UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA E DE MATERIAIS
CAMPUS CURITIBA

RICARDO CASAGRANDE FAUST

MÉTODO PARALELO DE CÁLCULO DA TRAJETÓRIA DE PREENCHIMENTO EM ZIGUE-ZAGUE PARA MANUFATURA ADITIVA

DISSERTAÇÃO

CURITIBA
2019

RICARDO CASAGRANDE FAUST

MÉTODO PARALELO DE CÁLCULO DA TRAJETÓRIA DE PREENCHIMENTO EM ZIGUE-ZAGUE PARA MANUFATURA ADITIVA

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Engenharia, do Programa de Pós-Graduação em Engenharia Mecânica e de Materiais, Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Neri Volpato, Ph.D.
Coorientador: Prof. Rodrigo Minetto, Ph.D.

CURITIBA
2019

Dados Internacionais de Catalogação na Publicação

Faust, Ricardo Casagrande
Método paralelo de cálculo da trajetória de preenchimento em zigue-zague para manufatura aditiva [recurso eletrônico] / Ricardo Casagrande Faust. -- 2019.
1 arquivo de texto (83 f.): PDF; 1,76 MB.

Texto em português com resumo em inglês
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Mecânica e de Materiais, Curitiba, 2019
Bibliografia: p. 74-83.

1. Engenharia mecânica - Dissertações. 2. Impressão tridimensional. 3. Processos de fabricação - Planejamento. 4. Algoritmos. 5. Unidade de processamento gráfico. 6. OpenCL (Linguagem de programa de computador). 7. Programação paralela (Computação)I. Volpato, Neri, orient. II. Minetto, Rodrigo, coorient. III. Universidade Tecnológica Federal do Paraná – Programa de Pós-graduação em Engenharia Mecânica e de Materiais. IV. Título.

CDD: Ed. 23 -- 620.1

Biblioteca Ecoville da UTFPR, Câmpus Curitiba
Bibliotecária: Lucia Ferreira Littiere – CRB 9/1271

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação

TERMO DE APROVAÇÃO DE DISSERTAÇÃO Nº379

A Dissertação de Mestrado intitulada: **Método Paralelo de Cálculo da Trajetória de Preenchimento em Zigue-Zague para Manufatura Aditiva**, defendida em sessão pública pelo Candidato **Ricardo Casagrande Faust**, no dia 06 de dezembro de 2019, foi julgada para a obtenção do título de Mestre em Engenharia, área de concentração: Engenharia de Manufatura, e aprovada em sua forma final, pelo Programa de Pós-Graduação em Engenharia Mecânica e de Materiais – PPGEM.

BANCA EXAMINADORA:
Prof. Dr. Neri Volpato - Presidente - UTFPR
Prof. Dr. Milton Borsato - UTFPR
Prof. Dr. Eduardo Todt - UFPR

A via original deste documento encontra-se arquivada na Secretaria do Programa, contendo a assinatura da Coordenação após a entrega da versão corrigida do trabalho.

Curitiba, _____de _____de 20___.

Carimbo e assinatura do Coordenador do Programa

_____

AGRADECIMENTOS

Ao meu sócio Felipe e meus colegas de trabalho, por sua anuência e incentivo, lidando de forma tranquila com a minha ausência na empresa durante o desenvolvimento desta pesquisa.

À minha esposa Renata, pelo companheirismo e por estar ao meu lado, me dizendo que “vai dar tudo certo” e me apoiando especialmente nos momentos mais difíceis.

Aos meus pais, irmão e toda a minha família, pelo apoio e pela compreensão do meu direcionamento de atenção e de minhas ausências nos eventos familiares nos meses finais da pesquisa.

Aos meus orientadores Neri Volpato e Rodrigo Minetto, por confiar em mim e aceitar a orientação da minha pesquisa. Agradeço também pela sua dedicação de tempo e pelo esforço em tornar esta pesquisa cada vez melhor.

A todos os professores e colegas das disciplinas que cursei ao longo do mestrado, pelos valiosos ensinamentos e troca de experiências.

A todos que me ajudaram, apoiaram e contribuíram, ainda que de forma indireta, para a realização e conclusão deste trabalho.

Para todos vocês: Muito obrigado!

RESUMO

FAUST, Ricardo. Método paralelo de cálculo da trajetória de preenchimento em zigue-zague para manufatura aditiva. 2019. 83 f. Dissertação – Programa de Pós-Graduação em Engenharia Mecânica e de Materiais, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

A manufatura aditiva (*Additive Manufacturing - AM*) é um processo de fabricação através da adição sucessiva de camadas de material. O planejamento do processo é uma etapa computacional anterior à fabricação, na qual o modelo tridimensional CAD (*Computer Aided Design*) é processado e convertido em informações que serão utilizadas pela máquina de produção. Nesta fase, a peça é fatiada em camadas e, para cada camada, é identificada a área que deve ser preenchida com material. A escolha da estratégia de preenchimento e a definição de seus parâmetros de cálculo são de fundamental importância para a AM, pois conferem diferentes propriedades mecânicas ao objeto, além de impactar no tempo e custo de fabricação da peça. O desenvolvimento das máquinas de AM, que permitem a fabricação de peças maiores e com camadas mais finas, faz com que a demanda computacional do planejamento do processo se torne cada vez maior, de modo que em alguns casos este processamento pode levar horas. O desenvolvimento de algoritmos mais eficientes para o planejamento do processo abre uma série de possibilidades, como por exemplo, a otimização de parâmetros via simulações. Atualmente, a paralelização de algoritmos e a aceleração da computação por placas gráficas são técnicas muito utilizadas na literatura. Este trabalho tem como objetivo desenvolver um novo método paralelo para o cálculo de uma das principais estratégias de preenchimento utilizada na tecnologia de extrusão de material (tipo Modelagem por Fusão e Deposição - FDM), que é a estratégia zigue-zague (*raster*). Através de testes computacionais e de um estudo de aplicação, o método foi avaliado para verificar a sua capacidade em resolver o problema do cálculo do zigue-zague, medir o seu ganho computacional, e avaliar os benefícios da sua utilização na otimização de parâmetros do cálculo da trajetória. Os resultados mostraram que o método paralelo proposto é funcionalmente correto e que gera ganho computacional relevante, sendo até 22 vezes mais rápido que seu equivalente serial. O estudo de aplicação mostrou que a simulação de parâmetros da trajetória resulta em um aumento de até 38% no comprimento médio das retas de *raster*, reduzindo o tempo de fabricação da peça.

Palavras-Chave: manufatura aditiva, planejamento do processo, zigue-zague, algoritmo, paralelismo, gpu, opencl

ABSTRACT

FAUST, Ricardo. Parallel zigzag generation method for additive manufacturing. 2019. 83 p. Master thesis – Programa de Pós-Graduação em Engenharia Mecânica e de Materiais, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

Additive manufacturing (AM) is a manufacturing process by successively adding layers of material. Process planning is a pre-manufacturing computational step in which the three-dimensional CAD (Computer Aided Design) model is processed and converted into information that will be used by the machine. At this stage, the part is sliced into layers and, for each layer, the area to be filled with material is identified. The choice of the filling strategy and the definition of its calculation parameters is of fundamental importance for AM, as it confers different mechanical properties to the object, also impacting the time and cost of manufacturing. The development of AM machines, which allow the manufacture of larger parts with thinner layers, increases the computational demand of the process planning, which in some cases can take hours. Developing more efficient algorithms for process planning opens up a number of possibilities, such as parameter optimization via simulations. Currently, the parallelization of algorithms and the acceleration using graphics processors are techniques widely used in the literature. This work aims to develop a new parallel method for the calculation of one of the main filling strategies used in material extrusion technology (Fused Deposition Modeling - FDM), which is the zigzag (raster) strategy. Through computational testing and an application study, the method was evaluated to verify its ability to solve the zigzag calculation problem, to measure its speedup, and to evaluate the benefits of its use in optimizing the parameters of the trajectory. The results showed that the proposed parallel method is functionally correct and generates relevant computational gain, being up to 22 times faster than its serial counterpart. The application study showed that simulating path parameters results in up to 38% increase in average length of raster lines, reducing manufacturing time.

Keywords: additive manufacturing, process planning, zigzag, toolpath, algorithm, parallel, gpu, opencl

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

3DP	*3D Printing*
AM	Manufatura Aditiva (*Additive Manufacturing*)
CAD	*Computer-Aided Design*
CLIP	*Continuous Liquid Interface Projection*
CNC	*Computer Numeric Control*
CPU	Unidade de Processamento Central (*Central Processing Unit*)
FDM	Modelagem por fusão e deposição (*Fused Deposition Modeling*)
GPU	Unidade de Processamento Gráfico (*Graphics Processing Unit*)
HPC	Computação de alta performance (*High Performance Computing*)
NUFER	Núcleo de Manufatura Aditiva e Ferramental
SIMD	*Single Instrucion Multiple Data*
SL	Estereolitografia (*Stereolitography*)
SLS	*Selective Laser Sintering*
WAAM	*Wire and Arc Manufacturing*

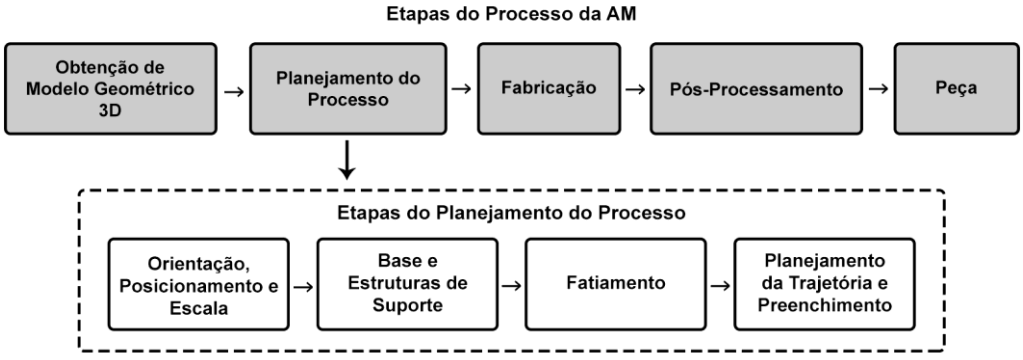
SUMÁRIO

1. INTRODUÇÂO

A manufatura aditiva (AM – *Additive Manufacturing*), também denominada de impressão 3D, é um processo de fabricação que consiste na adição sucessiva de camadas de material e que vem ganhando espaço nos últimos anos devido a sua simplicidade e por permitir a materialização de designs considerados complexos. Em uma etapa anterior a fabricação, chamada de planejamento do processo, um modelo tridimensional, que representa um objeto geralmente através de uma malha triangular definida por pontos e faces, é processado e convertido em instruções que serão utilizadas pela máquina de fabricação (KULKARNI; MARSAN; DUTTA, 2000).

O planejamento do processo pode ser dividido em quatro etapas principais: definição da orientação, posicionamento e escala da peça; cálculo de estruturas de suporte; fatiamento; e planejamento do preenchimento (Figura 1).

Figura 1 – Planejamento do processo da AM

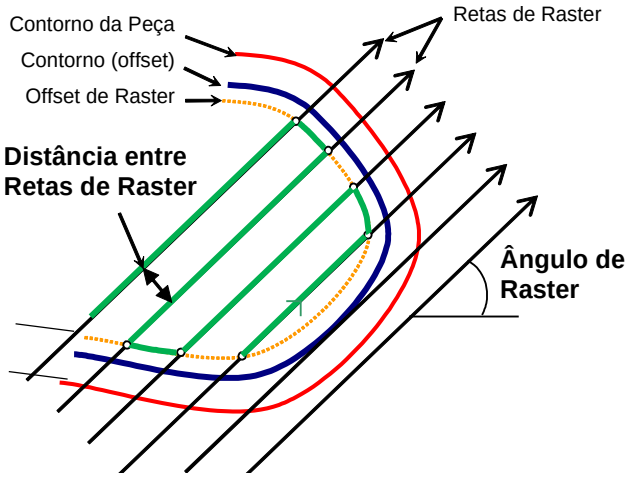


Fonte: Adaptado de Volpato e Da Silva (2017)

Na etapa de planejamento do preenchimento, a estratégia de zigue-zague (normalmente denominada de *raster*) é uma das mais utilizadas e tem como principais parâmetros o ângulo de *raster* e a distância entre as retas paralelas, conforme Figura 2 a seguir.

Com o avanço das tecnologias de manufatura, as máquinas de fabricação permitem resoluções cada vez maiores que necessitam do processamento de uma quantidade crescente de dados. Em alguns casos o planejamento do processo de uma peça pode levar horas (WANG et al., 2017).

Figura 2 – Parâmetros do cálculo do preenchimento zigue-zague



Fonte: Adaptado de Volpato e Foggiatto (2009)

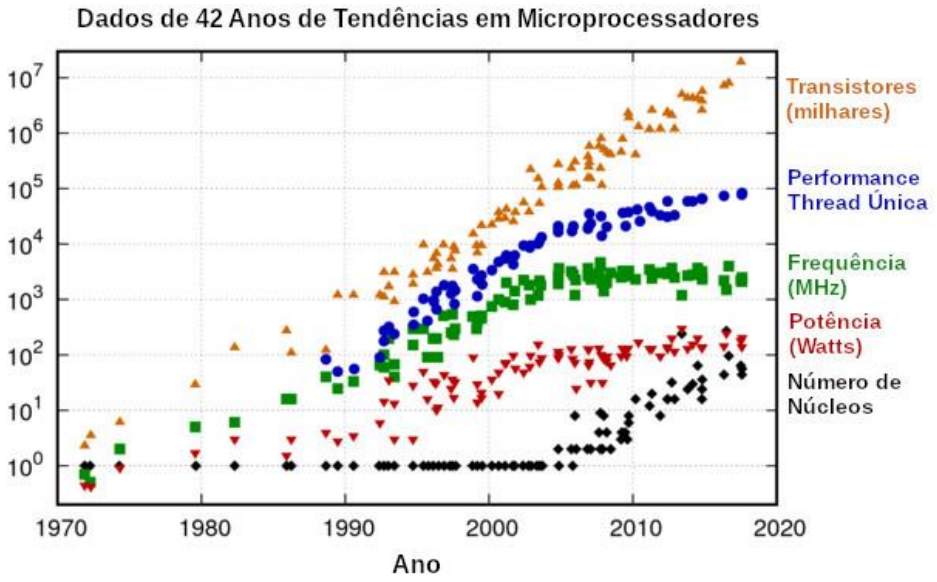
Além disso, para que o processo se torne mais eficiente e resulte em peças de maior qualidade, todas as etapas do planejamento do processo podem ser otimizadas. Uma das técnicas utilizadas para a otimização dos parâmetros é a realização de simulações numéricas, que visam calcular de forma antecipada a interação da máquina de fabricação com a peça, visando, por exemplo, melhorar a qualidade final da peça e reduzir o tempo total de fabricação (ALTINTAS et al., 2014). O número de simulações realizadas tem grande influência nos resultados e torna a demanda computacional ainda maior, sendo limitado principalmente pela capacidade computacional do computador e a eficiência computacional dos métodos utilizados.

Uma das técnicas mais antigas de aceleração computacional envolve a divisão da carga computacional em partes para que sejam executadas de forma paralela por um conjunto de máquinas (*cluster*). No entanto, o alto custo para aquisição das máquinas e manutenção do *cluster* faz com que esta técnica não seja interessante para acelerar o planejamento do processo da AM.

Ao longo das últimas décadas, o aumento na capacidade computacional dos computadores foi obtido principalmente através do aumento da frequência de processamento (*clock rate*). Porém, no início dos anos 2000 a frequência dos processadores atingiu uma limitação física, imposta pelo superaquecimento dos processadores operando em altas frequências e fazendo o *clock rate* se estabelecer na ordem dos 3 GHz (KONOBRYTSKYI, 2013). Para contornar este problema as empresas fabricantes de chips passaram a introduzir núcleos adicionais de

processamento em um mesmo chip, aumentando assim a capacidade computacional total, conforme Figura 3.

Figura 3 – Estagnação da frequência e aumento no número de núcleos de processamento



Fonte: Traduzido de Rupp (2018)

A recente adição de novos núcleos de processamento nos processadores permite que algoritmos sejam executados de forma paralela, com o uso de múltiplas linhas de execução (*threads*), reduzindo o seu tempo computacional. Porém, para usufruir deste benefício, os métodos devem ser projetados com o paralelismo em mente, ou então adaptados para serem paralelizados, e na maioria das vezes este problema não é trivial (KONOBRYTSKYI, 2013).

O desenvolvimento de placas gráficas que permitem o processamento de dados comum (GPGPU - *General Purpose Graphics Processing Unit*) impulsiona ainda mais o desenvolvimento de novos algoritmos paralelos, uma vez que este tipo de hardware adiciona um grande número de núcleos simples de computação com um custo relativamente baixo.

Atualmente, existe uma grande movimentação por parte de pesquisadores, com objetivo de desenvolver novos métodos computacionais que aproveitam ao máximo as tecnologias modernas e se adequam a este novo paradigma da computação.

1.1. OPORTUNIDADE DE PESQUISA

A utilização de métodos computacionais mais rápidos traz diversos benefícios à AM. Novas tecnologias de paralelismo e aceleração por placas gráficas vêm demonstrando grandes reduções no tempo computacional de algoritmos de diversas áreas. Estas tecnologias já são utilizadas em algumas etapas do planejamento do processo da manufatura assistida por computador, e em especial, na AM. No entanto, com base na revisão bibliográfica (detalhada no Capítulo 2), não foi encontrado, até o momento, nenhum estudo investigando a utilização de placas gráficas para a aceleração do cálculo da trajetória zigue-zague 2D.

1.2. OBJETIVOS DA PESQUISA

1.2.1. Objetivo Geral

Desenvolver método computacional para o cálculo da estratégia de preenchimento zigue-zague para manufatura aditiva por extrusão de material, explorando a paralelização de algoritmos e a aceleração do processamento por placas gráficas, e utilizá-lo para uma análise de otimização de um dos parâmetros necessários para a geração da estratégia (o ângulo de *raster*).

1.2.2. Objetivos Específicos

a) Desenvolver um algoritmo paralelo para o cálculo da trajetória zigue-zague.

b) Implementar o algoritmo utilizando biblioteca específica para aceleração em placas de processamento gráfico (GPU).

c) Avaliar a capacidade do método em resolver o problema do cálculo do zigue-zague.

d) Comparar os tempos computacionais do cálculo da trajetória entre o método serial e o método paralelo proposto.

e) Aplicar o método proposto em simulações para identificar o ângulo de *raster* que resulta no menor tempo de fabricação da peça no processo de AM por extrusão de material.

1.3. JUSTIFICATIVA

O desenvolvimento de um método paralelizável de cálculo de trajetória zigue-zague gera diversos benefícios à AM. Dentre os principais motivos estão a melhoria na experiência do usuário durante o planejamento do processo, a possibilidade de realizar um maior número de simulações do processo em mesmo tempo – o que facilita o uso de técnicas de otimização – e a possibilidade, se necessário, de realizar o planejamento do processo durante a fabricação, permitindo correções e ajustes. Por consequência disto, pode se esperar melhores resultados de eficiência e qualidade em processos que utilizam este método para otimizar os parâmetros de fabricação.

1.4. ESTRUTURA DO TRABALHO

O trabalho está dividido em 5 capítulos. No Capítulo 1 é apresentada a área de pesquisa, o contexto do problema a ser pesquisado e a oportunidade de pesquisa, bem como, os objetivos e a justificativa para este trabalho. O Capítulo 2 apresenta, principalmente, uma revisão bibliográfica sobre métodos de cálculo de trajetória zigue-zague, otimização de métodos computacionais utilizando placas de processamento gráfico e suas aplicações no planejamento de processo. O Capítulo 3 apresenta a metodologia e materiais utilizados. O Capítulo 4 apresenta os resultados da avaliação do método proposto, bem como, um estudo de aplicação do método na otimização de parâmetros do cálculo do zigue-zague. Por fim, o Capítulo 5 apresenta as conclusões sobre a pesquisa e indica novas oportunidades de trabalho.

2. REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta uma revisão bibliográfica sobre manufatura assistida por computador e o planejamento do processo da manufatura aditiva. Também foram abordados estudos relacionados ao planejamento do processo da usinagem CNC, em função das similaridades das estratégias utilizadas. Em seguida, apresenta uma revisão dos principais estudos que investigam o cálculo do preenchimento zigue-zague e suas otimizações. Por fim, mostram-se os estudos que exploram o paralelismo para reduzir o tempo computacional de métodos do planejamento do processo da manufatura aditiva, assim como suas principais aplicações.

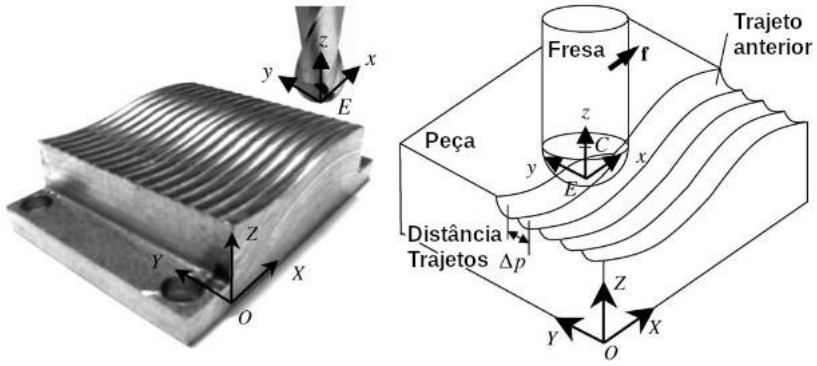
2.1. MANUFATURA ASSISTIDA POR COMPUTADOR

Para se manterem competitivas no cenário global, as empresas passaram a buscar uma maior redução nos custos de produção e aumento na qualidade de seus produtos, através de ciclos de desenvolvimento de produto mais curtos e processos de fabricação automatizados. Neste contexto, surgiu a manufatura assistida por computador (CAM – *Computer-Aided Manufacturing*).

Um dos processos de manufatura mais comuns é a usinagem CNC (*Computer Numeric Control*), também conhecida como manufatura subtrativa. Nesta, realiza-se a fabricação de peças através da remoção sucessiva de material da peça, seja de um bloco ou pré-forma. Na usinagem CNC, uma ferramenta de corte (por exemplo, uma fresa) segue um trajeto controlado por um computador, conhecido como *toolpath*, para obter o formato da peça. Existem várias estratégias de remoção (*toolpath*) de material, sendo uma bastante comum a usinagem em passes paralelos (*raster*), conforme Figura 4 a seguir.

A introdução do CNC revolucionou a indústria de manufatura, tornado possível a fabricação de peças com superfícies curvas e estruturas tridimensionais complexas com reduzida participação do operador da máquina, aumentando a consistência e qualidade da fabricação das peças (HUANG, 2011).

