Na junção dos segmentos, o vetor tangente P_2' torna-se interno à curva, possibilitando sua determinação por uma condição de continuidade, pois a curvatura é contínua através da junção, ou seja, a derivada segunda de P_2 (junção) é contínua:

$$P''(t) = \sum_{i=1}^4 (i-1).(i-2).B_i.t^{i-3} \quad t_1 \leq t \leq t_2 \quad (A24)$$

Para o primeiro segmento:

$$P''(t_2) = 2.B_3 + 6.B_4.t_2 \quad t_1 \leq t \leq t_2 \quad (A25)$$

Considerando a faixa de valores do parâmetro “ t ” de 0 à t_3 para o segundo segmento conectado por P_2 :

$$\text{para o segundo segmento: } P''(0) = 2.B_3 + 6.B_4.0 = 2.B_3 \quad t_2 \leq t \leq t_3 \quad (A26)$$

Como $P''(t_2) = P''(0)$

$$2.B_3 + 6.B_4.t_2^2 = 2.B_3 \quad (A27)$$

$$6t_2 \left(\frac{2(P_1 - P_2)}{t_2^3} + \frac{P_1'}{t_2^2} + \frac{P_2'}{t_2^2} \right) + 2 \left(\frac{3(P_2 - P_1)}{t_2^2} - \frac{2P_1'}{t_2} - \frac{P_2'}{t_2} \right) = 2 \left(\frac{3(P_3 - P_2)}{t_2^2} - \frac{2P_2'}{t_2} - \frac{P_3'}{t_2} \right) \quad (A28)$$

Generalizando as equações para os coeficientes B_n :

$$B_{1k} = P_k \quad (\text{A29})$$

$$B_{2k} = P'_k \quad (\text{A30})$$

$$B_{3k} = \frac{3(P_{k+1} - P_k) - 2P'_k - P'_{k+1}}{t_{k+1}^2} \quad (\text{A31})$$

$$B_{4k} = \frac{2(P_k - P_{k+1}) + P'_k + P'_{k+1}}{t_{k+1}^3} \quad (\text{A32})$$

Desenvolvendo chega-se à matriz dos coeficientes:

$$[\mathbf{B}] = \begin{bmatrix} \mathbf{B}_{1k} \\ \mathbf{B}_{2k} \\ \mathbf{B}_{3k} \\ \mathbf{B}_{4k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{2}{t_{k+1}^2} & -\frac{1}{t_{k+1}} & \frac{2}{t_{k+1}^2} & -\frac{1}{t_{k+1}} \\ \frac{2}{t_{k+1}^3} & \frac{1}{t_{k+1}^2} & -\frac{2}{t_{k+1}^3} & \frac{1}{t_{k+1}^2} \end{bmatrix} \begin{bmatrix} P_k \\ P'_k \\ P_{k+1} \\ P'_{k+1} \end{bmatrix} \quad (\text{A33})$$

A forma da equação matricial para a *spline* é:

$$P_k(t) = [1 \ t \ t^2 \ t^3] \cdot \begin{bmatrix} B_{1k} \\ B_{2k} \\ B_{3k} \\ B_{4k} \end{bmatrix} \quad (\text{A34})$$

Remanejando os termos e substituindo t/t_{k+1} por τ :

$$P_k(\tau) = [F_1(\tau) \ F_2(\tau) \ F_3(\tau) \ F_4(\tau)] \cdot \begin{bmatrix} P_k \\ P_{k+1} \\ P'_k \\ P'_{k+1} \end{bmatrix} \quad (\text{A35})$$

$$F_{1k}(\tau) = 2\tau^3 - 3\tau^2 + 1 \quad (\text{A36})$$

$$F_{2k}(\tau) = -2\tau^3 - 3\tau^2 \quad (\text{A37})$$

$$F_{3k}(\tau) = \tau(\tau^2 - 2\tau + 1)t_{k+1} \quad (\text{A38})$$

$$F_{4k}(\tau) = \tau(\tau^2 - \tau) \quad (\text{A39})$$

$$P_k(\tau) = [F][G] \quad (\text{A40})$$

A matriz F é chamada matriz das funções de mistura e a matriz G contém a informação geométrica para a *spline*.

$$[G]^T = [P_k \ P_{k+1} \ P'_k \ P'_{k+1}] \quad (\text{A41})$$

As funções *splines* podem ser utilizadas também como funções de mistura. Considere a curva $p(t)$ baseada no polígono de controle p_0, p_1, p_2, p_3, p_4 :

$$\mathbf{p}(t) = \sum_{k=0}^4 \mathbf{p}_k \cdot g_k(t) \quad (\text{A42})$$

Com $g_k(t) = g(t - k)$ para $k = 0, 1, 2, 3, 4$, como ilustrado na Figura A7e Figura A8.

Figura A7 - Funções *spline* de mistura. Adaptado de: (ROGERS E ADAMS, 1990).

Figura A8 - Curva *spline* obtida das funções de mistura. Adaptado de: (ROGERS E ADAMS, 1990).

A curva apresentada pela Figura A8 pode ser generalizada para utilizar “ $L + 1$ ” pontos de controle, variando “ k ” de 0 a L :

$$p(t) = \sum_{k=0}^L p_k \cdot g(t - k) \quad (\text{A43})$$

Figura A9 – Curva *Spline* fechada com continuidade no ponto inicial p_0 , dada pela repetição dos pontos finais p_5 e p_6 pelos pontos iniciais p_0 e p_1 . Adaptado de: (ROGERS E ADAMS, 1990).

