UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA MECÂNICA

UMA PROPOSTA DE SISTEMATIZAÇÃO DO PROCESSO DE PLANEJAMENTO DE TRAJETÓRIAS PARA O DESENVOLVIMENTO DE TAREFAS DE ROBÔS MANIPULADORES

Dissertação submetida à

UNIVERSIDADE FEDERAL DE SANTA CATARINA

para a obtenção do grau de

MESTRE EM ENGENHARIA MECÂNICA

CRISTIANE PESCADOR TONETTO

Florianópolis, março de 2007.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA MECÂNICA

UMA PROPOSTA DE SISTEMATIZAÇÃO DO PROCESSO DE PLANEJAMENTO DE TRAJETÓRIAS PARA O DESENVOLVIMENTO DE TAREFAS DE ROBÔS MANIPULADORES

CRISTIANE PESCADOR TONETTO

Esta dissertação foi julgada adequada para a obtenção do título de
MESTRE EM ENGENHARIA
ESPECIALIDADE ENGENHARIA MECÂNICA
sendo aprovada em sua forma final.

_____

Altamir Dias, D.Sc. – Orientador

_____

Fernando Cabral, Ph.D. – Coordenador do Curso

BANCA EXAMINADORA

_____

Raul Guenther, D.Sc. – Presidente

_____

Antônio Carlos Ribeiro Nogueira, D.Sc.

_____

Daniel Martins, Dr.Eng.

Aos meus pais, Humberto e Olívia.

À minha irmã, Cláudia.

Ao meu namorado, Mathias.

AGRADECIMENTOS

A Deus, acima de tudo!

Aos meus pais Humberto e Olívia e à minha irmã Cláudia, por todo carinho e amor que dedicam a mim e por acreditarem na realização desse sonho.

Ao meu namorado, Mathias, por seu amor, carinho e dedicação em todos os momentos.

Ao professor e orientador Altamir Dias, sempre indicando a direção nos momentos confusos encontrados durante este trabalho de pesquisa, gostaria de deixar meus sinceros agradecimentos pela dedicação, conhecimento compartilhado e incentivo. Muito obrigada pela confiança.

Aos professores Antônio Carlos Ribeiro Nogueira, Daniel Martins e Raul Guenther, pela dedicação e disponibilidade ao lerem o trabalho. Muito obrigada pelas contribuições.

Aos meus colegas e amigos Antônio Dourado, Cassiano Guerra, Cindy Ibarra, Felipe Barata, Francisco Santos, Luiz Ribeiro, Renato Bodanese, Tiago Pinto, pelo incentivo, ajuda e discussões que foram fundamentais durante a realização desse trabalho. Muito obrigada a todos e sucesso em seus caminhos.

À minha grande amiga Edilênia Frezza, cuja amizade não se desfez apesar da distância e do tempo. Agradeço pela amizade e peço desculpas pelos momentos de ausência.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), por ter financiado este projeto pelo período de dois anos.

Sumário

Lista de Figuras

Lista de Tabelas

Simbologia

$d(p, q)$	região de dominância de p sobre q
$\delta(x, p)$	distância Euclidiana entre x e p
$M(p, q)$	mediatriz de p e q
R_V	região de Voronoi
$V(S)$	diagrama de Voronoi
D	triangularização de Delaunay
$\overline{AB}$	segmento de reta passando pelos pontos A e B
$P(t)$	segmento de reta paramétrico
e_i	aresta i
N_i	normal referente à aresta e_i
P_{e_i}	ponto qualquer pertencente à aresta e_i do polígono
t	parâmetro da reta
V	conjunto de vértices
$\overline{CD}$	segmento de reta passando pelos pontos C e D
t_E	parâmetro de entrada
t_S	parâmetro de saída
$P(u)$	vetor posição de um ponto qualquer no segmento de curvas de forma-livre
$B_{n,i}(u)$	função base de Bernstein
$C(n, i)$	coeficiente binomial
$N_{i,k}$	funções base definidas pelas fórmulas de recursão de Boor e Cox
$r_i(u)$	segmento de curva B-Spline cúbico
u	parâmetro da superfície
w	parâmetro da superfície
$Q_{i,j}$	conjunto de vértices do poliedro de controle na direção do parâmetro u
$V_{i,j}$	conjunto de vértices do poliedro de controle na direção do parâmetro w
C	centro da ferramenta
R	raio da ferramenta
T	ponta da ferramenta
P	posição de contato da ferramenta com a superfície

f	avanço
L	passo
h	altura da rugosidade
Δu	incremento no parâmetro u da curva
δ	erro de aproximação