Figura 4 – Movimento de uma fresa usinando o formato de uma peça



Fonte: Traduzido de Fontaine *et al.* (2006)

2.2. MANUFATURA ADITIVA

Nos últimos anos nota-se um crescente interesse por outro tipo de fabricação auxiliada por computador, a manufatura aditiva (AM – *Additive Manufacturing*) ou impressão 3D. Esta é uma tecnologia que, apesar de ser apresentada ao público geral recentemente, existe desde a década de 1980 e tem aplicações em diversas áreas, como por exemplo, na medicina (JARDINI et al., 2014; LEE et al., 2011), na construção civil (CAMACHO et al., 2017; LIM et al., 2012) e no setor aeroespacial (DEHOFF et al., 2013; URIONDO; ESPERON-MIGUEZ; PERINPANAYAGAM, 2015).

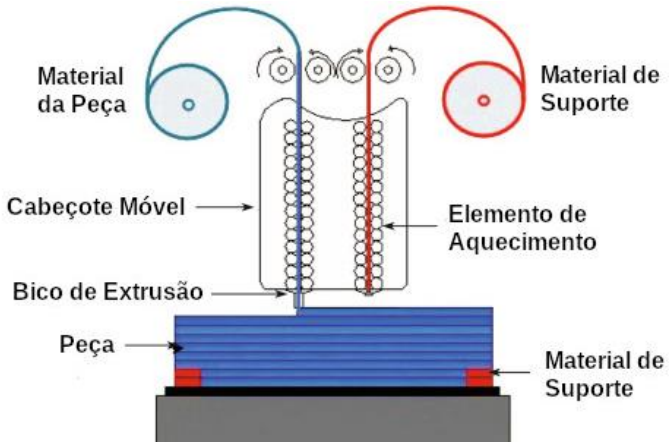
Ao contrário da manufatura subtrativa, neste processo trata-se da produção de peças através da adição sucessiva de camadas de material. Com dados obtidos diretamente do processamento de modelos geométricos tridimensionais CAD (*Computer-Aided Design*), a fabricação pode ser realizada através de diversos princípios de adição (VOLPATO; DA SILVA, 2017), conforme classificados na Figura 5, cada qual com vantagens, limitações e aplicações específicas.

Figura 5 – Princípios das tecnologias de AM

Fonte: Adaptado de Volpato e Da Silva (2017)

Um dos princípios de AM mais conhecidos é o de Extrusão de Material (TURNER; GOLD, 2015), onde se destaca a tecnologia comercial denominada de modelagem por fusão e deposição (FDM – *Fused Deposition Modeling*). Neste princípio de AM, a máquina movimenta um bico de extrusão que deposita material fundido na área de cada camada da peça. O material, geralmente termoplástico, é aquecido a um estado semilíquido e se solidifica após a extrusão, aderindo à camada anterior e à filamentos vizinhos da mesma camada, conforme Figura 6.

Figura 6 – Princípio do processo FDM



Fonte: Traduzido de Mohamed *et al.* (2015)

A grande vantagem das tecnologias que se baseiam na extrusão de material é a simplicidade do princípio de adição, que pode gerar peças de alta resistência a um custo baixo para alguns materiais (WONG; HERNANDEZ, 2012). Esta tecnologia se tornou amplamente conhecida devido à popularização das máquinas de fabricação de baixo custo, que ocupam pequeno espaço e permitem a criação de peças de plásticos com qualidade razoável em ambientes de escritórios e residenciais.

O modo pelo qual é realizado o preenchimento de cada camada é uma característica muito importante de cada princípio de adição. Os princípios de Fusão de Leito de Pó, Deposição Direcionada de Energia e alguns processos da Fotopolimerização em Cuba, representados pelas tecnologias SLS (*Selective Laser Sintering*), WAAM (*Wire and Arc Manufacturing*) e Estereolitografia (SL), respectivamente, necessitam de dados geométricos para o preenchimento que são similares aos da Extrusão de Material.

Nestas tecnologias, a adição do material é determinada pela trajetória de uma espécie de ferramenta (bico de extrusão, feixe de laser e cabeçote de deposição) que

percorre uma plataforma de fabricação seguindo um *toolpath,* na sua maioria através de passos paralelos. O cálculo deste tipo de preenchimento será abordado em detalhes na Seção 2.4.

Além destas já mencionadas, existem outras tecnologias de AM que merecem destaque e possuem ampla utilização, a 3DP (*3D Printing*) por apresentar funcionamento similar a uma impressora convencional e permitir a impressão de peças coloridas (Dimitrov *et al.*, 2006); a Polyjet por permitir a criação de peças com múltiplos materiais, inclusive a composição gradual dentre eles (UDROIU; BRAGA, 2017); e a CLIP (*Continuous Liquid Interface Projection*) por apresentar uma altíssima resolução de camada (JANUSZIEWICZ et al., 2016; TUMBLESTON et al., 2015). Estas tecnologias não estão incluídas no escopo deste estudo por não possuírem um planejamento de preenchimento definido por um trajeto de uma ferramenta.

2.3. PLANEJAMENTO DO PROCESSO

Independente do princípio de adição utilizado, a fabricação de um objeto percorre etapas comuns que servem para converter um modelo computacional CAD em instruções que serão utilizadas pela máquina de fabricação. Este procedimento é chamado de planejamento do processo (KULKARNI; MARSAN; DUTTA, 2000) e é dividido em quatro etapas: a definição da orientação, posicionamento e escala da peça; o cálculo de estruturas que servem de base e suporte para a peça; o fatiamento, que divide a peça em camadas sucessivas; e o cálculo do preenchimento da camada, que identifica a área interna a ser preenchida com o material.

O planejamento do processo é realizado por uma série de métodos computacionais, cada qual com parâmetros específicos, como por exemplo, orientação da peça e espessura da camada. Os parâmetros utilizados nos cálculos possuem grande influência no processo de fabricação afetando a qualidade da superfície, precisão dimensional, tempo de fabricação e propriedades mecânicas da peça (MOHAMED; MASOOD; BHOWMIK, 2015).

Por este motivo, diversos estudos buscam realizar a otimização destes parâmetros, como por exemplo, a simulação de diferentes orientações da peça para reduzir as estruturas de suporte (DAS et al., 2015; STRANO et al., 2013), a otimização do posicionamento das peças para maximizar a quantidade de peças produzidas em um único ciclo (ZHANG; GUPTA; BERNARD, 2016) e a realização de um fatiamento

adaptativo para a redução do efeito escada (PANDEY; REDDY; DHANDE, 2003; TYBERG; HELGE BØHN, 1998).

2.4. ESTRATÉGIAS DE PREENCHIMENTO

Durante o planejamento de processo, a etapa de cálculo de preenchimento da camada tem como objetivo identificar a região ou área na qual o material deve ser depositado, gerando dados que possam ser convertidos em instruções utilizadas pela máquina de fabricação.

Nas tecnologias de extrusão de material, esta área é transformada em uma série de pontos que definem o trajeto cujo bico de extrusão da máquina deve percorrer e depositar o material na forma de filamento. Tal cálculo é realizado para cada uma das camadas a serem fabricadas e segue um percurso específico, denominado estratégia de preenchimento.

O planejamento do preenchimento apresenta diversas similaridades com outras áreas de pesquisa, como por exemplo, planejamento da trajetória da ferramenta na usinagem de cavidades (PARK; CHOI, 2000) e o planejamento da trajetória de robôs (YAKOUBI; LASKRI, 2016). No entanto, apesar de similares e em muitos casos utilizarem processos computacionais muito semelhantes, Tarabanis (2001) e Urbanic *et al.* (2016) citam diversas diferenças entre o planejamento do preenchimento da AM e o planejamento de trajetória da usinagem de cavidades, dentre eles:

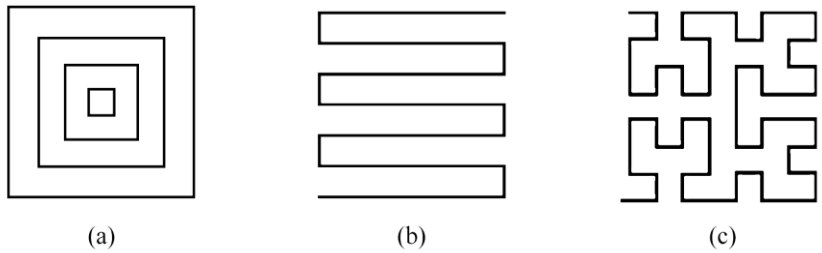
- Os pontos do trajeto utilizado na AM devem ser percorridos apenas uma única vez. Caso contrário a máquina de fabricação poderá colidir com a peça ou então realizar deposição excessiva de material. Na usinagem de cavidades este não é um problema, e a ferramenta pode percorrer livremente regiões já usinadas.
- As regiões de deposição devem ser gravitacionalmente estáveis. Em algumas tecnologias isto significa a necessidade do planejamento de estruturas de suporte.
- O formato do filamento de material e a quantidade material depositado nos momentos de liga / desliga da máquina devem ser levados em consideração para evitar deposição excessiva ou deficiente de material.

- Na AM, existe acúmulo de calor na peça. O planejamento do preenchimento deve levar este fator em conta para evitar empenamentos e distorções. Além disso, o resfriamento desigual de determinadas regiões da peça pode causar diferentes microestruturas no material, conferindo diferentes propriedades mecânicas.
- Dependendo da tecnologia de AM utilizada, propriedades anisotrópicas podem surgir da forma com que o material é adicionado.

2.4.1. Tipos de Estratégias de Preenchimento

O trajeto (*toolpath*) percorrido pela ferramenta pode ser realizado de diversas formas, denominadas de estratégias de preenchimento. As principais estratégias de preenchimento se encaixam em três grupos (LIVESU et al., 2017): curvas de contorno paralelo (*contour-parallel*), curvas de direção paralela (*direction-parallel*) e curvas de Hilbert (*Hilbert curves*), conforme apresentado na Figura 7.

Figura 7 – Estratégias de preenchimento: (a) contorno paralelo, (b) direção paralela, (c) curvas de Hilbert



Fonte: Adaptado de Livesu et al. (2017)

As estratégias de contorno paralelo, apresentadas na Figura 7(a), preenchem o interior de cada camada através de sucessivas curvas paralelas e equidistantes (*offsets*) ao contorno da superfície da peça. Um exemplo de possível melhoria desta estratégia é realizar a conexão do fim de uma curva com o início da próxima, criando assim um formato de espiral. O cálculo do preenchimento deste grupo de estratégias é computacionalmente intensivo, mas tem como grande vantagem a garantia de uma qualidade superior das superfícies externas (perímetro que delimita a peça é contornado integralmente nesta estratégia). Yang *et al.* (2002) apresentam um método para cálculo do preenchimento com curvas equidistantes (*offsets*). Yao (2006) utiliza

a transformação de eixo medial para gerar curvas de preenchimento otimizadas com formato de espiral. Zhao *et al.* (2016) apresentam um método para cálculo de uma curva contínua de preenchimento de contornos paralelos utilizando espirais de Fermat conectadas, que resultam em menor tempo de fabricação e maior qualidade da peça quando comparado a padrões de preenchimento convencionais.

Nas estratégias de direção paralela, demonstradas na Figura 7(b), o preenchimento é feito com uma trajetória que segue linhas paralelas a uma determinada direção. Esta é também comumente denominada de estratégia tipo *raster*. Os dois principais métodos de direção paralela são o zigue e o zigue-zague, sendo o primeiro deles inadequado a tecnologia FDM, devido às inúmeras interrupções que seriam geradas. Além de ser computacionalmente mais simples, esta estratégia gera filamentos (linhas das trajetórias) mais longos, permitindo a máquina atingir sua velocidade máxima de fabricação. O cálculo e otimizações desta estratégia de preenchimento é abordado de forma aprofundada na próxima seção.

As curvas de Hilbert, apresentadas na Figura 7(c), são um grupo de curvas contínuas com formato fractal que preenchem um polígono e foram descritas inicialmente pelo matemático David Hilbert em 1891. Elas têm como grande vantagem a fácil implementação e, em alguns casos, conferem à peça uma resistência mecânica superior (CATCHPOLE-SMITH et al., 2017). Porém, devido ao grande número de mudanças de direção necessárias, o tempo de fabricação é o maior dentre as estratégias de preenchimento de sólidos e é recomendado a apenas algumas aplicações específicas. O cálculo de trajetórias de preenchimento fractal, do tipo curva de Hilbert é estudada por Liu (2004) e Kapil *et al.* (2016).

Além das estratégias mencionadas, existem outros formatos de curvas de preenchimento adequadas ao processo de extrusão de material, como por exemplo, formato sinusoidal (JIN; HE; DU, 2017), cordas de Arquimedes, espiral de octagrama e padrão colmeia (HODGSON, 2018).

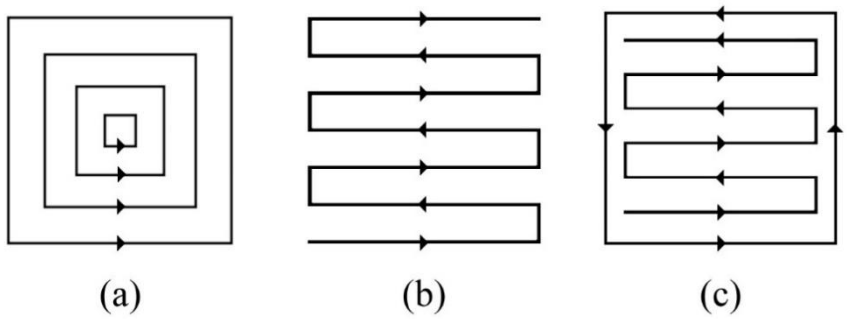
2.4.2. Otimizações do Preenchimento

A escolha da estratégia de preenchimento é de fundamental importância para o planejamento do processo, visto que impacta no tempo de fabricação da peça e em suas propriedades físicas, como por exemplo, resistência mecânica, qualidade da

superfície e peso. Em alguns casos a escolha inadequada da estratégia pode resultar em um tempo de fabricação duas vezes maior (KIM; CHOI, 2002).

Na tecnologia FDM, é comum a combinação da estratégia de Contorno Paralelo e de Direção Paralela (Figura 8), pois a escolha de uma única estratégia de preenchimento pode gerar problemas à peça.

Figura 8 – (a) Contorno paralelo, (b) Zigue-zague, (c) Abordagem hibrida



Fonte: Adaptado de Jin *et al.* (2011)

Neste princípio, se utilizadas sozinhas, as curvas de contorno paralelo apresentam a melhor qualidade de superfície externa, porém podem apresentar um tempo de fabricação maior e resistência mecânica reduzida. Já, se forem empregadas somente as curvas de direção paralela, os problemas da resistência mecânica e tempo de fabricação são resolvidos, porém diminuem a qualidade das superfícies externas da peça (o perímetro externo da peça seria obtido pelas ligações entre as linhas do *raster*, não sendo preenchido totalmente).

Por este motivo, diversos estudos (JIN et al., 2011; TARN; CHEN, 2015) recomendam uma abordagem híbrida, na qual se utiliza uma curva de contorno paralelo na borda da camada, para se obter uma qualidade de superfície externa superior, e curvas de direção paralela, com formato zigue-zague, no preenchimento do interior da camada, para se reduzir o tempo de fabricação e obter uma resistência mecânica maior. Desta forma, o preenchimento é normalmente composto por uma ou mais curvas de contorno e a trajetória de preenchimento zigue-zague no interior da camada.

Outra otimização do planejamento de trajetória muito comum é a redução do comprimento dos trajetos em vazio entre os contornos (reposicionamento ou *jumps*), nos quais o laser ou o cabeçote de deposição não estão realizando o trabalho, através

da reordenação e alteração da direção dos trajetos (TANG; PANG, 2003; VOLPATO et al., 2019).

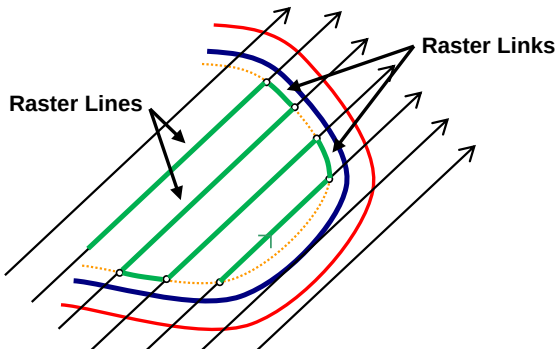
2.5. ESTRATÉGIA ZIGUE-ZAGUE

Nos sistemas CAM (softwares de manufatura auxiliada por computador), o trajeto da ferramenta no formato zigue-zague possui duas variações: o zigue-zague 2D e o zigue-zague 3D. Na primeira, a ferramenta percorre pontos com posição constante no eixo Z (direção de fabricação) e o zigue-zague é delimitado e segue os contornos da camada. Na segunda, comum na usinagem de peças, o zigue-zague é delimitado pelas extremidades do bloco de material e a ferramenta varia sua posição no eixo Z para conferir o formato desejado à peça. Nesta seção será abordado somente o estudo e geração da estratégia zigue-zague 2D por ser a principal estratégia utilizada na AM.

2.5.1. Definição do Problema Relacionado ao Cálculo do Zigue-Zague

Com base em uma lista de contornos de uma camada (polígonos) e parâmetros de cálculo, como por exemplo, o ângulo de *raster* e a distância entre as retas, o algoritmo deve gerar um trajeto com formato zigue-zague que percorre completamente a área definida pelos contornos da camada. Este trajeto é definido por uma série de pontos com altura Z constante, e composto por retas paralelas de *raster* (*raster lines*) e conexões de *raster* (*raster links*), que seguem os contornos da camada (Figura 9).

Figura 9 – Tipos de conexões em um zigue-zague



Fonte: Adaptado de Volpato e Foggiatto (2009)

O cálculo do zigue-zague para contornos simples pode ser realizado de forma trivial, identificando as intersecções entre os contornos e as retas de *raster* e percorrendo os pontos com um formato zigue-zague.

No entanto, normalmente os contornos são complexos, com ângulos entre as arestas dos polígonos maiores que 180º e contornos internos, também chamados de ilhas, que definem regiões vazadas que não devem ser preenchidas. Nestes casos, é necessário a geração de trajetos zigue-zague não contínuos (denominados de subtrajetos) para preencher a camada. Dependendo da complexidade do polígono, pode ser necessário um grande número de subtrajetos (Figura 10).

Figura 10 – Contornos complexos necessitam de mais de um zigue-zague

Fonte: Adaptado de Faust, Volpato e Minetto (2018)

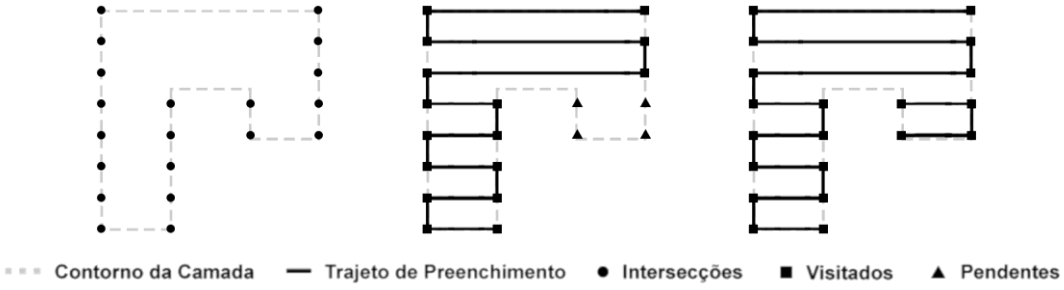
O grande desafio do cálculo do trajeto zigue-zague é gerar todos os subtrajetos necessários para preencher completamente uma camada, sem deixar regiões não preenchidas, independente do formato dos contornos, e a existência de ilhas. Alguns pesquisadores apresentaram soluções para o problema de geração do trajeto zigue-zague, que são descritas nas próximas subseções.

2.5.2. Métodos Iterativos de Retas de *Raster*

Um dos primeiros trabalhos que investiga o cálculo da estratégia de zigue-zague foi realizado por Martin Held em 1991. O método proposto (HELD, 1991), ainda que focado na usinagem, utiliza um algoritmo de varredura (PREPARATA; SHAMOS, 1985) para inicialmente identificar as intersecções entre as retas paralelas (retas de *raster*) e os contornos da peça, criando assim uma rede não-direcionada de pontos. Em seguida, tal rede é percorrida, conectando os pontos e gerando o *toolpath* em zigue-zague. O algoritmo marca cada ponto percorrido como “visitado” e ao encontrar mais de duas intersecções em uma única reta, marca os pontos que não foram

conectados como “pendentes” e adiciona a uma pilha de pendências. Ao concluir um *raster*, o algoritmo reinicia o cálculo selecionando um ponto “pendente” da pilha, até que todas as pendências sejam resolvidas e toda a área da camada seja preenchida (Figura 11).

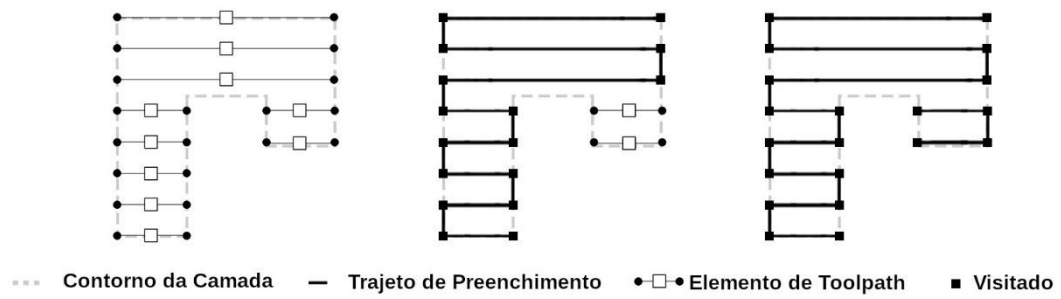
Figura 11 – Método proposto por Held



Fonte: Adaptado de Held (1991)

Seguindo este trabalho, Park e Choi (2000) apresentaram um método de cálculo aprimorado que executa as mesmas etapas básicas de Held (1991). Este identifica, inicialmente, as intersecções entre as retas e o contorno, e realiza a conexão entre os elementos (Figura 12). A grande contribuição deste estudo está em realizar um processamento inicial do contorno identificando os vértices reflexivos – com ângulo interno entre as respectivas arestas maior que 180º – que causam a divisão do *toolpath*. O artigo apresenta um maior nível de detalhamento e definição das etapas do algoritmo e leva em conta um número maior de objetivos, como por exemplo, minimizar o número de segmentos do *toolpath*, maximizar o tamanho médio dos elementos do *toolpath* e identificar a inclinação ótima da direção de referência, conhecida como ângulo de *raster*. A estrutura de dados das intersecções gerada no algoritmo de Park e Choi (2000) é mais robusta e permite a conexão dos elementos do *toolpath* com mais facilidade.