As equações A42 e A43 evidenciam que a escolha dos valores dos parâmetros, para uma *spline*, é de tal importância a ponto de alterar a precisão e a qualidade final da superfície. Na geração de uma superfície é possível utilizar valores de parâmetros que variam uniformemente, como $t = 0, 1, 2, 3, \dots$

Nos casos de interpolação, em que a superfície a ser gerada deve passar ou aproximar um conjunto de pontos, a uniformidade na distribuição dos parâmetros pode resultar em imprecisão ou má qualidade na forma final da superfície, caso os pontos fornecidos para a interpolação tenham uma distribuição não uniforme. Neste caso, valores de parâmetros que acompanhem esta distribuição são mais convenientes. Na Figura A10 observa-se a diferença na escolha da distribuição dos valores do parâmetro: na figura “a” é dado o conjunto de pontos a ser interpolado, e as figuras “c” e “d” mostram a imagem final com a distribuição uniforme e não uniforme dos valores do parâmetro. A distribuição não uniforme apresenta uma qualidade final melhor, porque os pontos fornecidos para a imagem concentram-se principalmente em detalhes, como as curvas que formam contornos de faces e extremidades.

Figura A10 - Influência dos valores dos parâmetros na curva final obtida. Fonte: (ROGERS E ADAMS, 1990).

A.2. B-splines

Existem famílias de funções de mistura que podem ser utilizadas para gerar qualquer curva *spline* para um determinado vetor de nós. Tais famílias são chamadas de base, com o significado de que toda e qualquer curva *spline* pode ser obtida por uma fórmula geral com os pontos de controle adequados. Existem muitas famílias de funções de mistura que são bases, mas existe uma em especial que oferece o menor suporte e consequentemente o melhor controle local: as *basis-splines*, ou apenas *b-splines*. As curvas *b-spline* são definidas por:

$$p(t) = \sum_{k=0}^L p_k \cdot N_{k,m}(t) \quad (\text{A44})$$

E as funções *b-spline* por:

$$N_{k,m}(t) = \left(\frac{t - t_k}{t_{k+m-1} - t_k} \right) \cdot N_{k,m-1}(t) + \left(\frac{t_{k+m} - t}{t_{k+m} - t_{k+1}} \right) \cdot N_{k+1,m-1}(t) \quad (\text{A45})$$

- sendo:
- t_k : os valores do parâmetro;
 - $T = (t_0, t_1, t_2, \dots)$: vetor de nós;
 - $L = k+1$: número de pontos de controle;
 - “ m ”: é a ordem das funções *B-splines*.

Observando que a ordem indica o número de termos no polinômio, ou seja, é o grau do polinômio adicionado de uma unidade; i.e., se a ordem dos polinômios de uma função *b-spline* é 3, o grau destes mesmos polinômios é 2; se a ordem é 4, o grau do polinômio é 3. A equação A45 é uma definição recursiva para a construção de uma função de ordem “ m ” a partir de duas funções *B-spline* de ordem “ $m - 1$ ”. A função de primeira ordem é igual à 1 dentro dos seguintes intervalos:

$$N_{k,1}(t) = \begin{cases} 1 & \text{para } t_k \leq t \leq t_{k+1} \\ 0 & \text{para outro intervalo} \end{cases} \quad (\text{A46})$$

$$N_{0,2}(t) = \frac{t}{1} \cdot N_{0,1}(t) + \frac{2-t}{1} \cdot N_{1,1}(t) \quad (\text{A47})$$

Características das *b-splines*:

- permitem um controle local melhor porque cada ponto de controle é associado à uma função base *b-spline*. Então cada ponto de controle exerce influência na curva

somente no intervalo de nós no qual sua respectiva função base *B-spline* não é nula (no seu suporte);

- possibilita a mudança na ordem da função base e conseqüentemente no grau da curva, sem modificar o número de pontos de controle;
- as funções base de Bernstein são um caso especial da função base *b-spline*.

Um exemplo é a forma da função de $N_{0,2}(t)$, dada pela equação A48, que é a primeira função *b-spline* de ordem $m=2$, para os nós equidistantes.

$$N_{0,2}(t) = \begin{cases} t & \text{para } 0 \leq t \leq 1 \\ 2 - t & \text{para } 1 \leq t \leq 2 \\ 0 & \text{para outro intervalo} \end{cases} \quad (\text{A48})$$

$N_{1,2}(t)$ é um pulso triangular deslocado de uma unidade de “ t ”, conforme o gráfico da figura A11:

$$N_{i,2}(t) = N_{0,2}(t) \cdot (t - i) \quad (\text{A49})$$

Figura A11 - Pulso triangular $N_{0,2}(t)$. Adaptado de: (ROGERS E ADAMS, 1990).

$$N_{0,3}(t) = \frac{t}{2} \cdot N_{0,2}(t) + \frac{3-t}{2} \cdot N_{1,2}(t) \quad (\text{A50})$$

Para funções *B-spline* quadráticas ($m = 3$) baseadas nos mesmos nós equidistantes, é necessário obter somente $N_{0,3}(t)$, pois as outras funções são somente translações desta: o primeiro termo é uma rampa vezes o primeiro pulso triangular, e o segundo termo é uma rampa decrescente, multiplicada pelo segundo pulso, produzindo duas parábolas coincidentes em uma extremidade. Quando os dois termos em $N_{0,3}(t)$ são somados, as extremidades desaparecem e o pulso resultante tem uma derivada contínua. O segmento médio envolve a soma de duas quadráticas resultando (Figura A12):

$$N_{0,3}(t) = \begin{cases} 0 & \text{para } 0 \leq t \leq 1 \\ \frac{1}{2}t^2 & \text{para } 1 \leq t \leq 2 \\ \frac{3}{4} - \left(t - \frac{3}{2}\right)^2 & \text{para } 2 \leq t \leq 3 \\ \frac{1}{2}(3-t)^2 & \text{para outro intervalo} \end{cases} \quad (\text{A51})$$

Figura A12 – Função base triangular $N_{0,3}(t)$. Adaptado de: (ROGERS E ADAMS, 1990).