Siglas

CAD	*Computer Aided Design*
CAM	*Computer Aided Manufacturing*
CMM	*Coordinate-Measuring Machine*
NC	*Numerical control* ou *numerically controlled*
ASCII	*American Standard Code for Information Interchange*
STEP	*Standard for the Exchange of Product model data*
IGES	*Initial Graphics Exchange Specification*
GIS	*Geographic Information System*
PE	Potencialmente de Entrada
PS	Potencialmente de Saída
NURBS	B-Splines racionais não uniformes
JDK	*Java(TM) Development Kit*
JRE	*Java Runtime Environment*
VRML	*Virtual Reality Modeling Language*
GLP	*General Public License*
UFSC	Universidade Federal de Santa Catarina
UML	*Unified Modeling Language*

Resumo

O processo de planejamento de trajetórias é uma atividade comum em operações de fabricação, que incluem soldagem, pintura, fresamento, entre outras, tanto com o uso de equipamentos de controle de operações por comando numérico (NC) quanto com robôs manipuladores. Este trabalho propõe um estudo do processo de planejamento de trajetórias para tarefas em ambientes robóticos. Para o caso em que a ferramenta descreve uma trajetória sobre a superfície, o processo de planejamento de trajetórias envolve duas etapas principais: a descrição da superfície de interesse e a definição precisa da tarefa através do cálculo da trajetória sobre a superfície.

A descrição da superfície pode ser obtida a partir de uma nuvem de pontos ou de um arquivo gerado em sistemas CAD/CAM. Neste trabalho, a pesquisa é focada na reconstrução de superfícies a partir de nuvens de pontos. Para isso, são utilizados métodos de geração de malhas como o diagrama de Voronoi e o seu dual, a triangularização de Delaunay. Esses métodos permitem obter uma representação topológica e ordenada da nuvem de pontos. Assim, é possível se criar um conjunto de segmentos de superfícies B-Splines para obter uma representação matemática mais precisa e melhor tratável do ponto de vista do planejamento de trajetórias. Isso permite que propriedades de derivadas e curvaturas sejam melhor descritas para a superfície, tornando possível manipulações e modificações.

A partir da descrição da superfície, pode-se trabalhar no planejamento da tarefa. Para a abordagem utilizada no planejamento de trajetórias, é definida a trajetória no espaço de trabalho do robô, indicando o caminho a ser seguido pela ferramenta sobre a superfície. A metolodogia do cálculo da trajetória utiliza informações sobre o avanço e passo da ferramenta para calcular as trajetórias sobre a superfície a ser recuperada.

Para se concretizar o trabalho de pesquisa, foi implementado um aplicativo com um conjunto de algoritmos que reconstrói a superfície de forma-livre, o caminho a ser percorrido pelo manipulador do robô para a execução da tarefa, incluindo a visualização 3D da nuvem de pontos, do diagrama de Voronoi e da triangularização de Delaunay. Além disso, a superfície reconstruída pode ser visualizada tanto em *wireframe* quanto sombreada. O aplicativo foi implementado, utilizando-se a linguagem de programação JAVA.

Abstract

The trajectory planning process is a common activity in manufacturing operations, such as welding, painting, milling, and others, used frequently to control equipments with numerical command (NC), as well as with robot manipulators. This work proposes an approach of the trajectory planning process for tasks within robotic environments. In particular, the trajectory planning process for a tool movement over surfaces contains two main steps: the interest surface description and the accurate definition of the task through trajectory computation over the surface.

The surface description can be obtained from a points cloud or from a file generated in CAD/CAM system. In this work, the research focused the surface reconstruction from points cloud. For this purpose, mesh generation methods were used, as Voronoi diagram and its dual, the Delaunay triangulation. These methods make possible to obtain a topological and ordered representation of the points cloud. This way, it was possible to create a set of B-Splines patches to get an accurater mathematical representation according to the trajectory planning point of view. This representation allows derivates and curvatures to be described for the surface, making manipulations and modifications possible.

From a surface description it is possible to work in the task planning. The approach used for the trajectory planning was to define the trajectory in the robot workspace, defining the path to be followed by the tool over the surface. The methodology of the trajectory computation through paralel paths uses information about the forward step and side step of the tool to compute the trajectories over the surface to be remanufactured.

To reinforce the work of research an application was implemented, with a set of algorithms that reconstruct a free form surface, the path to be followed by the robot manipulator to execute the task and 3D visualization of the points cloud, Voronoi diagram and Delaunay triangularization. Besides this, the reconstructed surface can be visualizated as wireframe or as shaded model. The application was implemented using the JAVA programming language.

1 Introdução

Diante de um mercado cada vez mais competitivo e com a necessidade de oferecer produtos com baixo custo, alta qualidade e em menor tempo, as indústrias têm investido em novas estratégias para que o processo de produção atenda aos seus objetivos e aos do consumidor.