Figura 12 – Método proposto por Park e Choi



Fonte: Adaptado de Park e Choi (2000)

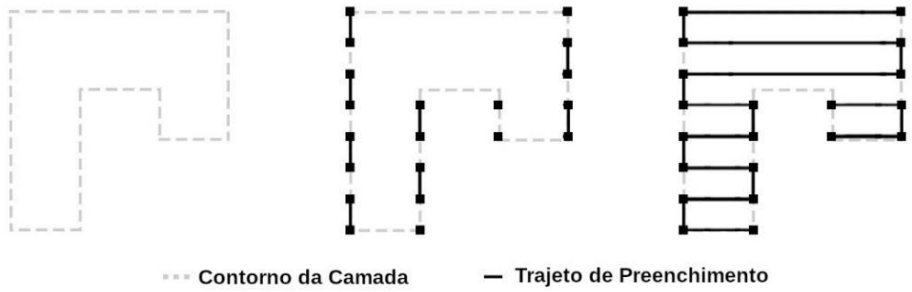
O estudo de Kim (2010) tem como objetivo calcular uma trajetória zigue-zague que resulte em um esforço de corte constante na máquina de usinagem e segue esta mesma abordagem para o cálculo da trajetória zigue-zague.

2.5.3. Métodos Iterativos de Conexões de *Raster*

Projetos de código aberto utilizados em impressoras FDM de baixo custo realizam o cálculo do preenchimento zigue-zague de diferentes formas. Um dos projetos que merece maior destaque é o Slic3r (RANELLUCCI, 2019), compatível com impressoras do tipo RepRap (JONES et al., 2011).

Ao contrário dos métodos iterativos de retas de *raster*, o algoritmo implementado no Slic3r utiliza a lista de segmentos do contorno como base de suas iterações, identificando interseções para gerar primeiramente as conexões de *raster* (*raster links*). Após isso conecta as interseções para gerar as retas de *raster*, conforme Figura 13.

Figura 13 – Método implementado no Sistema Slic3r



Fonte: Adaptado do código-fonte do Slic3r

Este método é conceitualmente mais simples que os métodos propostos por Held (1991) e Park e Choi (2000), pois elimina a necessidade do pré-processamento dos contornos e a criação de uma lista de pendências.

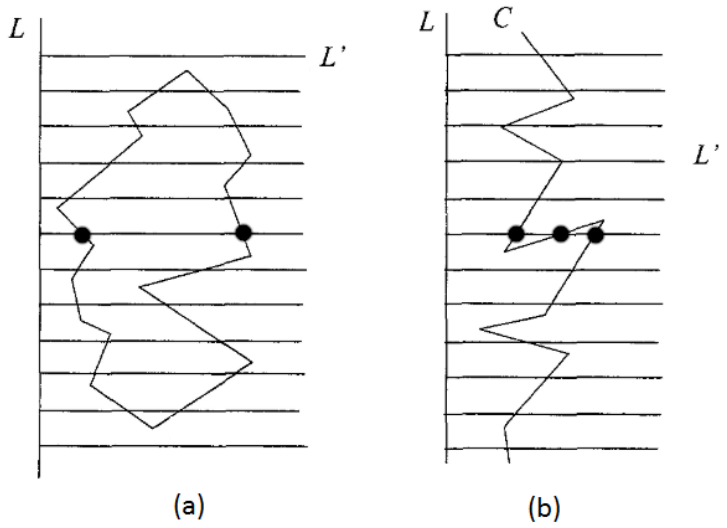
2.5.4. Métodos com Decomposição do Contorno

Para reduzir a complexidade do cálculo da trajetória zigue-zague de contornos complexos, alguns pesquisadores realizaram a decomposição dos contornos em polígonos mais simples, que podem ser preenchidos com um único trajeto zigue-zague.

O estudo de Dwivedi e Kovacevic (2004) realiza a decomposição dos contornos em polígonos monótonos para facilitar o cálculo do zigue-zague ao investigar o cálculo do preenchimento para tecnologias de AM baseadas em soldagem, na qual é necessário gerar um trajeto de deposição fechado e contínuo.

Um polígono é considerado monótono referente a uma determinada direção (L) quando todas as retas perpendiculares a esta direção (L') intersectam o polígono no máximo duas vezes, conforme Figura 14(a). Esta propriedade garante que um polígono monótono pode ser percorrido com um único trajeto de *raster* (único subtrajeto) que pode ser calculado facilmente com o algoritmo trivial. A Figura 14(b) apresenta um exemplo de cadeia de pontos não-monótona na direção (L), pois existem regiões em que as linhas perpendiculares (L') intersectam a cadeia em mais de dois pontos.

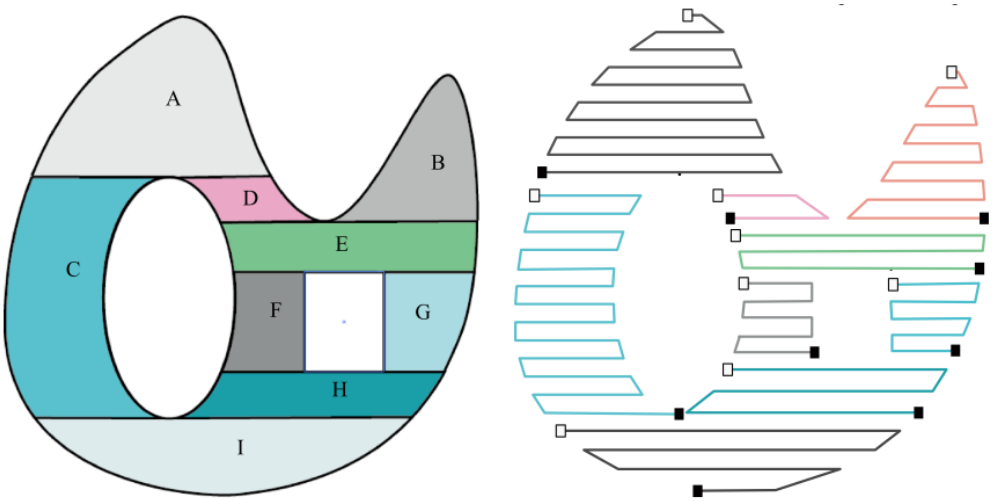
Figura 14 – (a) Polígono monótono, (b) Cadeia não monótona



Fonte: Adaptado de Dwivedi e Kovacevic (2004)

A solução apresentada decompõe o contorno em polígonos monótonos tornando o cálculo do preenchimento muito mais simples. O algoritmo então calcula o *toolpath* para cada um dos polígonos separadamente e ao fim conecta os *toolpaths* gerando um trajeto único. O estudo de Jin *et al.* (2015) segue uma abordagem muito similar ao dividir o contorno em sub-regiões que serão posteriormente conectadas, conforme Figura 15.

Figura 15 – Divisão do contorno em polígonos monótonos



Fonte: Jin et al. (2015)

Ding et al. (2014) generalizam a decomposição de contornos proposto por Dwivedi e Kovacevic ao estudar o cálculo de *toolpath* para a tecnologia WAAM (*Wire and Arc Additive Manufacturing*). Neste estudo é realizada a divisão do contorno em polígonos convexos, o autor aplica otimizações específicas a cada um dos polígonos, gerando um *toolpath* otimizado para cada região do contorno, demonstrando assim, o grande benefício da decomposição do contorno (Figura 16).

Polígonos convexos são aqueles em que a soma de seus ângulos internos é exatamente igual a 180º. Uma consequência importante desta propriedade é que os polígonos convexos são considerados monótonos para todas as direções, ou seja, necessitam de um único trajeto zigue-zague para seu preenchimento independente da direção das retas de *raster*.

Figura 16 – Decomposição em polígonos convexos

Fonte: Adaptado de Ding et al. (2014)

2.5.5. Otimizações e Aplicações Específicas da Estratégia Zigue-Zague

Outros estudos apresentam métodos de cálculo similares aos métodos anteriores ou buscam otimizar o trajeto para uma tecnologia específica de manufatura (JIN et al., 2011; SELVARAJ; RADHAKRISHNAN, 2006).

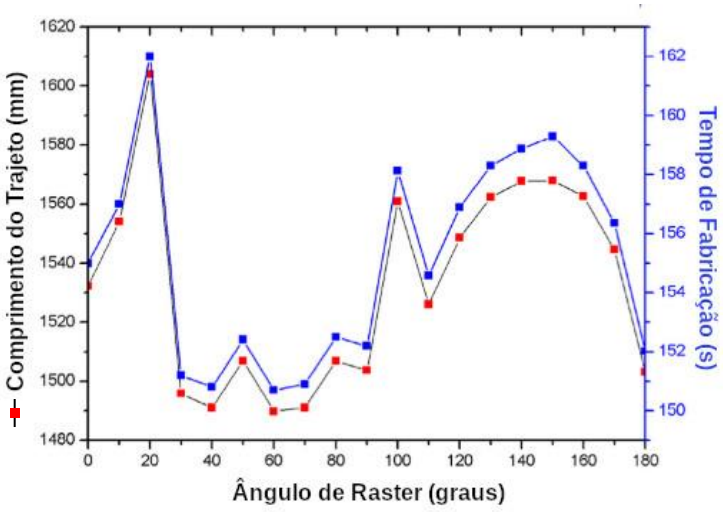
Rajan, Srinivasan e Tarabanis (2001) apresentam um método para calcular a direção de preenchimento ótima, minimizando o número de linhas de *raster* e por consequência diminuindo o tempo de fabricação.

O estudo de Qiu e Langrana (2002) apresenta um método para cálculo e otimização do preenchimento com o objetivo de minimizar os vazios presentes entre as curvas de contorno e as curvas de zigue-zague, fazendo alterações na distância entre retas em diferentes seções da camada.

Jin *et al.* (2017) apresentam uma revisão do estado-da-arte na geração de trajetórias fechadas. Os métodos apresentados permitem a fabricação de peças com contornos conectados através um único *toolpath* contínuo e fechado.

Jin *et al.* (2014) buscam minimizar vazios no preenchimento e o tempo de fabricação de uma peça através da otimização de parâmetros do planejamento do preenchimento. O método calcula de forma iterativa o preenchimento zigue-zague para diferentes valores de ângulo de *raster*, selecionando aquele com melhores resultados (Figura 17). Esta abordagem é utilizada de forma similar no estudo de Habib e Khoda (2017) para identificar o ângulo de *raster* que resulta no menor tempo de fabricação.

Figura 17 – Otimização de parâmetros do zigue-zague através de simulações



Fonte: Traduzido de Jin et al. (2014)

2.6. COMPUTAÇÃO DE ALTA PERFORMANCE

A computação de alta performance (*High Performance Computing* – HPC) é um conjunto de técnicas que visa acelerar a solução de problemas computacionais, sendo que uma das principais técnicas é a paralelização de algoritmos, que divide uma tarefa para ser executada por múltiplas unidades de processamento simultaneamente.

A principal métrica utilizada para avaliar a performance de um algoritmo paralelo é o seu ganho computacional (*speedup*), que representa o quão mais rápido é o algoritmo paralelo quando comparado ao seu equivalente serial, e é calculado pela razão entre os tempos computacionais dos algoritmos paralelo e serial (NAVARRO; HITSCHFELD-KAHLER; MATEU, 2014).

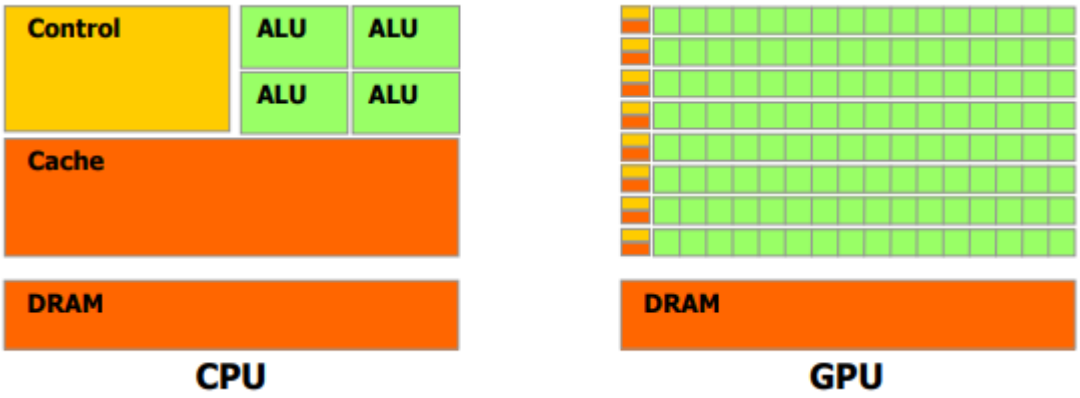
Com o desenvolvimento das redes de computadores e da internet, problemas anteriormente considerados inviáveis para serem em um único computador passaram a ser resolvidos por conjuntos de computadores agindo como uma única máquina virtual (*clusters*).

Já nas aplicações de uso pessoal, o paralelismo foi obtido inicialmente com a introdução do conjunto de instruções vetoriais do tipo SIMD (*Single Instrucion Multiple Data*) que permitem a aceleração de cálculos matemáticos realizando uma mesma operação sobre múltiplos dados em um único ciclo de processamento (HERN, 2013). O reduzido conjunto de instruções não é suficiente para a implementação de algoritmos complexos e, por este motivo, o seu uso foi limitado a aplicações específicas, como por exemplo, o processamento de áudio e vídeo.

Com a adição de novos núcleos de processamento nos processadores foi possível executar o conjunto completo de instruções do processador de forma paralela, permitindo que tarefas complexas fossem executadas simultaneamente por meio de diferentes linhas de execução (*threads*). Ainda assim, a baixa quantidade de núcleos de processamento limitou os ganhos computacionais**.**

Uma forma emergente de aceleração da velocidade computacional de algoritmos é a utilização de processadores gráficos (*GPU – Graphics Processing Unit*). Ao contrário dos processadores convencionais (CPU) que possuem poucos núcleos altamente complexos, ideais para sistemas multitarefa, os GPUs possuem centenas de núcleos simples (representados em verde na Figura 18), que são especializados em realizar cálculos matemáticos e executar em paralelo de uma única tarefa.

Figura 18 – Diferenças nas arquiteturas do CPU e GPU



Fonte: Adaptado de Hsieh e Chu (2011)

O relativo baixo custo de aquisição (quando comparado à montagem de um *cluster*) e o alto nível de paralelismo (quando comparado ao uso de *threads* e instruções vetoriais), faz com que as GPUs sejam muito atraentes para a computação paralela de uso pessoal.

No entanto, o paradigma da computação paralela, definido pela independência de cálculo entre os elementos de computação, e as limitações impostas pela arquitetura das GPUs fazem com que a conversão de um método serial em um método paralelo que utiliza aceleração gráfica não seja trivial, sendo que na maioria das vezes o método tem de ser repensado do zero para se adequar à esta nova estrutura de processamento (KONOBRYTSKYI, 2013).

Na usinagem, o uso de GPUs para acelerar métodos computacionais do planejamento do processo é utilizado há mais de uma década. Uma das aplicações mais estudadas é a aceleração na geração de *toolpaths*. Morell-Giménez *et al.* (2013), Lee *et al.* (2013) e Tarbutton *et al.* (2013) utilizam GPU para o cálculo de *toolpaths* de contornos paralelos no formato espiral. Bi *et al.* (2010), Hsieh *et al.* (2011) e Wang *et al.* (2018) buscam acelerar a geração de *toolpaths* sem colisões para máquinas de 5 eixos.

As GPUs também são utilizadas na usinagem para otimização do planejamento do processo, como por exemplo, na otimização da velocidade de avanço (CHAN; WANG, 2015), na otimização da escolha da ferramenta (BALABOKHIN, 2016) e na usinagem virtual através de simulações do processo (INUI; UMEZU, 2016; KONOBRYTSKYI, 2013; TUKORA, B.; SZALAY, 2010; TUKORA, BALÁZS; SZALAY, 2012).

Já na AM a aplicação desta tecnologia é mais recente e é utilizada para otimizar a orientação de fabricação (GARCÍA GALICIA; BENES, 2018; HUANG et al., 2017; LI et al., 2019), acelerar o fatiamento (LEFEBVRE, 2013; WANG et al., 2017; ZHANG et al., 2017b), realizar otimizações topológicas do preenchimento interno (SURESH, 2014; ZHAO; LI; LIU, 2017), simular o processo de fabricação (HUANG et al., 2018; UENG; CHEN; JEN, 2018; ZHANG et al., 2017a) ou monitorar o processo de fabricação (SAMPSON et al., 2017).

Poucos estudos exploram o paralelismo no planejamento da trajetória da ferramenta, sendo que a grande maioria estuda apenas as estratégias de contorno paralelo. O estudo que mais se assemelha a este trabalho é o de Konobrytskyi (2013), que investiga métodos que utilizam a GPU para acelerar o cálculo de *toolpaths* e de simulações da usinagem CNC de 3 e 5 eixos. Dentre os métodos de cálculo de *toolpath* apresentados estão o de contornos paralelos 2D e o zigue-zague 3D. Ao analisar o preenchimento zigue-zague, Konobrytskyi escolhe paralelizar apenas o cálculo do zigue-zague 3D por considerar este muito mais computacionalmente intensivo do que o cálculo do zigue-zague 2D. O algoritmo apresentado calcula de forma serial o zigue-zague 2D, e em seguida, calcula de forma paralela, para cada ponto do trajeto, a distância entre a superfície da peça e a superfície da ferramenta de corte, ajustando assim a altura da ferramenta.

2.7. CONSIDERAÇÕES SOBRE A REVISÃO

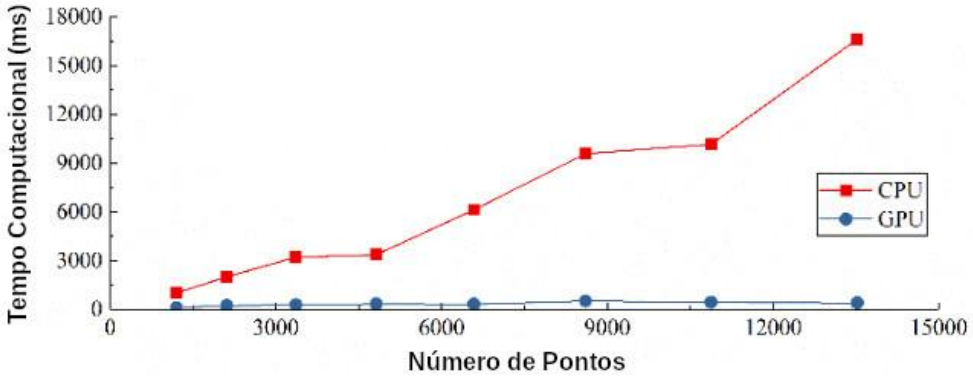
A otimização do planejamento do processo tem um enorme impacto positivo nos principais objetivos da fabricação de peças: redução do tempo e custo de fabricação; e o aumento na qualidade das peças.

A redução do tempo computacional torna viável o cálculo de um maior número de simulações de parâmetros e variações do processo de fabricação (permitindo a simulação de vários casos), identificando os parâmetros que geram os melhores resultados. Além de permitir que processos de fabricação monitorados possam fazer rápidos ajustes no preenchimento para evitar defeitos de fabricação.

A evolução dos processadores segue no caminho da adição de núcleos adicionais de processamento. Fica evidente que o desenvolvimento de algoritmos altamente paralelizáveis é necessário para que se utilize por completo a capacidade computacional dos computadores modernos.

A revisão bibliográfica aponta que a utilização de algoritmos paralelos resulta em ganhos computacionais de até 250 vezes na aceleração do planejamento do processo da AM e da usinagem, sendo mais comuns ganhos computacionais no intervalo de 20 a 60 vezes, conforme Figura 19.

Figura 19 – Comparação do tempo computacional entre CPU e GPU



Fonte: Traduzido de Wang, Luo e Zhang (2018)

Apesar do cálculo e otimização da estratégia zigue-zague 2D ser uma área de pesquisa muito ativa, existe pouca literatura referente à busca de algoritmos mais rápidos e computacionalmente eficientes para a solução deste problema. Um possível motivo para isso é de que o planejamento do processo é geralmente realizado uma única vez em um momento anterior a fabricação, no qual o tempo de processamento tem pouco influência no tempo total da fabricação. No entanto, existem situações em que o planejamento do processo pode ser executado de forma iterativa ou então, dependendo da complexidade do modelo geométrico, demorar um longo tempo para ser processado, tornando custosa a realização de ajustes.

Diversos estudos abordaram o cálculo do preenchimento zigue-zague e diversos outros pesquisadores utilizaram o paralelismo oferecido por placas gráficas para acelerar o planejamento do processo. No entanto, nenhum estudo buscou aplicar esta técnica de otimização computacional no cálculo da trajetória zigue-zague 2D, muito utilizada na AM. Sendo assim, uma oportunidade de pesquisa foi identificada no desenvolvimento de um método paralelo do cálculo da trajetória zigue-zague 2D, que possa ser acelerado pelo uso de placas gráficas, para planejamento do processo na AM e usinagem de cavidades.

3. METODOLOGIA DA PESQUISA

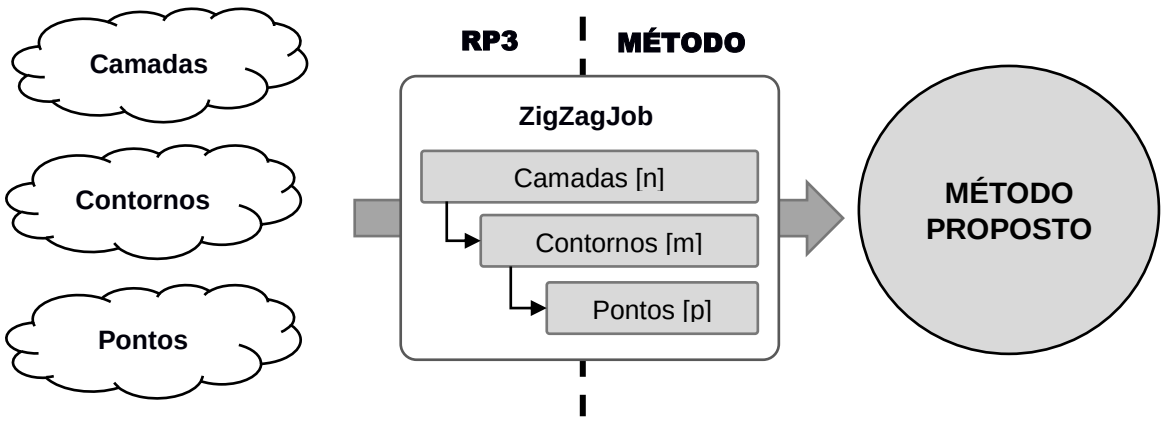
Este capítulo apresenta a metodologia utilizada no desenvolvimento da pesquisa. Na primeira seção é proposto um método para o cálculo da trajetória zigue-zague utilizando uma GPU como forma de aceleração. Em seguida são apresentados os testes que foram realizados para avaliar o método. Por fim, é descrito um estudo de aplicação do método para verificar seu impacto em peças produzidas por manufatura aditiva.