As outras formas *spline* quadráticas, $N_{k,3}(t)$, são obtidas facilmente quando os nós são equidistantes:

$$\text{se } t_k = k: \quad \text{então } N_{k,m}(t) = N_{0,m}(t - k) \quad (\text{A52})$$

A *spline* cúbica, com função base $N_{0,4}(t)$, é a *spline* mais utilizada frequentemente. Seu gráfico é mostrado na Figura A13, sendo simétrico ao redor de $t=2$. Pode ser representado compactamente como:

$$N_{0,4}(t) = \begin{cases} u(1-t) & \text{para } 0 \leq t \leq 1 \\ v(2-t) & \text{para } 1 \leq t \leq 2 \\ v(t-2) & \text{para } 2 \leq t \leq 3 \\ u(t-3) & \text{para } 3 \leq t \leq 4 \\ 0 & \text{para outro intervalo} \end{cases} \quad (\text{A53})$$

onde os dois segmentos $u(t)$ e $v(t)$ são dados por:

$$u(t) = \frac{1}{6} \cdot (1-t)^3 \quad (\text{A54})$$

$$v(t) = \frac{1}{6} \cdot (3t^2 - 6t + 4) \quad (\text{A55})$$

Figura A13 – Funções base para a *spline* cúbica. Adaptado de: (ROGERS E ADAMS, 1990).

O cálculo das derivadas mostra que a primeira e segunda derivadas da *spline* cúbica são contínuas. Baseado no raciocínio exposto nas funções anteriores, em geral a função $N_{k,m}(t)$ inicia em t_k e termina em t_{k+m} , seu suporte é $[t_k, t_{k+m}]$, e é sempre positivo.

A.3. Superfícies NURBS

Para o caso de superfícies *NURBS* os pontos de controle formam uma malha externa à superfície, e o controle que cada ponto exerce sobre a curva é determinado pelos pesos associados a cada um, ou seja, a malha de controle destes pontos define a superfície *NURBS*. Uma superfície *NURBS* é a generalização racional do produto tensor da superfície *B-Spline* não racional e é definida como:

$$S(u, v) = \frac{\sum_{i=0}^n \sum_{j=0}^m w_{i,j} \mathbf{P}_{i,j} N_{i,p}(u) N_{j,q}(v)}{\sum_{i=0}^n \sum_{j=0}^m w_{i,j} N_{i,p}(u) N_{j,q}(v)} \quad (\text{A56})$$

onde: - $w_{i,j}$ são os pesos;

- $\mathbf{P}_{i,j}$ são os pontos que formam uma rede de controle da superfície;

- $N_{i,p}(u)$ e $N_{j,q}(u)$ são as funções base *B-Spline* normalizadas de grau p e q nas direções u e v respectivamente, definidas por:

$$N_{i,0}(u) = \begin{cases} 1, & \text{se } u_i \leq u < u_{i+1} \\ 0, & \text{caso contrário} \end{cases} \quad (\text{A57})$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \quad (A58)$$

* a construção de $N_{j,q}(u)$ é similar;

- u e v são os valores dos nós que definem as direções da curva, fornecidos pelos vetores de nós:

$$U = \{0, 0, \dots, 0, u_{p+1}, \dots, u_{r-p-1}, 1, 1, \dots, 1\} \quad (A59)$$

$$V = \{0, 0, \dots, 0, v_{q+1}, \dots, v_{s-q-1}, 1, 1, \dots, 1\} \quad (A60)$$

Para uma superfície é necessário uma matriz de nós para as curvas que compõem a superfície em duas direções (dois vetores de nós). As superfícies *NURBS* podem ser consideradas também como sendo constituídas por curvas nos dois sentidos ou direções de formação da superfície, que satisfazem as condições de uma *NURBS*, ou seja, possuem pontos de controle que estão fora ou podem estar contidos na superfície. Estes pontos de controle formam uma malha externa à superfície, e o controle que cada ponto exerce sobre a curva é determinado pelos pesos associados a cada um: a “malha” ou “rede” de controle formada por estes pontos define a superfície *NURBS*. A utilização da *NURBS* deve-se às suas excelentes propriedades e a sua incorporação em padrões internacionais como:

IGES – International Standard Office Standard for the Exchange of Product Model Data (IGES, 1996);

PHIGS+ – Product Data Exchange Specification;

STEP – Standard for the Exchange of Product Model Data.

Algumas razões para a aceitação e utilização da *NURBS* são:

- oferta de uma forma comum para a representação e projeto de formas analíticas comuns (cônicas, quádricas, superfícies de revolução), assim como curvas e superfícies de forma livre. Deste modo, formas analíticas e formas livres são representadas precisamente, e podem ser armazenadas em um banco de dados unificado;
- representação precisa de formas analíticas e de formas livres;

- a manipulação via pontos de controle e pesos fornece a flexibilidade para o projeto de uma grande variedade de formas;
- evolução razoavelmente rápida e computacionalmente estável;
- interpretações geométricas claras, o que a torna útil para usuários que possuem bons conhecimentos de geometria – principalmente geometria descritiva;
- projeto intuitivo: quase toda ferramenta e algoritmo possui uma interpretação geométrica de fácil compreensão;
- ferramentas geométricas poderosas (inserção, refinamento e remoção de nós, elevação do grau, divisão) que podem ser utilizadas para projeto, análise, processo e objetos de interrogação;
- invariância sobre mudança de escala, rotação, translação, corte e nas projeções perspectiva e paralela;
- são generalizações genuínas das formas *B-splines* não racionais e das curvas e superfícies de Bezier racionais e não racionais;
- disponibilidade de biblioteca de algoritmos testados.

Desvantagens:

- armazenamento extra para definir superfícies tradicionais: como exemplo, para representar uma circunferência inteira são necessários sete pontos de controle formando um quadrado e dez nós, enquanto a representação tradicional requer somente o raio, posição do centro e o vetor normal ao plano que contém a circunferência. Em três dimensões isso significa o armazenamento de 38 números ao invés de sete;
- aplicação imprópria dos pesos pode resultar em uma parametrização muito ruim que pode destruir construções subsequentes da superfície;
- algumas técnicas de interrogação (PIEGL, 1991) trabalham melhor com formas tradicionais do que com *NURBS*. Um exemplo é a interseção de superfícies em que é particularmente difícil manusear casos de “toque correto” ou “envoltório”;
- algoritmos fundamentais, como mapeamento de ponto inverso, são sujeitos à instabilidade numérica.