3.1. MÉTODO PARALELO DE CÁLCULO DA TRAJETÓRIA ZIGUE-ZAGUE

O método desenvolvido tem como objetivo gerar a trajetória zigue-zague que preenche completamente os contornos das camadas obtidas do fatiamento de um modelo tridimensional e utiliza o alto paralelismo proporcionado por placas de processamento gráfico (GPU) para acelerar a computação da trajetória. Esta trajetória é também muito conhecida como trajetória tipo *raster*.

A implementação foi realizada como um módulo do sistema de planejamento de processo RP3, desenvolvido no NUFER (Núcleo de Manufatura Aditiva e Ferramental) da UTFPR. Antes da execução do método, as informações resultantes do fatiamento (camadas, contornos e pontos) são convertidas em uma nova estrutura de interface, chamada *ZigZagJob*, que contém apenas os dados relevantes ao cálculo do preenchimento, como por exemplo, pontos de contorno e parâmetros do *raster*. Em seguida, o RP3 invoca o método de cálculo paralelo passando como argumento a estrutura criada, conforme Figura 20 a seguir.

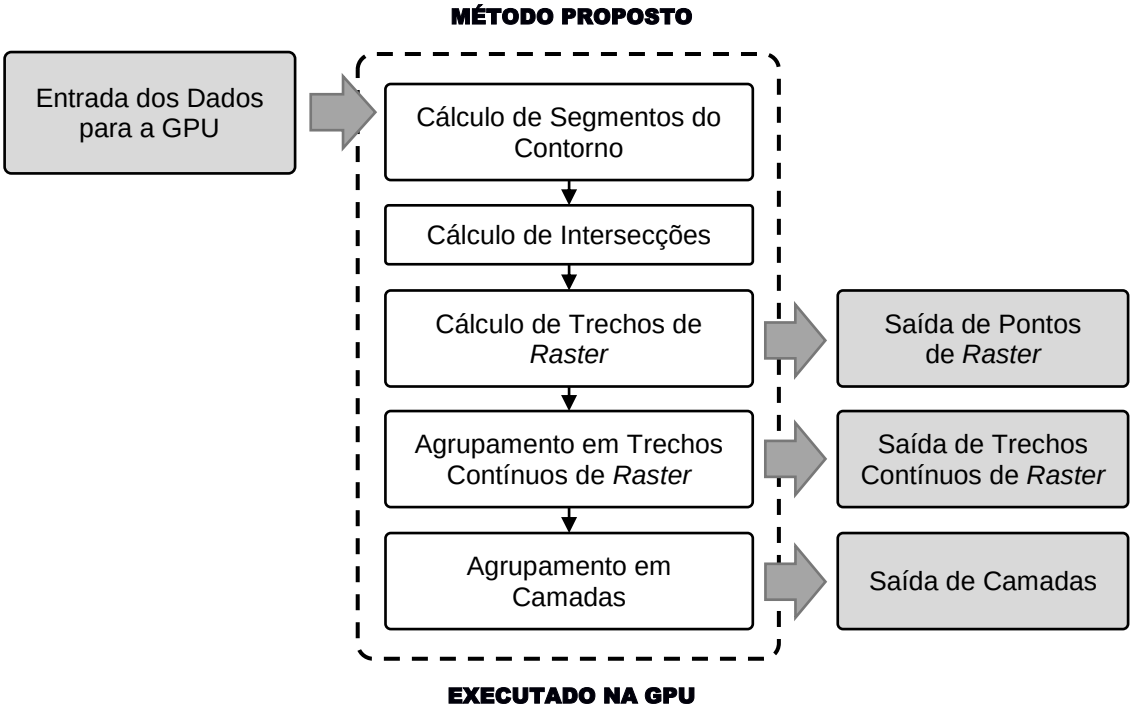
Figura 20 – Interface de dados RP3 / método proposto



Fonte: Autoria Própria

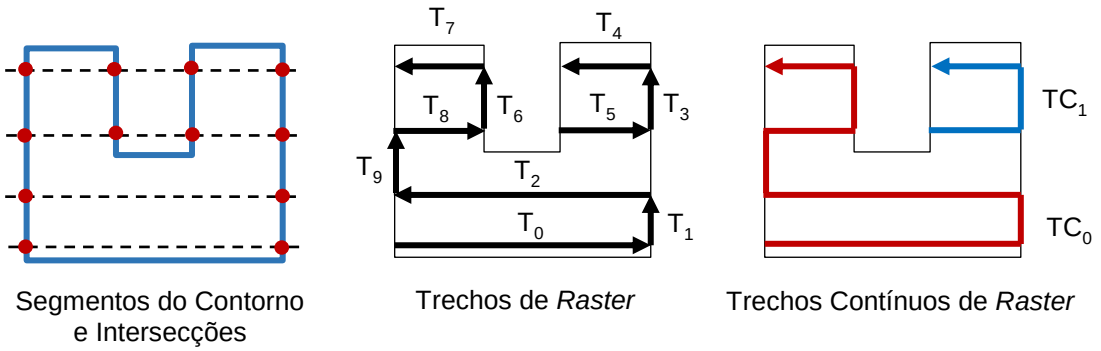
O cálculo do zigue-zague é realizado através de cinco etapas que são realizadas de forma sequencial e são: cálculo de segmentos do contorno, cálculo de intersecções entre os segmentos do contorno e retas de *raster*, cálculo dos trechos de *raster*, agrupamento em trechos contínuos de *raster* e agrupamento em camadas, conforme Figura 21 e Figura 22.

Figura 21 – Etapas do cálculo da trajetória zigue-zague



Fonte: Autoria Própria

Figura 22 – Elementos da trajetória zigue-zague



Fonte: Autoria Própria

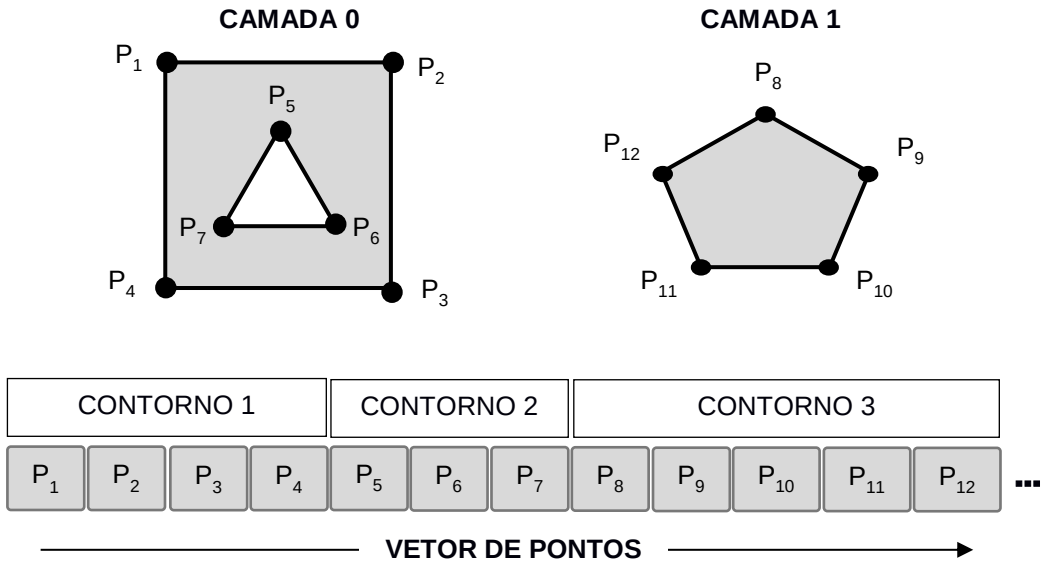
Cada uma destas etapas é dividida em subetapas que executam métodos paralelos para gerar de forma gradativa a trajetória zigue-zague e que serão descritas de forma detalhada nas seções a seguir.

3.1.1. Entrada dos Dados para a GPU

Ao receber a chamada com os dados do RP3, o método processa o *ZigZagJob* e reorganiza as informações em três novos vetores de dados que são copiados para a GPU e servem como entrada dos algoritmos paralelos.

O primeiro vetor é formado pela concatenação de todos os pontos dos contornos da peça, independente de qual contorno ou camada em que o ponto se encontra, conforme exemplo apresentado na Figura 23. Para efeito do programa, a camada “0” significa a primeira camada da peça.

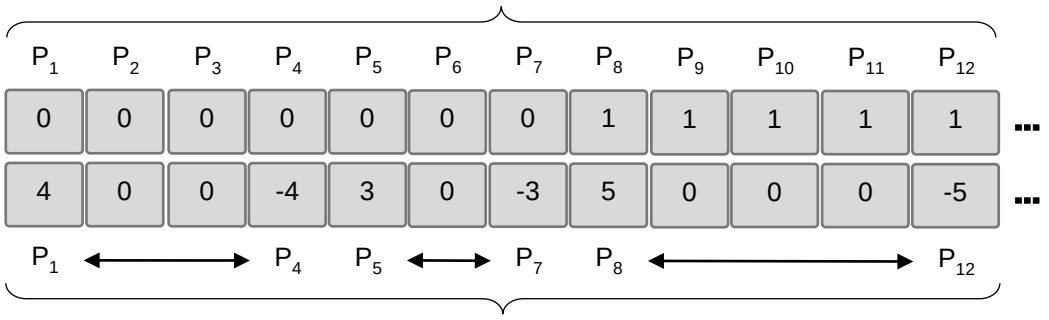
Figura 23 – Vetor de pontos dos contornos



Fonte: Autoria Própria

Ao longo deste processo, é criado um segundo vetor que relaciona cada ponto a uma estrutura que combina duas informações adicionais, conforme Figura 24 a seguir. A primeira informação se refere a qual camada cada ponto pertence (índice de camada). A segunda informação é preenchida apenas para o primeiro e o último ponto do contorno, com o número de pontos de cada contorno. Esta informação é utilizada pelo método durante o cálculo da trajetória para percorrer de forma fechada os pontos de cada contorno, identificando os seus pontos de início e fim. Este número de pontos contém ainda o índice positivo para identificar o início do contorno e um índice negativo para identificar o fim do contorno.

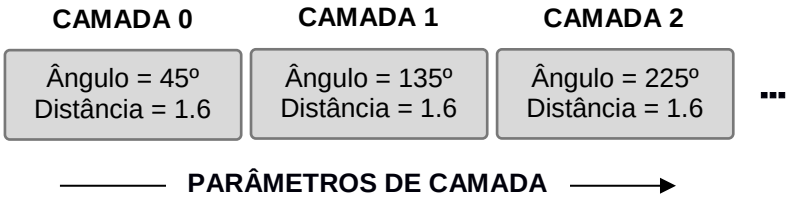
Figura 24 – Vetor de informações adicionais dos pontos



Fonte: Autoria Própria

Por fim, é criado um último vetor que armazena os parâmetros de cada camada (ângulo de *raster* e distância entre retas), conforme Figura 25. Estes três vetores são copiados para a memória da GPU e servem como entrada do método paralelo de cálculo.

Figura 25 – Vetor de parâmetros das camadas

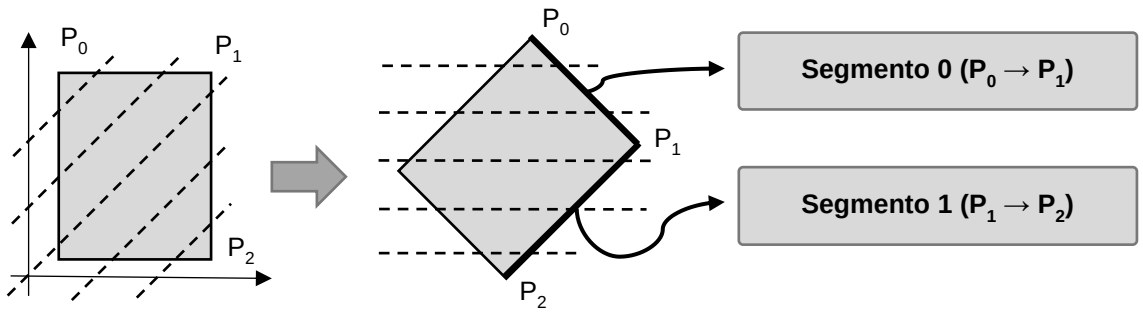


Fonte: Autoria Própria

3.1.2. Etapa de Cálculo de Segmentos do Contorno

A primeira etapa tem como principal objetivo calcular informações sobre os segmentos de reta formados por dois pontos consecutivos nos contornos da camada (*segments*), conforme Figura 26. Esta etapa é realizada em duas subetapas denominadas cálculo de segmentos do contorno (*calcSegments*) e scan dos segmentos do contorno (*scanSegments*), que serão detalhadas nesta seção.

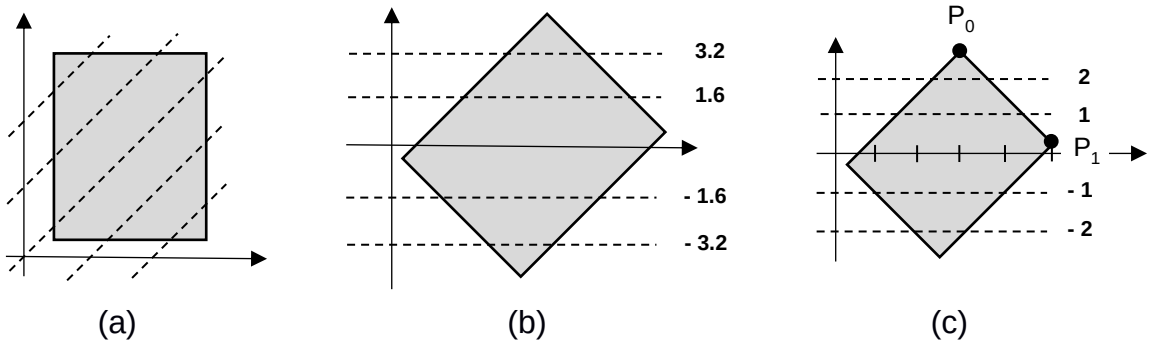
Figura 26 – Etapa de cálculo de segmentos do contorno



Fonte: Autoria Própria

Para simplificar o cálculo das intersecções que será realizado na próxima etapa, os pontos do contorno passam por transformações lineares de rotação e escala. Antes do cálculo dos coeficientes paramétricos (linear e angular) da reta que contém o segmento do contorno, cada ponto é rotacionado por um ângulo igual ao ângulo de *raster* com pivô de rotação na origem e aplicado uma escala igual a distância entre retas. Neste novo domínio, o cálculo das intersecções se torna muito mais fácil, pois as retas de *raster* se tornam paralelas ao eixo horizontal e estão situadas nos números inteiros do eixo vertical, conforme Figura 27. No contexto do método, a coordenada vertical da reta é chamada de índice da reta.

Figura 27 – (a) Peça original, (b) Peça rotacionada, (c) Peça com escala

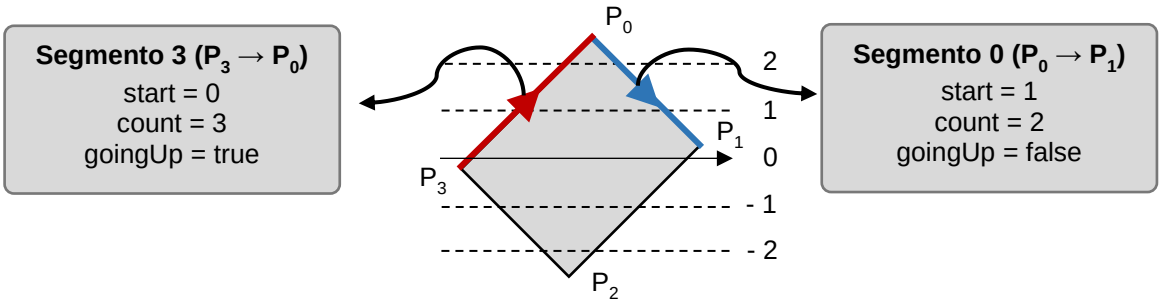


Fonte: Autoria Própria

Neste cenário, o cálculo das intersecções se torna trivial. Por exemplo, considerando a Figura 27 (c), em que todas as retas de *raster* são paralelas ao eixo horizontal e estão posicionados em números inteiros no eixo vertical, e considerando o segmento formado pelos pontos P_0 (3, 2.8) e P_1 (5, 0.2), pode-se identificar facilmente que as retas que intersectam o segmento são aquelas com coordenada vertical de número inteiro entre 0.2 e 2.8, ou seja, retas com ordenada igual a 1 e 2.

Além dos coeficientes paramétricos da reta, são calculadas outras informações importantes para o cálculo das intersecções, que são, o menor índice de reta que intersecta o segmento (*start*), o número de retas intersectantes (*count*) e se o segmento rotacionado possui uma direção vertical positiva ou negativa (*goingUp*), conforme exemplificado na Figura 28 e descrito no Algoritmo 1 a seguir.

Figura 28 – Exemplos de segmentos e suas informações importantes



Fonte: Autoria Própria

Algoritmo 1 – Cálculo de segmentos do contorno (*calcSegments*)

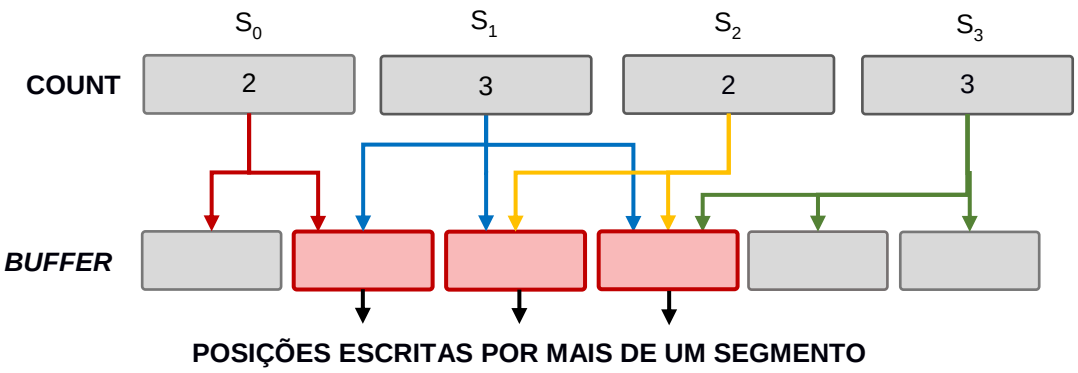
```
 1: executar em paralelo para cada point
 2:   id ← índice do point
 3:
 4:   P₁ ← points[id]                                  P₁ (x₁, y₁)
 5:   id ← próximoPonto(points, id, direção)
 6:   P₂ ← points[id]                                  P₂ (x₂, y₂)
 7:
 8:   aplicaRotação (P₁, P₂)
 9:   aplicaEscala (P₁, P₂)
10:
11:   segment.a = (y₂ - y₁) / (x₂ - x₁)                coefAng
12:   segment.b = y₁ - (segment.a - x₁)                coefLin
13:
14:   start ← floor(y₁)
15:   end ← floor(y₂)
16:
17:   segment.start ← min(start, end) + 1
18:   segment.count ← abs(end - start)
19:   segment.goingUp ← (end >= start)
20:
21:   segments[id] ← segment
22: fim
```

Fonte: Autoria Própria

A função “próximoPonto” utilizada na linha 5 do Algoritmo 1 retorna o índice do próximo ponto de um contorno, levando em consideração a conexão entre o último e o primeiro ponto do contorno, e para isso, utiliza o tamanho do contorno salvo no vetor de informações adicionais dos pontos (Figura 24).

Antes de realizar o cálculo das intersecções, é necessário fazer a divisão prévia do *buffer* em que serão salvas as intersecções. Diferente de algoritmos seriais, em algoritmos paralelos os elementos que estão sendo processados geralmente não têm acesso ao resultado da computação de elementos anteriores e por este motivo não sabem o quanto do *buffer* de saída já foi utilizado. A escrita do *buffer* de saída sem a devida pré-divisão pode fazer com que diferentes elementos de entrada escrevam nas mesmas posições de saída, conforme esquematizado na Figura 29.

Figura 29 – Problema da divisão do *buffer*



Fonte: Autoria Própria

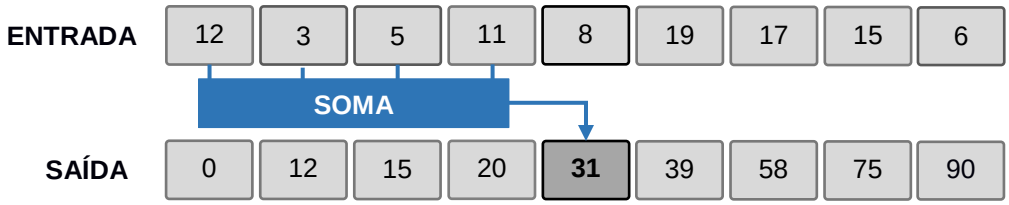
A solução para este problema é realizar a divisão prévia do *buffer* em que serão escritos os dados e esta pode ser obtida através de um algoritmo de soma cumulativa, que pertence a um grupo de algoritmos chamado de *scan*.

O *additive scan*, também chamado de *prefix sum*, é um algoritmo muito simples quando executado de forma serial e sua implementação é trivial (HARRIS; SENGUPTA; OWENS, 2007). Neste algoritmo, cada elemento do vetor de saída representa a soma cumulativa de todos os elementos anteriores do vetor de entrada, conforme Figura 30.

No entanto, a execução do algoritmo *scan* serial em processadores paralelos é altamente ineficiente, sendo necessária a implementação de um algoritmo paralelo de *scan*. Na literatura, o *scan* paralelo é muito estudado e existem diversas soluções

para este problema, cada qual com suas vantagens, limitações e aplicações adequadas (BLELLOCH, 1990).

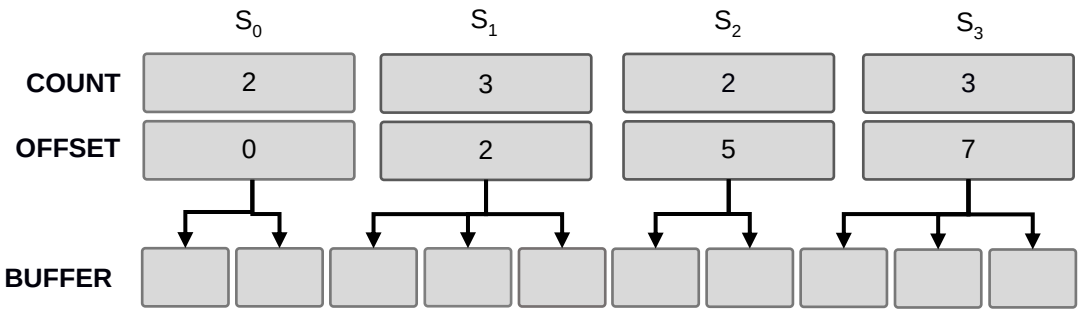
Figura 30 – Exemplo de a*dditive scan*



Fonte: Autoria Própria

Desta forma, na subetapa *scanSegments* é utilizado um algoritmo paralelo de *scan* baseado em Blelloch (1990) e descrito por Harris, Sengupta e Owens (2007) para computar a soma cumulativa do número de intersecções de cada segmento, preparando e dividindo o *buffer* de intersecções dentre todos os segmentos. O resultado desta soma é salvo na variável *offset* de cada segmento, que em conjunto com o número de intersecções (*count*) define os endereços no *buffer* de intersecções que serão escritos por cada segmento, conforme exemplo ilustrado na Figura 31.

Figura 31 – Divisão do *buffer* de intersecções

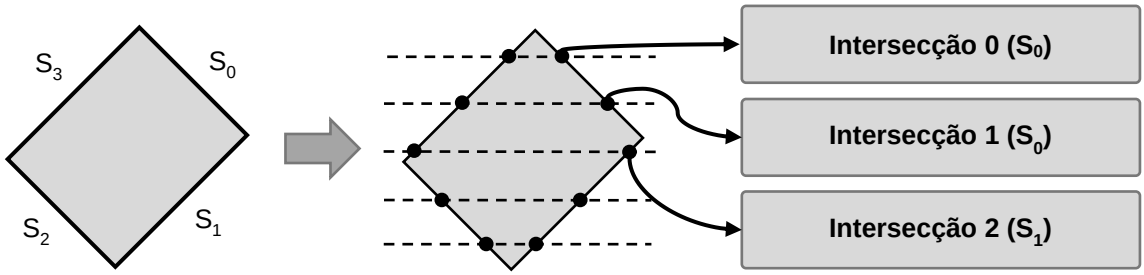


Fonte: Autoria Própria

3.1.3. Etapa de Cálculo de Intersecções

Com base nas informações geradas na etapa anterior, são calculadas as intersecções entre as retas de *raster* e os segmentos do contorno (*intersections*), conforme Figura 32, que são salvas e ordenadas em um vetor de intersecções. Esta etapa é dividida em duas subetapas denominadas de cálculo das intersecções (*calcIntersections*) e ordenação das intersecções (*sortIntersections*), que serão detalhadas nesta seção.

Figura 32 – Etapa de cálculo de intersecções



Fonte: Autoria Própria

Na subetapa *calcIntersections*, são utilizadas as informações *start* e *count* (Figura 28) de cada segmento para executar em paralelo laços de cálculo de intersecções. Com a transformação de eixo realizada na primeira etapa, as retas de *raster* estão paralelas ao eixo horizontal, e o cálculo da intersecção pode ser feito inserindo a coordenada y da reta na equação paramétrica do segmento do contorno, obtendo a coordenada x da intersecção. Após o cálculo destas coordenadas, a transformação linear de escala e rotação é removida para obter as coordenadas da intersecção no domínio original da peça. Um pseudocódigo do cálculo de intersecções pode ser visto no Algoritmo 2.

Além do cálculo das coordenadas da intersecção, nesta subetapa são calculadas e salvas em uma variável (*flags*) informações auxiliares. A primeira *flag* (END – linhas 15 e 16 do Algoritmo 2) é marcada quando a intersecção se encontra na última reta que intersecta o segmento, isto é, na reta intersectante de maior índice (intersecções 9 e 0 da Figura 33). A segunda *flag* (HORIZONTAL – linhas 18 e 19 do Algoritmo 2) é marcada quando o índice da reta é par e o segmento está subindo (variável *goingUp* da Figura 28 – intersecção 9 da Figura 33), ou, quando o índice da reta é ímpar e o segmento está descendo (intersecção 1 da Figura 33). Esta *flag* faz com que a trajetória intercale movimentos horizontais e verticais, formando o zigue-zague.

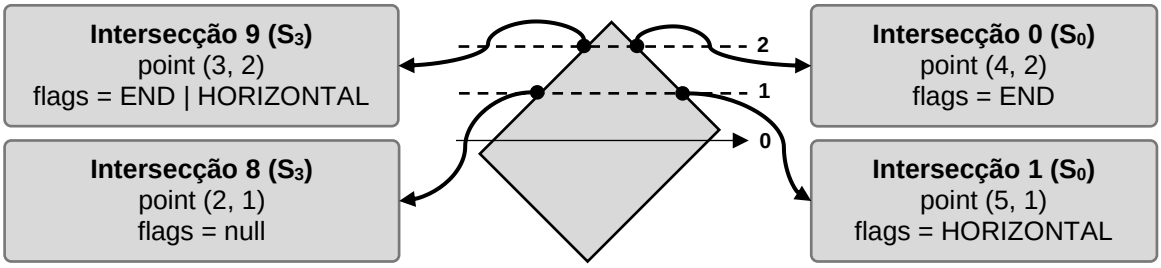
Algoritmo 2 – Cálculo de intersecções (*calcIntersections*)

```
1:  executar em paralelo para cada segment
2:    start ← segment.start
3:    end ← start + segment.count
4:    offset ← segment.offset
5:
```

```
 6:   para i ← start até end
 7:     x ← (i - segment.b) / segment.a
 8:     y ← i
 9:
10:     intersection.point ← (x, y)
11:     removeEscala (x, y)
12:     removeRotação (x, y)
13:     intersection.intersection ← (x, y)
14:
15:     se i + 1 == end então
16:       intersection.flags seta END
17:
18:     se (i % 2 == 0) ^ !segment.goingUp então
19:       intersection.flags seta HORIZONTAL
20:
21:     intersections[offset] ← intersection
22:     offset ← offset + 1
23: fim
```

Fonte: Autoria Própria

Figura 33 – Exemplos de intersecções com suas principais informações



Fonte: Autoria Própria

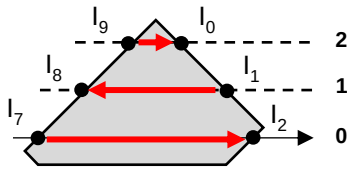
Na segunda subetapa, *sortIntersections*, as intersecções são ordenadas a fim de identificar a ordem em que serão percorridas na trajetória zigue-zague. De modo similar ao *scan*, para realizar uma ordenação eficiente em processadores paralelos é necessário utilizar um algoritmo de ordenação adequado a este paradigma de computação. Existem diversos algoritmos seriais que podem ser adaptados para o processamento paralelo, como por exemplo, *bitonic sort* e *counting sort* (SATISH; HARRIS; GARLAND, 2009). Para realizar a ordenação das intersecções foi utilizada a implementação padrão do *bitonic sort* que está incluída na biblioteca de exemplos da fabricante da GPU.

Neste caso de ordenação de intersecções de um zigue-zague, a definição de que uma intersecção deve ser considerada anterior à outra é baseada em dois critérios, relacionados às coordenadas y e x das intersecções, respectivamente. Dadas duas intersecções M e N, considera-se que a interseção M é anterior a N quando:

- A coordenada y de M é menor que de N, ou:
- A coordenada y de M é igual a de N, e:
  - A coordenada x de M é menor ou igual a de N, quando y é par.
  - A coordenada x de M é maior que de N, quando y é ímpar.

A verificação de paridade da coordenada y no segundo critério serve para criar o efeito zigue-zague, de modo que retas pares são percorridas da esquerda para a direita (coordenada x crescente) e as retas ímpares da direita para a esquerda (coordenada x decrescente), conforme exemplo na Figura 34.

Figura 34 – Exemplo de ordenação de intersecções



- I_2 é anterior a I_1 pois possui y menor
- I_1 é anterior a I_8 pois possui y igual e possui x maior estando em uma reta com índice ímpar (1)
- I_9 é anterior a I_0 pois possui y igual e possui x menor estando em uma reta com índice par (2)

Fonte: Autoria Própria

Para armazenar o resultado da ordenação realizada na subetapa *sortIntersections* foi criada uma estrutura chamada *IntersectionSort*, que permite acessar facilmente qual a posição de determinada intersecção e qual intersecção está em determinada posição. A Figura 35 exibe a estrutura de dados e como ela pode ser utilizada.

Figura 35 – Estrutura IntersectionSort

```
typedef struct IntersectionSort {
    int position
    int intersection;
} IntersectionSort;
```

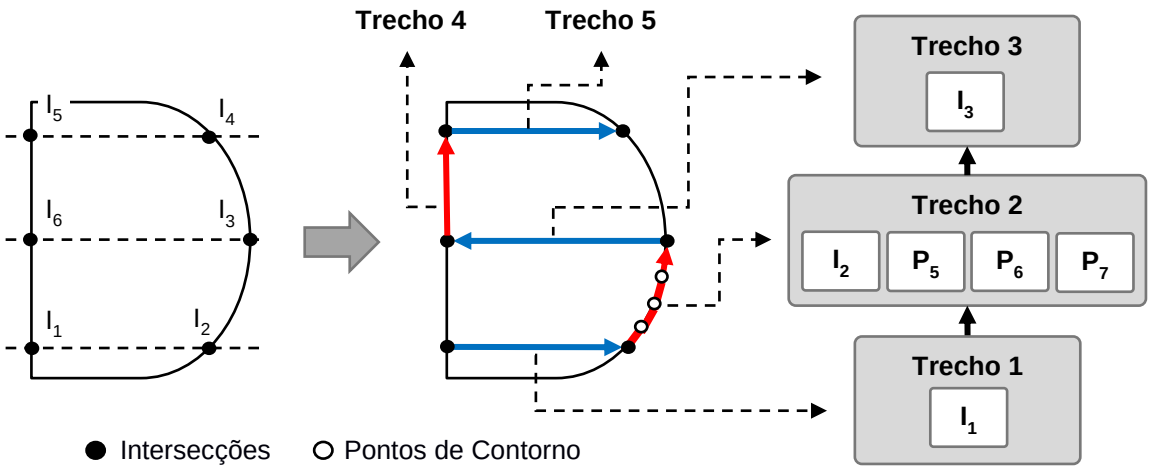
```
// Considerando um vetor com 100 intersecções
IntersectionSort sorted[100];

// Qual a posição da intersecção 5 ?
position = sorted[5].position

// Qual a intersecção na posição 10 ?
intersection = sorted[10].intersection
```

Fonte: Autoria Própria

3.1.4. Etapa de Cálculo de Trechos de *Raster*

O objetivo desta etapa é calcular e ordenar trechos de *raster* (*chunks*), que no contexto do método proposto, são considerados como sendo a sequência de pontos que são percorridos entre duas intersecções consecutivas da trajetória zigue-zague, incluindo a primeira destas intersecções, conforme Figura 36 abaixo. Esta etapa é realizada através de cinco subetapas: inicialização de trechos (*initChunks*), posicionamento de trechos (*rankChunks*), ordenação de trechos (*sortChunks*), scan de trechos (*scanChunks*) e leitura de trechos (*readChunks*), que serão detalhadas ao longo desta seção.

Figura 36 – Etapa de cálculo de trechos de *raster*



Fonte: Autoria Própria

Na subetapa *initChunks*, cada intersecção é utilizada como ponto de partida de um novo trecho de *raster* e um algoritmo é executado para identificar o número de pontos que serão percorridos no trecho e criar uma espécie de lista encadeada de trechos. O Algoritmo 3 analisa três possíveis casos para identificar estas informações (número de pontos e ponteiros da lista encadeada) do trecho de *raster*. No caso em que a intersecção está marcada com a *flag* HORIZONTAL, o trecho tem tamanho igual a um, pois conecta-se diretamente com outra intersecção, que é a próxima intersecção no vetor *IntersectionSorted* (Trechos 1, 3 e 5 da Figura 36 – linhas 6 a 12 do Algoritmo 3). No caso em que a intersecção não está marcada com a *flag* END, o trecho também tem tamanho igual a um, sendo o próximo trecho correspondente à próxima intersecção do próprio segmento (Trecho 4 da Figura 36 – linhas 14 a 19 do Algoritmo

3). Em último caso, o algoritmo deve percorrer o contorno, contando quantos pontos foram percorridos até encontrar uma nova intersecção, que será marcada como próximo trecho (Trecho 2 da Figura 36 – linhas 25 a 40 do Algoritmo 3).

Algoritmo 3 – Inicialização de trechos de *raster* (*initChunks*)

```
 1: executar em paralelo para cada intersection
 2:   chunkId ← índice do chunk
 3:   chunk.size ← 1
 4:   chunk.next ← -1
 5:
 6:   se intersection.flags contém HORIZONTAL então
 7:     position ← intersectionsSorted[chunkId].position
 8:     next ← intersectionsSorted[position + 1].intersection
 9:
10:     chunk.next ← next
11:     chunks[next].prev ← chunkId
12:     retorna
13:
14:   se intersection.flags não contém END então
15:     next ← chunkId + 1
16:
17:     chunk.next ← next
18:     chunks[next].prev ← chunkId
19:     retorna
20:
21:   segId ← intersection.segmentId
22:   direction ← segments[segId].goingUp ? 1 : -1
23:   count ← 1
24:
25:   laço
26:     segId ← próximoPonto(segments, segId, direction)
27:     count ← count + 1
28:
29:     se segments[segId] possui intersecções então
30:       next ← segments[segId].offset
31:       nextIntY ← intersections[next].rotated.y
32:       thisIntY ← intersections[chunkId].rotated.y
33:
34:       se nextIntY <= thisIntY então
35:         retorna
36:
37:       chunk.size ← count
```

```
38:         chunk.next ← next
39:         chunks[next].prev ← chunkId
40:         retorna
41: fim
```

Fonte: Autoria Própria

Na subetapa *rankChunks*, que pode ser vista no Algoritmo 4, cada elemento do vetor de trechos percorre de forma reversa a lista encadeada formada na etapa anterior, contando o número de trechos que existem antes do elemento e identificando sua posição na lista encadeada (*rank*). Ao fim deste processo é salvo o índice do primeiro trecho da lista encadeada (start), que determina o início de um trecho contínuo de *raster* e que será utilizado na próxima etapa.

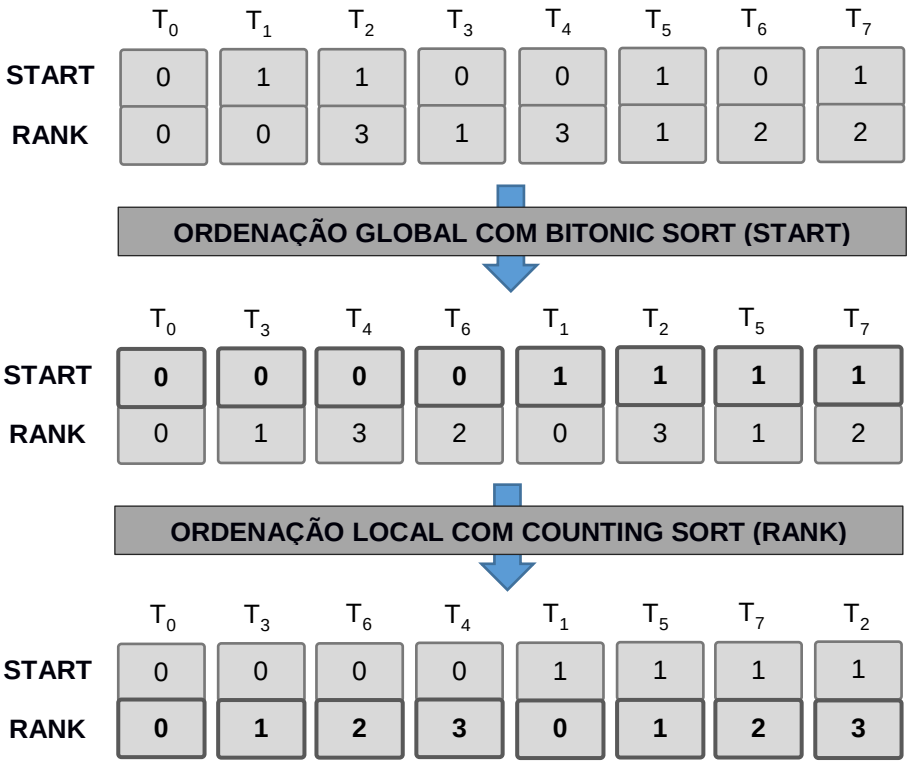
Algoritmo 4 – Posicionamento de trechos (*rankChunks*)

```
 1: executar em paralelo para cada chunk
 2:   start ← índice do chunk
 3:   rank ← 0
 4:
 5:   enquanto chunks[start] tem anterior
 6:     start ← chunks[start].prev
 7:     rank ← rank + 1
 8:
 9:   chunk.start ← start
10:   chunk.rank ← rank
11: fim
```

Fonte: Autoria Própria

Em seguida é realizada a ordenação dos trechos para identificar a ordem em que serão percorridos no zigue-zague. Esta ordenação é feita em duas etapas, conforme Figura 24. Na primeira etapa, é utilizado o algoritmo *bitonic sort* (KHAN et al., 2011) para fazer uma ordenação global baseada na variável start de cada trecho, agrupando todos os trechos pertencentes a um mesmo trecho contínuo de *raster*. Em seguida, é feita uma ordenação local, comparando apenas os elementos de um mesmo trecho contínuo de *raster* (*start*), com um algoritmo similar ao *counting sort*, com base na variável *rank*.

Figura 37 – Ordenação de trechos em duas etapas



Fonte: Autoria Própria

Novamente, antes de realizar a escrita dos pontos de cada trecho no *buffer* de pontos *raster*, na subetapa *scanChunks* é realizado um *scan* nos tamanhos dos trechos ordenados, de modo a dividir o *buffer* de pontos de *raster* dentre todos os trechos.

Por fim, na subetapa *readChunks* os pontos de cada trecho são percorridos e escritos em um vetor de pontos que é copiado da memória da GPU para a memória RAM do computador (Algoritmo 5).

Algoritmo 5 – Leitura de trechos (*readChunks*)

```
1:  executar em paralelo para cada chunk
2:    chunkId ← índice do chunk
3:
4:    offset ← chunk.offset
5:    size ← chunk.size
6:
7:    segId ← intersections[chunkId].segmentId
8:    rasterPoints[offset] = intersections[segId].intersection
9:
```

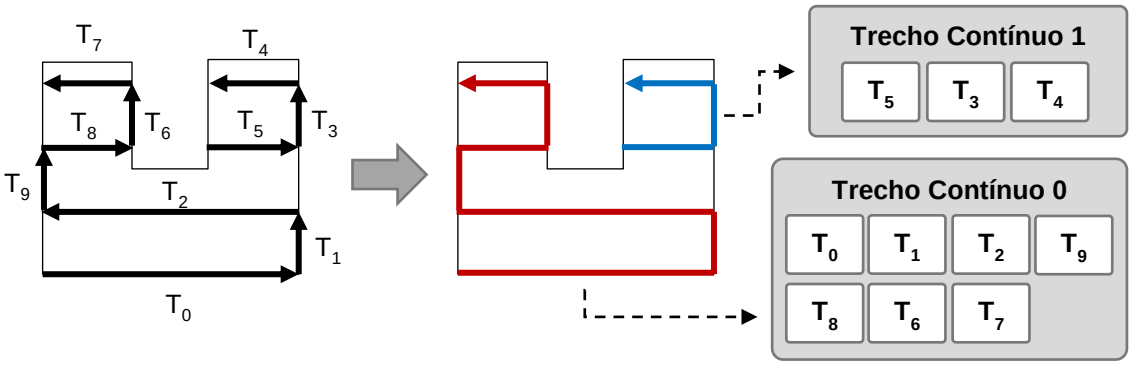
```
10:   enquanto size > 0
11:     rasterPoints[offset] = points[segId]
12:
13:     segId ← próximoPonto(segments, segId, direction)
14:     offset ← offset + 1
15:     size ← size - 1
16: fim
```

Fonte: Autoria Própria

3.1.5. Etapa de Agrupamento de Trechos Contínuos de *Raster*

Esta etapa tem como objetivo gerar vetores que agrupam os trechos de *raster* em trechos contínuos de *raster*, conforme Figura 38, e é realizada através de quatro subetapas: identificação dos inícios dos trechos contínuos (*compactRasters*), inicialização dos trechos contínuos (*initRasters*), scan dos trechos contínuos (*scanRasters*) e leitura do trechos contínuos (*readRasters*), que serão descritas nesta seção.

Figura 38 – Etapa de agrupamento de trechos contínuos de *raster*



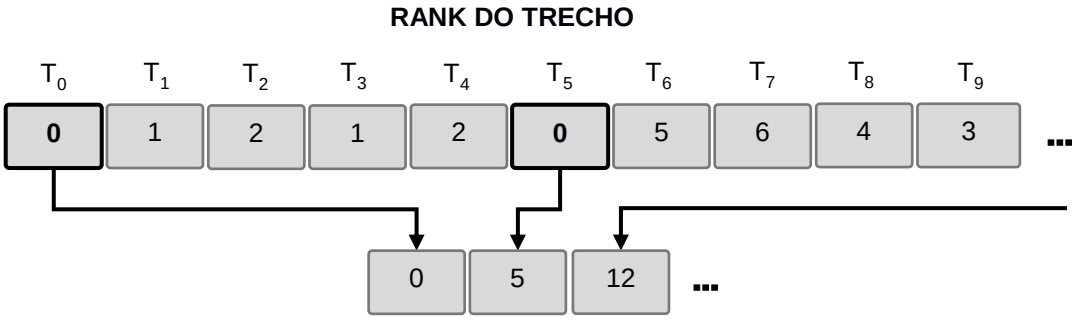
Fonte: Autoria Própria

A subetapa inicial *compactRasters* tem como objetivo extrair em um novo vetor os índices dos trechos que são o início de um trecho contínuo de *raster*. Para extrair estes índices, um método paralelo verifica a variável *rank* de cada trecho, conforme Figura 39.