As *NURBS* tem se tornado padrão para a representação, projeto e troca de informação geométrica processada por computadores. As excelentes propriedades matemáticas e

algorítmicas, combinadas com aplicações industriais de sucesso, contribuíram para a popularidade das *NURBS*.

APÊNDICE B – Diagramas RTL

Os circuitos sintetizados são apresentados na forma de diagrama esquemático *Register Transfer Level*, que representa os *cores* de sistemas e subsistemas em forma hierárquica em diagramas de blocos, de funções, operações e circuitos lógicos básicos. Para simplificação do diagrama apresentado, o circuito (*core*) apresentado para a geração de *b-splines* é um exemplo de um circuito para gerar 4 pontos de uma curva *b-spline* bidimensional de grau 2, com 3 circuitos sequenciais de geração de pontos de controle para cada camada do algoritmo Cox-de Boor.

Dois multiplexadores de 4 entradas são utilizados para enviar os dados para outros circuitos, um multiplexador para a coordenada euclidiana “x” e outro para a “y”. Os subsistemas deste circuito *b-spline* se repetem para cada ponto gerado, e para cada camada de geração dos pontos de controle intermediários se repetem os circuitos para o cálculo da combinação convexa de *alfa* com estes pontos de controle.

Figura B1 – Circuito principal para 4 cores fast Cox-de Boor.

Figura B2 – Circuito das três camadas para um *core fast* Cox-de Boor de grau 2.

Figura B3 – Entrada dos pontos de controle para o *core fast* Cox-de Boor.

Figura B4 – Circuito da combinação convexa do core fast Cox-de Boor.

Figura B5 – Circuito para o cálculo de α do algoritmo Cox-de Boor.

APÊNDICE C – Dados Técnicos e Implementação GPU

Especificações técnicas da *GPU GeForce G105M* (NVIDIA CORPORATION, 2009):

- 8 *cores CUDA*;
- frequência do sinal de *clock* do núcleo de 640 MHz;
- frequência do sinal de *clock* do *shader*: 1,6 GHz;
- frequência da memória: 702 MHz, com até 1,404 GHz de taxa de comunicação de dados;
- memória de vídeo dedicada de 256 MB tipo *GDDR3*;
- interface com a memória de 64 *bits*;
- memória compartilhada com o sistema de 1,262 GB;
- memória de gráficos total disponível de 1,518 GB;
- barramento padrão *PCI Express x16*.

A implementação do sistema de geração de curvas e superfícies *b-splines* na *GP-GPU* utiliza a arquitetura *CUDA* (NVIDIA CORPORATION, 2011) de acordo com o modelo de *GPU* utilizado, configurada para a mesma capacidade de processamento compatível com o *FPGA*, seguindo os mesmos princípios de implementação. A linguagem de programação *C++* foi utilizada, possibilitando a compilação de funções e classes para o *host* (computador) e *device* (*GPU*) no mesmo código fonte. Deste modo, a implementação em sistemas de arquitetura convencional do algoritmo Cox-de Boor é feita de dois modos, sequencial e paralela, no mesmo código fonte (KIRK e HWU, 2010). Para a implementação na *GPU* são executadas as seguintes etapas:

1. Busca do índice dos parâmetros no vetor de nós correspondentes;
2. Alocação na memória dos parâmetros, vetor de índice e vetor de nós;
3. Cálculo dos valores alfa do algoritmo *Cox-deBoor* (dependem somente dos valores dos parâmetros e dos nós);
4. Alocação dos pontos de controle junto aos valores alfa calculados;
5. Repetição do processo do item 3 até o número de vezes do grau da curva.

De acordo com a disponibilidade de diversos modelos e versões de *GP-GPUs*, e sendo uma característica destas a execução de diversos *threads* paralelamente via *hardware*, é necessário o ajuste do *software* aos recursos disponíveis pelo tipo de *GPU* a ser utilizada, de modo a otimizar o tempo de execução do sistema. Os dois principais recursos a serem especificados pelo programa são:

- o número máximo de *threads* que podem ser executados simultaneamente;
- a quantidade disponível, tipo, e endereçamento da memória disponível na *GPU*, otimizando a transferência de dados entre os circuitos de *threads* e as memórias.

Os *threads* em *GPUs* geralmente são multi-dados, executando o mesmo código em diversos *threads*, mas utilizando e produzindo dados diferentes. Os dados a serem processados podem ser vistos como elementos em matrizes de *threads*, e o processamento como operações matriciais na qual todos os elementos de dados são processados ao mesmo tempo. Caso o volume de dados ultrapasse a capacidade máxima da *GPU*, um novo processamento dever ser efetuado, resultando no *thread swap*, processo que demanda tempo de execução e deve ser reduzido ao máximo. Além do número de *threads*, a memória disponível na própria *GPU* geralmente é reduzida em relação ao sistema do *host*, mas deve ser utilizada ao máximo para evitar a troca de dados entre o *device* e o *host*, sendo os dados transferidos no início da execução do *kernel*. Deste modo, a matriz dos pontos de controle e os vetores de nós são transferidos na chamada do *kernel* no programa principal para a *GPU*, sendo alocados dois vetores na memória compartilhada para o algoritmo *Cox de-Boor* como demonstrado na Figura C1. Dois vetores iniciais, com os valores dos parâmetros e o respectivo índice no vetor de nós são utilizados.

Os *threads* são agrupados em blocos, de no mínimo 8 *threads*, com processamento concorrente entre os blocos. Deste modo, o número de *threads* a ser utilizado é igual ao número de parâmetros, i.e., o tamanho do vetor inicial de parâmetros, arredondado para o maior inteiro divisível por 8. Estes vetores, de parâmetros e dos índices no vetor de nós, são sobrescritos (HARRIS, 2007) até o valor dos pontos finais resultantes de cada parâmetro, sendo $den1_n$ e $den2_n$ na Figura C1, o numerador e denominador no cálculo do valor de alfa.