Este problema é conhecido na literatura como *stream compaction* e pode ser resolvido de várias formas (BAKUNAS-MILANOWSKI et al., 2017; BILLETER; OLSSON; ASSARSSON, 2009). Nesta pesquisa optou-se por utilizar operação atômicas de incremento para resolver este problema devido a sua simplicidade de

implementação. A operação atômica utilizada garante de forma simples que o *buffer* de saída seja preenchido sem que dois elementos escrevam no mesmo endereço de memória. Esta operação pode resultar numa queda de performance quando utilizada de forma intensiva, porém neste caso em específico, o número trechos que iniciam um trecho contínuo de *raster* é muito menor que o número de trechos avaliados e estima-se que o prejuízo de performance seja muito pequeno.

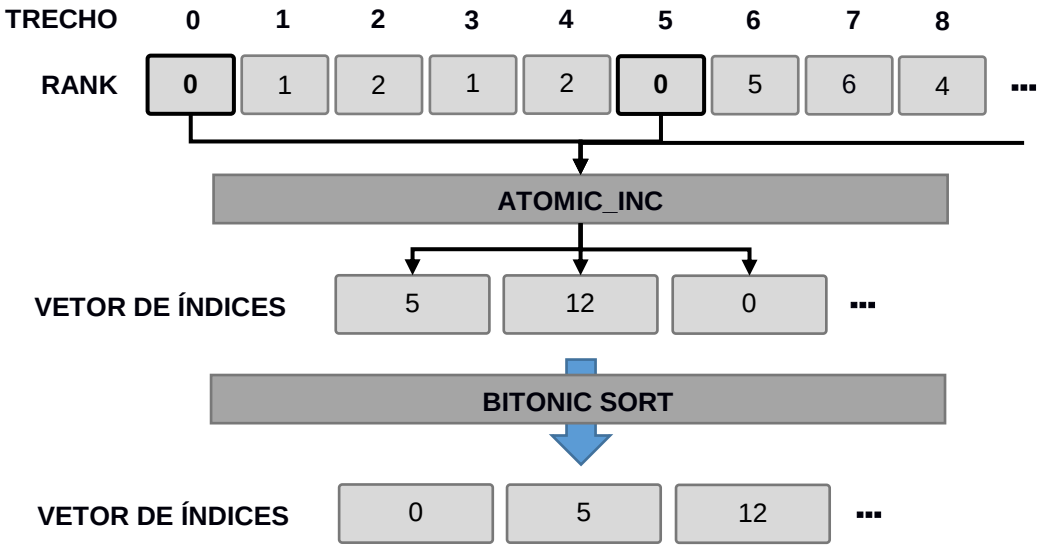
Figura 39 – Subetapa *compactRasters*



Fonte: Autoria Própria

Um resultado indesejado da utilização desta operação é que a sequência dos dados não é mantida e se torna imprevisível. Para resolver este novo problema, uma ordenação utilizando *bitonic sort* é feita no *buffer*, conforme Figura 40.

Figura 40 – Exemplo da subetapa *compactRasters*



Fonte: Autoria Própria

Na subetapa *initRaster*, cada índice do vetor gerado na etapa anterior é utilizado como ponto inicial de um trecho contínuo de *raster*. De forma paralela, um laço caminha pelos trechos de cada trecho contínuo de *raster*, somando seus números de pontos (variável *size*). Ao fim deste processo, é gerado um vetor com os pontos iniciais e o tamanho de cada *raster*, conforme Algoritmo 6.

Algoritmo 6 – Inicialização dos trechos contínuos de *raster* (*initRasters*)

```
 1: executar em paralelo para cada rasterStart
 2:     id ← índice do rasterStart
 3:
 4:     rasters[id].start ← rasterStart
 5:
 6:     chunkId ← rasterStart
 7:     size ← 0
 8:
 9:     laço
10:       size += chunks[chunkId].size
11:       chunkId ← chunks[rid].next
12:     enquanto chunkId != -1
13:
14:     rasters[id].size ← size
15: fim
```

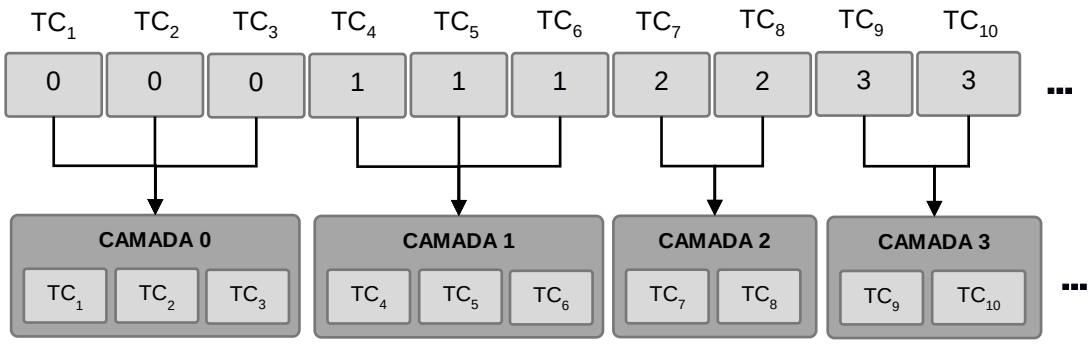
Fonte: Autoria Própria

Em seguida, é feito um *scan* no tamanho de cada trecho contínuo de *raster* (*size*) para dividir o *buffer* de saída desta etapa. Por fim, a subetapa *readRasters* faz a escrita dos *rasters* em um *buffer* e a respectiva leitura destas informações para a memória RAM do computador.

3.1.6. Etapa de Agrupamento em Camadas

Esta etapa final tem como objetivo agrupar trechos contínuos de raster pelos seus índices de camada e montar os vetores que descrevem o preenchimento de cada camada, conforme Figura 41. É a etapa mais simples do método e é realizada em apenas duas subetapas denominadas de inicialização das camadas (*initSlices*) e leitura das camadas (*readSlices*).

Figura 41 – Etapa de agrupamento em camadas



Fonte: Autoria Própria

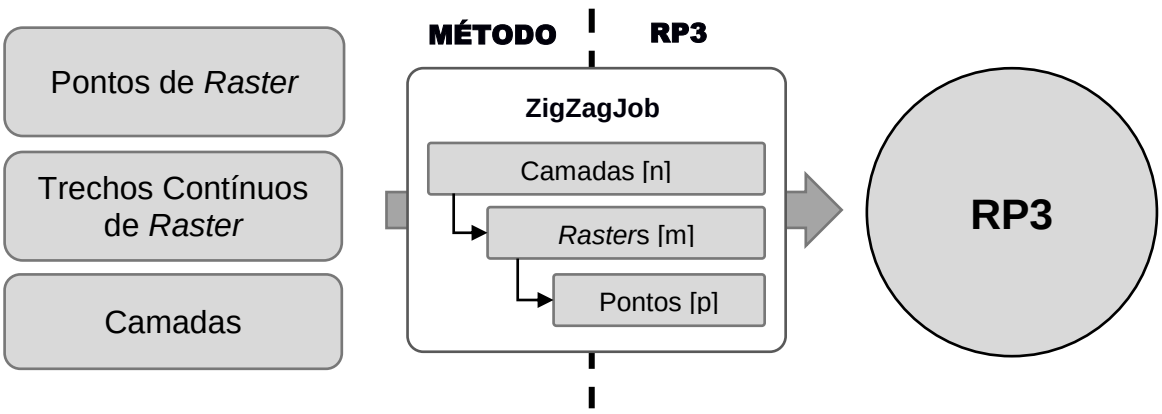
Na subetapa *initSlices*, os índices de camada do ponto inicial de cada trecho contínuo de *raster* consecutivo são comparados para identificar o índice do primeiro trecho contínuo de *raster* de cada camada. Esta informação é salva em um simples vetor de inteiros chamado *sliceOffsets*.

Por fim, de forma similar à etapa *readRasters*, na subetapa *readSlices* é feito a escrita do vetor que agrupa os trechos contínuos de *raster* de cada camada e a respectiva leitura destes dados para a memória RAM.

3.1.7. Saída de Dados para o RP3

O método tem como saída da GPU três grandes vetores de dados (camadas, trechos contínuos de *raster* e pontos de *raster*) que são utilizados para preencher a estrutura *ZigZagJob* que é retornada ao RP3, conforme Figura 42.

Figura 42 – Interface de saída Método / RP3



Fonte: Autoria Própria

3.2. AVALIAÇÃO DO MÉTODO

Inicialmente, o método proposto foi avaliado por um viés computacional. Foram feitos testes para avaliar a sua capacidade em resolver o problema do cálculo da trajetória zigue-zague e medir a sua velocidade computacional, que são descritos nas próximas seções.

3.2.1. Equipamentos Utilizados

Por se tratar de uma pesquisa que avalia a performance de um método computacional, é necessário identificar as especificações do computador (*hardware* e *software*), que seguem abaixo:

- Processador Intel Core i5-4570 com 4 núcleos de 3.2GHz.
- Memória DDR3 com capacidade de 16GB.
- Placa Gráfica GeForce GTX 750 Ti com 2048MB de memória GDDR5 e 640 núcleos CUDA de 1020 MHz.
- Sistema Operacional: Windows 10
- Ambiente de Desenvolvimento: Microsoft Visual Studio 2012
- Linguagem de Programação: C++
- CUDA Toolkit versão 10.1
- OpenCL versão 1.2 (fornecida pelo CUDA Toolkit)

A implementação do método paralelo poderia ter sido realizada utilizando duas diferentes tecnologias: a CUDA, tecnologia proprietária que pode ser executada apenas em placas gráficas da marca NVIDIA; ou o OpenCL, tecnologia de código aberto que pode ser executada em diferentes ambientes de computação. Por se tratar de uma pesquisa acadêmica e para se obter uma maior generalidade, optou-se pela implementação utilizando o OpenCL. O CUDA Toolkit citado na lista acima é utilizado apenas para fornecer a versão da biblioteca OpenCL adequada à placa gráfica utilizada.

Com relação ao *hardware*, a escolha desta configuração específica não é definida como um requisito obrigatório. A utilização de um computador com especificações diferentes é aceitável, no entanto, os resultados de tempos

computacionais podem ser diferentes dos obtidos nesta pesquisa. É necessário apenas que o computador execute o RP3 e possua uma placa gráfica compatível com a tecnologia OpenCL.

3.2.2. Modelos Geométricos

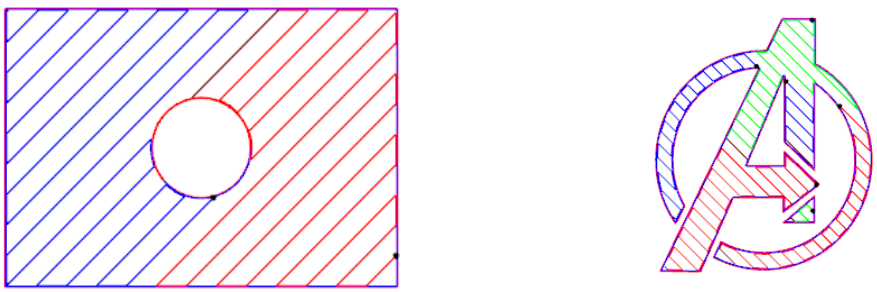
Para avaliar o método e servir como sua entrada foram selecionados dez modelos tridimensionais de diferentes complexidades geométricas, que podem ser vistos na forma de dados na Tabela 1. As Figura 44 e Figura 44 apresentam uma camada característica dos quatro primeiros modelos da tabela.

Tabela 1 – Modelos geométricos

Modelo 3D	Camadas	Pontos de Contorno
Chapa Furada	79	7.936
Avengers	40	17.152
Labirinto	158	47.104
Peça 7	79	155.904
Dragon	8	42.240
Volvo	79	32.512
Brasil	71	71.424
Labrador	446	40.704
Wrench	44	37.632
Yoda	268	28.416

Fonte: Autoria Própria

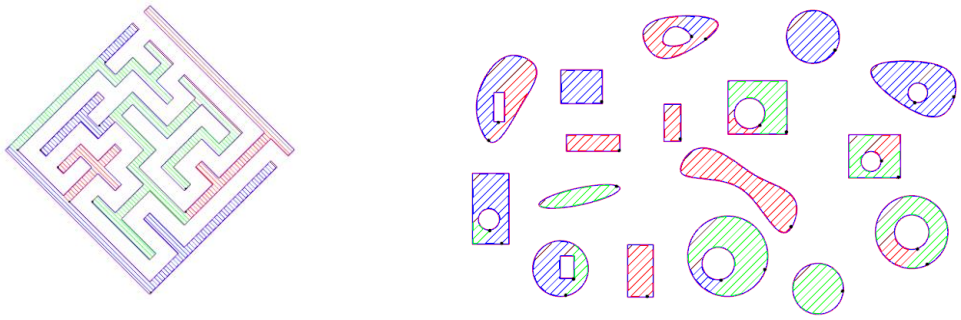
Figura 43 – Modelos geométricos: (a) Chapa Furada, (b) Avengers



(a) (b)

Fonte: Autoria Própria

Figura 44 – Modelos geométricos: (a) Labirinto, (b) Peça 7



(a) (b)

Fonte: Autoria Própria

3.2.3. Testes de Corretude Funcional

A corretude de um algoritmo está relacionada a sua capacidade em resolver determinado problema. Um algoritmo é considerado correto quando sempre encerra sua computação retornando um resultado válido (CORMEN, 2012). A prova formal da corretude de um algoritmo nem sempre pode ser obtida facilmente e necessita de um rigoroso formalismo matemático. Por este motivo, nesta pesquisa optou-se por verificar apenas a corretude funcional do método, isto é, dada uma entrada válida, o algoritmo retorna uma saída válida.

Para fazer esta verificação foram utilizados os dez modelos geométricos descritos anteriormente, com os quais foi realizado o cálculo do zigue-zague, utilizando os métodos serial e paralelo.

Primeiramente, foram comparados os comprimentos totais das trajetórias zigue-zague calculadas pelos métodos. A diferença percentual entre os comprimentos foi avaliada para identificar possíveis diferenças nas trajetórias calculadas.

Em seguida, as trajetórias resultantes do método paralelo foram inspecionadas visualmente para comprovar o preenchimento completo de todas camadas pela trajetória zigue-zague e identificar possíveis falhas no cálculo.

Por fim, com base nestas observações foi avaliada a corretude funcional do método.

3.2.4. Testes de Tempo Computacional

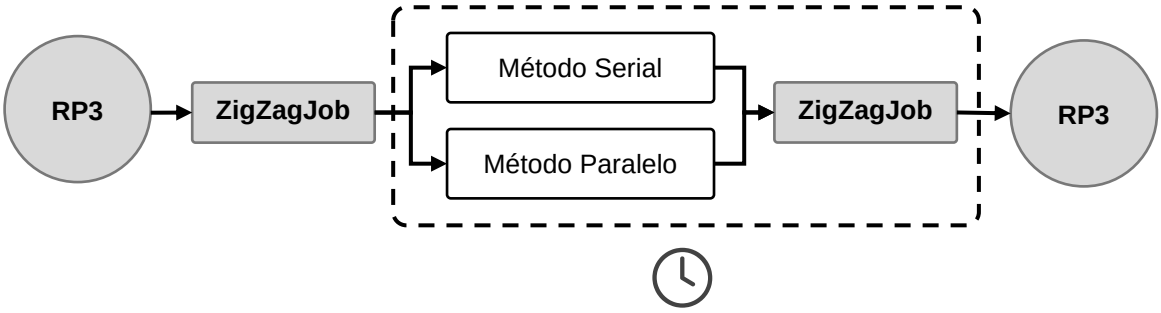
A grande motivação do desenvolvimento de um método paralelo de cálculo do preenchimento é a provável redução no tempo computacional e os benefícios que podem ser alcançados por esta redução. Por este motivo, foram realizados testes para comparar o tempo necessário para cálculo do zigue-zague entre o antigo método serial e o novo método paralelo.

O primeiro experimento foi realizado para identificar o ganho computacional do método, comparando os tempos computacionais do método serial implementado no RP3 e o método paralelo proposto. Para garantir condições iguais de avaliação, o método serial do RP3 foi adaptado para receber como entrada um *ZigZagJob*, assim como o método paralelo. Para este experimento foram utilizados apenas os quatro primeiros modelos geométricos da Tabela 1.

Para avaliar a influência do tamanho do *ZigZagJob* no tempo computacional, foram realizadas três medições para cada modelo geométrico, considerando diferentes números de simulações do ângulo de *raster*. Devido às limitações da GPU utilizada, foram realizadas até 50 simulações de parâmetros para cada modelo.

A medição de tempo foi feita sob o cálculo do zigue-zague e a escrita da trajetória resultante no *ZigZagJob*, ignorando tempos de inicialização dos métodos, conversões de estrutura de dados e etapas anteriores ao preenchimento. Desta forma, o contador de tempo inicia no instante que o método recebe o *ZigZagJob* e encerra no instante em que o *ZigZagJob* é devolvido ao RP3 com a trajetória calculada, conforme Figura 45.

Figura 45 – Medição de tempo



Fonte: Autoria Própria

O segundo experimento foi realizado para medir o tempo computacional de cada etapa e subetapa envolvida no cálculo paralelo do zigue-zague. Esta medição

tem como objetivo identificar quais etapas consomem o maior tempo computacional, indicando em quais delas deve-se focar os esforços de melhoria do método.

Para este experimento, foi utilizado o modelo tridimensional que resultou no maior ganho computacional no experimento anterior, com o qual o cálculo do zigue-zague foi novamente realizado, porém desta vez medindo o tempo computacional de cada uma das etapas e subetapas.

3.3. ESTUDO DE APLICAÇÃO DO MÉTODO

Após a avaliação computacional, o método foi avaliado por um viés da manufatura aditiva, focado no estudo de uma aplicação do método e nos benefícios que este traz na redução dos custos e tempo de fabricação de peças.

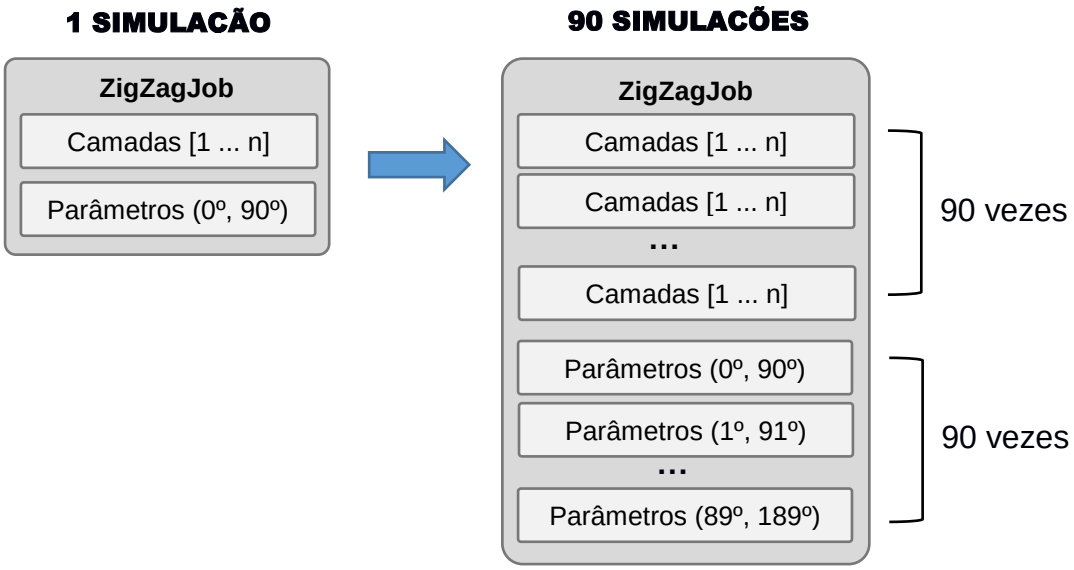
Foi realizado um estudo de aplicação que avaliou o impacto deste na otimização de um dos parâmetros do cálculo de preenchimento zigue-zague: o ângulo de *raster*. Este estudo serve apenas como um exemplo de aplicação do método paralelizado, sendo que outras aplicações poderão ser identificadas. Novamente, apenas os quatro primeiros modelos geométricos da Tabela 1 foram utilizados neste estudo.

Para exemplificar o potencial de aceleração dos cálculos, foram realizadas simulações de cálculo do preenchimento, variando o parâmetro ângulo de *raster* e obtendo o comprimento médio dos filamentos das retas de *raster*, que está diretamente relacionado ao tempo total de fabricação da peça, de modo similar ao estudo realizado por Jin *et al.* (2014).

Assim como o estudo de Jin *et al.* (2014), o objetivo da otimização foi de encontrar o ângulo de *raster* que gera o maior valor de comprimento médio das retas de *raster*, resultando no menor tempo de fabricação da peça. Diferente de Jin, neste estudo de aplicação do método foi mantida uma variação exata de 90º no ângulo de *raster* entre duas camadas consecutivas, de modo a garantir uma maior resistência da peça (VOLPATO; DA SILVA, 2017).

Desta forma, para cada modelo avaliado foi gerado um novo *ZigZagJob*, com cópias das camadas e com variações de parâmetros de *raster*, conforme Figura 46. Este único *ZigZagJob* contendo todas as variações de parâmetros foi enviado ao método paralelo para cálculo do zigue-zague.

Figura 46 – ZigZagJob com variações de parâmetros



Fonte: Autoria Própria

Após o cálculo, o *ZigZagJob* retornado é processado e o comprimento médio das retas de *raster* é calculado. Por fim, os resultados são exibidos em gráficos que mostram o comprimento médio em função do ângulo de *raster*.

Uma busca exaustiva nos resultados das simulações foi realizada para identificar os melhores e piores ângulos de *raster*, que resultam, respectivamente, nos maiores e nos menores comprimentos médios das retas de *raster*.

Para os dois primeiros modelos, foi feita a inspeção visual das trajetórias zigue-zague geradas com os melhores e piores ângulos de *raster*, permitindo a visualização das retas de *raster* e a validação dos resultados obtidos.

4. RESULTADOS E DISCUSSÕES

Este capítulo apresenta os resultados das avaliações e do estudo de aplicação do método desenvolvido, conforme descrito no Capítulo 3.

4.1. AVALIAÇÃO DO MÉTODO

4.1.1. Testes de Corretude Funcional

Na Tabela 2 são exibidos e comparados os comprimentos totais das trajetórias zigue-zague geradas pelo método serial (RP3) e pelo método paralelo proposto (GPU).

Tabela 2 – Comprimento total da trajetória zigue-zague (mm)

Modelo	Método Serial (RP3)	Método Proposto (GPU)	Diferença
Chapa Furada	219.430	219.550	0.0547%
Avengers	21.287	21.286	- 0.0047%
Labirinto	916.497	916.297	- 0.0218%
Peça 7	202.883	202.722	- 0.0794%
Dragon	7.738	7.745	0.0905%
Volvo	242.643	242.678	0.0144%
Brasil	366.240	366.288	0.0131%
Labrador	310.873	310.792	- 0.0261%
Wrench	41.084	41.063	- 0.0511%
Yoda	171.131	171.210	0.0462%

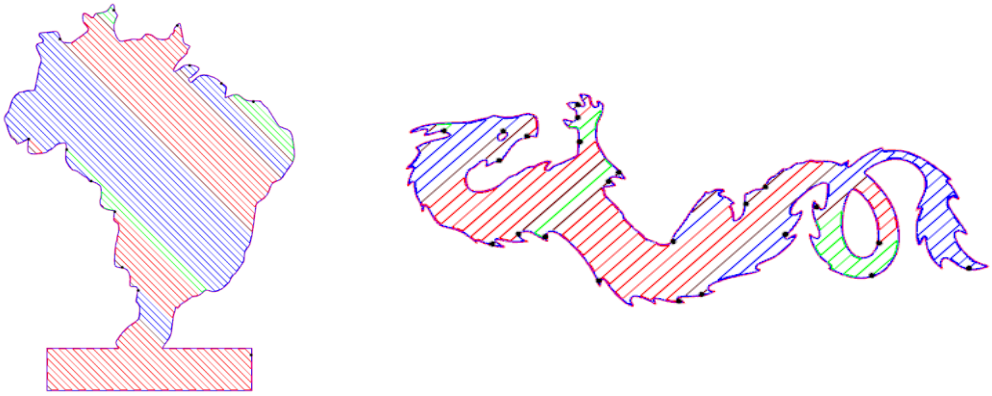
Fonte: Autoria Própria

Os resultados da Tabela 2 mostram que a diferença entre os comprimentos das trajetórias geradas pelos métodos é inferior a 0,1% em todos os casos avaliados, indicando que ambos geram trajetórias similares. Considerando que o método serial gera trajetórias zigue-zague válidas este é o resultado esperado para o método proposto.

Em seguida, foi feita uma verificação através da inspeção visual de todas as trajetórias calculadas pelo método proposto, a fim identificar possíveis áreas com

preenchimento deficiente ou incorreto. Em todos os casos verificados não foram encontrados problemas na trajetória zigue-zague gerada pelo método, de modo que todas as camadas das peças foram preenchidas completamente, conforme exemplificado na Figura 47.

Figura 47 – Exemplos do preenchimento correto de camadas



Fonte: Autoria Própria

O resultado destas verificações em conjunto com os dados da Tabela 2, indicam que o método proposto é funcionalmente correto, visto que para todos os modelos que o sistema RP3 forneceu uma entrada válida, o resultado também foi válido, com o preenchimento completo de todas as camadas por uma trajetória zigue-zague.

4.1.2. Testes de Tempo Computacional

Na Tabela 3 são exibidos os tempos computacionais dos métodos de cálculo da trajetória zigue-zague, os ganhos computacionais resultantes e a redução no tempo de cálculo, considerando diferentes números de simulações de parâmetros calculadas.

Com base nos resultados da Tabela 3, e conforme o esperado, pode-se notar que enquanto o método serial cresce o seu tempo computacional de forma linear e proporcional ao número de simulações realizadas, o método paralelo tende a crescer de forma mais lenta, resultando em ganhos maiores na medida que a carga computacional aumenta.

O maior ganho computacional obtido neste experimento foi de 22 vezes no modelo “Peça 7”, com o cálculo de 50 simulações de parâmetros, e que representa

uma redução de 95% no tempo computacional do cálculo do zigue-zague quando comparado ao método serial. Já nas simulações com um número menor de parâmetros, o ganho computacional foi menor. Ainda assim, na situação com pior ganho, o resultado foi relevante, com uma redução de 33% no tempo computacional.

Tabela 3 – Tempos computacionais dos métodos de cálculo (ms)

Modelo 3D	Simulações	RP3	GPU	Ganho	Redução
Chapa Furada	1	25	15	1,67	40%
Chapa Furada	10	257	24	10,71	91%
Chapa Furada	50	1.340	82	**16,34**	**94%**
Avengers	1	18	11	1,50	33%
Avengers	10	189	41	4,61	78%
Avengers	50	979	58	**16,88**	**94%**
Labirinto	1	185	31	5,97	83%
Labirinto	10	2.091	151	13,84	93%
Labirinto	50	10.162	618	**16,43**	**94%**
Peça 7	1	146	38	3,84	74%
Peça 7	10	1.511	96	15,73	94%
Peça 7	50	9.057	409	**22,13**	**95%**

Fonte: Autoria Própria

A utilização de uma placa gráfica moderna tende a aumentar ainda mais o ganho computacional do método, visto que a placa utilizada possui apenas 640 núcleos de processamento e atualmente é comum encontrar placas gráficas com mais de 3.000 núcleos de processamento (NVIDIA, 2019). A atualização do processador, no entanto, não deve gerar impacto tão grande no ganho computacional, visto que um processador mais moderno possui performance 40% superior ao processador utilizado nos experimentos (USERBENCHMARK, 2019). Desta forma, com a atualização da placa gráfica pode-se esperar ganhos computacionais de 2 a 3 vezes maiores que os ganhos medidos.

O segundo experimento que visou identificar o tempo gasto na realização de cada uma das etapas e subetapas do método é apresentado na Tabela 4. Foi utilizado o modelo “Peça 7” por ter resultado de maior ganho computacional no primeiro experimento.

Tabela 4 – Tempos computacionais por etapa e subetapa (ms)

	Peça 7 (Simulações: 1)		Peça 7 (Simulações: 50)	
Etapa	**Tempo**	**%**	**Tempo**	**%**
Entrada: Cópia para a GPU	**3**	**9%**	**71**	**18%**
Segmentos	**2**	**6%**	**58**	**15%**
calcSegments	1	3%	36	9%
scanSegments	1	3%	22	6%
Intersecções	**5**	**14%**	**148**	**38%**
calcIntersections	1	3%	10	3%
sortIntersections	4	11%	138	35%
Trechos de *Raster*	**12**	**34%**	**104**	**27%**
initChunks	1	3%	12	3%
rankChunks	1	3%	3	1%
sortChunks	8	23%	65	17%
scanChunks	1	3%	6	2%
readChunks	1	3%	18	5%
Trechos Contínuos de *Raster*	**11**	**31%**	**9**	**2%**
compactRasters	6	17%	4	1%
initRasters	1	3%	3	1%
scanRasters	3	9%	1	0%
readRasters	1	3%	1	0%
Camadas	**2**	**6%**	**2**	**1%**
initSlices	1	3%	1	0%
readSlices	1	3%	1	0%
Saída: Cópia para o ZigZagJob	**0**	**0%**	**0**	**0%**
TOTAL	**35**	**100%**	**392**	**100%**

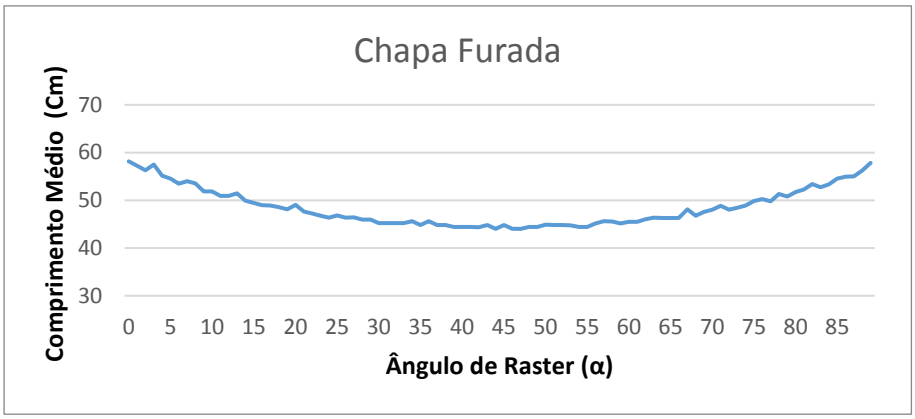
Fonte: Autoria Própria

Com os resultados da Tabela 4 é possível notar que as etapas de ordenação são as mais custosas do método, representando 52% do tempo total da computação no experimento com 50 simulações de parâmetros e 34% no experimento com 1 simulação de parâmetro. A etapa de entrada dos dados, que copia de forma serial os dados de entrada para a GPU, representa 18% do tempo computacional. Desta forma, esforços para acelerar ainda mais o método devem se concentrar na avaliação de melhores algoritmos para estas etapas.

4.2. ESTUDO DE APLICAÇÃO DO MÉTODO

O primeiro resultado mostrou que, para o modelo “Chapa Furada”, o ângulo de *raster* inicial que resulta no maior comprimento médio de retas de *raster* é de 0º (Figura 48). Neste cenário, o comprimento médio das retas de *raster* foi de aproximadamente 58 mm, o que representa uma melhoria de 32% sob o pior resultado, que foi de 44 mm com um ângulo de *raster* de 46º.

Figura 48 – Comprimento médio de retas de *raster* para o modelo “Chapa Furada”



Melhor Resultado

α = 0º

C_m = 58,15 mm

Pior Resultado

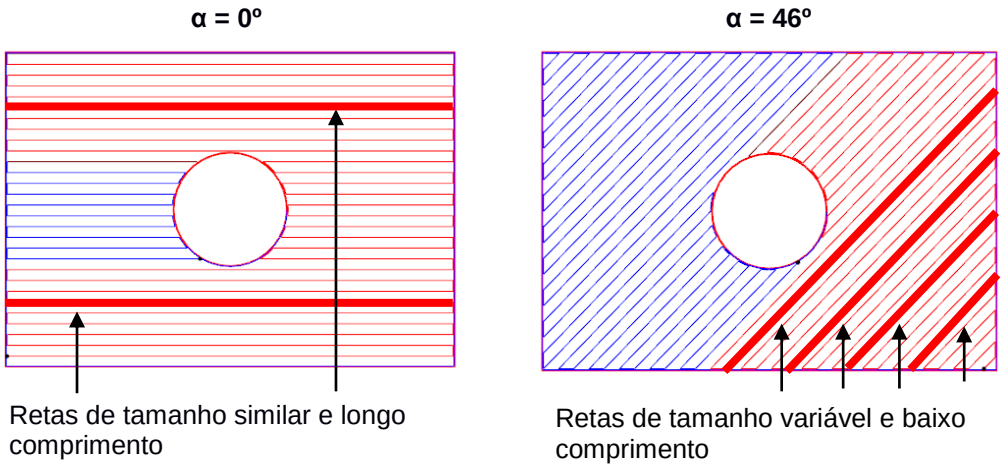
α = 46º

C_m = 44,02 mm

Fonte: Autoria Própria

Ao inspecionar visualmente ambos os resultados, foi possível comprovar que o ângulo de *raster* de 0º gera o maior comprimento médio das retas de *raster*, conforme Figura 49 abaixo. Na situação com ângulo de *raster* de 0º é possível notar que a maioria das retas de *raster* possuem comprimento maximizado e similar. Já no experimento com ângulo de *raster* de 46º as retas de *raster* tem comprimento crescente e nunca atingem o comprimento máximo da situação anterior.

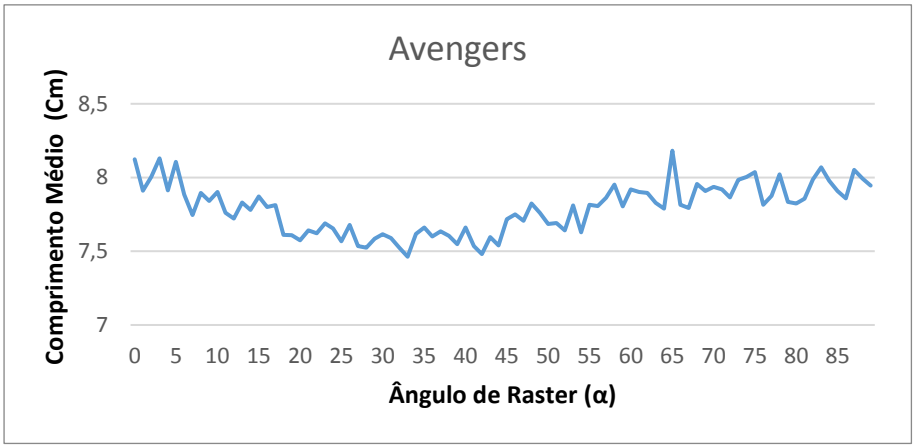
Figura 49 – Comparativo entre o melhor e o pior resultado do preenchimento (Chapa Furada)



Fonte: Autoria Própria

O modelo “Avengers” apresentou um resultado no qual o melhor ângulo de *raster* é de 65º, com um comprimento médio de 8,18 mm e uma melhoria de 9% sob o pior resultado, conforme Figura 50 a seguir. Assim como no modelo anterior, a inspeção visual das duas situações permitiu comprovar o melhor e o pior resultado, conforme Figura 51 a seguir.

Figura 50 – Comprimento médio de retas de *raster* para o modelo “Avengers”



Melhor Resultado

α = 65º

C_m = 8,18 mm

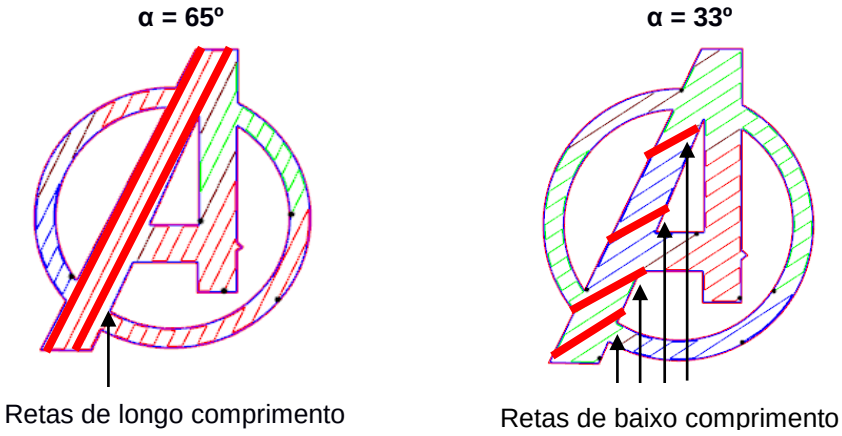
Pior Resultado

α = 33º

C_m = 7,46 mm

Fonte: Autoria Própria

Figura 51 – Comparativo entre o melhor e o pior resultado do preenchimento (Avengers)

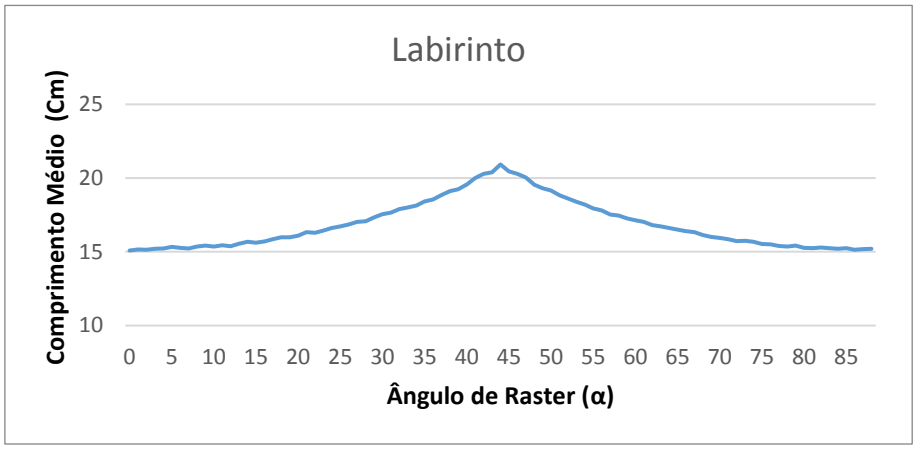


Fonte: Autoria Própria

É importante lembrar que neste estudo de aplicação foi mantida uma variação de 90º no ângulo de *raster* entre camadas consecutivas. Com isso, o ângulo de *raster* que gera retas com maior comprimento em determinada camada, geralmente resulta em retas de baixo comprimento na camada seguinte. Por este motivo, a busca pela combinação de parâmetros que maximiza o comprimento médio das retas de *raster* pode não ser facilmente identificada pela inspeção visual em todos os casos.

Por fim, os modelos “Labirinto” e “Peça 7” apresentaram diferença entre o melhor e o pior resultado de 38% e 17%, respectivamente, conforme Figura 52 e Figura 53 a seguir.

Figura 52 – Comprimento médio de retas de *raster* para o modelo “Labirinto”

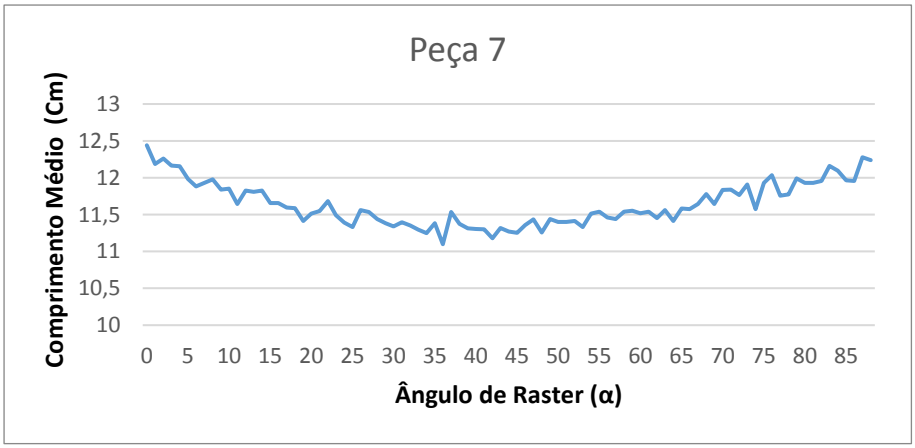


Melhor Resultado
α = 45º
C_m = 20,92 mm

Pior Resultado
α = 1º
C_m = 15,08 mm

Fonte: Autoria Própria

Figura 53 – Comprimento médio de retas de *raster* para o modelo “Peça 7”



Melhor Resultado
α = 1º
C_m = 12,83 mm

Pior Resultado
α = 37º
C_m = 11,09 mm

Fonte: Autoria Própria

Os resultados do estudo de caso mostraram que a escolha otimizada do ângulo de *raster* pode aumentar o comprimento médio das retas de *raster* em até 38%, reduzindo o tempo total de fabricação da peça. É importante notar que tal melhoria é altamente dependente da geometria, sendo que no caso com menor diferença (Avengers), a diferença entre o melhor e pior resultado foi de apenas 9%.

Além da redução no tempo de fabricação da peça, os ângulos de *raster* otimizados podem ser utilizados na rotação da peça como um todo, de modo que após a rotação, as retas de *raster* estejam alinhadas com os eixos principais (x e y) da

máquina de fabricação. Nesta situação específica pode-se obter outros benefícios como a redução de erros dimensionais e um menor desgasta nos motores que movimentam o bico de extrusão.

5. CONCLUSÕES

Esta pesquisa propôs um novo método paralelo para o cálculo da trajetória zigue-zague que explora o alto paralelismo de placas de processamento gráfico (GPU) para aceleração da computação. O método foi implementado como um módulo do sistema RP3, utilizando a biblioteca OpenCL de computação.

A capacidade do método em resolver o problema do cálculo do zigue-zague foi inicialmente verificada através da comparação entre o comprimento total das trajetórias geradas pelo método serial e pelo método paralelo. Os resultados mostraram que, para todas as peças, o método paralelo gera trajetórias com comprimento muito próximo ao gerado pelo método serial. A inspeção visual das trajetórias calculadas foi utilizada para comprovar os resultados e não identificou nenhum problema no zigue-zague gerado pelo método paralelo. Desta forma, o método foi considerado funcionalmente correto.

Experimentos mostraram que o método proposto apresenta ganho computacional relevante, sendo 22 vezes mais rápido que o seu equivalente serial e reduzindo o tempo para o cálculo do zigue-zague em mais de 95%. A utilização de placas gráficas mais modernas pode aumentar ainda mais o ganho computacional do método, que varia de acordo com a complexidade do modelo tridimensional e com o número de simulações de parâmetros avaliadas, sendo menor nos modelos mais simples.

A etapa mais custosa do método é a ordenação paralela de vetores, que pode ser acelerada com a implementação de algoritmos paralelos mais eficientes descritos na literatura. Além da ordenação, algoritmos paralelos mais eficientes para problemas como o *scan* e o *stream compaction* podem ser investigados para aceleração do método.

Um estudo de aplicação mostrou que através de simulações do cálculo do zigue-zague foi possível aumentar o comprimento médio das retas de *raster* em até 38%, reduzindo o tempo de fabricação de peça nos casos avaliados. O uso do método paralelo permitiu a realização de um número maior de simulações do cálculo do zigue-zague com diferentes parâmetros, quando comparado ao método serial, e possibilitou a otimização dos parâmetros do preenchimento.

5.1. CONSIDERAÇÕES FINAIS

O desenvolvimento e implementação do método proposto nesta pesquisa pode ser considerado não trivial, consumindo grande parte do tempo disponível. Durante todo este período, não se teve certeza da possibilidade de implementação de um método paralelo eficiente para o cálculo do zigue-zague e se este método resultaria em um ganho computacional relevante.

O cálculo paralelo do zigue-zague envolve a solução de diversos problemas, sendo que para alguns destes, foi necessária a revisão bibliográfica e implementação de um algoritmo paralelo descrito na literatura.

Para resolver as etapas iniciais do método (cálculo de segmentos e intersecções) foi necessário revisar os métodos seriais e identificar todos os cálculos matemáticos e computações geométricas, que foram organizados e separados nas duas primeiras etapas do método.

A etapa de cálculo dos trechos de *raster* foi considerada a mais difícil, pois diferente das etapas anteriores, não havia nenhuma similaridade com o método serial que pudesse ser utilizada como referência e uma solução completamente nova teve de ser desenvolvida.

A solução das etapas finais de agrupamento em trechos de contínuos de *raster* e agrupamento em camadas, ainda que mais simples, necessitaram de uma atenção especial para que a criação e leitura dos vetores de dados fosse realizada de forma eficiente.

A impossibilidade da depuração de código (*debug*) no ambiente OpenCL dificultou muito a solução de erros que foram encontrados ao longo da implementação do método. Para cada novo erro de cálculo, foi necessária a inspeção manual dos dados de saída de cada uma das etapas a fim de validar o seu correto funcionamento, fazendo com que a sua correção se tornasse difícil e demorada.

A implementação do método levou em consideração a velocidade de acesso dos tipos de memória da GPU, fazendo sempre que conveniente, a cópia dos dados da memória global para a local. Foi feita uma breve revisão dos *kernels* para identificar possíveis conflitos de banco e otimizações no padrão de acesso às memórias.

De modo geral, a implementação de um método eficiente torna necessária a preocupação de que todas as etapas deste método também sejam eficientes e implementadas de forma otimizada, visto que uma única etapa ineficiente pode reduzir drásticamente o ganho computacional do método.