Para o modelo de *GPU* utilizado, a área de memória é dividida e classificada de acordo com o formato dado pela Figura C2, sendo a classificação e o tipo descritos na Tabela C1, observando-se a memória do *host*, que apresenta a menor taxa de transferência de dados pela dependência do barramento da *CPU*. A medida do tempo de execução é efetuada pela

própria *GPU* por meio de temporizadores controlados por comandos de alto nível disponibilizados pelo *CUDA*.

par:	t_0	t_1	t_2	t_3	t_4	t_5		t_{n-2}	t_{n-1}	t_n
ind:	u_0	u_0	u_1	u_1	u_2	u_2		u_{n-1}	u_n	u_n
par:	$t_0 - u_0$	$t_1 - u_0$	$t_2 - u_1$	$t_3 - u_1$	$t_4 - u_2$	$t_5 - u_2$		$t_{n-2} - u_{n-1}$	$t_{n-1} - u_n$	$t_n - u_n$
	$den1_0$	$den1_1$	$den1_2$	$den1_3$	$den1_4$	$den1_5$		$den1_{n-2}$	$den1_{n-1}$	$den1_n$
ind:	$u_{0+k-1}-u_0$	$u_{0+k-1}-u_0$	$u_{0+k-1}-u_0$	$u_{1+k-1}-u_1$	$u_{1+k-1}-u_1$	$u_{1+k-1}-u_1$		$u_{n+k-3}-u_n$	$u_{n+k-2}-u_n$	$u_{n+k-1}-u_n$
	$den2_0$	$den2_1$	$den2_2$	$den2_3$	$den2_4$	$den2_5$		$den2_{n-2}$	$den2_{n-1}$	$den2_n$
par=	$\frac{den1_0}{den2_0}$	$\frac{den1_1}{den2_1}$	$\frac{den1_2}{den2_2}$	$\frac{den1_3}{den2_3}$	$\frac{den1_4}{den2_4}$	$\frac{den1_5}{den2_5}$		$\frac{den1_{n-2}}{den2_{n-2}}$	$\frac{den1_{n-1}}{den2_{n-1}}$	$\frac{den1_n}{den2_n}$
alfa:	$alfa_0$	$alfa_1$	$alfa_2$	$alfa_3$	$alfa_4$	$alfa_5$		$alfa_{n-2}$	$alfa_{n-1}$	$alfa_n$
par:	$P_0 * alfa_0$	$P_0 * alfa_1$	$P_0 * alfa_2$	$P_1 * alfa_3$	$P_1 * alfa_4$	$P_1 * alfa_5$		$P_{n-1} * alfa_{n-2}$	$P_{n-1} * alfa_{n-1}$	$P_{n-1} * alfa_n$
ind:	$1 - alfa_0$	$1 - alfa_1$	$1 - alfa_2$	$1 - alfa_3$	$1 - alfa_4$	$1 - alfa_5$		$1 - alfa_{n-2}$	$1 - alfa_{n-1}$	$1 - alfa_n$
ind:	$P_0 * (1 - alfa_0)$	$P_0 * (1 - alfa_1)$	$P_0 * (1 - alfa_2)$	$P_1 * (1 - alfa_3)$	$P_1 * (1 - alfa_4)$	$P_1 * (1 - alfa_5)$		$P_{n-1} * (1 - alfa_{n-2})$	$P_{n-1} * (1 - alfa_{n-1})$	$P_{n-1} * (1 - alfa_n)$
par:	P_0^{-1}	P_1^{-1}	P_2^{-1}	P_3^{-1}	P_4^{-1}	P_5^{-1}		P_{n-2}^{-1}	P_{n-1}^{-1}	P_n^{-1}
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$

Figura C1 – Alocação matricial dos dados para o processamento do circuito Cox-de Boor.

Essa temporização pode ser feita via sincronia das *threads*, criando-se eventos para a medida do intervalo de tempo de execução de determinados trechos do *software*. Depois da transferência de dados, o código sequencial do *software* pode ser executado no *host* e o respectivo código para a *GPU* (*kernel*) em paralelo, dividindo-se as medidas de tempo dos

dois sistemas. Deste modo, neste trabalho são utilizados os comandos de temporização para a *GPU* que levam em conta somente o processamento interno das *threads*, sendo outro fator importante a largura de banda da transferência de dados entre as memórias disponíveis no sistema, incluindo as diferentes áreas de memória da *GPU* e a memória principal do *host* (CPU).

Figura C2 – Organização dos tipos de memória na placa *GPU*.

Tabela C1 – Características dos tipos de memória na placa *GPU*.

Memória	Local	Cache	Acesso	Escopo	Duração
Registrador	on chip	Não	Leitura/escrita	1 thread	Thread
Local	off chip	Dispositivos de maior capacidade	Leitura/escrita	1 thread	Thread
Compartilhada	on chip	Não	Leitura/escrita	Todos threads em um bloco	Bloco
Global	off chip	Dispositivos de maior capacidade	Leitura/escrita	Todos threads e o host	Alocação do host
Constante	off chip	Sim	Leitura	Todos threads e o host	Alocação do host
Textura	off chip	Sin	Leitura	Todos threads e o host	Alocação do host

APÊNDICE D – Dados Técnicos – Sistema *FPGA*

D.1. Placa DE2-70 (TERASIC, 2007):

- Dispositivo FPGA Altera Cyclone II 2C70;
- Dispositivo de Configuração Serial Altera - EPCS16;
- USB Blaster (on board) para controle *API* do usuário e programação; modos de programação JTAG e Active Serial (AS);
- SSRAM de 2-Mbytes;
- Duas SDRAM de 32-Mbytes;
- Memória Flash de 8-Mbytes;
- Soquete para cartão SD;
- 4 chaves *pushbutton*;
- 18 chaves comutadoras;
- 18 LEDs vermelhos;
- 9 LEDs verdes;
- Osciladores de 50-MHz e 28.63-MHz para fontes de sinal de *clock*;
- *Codec* de áudio de 24-bits qualidade de *CD* com saídas de áudio e entrada de áudio e microfone;
- *VGA DAC* (*DACs* triplos de 10-bits de alta velocidade) com conector de saída *VGA*;
- 2 *Decoders* de TV (*NTSC/PAL/SECAM*) com conector de entrada para sinais de TV;
- Controlador Ethernet 10/100 com conector modular;
- Controlador USB Host/Slave com conectores USB tipo A e tipo B;
- Transceiver RS-232 e conector de 9 pinos;
- Conector PS/2 para mouse/teclado;
- Transceiver infra-vermelho (*IrDA*);
- Conector *SMA*;
- Duas barras de conectores de expansão de 40 pinos com proteção a diodo.

D.2. *FPGA* Altera Cyclone II EP2C70F896C6N (ALTERA, 2008):

- 68.416 elementos lógicos;
- 250 blocos de *RAM* M4K;
- 1.152.000 de bits total da *RAM*;
- 150 multiplicadores embarcados;
- 4 *PLLs*;
- 622 pinos de entrada/saída disponíveis ao usuário;
- Invólucro *BGA* de 896 pinos FineLine.

D.3. *SDRAM* (ALTERA, 2008):

- *Chips* de memória dinâmica síncrona *RAM* de 32-Mbytes de taxa de dados simples (2x);
- Organizada em 4 bancos de 4 M por 2 *bytes* de célula de memória;

D.4. *VGA DAC* (ADV123, 2008):

- *DAC* de vídeo ADV7123 240-MHz de 10-*bits* de alta velocidade;
- Conector *sub-D* de 15 pinos de alta densidade;
- Taxa máxima de varredura de 1600 x 1200 *px* de 100 Hz.

APÊNDICE E – Operação da SDRAM

E.1. Inicialização

A inicialização para o modo normal de operação ocorre na alimentação do circuito até o sinal do *clock* se tornar estável. É necessário um *delay* de 200 μ s anteriormente à execução de qualquer comando, sendo aplicado posteriormente um comando *PRECHARGE* (*PALL*) para ajustar todos os bancos de memória para um estado ocioso, e no mínimo oito ciclos de *AUTO REFRESH* (*CBR*) da memória devem ser executados, após os quais o registrador de modo estará pronto para a operação (deve ser configurado antes da operação para não resultar em um estado indefinido depois desta inicialização).

E.2. Ativação do Banco/Linha

Uma linha deve ser “aberta” em um banco de memória SDRAM antes de um comando de leitura ou escrita, via o comando *ACTIVE* (*ACT*), que seleciona o banco e a linha a serem ativados para o acesso. Após a abertura de uma linha, um comando está sujeito à especificação de tempo t_{RCD} , conforme a Tabela E1. A linha acessada deve ser desativada antes do acesso à outra linha no mesmo banco da memória, com um intervalo de tempo mínimo entre a desativação e o acesso à outra linha definido por t_{RC} . Uma linha em outro banco de memória pode ser acessada enquanto o primeiro banco está sendo acessado, o que resulta em uma redução na sobrecarga do acesso total das linhas, e o intervalo de tempo mínimo entre estes acessos sucessivos a diferentes bancos é definido por t_{RRD} (ISSI, 2007).

Tabela E1 – Frequências de operação e relações de latência. Fonte: (ISSI, 2007).

Tempo do ciclo de <i>clock</i>	6 ns	7 ns	8 ns	10ns
Frequência de operação	166 MHz	143 MHz	125 MHz	100 MHz
Latência CAS (t_{CAC})	3 ciclos	3 ciclos	2/3 ciclos	2/3 ciclos
Tempo de atraso entre o comando ACTIVE e o comando READ/WRITE (t_{RCD})	3 ciclos	3 ciclos	3 ciclos	2 ciclos
Latência RAS ($t_{RCD}+t_{CAC}$)	Latência CAS=3	6 ciclos	6 ciclos	6 ciclos
	Latência CAS=2	--	--	5 ciclos
Período de comando REF para REF ou ACT para ACT (t_{RC})	10 ciclos	10 ciclos	8 ciclos	7 ciclos
Período de comando ACT para diferentes bancos (t_{RRD})	2 ciclos	2 ciclos	2 ciclos	2 ciclos

E.3. READ

Os endereços da coluna inicial e do banco de memória devem ser fornecidos juntamente com o comando de leitura nos *burst READ*, possibilitando habilitar a pré-carga ao completar o *burst*. Os dados lidos da coluna inicial estarão disponíveis depois da latência *CAS*, seguinte ao comando de leitura, e os dados subsequentes estarão disponíveis no próximo sinal de *clock*, finalizando com os pinos de saída no nível lógico alto.

Um *burst* do tipo *full page* continua até o final da página ou se desloca para a coluna 0, dando continuidade ao processo. Dados de *bursts READ* podem ser truncados com um comando *READ*, *WRITE* ou *PRECHARGE* subsequente, e um fluxo contínuo de dados pode ser mantido. Um novo comando *READ* deve ser acionado um número de ciclos do sinal de *clock* igual à latência *CAS* menos um, antes da borda do sinal de *clock* com o último elemento dos dados. Um comando pode ser iniciado em qualquer ciclo de *clock* seguinte a um comando *READ*. O *burst WRITE* pode ser inicializado no *clock* seguinte ao último elemento de dados do *burst READ*, para evitar a contenção *I/O*: em um sistema pode haver a possibilidade de que o dispositivo que direciona os dados de entrada mude para o nível lógico baixo antes que os terminais de dados *DQs* comutem para o nível lógico alto. Neste caso, deve ocorrer um atraso de no mínimo um ciclo entre o último dado lido e o comando *WRITE*. A entrada *DQM* é utilizada para evitar contenção *I/O*.