A utilização da biblioteca OpenCL e de uma estrutura de interface de dados genérica permite que o método seja facilmente integrado a outros sistemas planejadores de processo e executado em placas gráficas de outras fabricantes.

O método pode ser futuramente utilizado em conjunto com as técnicas de otimização de preenchimento descritos no Capítulo 2, e em especial, na ordenação dos trechos contínuos buscando minimizar a distância percorrida com o bico de extrusão desligado (VOLPATO et al., 2019).

5.2. SUGESTÕES PARA TRABALHOS FUTUROS

Com a conclusão deste trabalho foram identificadas diversas oportunidades de pesquisa e melhorias do método proposto, que são listadas a seguir.

- Avaliar diferentes algoritmos paralelos de ordenação de vetores, visto que esta etapa é a mais custosa do método, visando o aumento do ganho computacional.
- Utilizar o paralelismo da CPU (*threads*) para acelerar a cópia dos dados da entrada para a memória da GPU.
- Testar diferentes placas gráficas e verificar a influência do número de núcleos de processamento na velocidade de computação.
- Implementar o método utilizando a tecnologia proprietária CUDA e avaliar o impacto da tecnologia no ganho computacional.
- Aplicar o paralelismo de GPUs nas demais etapas do planejamento do processo ou em outras estratégias de preenchimento.

REFERÊNCIAS BIBLIOGRÁFICAS

ALTINTAS, Y.; KERSTING, P.; BIERMANN, D.; BUDAK, E.; DENKENA, B.; LAZOGLU, I. Virtual process systems for part machining operations. **CIRP Annals**, v. 63, n. 2, p. 585–605, 2014.

BAKUNAS-MILANOWSKI, D.; REGO, V.; SANG, J.; CHANSU, Y. Efficient Algorithms for Stream Compaction on GPUs. **International Journal of Networking and Computing**, v. 7, n. 2, p. 208–226, 2017.

BALABOKHIN, A. **Automated Tool Selection and Tool Path Planning for Free-Form Surfaces in 3-Axis CNC Milling using Highly Parallel Computing Architecture**. [s.l.] University of South Carolina, 2016.

BI, Q.-Z.; WANG, Y.-H.; DING, H. A GPU-based algorithm for generating collision-free and orientation-smooth five-axis finishing tool paths of a ball-end cutter. **International Journal of Production Research**, v. 48, n. 4, p. 1105–1124, 15 fev. 2010.

BILLETER, M.; OLSSON, O.; ASSARSSON, U. Efficient stream compaction on wide SIMD many-core architectures. **Proceedings of the 1st ACM conference on High Performance Graphics - HPG ’09**, p. 159, 2009.

BLELLOCH, G. E. Prefix sums and their applications. p. 35–60, 1990.

CAMACHO, D.; CLAYTON, P.; O’BRIEN, W.; FERRON, R.; JUENGER, M.; SALAMONE, S.; SEEPERSAD, C. Applications of Additive Manufacturing in the Construction Industry - A Prospective Review. **Proceedings of the 34th International Symposium on Automation and Robotics in Construction**, n. Isarc, 1 jul. 2017.

CATCHPOLE-SMITH, S.; ABOULKHAIR, N.; PARRY, L.; TUCK, C.; ASHCROFT, I. A.; CLARE, A. Fractal scan strategies for selective laser melting of ‘unweldable’ nickel superalloys. **Additive Manufacturing**, v. 15, p. 113–122, 2017.

CHAN, K.-C.; WANG, C. C. L. Progressive segmentation for MRR-based feed-rate optimization in CNC machining. **Proceedings of the 2015 IEEE International Conference on Automation Science and Engineering (CASE)**, v. 2015- Octob, p.

691–696, ago. 2015.

CORMEN, T. **Introduction to Algorithms**. 3rd. ed. [s.l.] Massachusetts Institute of Technology, 2012.

DAS, P.; CHANDRAN, R.; SAMANT, R.; ANAND, S. Optimum Part Build Orientation in Additive Manufacturing for Minimizing Part Errors and Support Structures. **Procedia Manufacturing**, v. 1, p. 343–354, 2015.

DEHOFF, R.; DUTY, C.; PETER, W.; YAMAMOTO, Y.; CHEN, W.; BLUE, C.; TALLMAN, C. Case study: Additive manufacturing of aerospace brackets. **Advanced Materials and Processes**, v. 171, n. 3, p. 19–22, 2013.

DIMITROV, D.; SCHREVE, K.; DE BEER, N. Advances in three dimensional printing – state of the art and future perspectives. **Rapid Prototyping Journal**, v. 12, n. 3, p. 136–147, maio 2006.

DING, D.; PAN, Z.; CUIURI, D.; LI, H. A tool-path generation strategy for wire and arc additive manufacturing. **The International Journal of Advanced Manufacturing Technology**, v. 73, n. 1–4, p. 173–183, 11 jul. 2014.

DWIVEDI, R.; KOVACEVIC, R. Automated torch path planning using polygon subdivision for solid freeform fabrication based on welding. **Journal of Manufacturing Systems**, v. 23, n. 4, p. 278–291, jan. 2004.

FAUST, R.; VOLPATO, N.; MINETTO, R. A review of zigzag toolpath generation methods for additive manufacturing. **Proceedings of the 39th MATADOR Conference**, v. 1. 2018.

FONTAINE, M.; DEVILLEZ, A.; MOUFKI, A.; DUDZINSKI, D. Predictive force model for ball-end milling and experimental validation with a wavelike form machining test. **International Journal of Machine Tools and Manufacture**, v. 46, n. 3–4, p. 367–380, mar. 2006.

GARCÍA GALICIA, J. A.; BENES, B. Improving printing orientation for Fused Deposition Modeling printers by analyzing connected components. **Additive Manufacturing**, v. 22, p. 720–728, ago. 2018.

HABIB, M. A.; KHODA, B. Attribute driven process architecture for additive manufacturing. **Robotics and Computer-Integrated Manufacturing**, v. 44, n.

October 2016, p. 253–265, 2017.

HARRIS, M.; SENGUPTA, S.; OWENS, J. D. **Parallel Prefix Sum (Scan) with CUDA**. Disponível em: <https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch39.html>. Acesso em: 2 fev. 2019.

HELD, M. A geometry-based investigation of the tool path generation for zigzag pocket machining. **The Visual Computer**, v. 7, n. 5–6, p. 296–308, 1991.

HERN, A. F. Yet Another Survey on SIMD Instructions. p. 1–8, 2013.

HODGSON, G. **Slic3r Manual**. Disponível em: <https://manual.slic3r.org/expert-mode/print-settings#infill-patterns>. Acesso em: 25 nov. 2018.

HSIEH, H. T.; CHU, C. H. Particle swarm optimisation (PSO)-based tool path planning for 5-axis flank milling accelerated by graphics processing unit (GPU). **International Journal of Computer Integrated Manufacturing**, v. 24, n. 7, p. 676–687, 2011.

HUANG, G. Q. Integrating advanced computer-aided design, manufacturing, and numerical control: Principles and implementations. **International Journal of Production Research**, v. 49, n. 11, p. 3425–3426, jun. 2011.

HUANG, H.; CHEN, J.; CARLSON, B.; WANG, H.-P.; CROOKER, P.; FREDERICK, G.; FENG, Z. Stress and Distortion Simulation of Additive Manufacturing Process by High Performance Computing. **Proceedings of the ASME 2018 Pressure Vessels and Piping Conference**, p. 85045, 15 jul. 2018.

HUANG, R.; DAI, N.; LI, D.; CHENG, X.; LIU, H.; SUN, D. Parallel non-dominated sorting genetic algorithm-II for optimal part deposition orientation in additive manufacturing based on functional features. **Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science**, v. 0, n. 0, p. 1–12, 2017.

INUI, M.; UMEZU, N. Milling result visualization for assisting process planning of mold parts with complex pocket features. **Proceedings of the 2016 International Symposium on Flexible Automation (ISFA)**, p. 311–317, ago. 2016.

JANUSZIEWICZ, R.; TUMBLESTON, J. R.; QUINTANILLA, A. L.; MECHAM, S. J.;

DESIMONE, J. M. Layerless fabrication with continuous liquid interface production. **Proceedings of the National Academy of Sciences**, v. 113, n. 42, p. 11703–11708, 18 out. 2016.

JARDINI, A. L.; LAROSA, M. A.; FILHO, R. M.; ZAVAGLIA, C. A. D. C.; BERNARDES, L. F.; LAMBERT, C. S.; CALDERONI, D. R.; KHARMANDAYAN, P. Cranial reconstruction: 3D biomodel and custom-built implant created using additive manufacturing. **Journal of Cranio-Maxillofacial Surgery**, v. 42, n. 8, p. 1877–1884, 2014.

JIN, G. Q.; LI, W. D.; TSAI, C. F.; WANG, L. Adaptive tool-path generation of rapid prototyping for complex product models. **Journal of Manufacturing Systems**, v. 30, n. 3, p. 154–164, ago. 2011.

JIN, Y.; HE, Y.; FU, J.; GAN, W.; LIN, Z. Optimization of tool-path generation for material extrusion-based additive manufacturing technology. **Additive Manufacturing**, v. 1–4, p. 32–47, out. 2014.

JIN, Y.; HE, Y.; XUE, G.-H.; FU, J.-Z. A parallel-based path generation method for fused deposition modeling. **The International Journal of Advanced Manufacturing Technology**, v. 77, n. 5–8, p. 927–937, 29 mar. 2015.

JIN, Y.; HE, Y.; FU, G.; ZHANG, A.; DU, J. A non-retraction path planning approach for extrusion-based additive manufacturing. **Robotics and Computer-Integrated Manufacturing**, v. 48, n. August 2016, p. 132–144, 2017.

JIN, Y.; HE, Y.; DU, J. A novel path planning methodology for extrusion-based additive manufacturing of thin-walled parts. **International Journal of Computer Integrated Manufacturing**, v. 30, n. 12, p. 1301–1315, 2017.

JONES, R.; HAUFE, P.; SELLS, E.; IRAVANI, P.; OLLIVER, V.; PALMER, C.; BOWYER, A. RepRap – the replicating rapid prototyper. **Robotica**, v. 29, n. 1, p. 177–191, 14 jan. 2011.

KAPIL, S.; JOSHI, P.; YAGANI, H. V.; RANA, D.; KULKARNI, P. M.; KUMAR, R.; KARUNAKARAN, K. P. Optimal space filling for additive manufacturing. **Rapid Prototyping Journal**, v. 22, n. 4, p. 660–675, 20 jun. 2016.

KHAN, F. G.; KHAN, O. U.; MONTRUCCHIO, B.; GIACCONE, P. Analysis of Fast Parallel Sorting Algorithms for GPU Architectures. **Proceedings of the 2011**

Frontiers of Information Technology, p. 173–178, dez. 2011.

KIM, B. H.; CHOI, B. K. Machining efficiency comparison direction-parallel tool path with contour-parallel tool path. **CAD Computer Aided Design**, v. 34, n. 2, p. 89–95, 2002.

KIM, H. C. Optimum tool path generation for 2.5D direction-parallel milling with incomplete mesh model. **Journal of Mechanical Science and Technology**, v. 24, n. 5, p. 1019–1027, 2010.

KONOBRYTSKYI, D. Automated CNC Tool Path Planning and Machining Simulation on Highly Parallel Computing Architectures. **Automotive Engineering**, v. PhD, 2013.

KONOBRYTSKYI, D.; KURFESS, T.; TARBUTTON, J.; TUCKER, T. GPGPU Accelerated 3-Axis CNC Machining Simulation. **Proceedings of the ASME 2013 International Manufacturing Science and Engineering Conference**, p. 1–11, 10 jun. 2013.

KULKARNI, P.; MARSAN, A.; DUTTA, D. A review of process planning techniques in layered manufacturing. **Rapid Prototyping Journal**, v. 6, n. 1, p. 18–35, 2000.

LEE, C. Y.; LIN, H. S.; YAU, H. T. Using Graphic Hardware to Accelerate Pocketing Tool-Path Generation. **Applied Mechanics and Materials**, v. 311, p. 135–140, 2013.

LEE, J. W.; KANG, K. S.; LEE, S. H.; KIM, J. Y.; LEE, B. K.; CHO, D. W. Bone regeneration using a microstereolithography-produced customized poly(propylene fumarate)/diethyl fumarate photopolymer 3D scaffold incorporating BMP-2 loaded PLGA microspheres. **Biomaterials**, v. 32, n. 3, p. 744–752, 2011.

LEFEBVRE, S. Icesl : a Gpu Accelerated Csg Modeler. **Proceedings of AEFA'13, 18th European Forum on Additive Manufacturing**, n. June, p. 25–27, 2013.

LI, Z. et al. A GPU Based Parallel Genetic Algorithm for the Orientation Optimization Problem in 3D Printing*. **Proceedings of the 2019 International Conference on Robotics and Automation (ICRA)**, p. 2786–2792, maio 2019.

LIM, S.; BUSWELL, R. A.; LE, T. T.; AUSTIN, S. A.; GIBB, A. G. F.; THORPE, T. Developments in construction-scale additive manufacturing processes. **Automation in Construction**, v. 21, n. 1, p. 262–268, 2012.

LIU, X. Four alternative patterns of the Hilbert curve. **Applied Mathematics and**

Computation, v. 147, n. 3, p. 741–752, jan. 2004.

LIVESU, M.; ELLERO, S.; MARTÍNEZ, J.; LEFEBVRE, S.; ATTENE, M. From 3D models to 3D prints: an overview of the processing pipeline. **Computer Graphics Forum**, v. 36, n. 2, p. 537–564, 2017.

MOHAMED, O. A.; MASOOD, S. H.; BHOWMIK, J. L. Optimization of fused deposition modeling process parameters: a review of current research and future prospects. **Advances in Manufacturing**, v. 3, n. 1, p. 42–53, 25 mar. 2015.

MORELL-GIMÉNEZ, V.; JIMENO-MORENILLA, A.; GARCÍA-RODRÍGUEZ, J. Efficient tool path computation using multi-core GPUs. **Computers in Industry**, v. 64, n. 1, p. 50–56, 2013.

NAVARRO, C. A.; HITSCHFELD-KAHLER, N.; MATEU, L. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. **Communications in Computational Physics**, v. 15, n. 2, p. 285–329, 2014.

NVIDIA. **Geforce GTX 980 Specifications**. Disponível em: <https://www.geforce.com/hardware/notebook-gpus/geforce-gtx-980/specifications>. Acesso em: 20 ago. 2019.

PANDEY, P. M.; REDDY, N. V.; DHANDE, S. G. Real time adaptive slicing for fused deposition modelling. **International Journal of Machine Tools and Manufacture**, v. 43, n. 1, p. 61–71, 2003.

PARK, S. C.; CHOI, B. K. Tool-path planning for direction-parallel area milling. **CAD Computer Aided Design**, v. 32, n. 1, p. 17–25, 2000.

PREPARATA, F. P.; SHAMOS, M. I. **Computational Geometry**. New York, NY: Springer New York, 1985.

QIU, D.; LANGRANA, N. A. Void eliminating toolpath for extrusion-based multi-material layered manufacturing. **Rapid Prototyping Journal**, v. 8, n. 1, p. 38–45, 2002.

RAJAN, V. T.; SRINIVASAN, V.; TARABANIS, K. A. The optimal zigzag direction for filling a two-dimensional region. **Rapid Prototyping Journal**, v. 7, n. 5, p. 231–241, dez. 2001.

RANELLUCCI, A. **Slic3r: G-code generator for 3D printers**. Disponível em:

<https://slic3r.org/about/>.

RUPP, K. **42 Years of Microprocessor Trend Data**. Disponível em: <https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/>. Acesso em: 22 nov. 2018.

SAMPSON, R.; LANCASTER, R.; WESTON, M.; CENTRE, T. W. I. T.; KINGDOM, U. Melt Pool Image Process Acceleration Using General Purpose Computing On Graphics Processing Units . **Solid Freeform Fabrication Symposium**, p. 1557–1571, 2017.

SATISH, N.; HARRIS, M.; GARLAND, M. Designing efficient sorting algorithms for manycore gpus. **Proceedings of the 2009 IEEE International Parallel and Distributed Processing Symposium**, 2009.

SELVARAJ, P.; RADHAKRISHNAN, P. Algorithm for Pocket Milling using Zig-zag Tool Path. **Defence Science Journal**, v. 56, n. 2, p. 117–127, 28 abr. 2006.

STRANO, G.; HAO, L.; EVERSON, R. M.; EVANS, K. E. A new approach to the design and optimisation of support structures in additive manufacturing. **International Journal of Advanced Manufacturing Technology**, v. 66, n. 9–12, p. 1247–1254, 2013.

SURESH, K. Efficient Microstructural Design for Additive Manufacturing. **Proceedings of the ASME 2014 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference**, p. 1–9, 2014.

TANG, K.; PANG, A. Optimal connection of loops in laminated object manufacturing. **Computer-Aided Design**, v. 35, n. 11, p. 1011–1022, set. 2003.

TARABANIS, K. A. Path planning in the Proteus rapid prototyping system. **Rapid Prototyping Journal**, v. 7, n. 5, p. 241–252, 2001.

TARBUTTON, J.; KURFESS, T. R.; TUCKER, T.; KONOBRYTSKYI, D. Gouge-free voxel-based machining for parallel processors. **International Journal of Advanced Manufacturing Technology**, v. 69, n. 9–12, p. 1941–1953, 2013.

TARN, T.; CHEN, S. **Robotic Welding, Intelligence and Automation**. Cham: Springer International Publishing, 2015. v. 363

TUKORA, B.; SZALAY, T. Fully GPU-based volume representation and material removal simulation of free-form objects. **Innovative Developments in Design and Manufacturing - Advanced Research in Virtual and Rapid Prototyping**, n. January, 2010.

TUKORA, B.; SZALAY, T. Multi-dexel based material removal simulation and cutting force prediction with the use of general-purpose graphics processing units. **Advances in Engineering Software**, v. 43, n. 1, p. 65–70, 2012.

TUMBLESTON, J. R. et al. Continuous liquid interface production of 3D objects. **Science**, v. 347, n. 6228, p. 1349–1352, 20 mar. 2015.

TURNER, B. N.; GOLD, S. A. A review of melt extrusion additive manufacturing processes: II. Materials, dimensional accuracy, and surface roughness. **Rapid Prototyping Journal**, v. 21, n. 3, p. 250–261, 2015.

TYBERG, J.; HELGE BØHN, J. Local adaptive slicing. **Rapid Prototyping Journal**, v. 4, n. 3, p. 118–127, set. 1998.

UDROIU, R.; BRAGA, I. C. Polyjet technology applications for rapid tooling. **MATEC Web of Conferences**, v. 112, p. 03011, 3 jul. 2017.

UENG, S.-K.; CHEN, L.-G.; JEN, S.-Y. Voxel-based virtual manufacturing simulation for three-dimensional printing. **Advances in Mechanical Engineering**, v. 10, n. 6, 27 jun. 2018.

URBANIC, R. J.; HEDRICK, R. W.; BURFORD, C. G. A process planning framework and virtual representation for bead-based additive manufacturing processes. **The International Journal of Advanced Manufacturing Technology**, 2016.

URIONDO, A.; ESPERON-MIGUEZ, M.; PERINPANAYAGAM, S. The present and future of additive manufacturing in the aerospace sector: A review of important aspects. **Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering**, v. 229, n. 11, p. 2132–2147, 2015.

USERBENCHMARK. **CPU Compare: 4570 vs 9600k**. Disponível em: <https://cpu.userbenchmark.com/Compare/Intel-Core-i5-4570-vs-Intel-Core-i5-9600K/2770vs4031>. Acesso em: 20 ago. 2019.

VOLPATO, N.; GALVÃO, L. C.; NUNES, L. F.; SOUZA, R. I.; OGUIDO, K. Combining

heuristics for tool-path optimisation in material extrusion additive manufacturing. **Journal of the Operational Research Society**, p. 1–11, 16 maio 2019.

VOLPATO, N.; DA SILVA, J. V. L. Planejamento de processo para tecnologias de AM. In: **Manufatura aditiva: tecnologias e aplicações da impressão 3D**. [s.l.] Edgar Blucher, 2017.

VOLPATO, N.; FOGGIATTO, J. The development of a generic Rapid Prototyping process planning system. In: **Innovative Developments in Design and Manufacturing**. [s.l.] CRC Press, 2009.

WANG, A.; ZHOU, C.; JIN, Z.; XU, W. Towards scalable and efficient GPU-enabled slicing acceleration in continuous 3D printing. **Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC**, p. 623–628, 2017.

WANG, J.; LUO, M.; ZHANG, D. A GPU-Accelerated Approach for Collision Detection and Tool Posture Modification in Multi-Axis Machining. **IEEE Access**, v. 6, p. 35132–35142, 2018.

WONG, K. V.; HERNANDEZ, A. A Review of Additive Manufacturing. **ISRN Mechanical Engineering**, v. 2012, p. 1–10, 2012.

YAKOUBI, M. A.; LASKRI, M. T. The path planning of cleaner robot for coverage region using Genetic Algorithms. **Journal of Innovation in Digital Ecosystems**, v. 3, n. 1, p. 37–43, 2016.

YANG, Y.; LOH, H. T.; FUH, J. Y. H.; WANG, Y. G. Equidistant path generation for improving scanning efficiency in layered manufacturing. **Rapid Prototyping Journal**, v. 8, n. 1, p. 30–37, 2002.

YAO, Z. A novel cutter path planning approach to high speed machining. **Computer-Aided Design and Applications**, v. 3, n. 1–4, p. 241–248, 2006.

ZHANG, W.; MEHTA, A.; DESAI, P. S.; III, C. F. H. Machine Learning Enabled Powder Spreading Process Map for Metal Additive Manufacturing (Am). **Solid Freeform Fabrication Symposium**, p. 1235–1249, 2017a.

ZHANG, X.; XIONG, G.; SHEN, Z.; ZHAO, Y.; GUO, C.; DONG, X. A GPU-based parallel slicer for 3D printing. **IEEE International Conference on Automation Science and Engineering**, v. 2017- Augus, p. 55–60, 2017b.

ZHANG, Y.; GUPTA, R. K.; BERNARD, A. Two-dimensional placement optimization for multi-parts production in additive manufacturing. **Robotics and Computer-Integrated Manufacturing**, v. 38, p. 102–117, 2016.

ZHAO, D.; LI, M.; LIU, Y. Self-supporting Topology Optimization for Additive Manufacturing. **Arxiv**, 24 ago. 2017.

ZHAO, H. et al. Connected fermat spirals for layered fabrication. **ACM Transactions on Graphics**, v. 35, n. 4, p. 1–10, 2016.