O comando *PRECHARGE* deve ser acionado em um número de ciclos igual à latência *CAS* menos um, antes da borda de subida do sinal de *clock* com o último elemento de dado. Após o comando *PRECHARGE*, um comando subsequente ao mesmo banco não pode ser designado antes do intervalo específico t_{RP} (ISSI, 2007).

No caso de um *burst* de comprimento fixo sendo executado, um comando *PRECHARGE* designado no intervalo de tempo t_{rp} executa a mesma operação que resultaria do mesmo *burst* de comprimento fixo fornecido com auto pré-carga. O comando *PRECHARGE* requer os barramentos de comando e endereços disponíveis no momento apropriado para designar o comando; mas pode ser utilizado para truncar *bursts* de comprimento fixo ou de *full page*. O comando de terminação de *BURST* deve ser designado “x” ciclos antes da borda do sinal de *clock* no qual o último dado é válido, onde “x” é igual à latência *CAS* menos um.

E.4. WRITE

Bursts WRITE são iniciados com os endereços da coluna inicial e do banco fornecidos juntamente com este comando, e se a auto pré-carga for habilitada, a linha sendo acessada é

pré-carregada ao se completar o *burst*. O primeiro elemento de entrada será registrado coincidente com o comando *WRITE* e os subsequentes são registrados em cada borda positiva do sinal de *clock*. Na conclusão de um *burst*, os pinos *DQs* permanecerão no nível lógico alto e qualquer entrada de dados adicional será ignorada. Um *burst full page* continua sua execução até o final da página da memória, ou é deslocado para a coluna 0.

Dados para qualquer *burst WRITE* podem ser truncados com um comando *WRITE*, *READ* ou *PRECHARGE* subsequente, e o respectivo dado é fornecido junto a este novo comando. Uma vez executado o comando *READ*, as entradas de dados serão ignoradas, e comandos *WRITE* não serão executados. Após o comando *PRECHARGE*, um comando subsequente ao mesmo banco não pode ser designado até t_{RP} ser alcançado (ISSI, 2007). O comando *PRECHARGE* requer os barramentos de comando e endereços disponíveis no momento apropriado para designar o comando, mas pode ser utilizado para truncar *bursts* de comprimento fixo ou de *full page*.

Bursts WRITE de *full page* ou de comprimento fixo podem ser truncados com o comando *BURST TERMINATE*, e quando truncado com um *burst WRITE*, o dado de entrada aplicado coincidente com o comando *BURST TERMINATE* será ignorado. O último dado escrito (considerando que *DQM* está no nível lógico baixo neste instante) será o dado de entrada aplicado um ciclo de *clock* antes ao comando *BURST TERMINATE*.

E.5. Pré-Carga

O comando *PRECHARGE* é utilizado para desativar uma linha “aberta” em um banco, sendo disponibilizado seu acesso em um intervalo de tempo t_{RP} após a execução do comando. Somente um ou todos os bancos podem ser pré-carregados, e na pré-carga o banco se encontra no estado ocioso e deve ser ativado anteriormente a qualquer comando *READ* ou *WRITE*.

E.6. BURST READ/SINGLE WRITE

No modo de *burst* de leitura/escrita simples todos os comandos *WRITE* resultam no acesso de um único endereço de coluna (*burst* de um), apesar do comprimento do *burst* programado. Comandos *READ* acessam colunas de acordo com o comprimento e sequência para o *burst* programado.

E.7. Auto Pré-Carga Concorrente

Um comando de leitura ou escrita a outro banco de memória, enquanto outro comando de acesso com auto pré-carga está em execução, não é permitido pelas *SDRAMs*, a menos que

ofereça suporte a auto pré-carga concorrente. As *SDRAMs* utilizadas neste projeto suportam auto pré-carga concorrente em quatro casos onde a mesma ocorre:

- Leitura com auto pré-carga, interrompida por um comando de leitura ou interrompida por um comando de escrita;
- Escrita com auto pré-carga, interrompida por um comando de leitura ou interrompido por um comando de escrita.

APÊNDICE F – Sistema de Visualização

F.1. Interface de Vídeo

Um circuito para um sistema de visualização básico foi desenvolvido, configurado de acordo com a compatibilidade com o circuito integrado de interface de vídeo. Esta interface de vídeo consiste em um circuito integrado conversor digital-analógico controlador de vídeo *ADV7123* da *Analog Devices* (2010), com 16 pinos de saída conectados a um conector *D-SUB* para monitores de vídeo compatíveis com o padrão *VGA*, 30 pinos de entrada para os sinais *RGB* (10 *bits* para cada sinal) e 5 pinos para os sinais de controle de sincronização de acordo com as especificações de temporização para sinais *VGA*. O diagrama de conexão com a pinagem desta interface de vídeo é exibido na Figura F1.

Os sinais de controle da sincronia do vídeo são enviados diretamente de pinos configurados como saídas do *FPGA*, sendo gerados por um *core* responsável pela varredura da memória de vídeo em sincronia com a frequência dos feixes *RGB* do conversor *VGA* (SINGH ET. AL., 1994). Um sinal de *clock* de frequência diferente do sinal utilizado para o restante dos *cores* do sistema foi gerado, via *PLL*, para compatibilidade com as frequências determinadas para as configurações possíveis de resolução disponíveis no *hardware* de interface de vídeo. A resolução utilizada foi de 640 x 480 *pixels*, que é um valor de resolução comumente encontrado nos monitores *VGA* (alguns modelos mais antigos não suportam resoluções maiores).

Os pinos R0-R9, G0-G9 e B0-B9 são as entradas digitais para as componentes *RGB*, provenientes do *FPGA*, que controlam os sinais analógicos nas saídas analógicas *IOR*, *I OG* e *IOB* para o controle dos três feixes de elétrons de um monitor *VGA*. Os sinais de controle do sincronismo horizontal (*HS*), vertical (*VS*) e de transmissão dos dados (*SYNC*) também são provenientes do *FPGA*, além de um sinal de desativação das saídas (*BLANK*), necessário para o sincronismo. Estes sinais de controle possibilitam a operação em diferentes resoluções e dimensões da tela de exibição, de acordo com a frequência do sinal de *clock* de entrada do circuito proveniente do *FPGA*, controlado via programação, assim como os tempos de controle dos sinais de sincronismo.

O conversor utiliza o formato *RGB* de 10 bits e a temporização padrão para os sinais de sincronismo horizontal, determinando as linhas, e vertical, para os frames. A temporização

básica para as linhas horizontais, exibidas pelo controlador *VGA*, obedece aos padrões das resoluções de monitores de vídeo para o sincronismo horizontal (*hsync*) e vertical (*vsync*), executadas por um *loop* de contagem, interno a outro *loop* de temporização da varredura horizontal, utilizando como referência um sinal de *clock* de 27 MHz, produzido por um *PLL* a partir do sinal externo de referência de 50 MHz.

Figura F1 – Sinais e conexões para o circuito de interface de vídeo VGA. Fonte: (TERASIC, 2007).

A interface de vídeo digital realiza a varredura de uma memória que contém os dados necessários para exibição no monitor de vídeo, uma memória *on chip* do *FPGA*, devido ao requisito de varredura dos dados da memória no tempo requerido pela interface de vídeo, utilizada como um *framestore*. O circuito A/D de interface de vídeo *VGA*, apesar de oferecer 10 *bits* para cada sinal do sistema *RGB*, é configurado para sinais de vídeo monocromáticos, suportados pelo tamanho máximo da memória *on chip* do *FPGA*, de 1,152 Mb. No caso de uma resolução de 1024x768 *pixels*, com o número máximo de *cores* possível ($3 \times 10 \text{ bits} = 30 \text{ bits/px}$) do conversor, seria necessária uma memória com aproximadamente 23,6 MB. Portanto, de acordo com as configurações do tipo de controlador de vídeo, foi levado em conta o número de elementos lógicos disponíveis, o número de elementos de memória *RAM* total, a frequência máxima do sinal de *clock* a ser utilizado e a disponibilidade de circuitos periféricos como memórias e *PLLs* para o ajuste da frequência.

Os itens anteriores são relativos ao circuito do *FPGA*, sendo possível o incremento na configuração do sistema através de circuitos periféricos, como memórias e divisores de frequência. Quando não é possível o acesso simultâneo de dois ou mais circuitos à memória de vídeo, a prioridade de acesso é dada ao controlador de vídeo, para disponibilizar o pixel no tempo correto. A lógica programável possibilita que o algoritmo de controle, ou o circuito sintetizado de controle da varredura da memória de vídeo opere independente do restante do circuito sintetizado no *FPGA*. Assim a leitura da memória de vídeo sempre é efetuada constantemente, independente do resultado do processamento de outros circuitos.

F.2. Sistema de Visualização

Para a visualização de gráficos, principalmente em sistemas gráficos 3D, normalmente é executada a renderização pelo processo de um *pipeline* 3D separado em etapas (ou estágios) serializadas, cujo número pode variar entre diferentes implementações, mas pode ser dividida em quatro seções:

- Aplicação/Cena;
- Geometria;
- Configuração dos triângulos;
- Rasterização/Renderização.

Os *cores* desenvolvidos são semelhantes aos *minimal pipelines* (KNUTSSON, 2005) utilizados em *GPUs*, conforme o diagrama da Figura F2, excluindo a etapa de colorização, sendo minimizado para executar a visualização das primitivas geométricas: circunferências, *splines* e retas em módulos implementados para sistemas 2D e 3D de coordenadas.

O módulo básico de visualização consiste de um circuito para mapeamento (ou conversão) das coordenadas euclidianas 2D para a memória de vídeo, sendo utilizado também pelo sistema 3D. Este mapeamento é realizado pela transformação dos valores das coordenadas euclidianas em dois eixos cartesianos xy para o endereçamento linear da memória de vídeo (Figura F3). Como as células de memória possuem uma dimensão mínima de 8 *bits*, é necessário ler o valor contido no endereço de memória antes da operação de escrita nesta memória de vídeo. Cada endereço corresponde a 8 *pixels* monocromáticos, reduzindo o tamanho necessário da memória.

Figura F2 – Pipeline 3D mínimo.

Figura F3 – Mapeamento da memória de vídeo.

Para o *viewport* de transformação, funções trigonométricas são utilizadas baseadas em uma tabela de valores trigonométricos, para as operações de translação, rotação e mudança de escala, de acordo com os circuitos utilizados por Bensali (2003, 2006), cujas matrizes são ilustradas na Figura F4 e na Figura F5.

Para os recursos de *hardware* disponíveis do sistema *FPGA*, existe a possibilidade de configuração de acordo com os quesitos de resolução (número de linhas e colunas utilizadas para a rasterização), taxa de varredura horizontal, profundidade da paleta de cores disponíveis,

além de ser possível implementar *buffers* múltiplos de vídeo de acordo com a capacidade máxima de armazenamento das memórias utilizadas (a memória *onchip* suporta somente mais um *buffer*).

Figura F4 – Operações de geometria com os pontos das primitivas geométricas no estágio de geometria de um pipeline gráfico.

Figura F5 – Operações de transformação de vista dos gráficos.

Genericamente, as placas gráficas 3D são a implementação física do *pipeline* 3D. O processamento, em alguns estágios do sistema de visualização, pode ser sobreposto para incremento do tempo de execução. O processamento de dois ou mais vértices no estágio de geometria ou o processamento de dois ou mais *pixels* do mesmo triângulo no estágio de rasterização podem ser executados ao mesmo tempo. Como nenhum dado é transferido de um vértice ao outro no estágio de geometria ou de um *pixel* ao outro no estágio de renderização, é possível implementar múltiplos canais de *pixels*, resultando em um incremento significativo de tempo de execução utilizando o processamento paralelo nestas entidades independentes.