Para isso, tecnologias, como o projeto e a manufatura assistidos por computador (CAD/CAM) são amplamente utilizadas nas indústrias nas fases de planejamento de projeto e manufatura de produtos. Essas tecnologias devem ser cada vez mais automatizadas e interligadas para que o processo ocorra de forma mais veloz e eficiente.

Dessa forma, os robôs industriais são considerados elementos importantes no ramo industrial por agilizarem algumas das tarefas do processo de fabricação de produtos, principalmente, dentro de uma abordagem automatizada.

Várias referências existem na literatura sobre métodos computacionais para o desenvolvimento de produtos em forma de planejamento de tarefas executadas por robôs industriais ou comandos numéricos [3, 4, 7, 11, 17, 16, 29, 43, 45, 50]. A ênfase das referências está no planejamento do movimento do manipulador ou na programação da execução de tarefas. Porém, existem poucas referências que contemplem detalhes do processo completo, desde a busca de informações sobre a tarefa a ser executada até a programação dos controladores que comandam a estrutura mecânica.

1.1 Objetivo da dissertação

O objetivo principal deste trabalho é organizar sistematicamente o processo de planejamento de trajetórias para o desenvolvimento de tarefas para robôs manipuladores, propondo-se um modelo teórico e implementando as suas etapas visando exemplificar uma aplicação real.

Será desenvolvido um protótipo computacional para a geração de caminhos sobre a

superfície de trabalho. A implementação deste protótipo envolve o estudo de:

- procedimentos, métodos e técnicas para organizar e transformar topologicamente a nuvem de pontos representativa da superfície de interesse;
- representações de superfícies de forma livre;
- técnicas já desenvolvidas para a geração do caminho sobre a superfície;
- técnicas computacionais de recortes de retas;
- linguagens de programação e padrões de implementação.

Por fim, será desenvolvida uma interface que proporcionará a interação do usuário com os algoritmos desenvolvidos.

1.2 Estrutura do texto

O trabalho foi elaborado em cinco partes, expostos ao longo dos capítulos desta pesquisa.

No capítulo 2, são apresentadas as características gerais da estrutura de robôs manipuladores, a descrição de alguns métodos utilizados no processo de planejamento de trajetórias e na reconstrução de superfícies.

No capítulo 3, são detalhadas as ferramentas que serão utilizadas no desenvolvimento de um protótipo para a geração de caminhos da ferramenta sobre a superfície. As principais ferramentas descritas são a teoria do diagrama de Voronoi, a triangularização de Delaunay e sua relação com o diagrama de Voronoi, a técnica de recorte de Cyrus-Beck, as superfícies B-Splines e as definições necessárias para a geração do caminho da ferramenta.

No capítulo 4, são descritas em detalhes as etapas da proposta de sistematização do processo de planejamento de trajetórias e o desenvolvimento de um programa que envolve as etapas da proposta de sistematização. Essa descrição ocorre através de diagramas de atividades, de ferramentas comentadas no capítulo 3 e de pseudo-algoritmos.

No capítulo 5, é apresentada a relação entre a proposta de sistematização e o programa desenvolvido, também é exemplificada a instalação, utilização e aplicação do programa para alguns casos reais. Ao fim, é apresentada a estrutura do código-fonte do protótipo.

Finalmente, no capítulo 6, são feitas as considerações finais sobre o estudo realizado e a proposta apresentada e os comentários sobre os resultados obtidos. São analisadas também algumas limitações e recomendações para melhorias do protótipo, e dadas sugestões para trabalhos futuros relacionados ao processo de planejamento de trajetórias de robôs manipuladores.

No apêndice A, é fornecido um modelo de conjunto de pontos representativos de uma superfície com 36 pontos, estruturado de acordo com o formato aceito pelo programa (o conjunto foi fornecido pelo projeto Roboturb-UFSC), e um cenário para a utilização do programa.

No apêndice B, está a licença GPL do protótipo desenvolvido.

O apêndice C contém o diagrama de classes com as principais classes implementadas.

2 Revisão bibliográfica

Neste capítulo, é apresentada uma revisão sobre o processo de planejamento de trajetórias de robôs manipuladores. Nele, será vista a estrutura robótica, enfatizando a programação e planejamento de trajetórias, incluindo os métodos de geração de caminhos e reconstrução de superfícies. A revisão possibilita a estruturação das informações necessárias para o desenvolvimento do modelo do processo de planejamento de trajetórias.

Durante a etapa de planejamento de trajetórias, é que são definidos todos os detalhes do movimento a ser executado pelo robô para a realização da tarefa. Esses detalhes consistem em determinar quais pontos o efetuador deve seguir, além de informações como as velocidades e acelerações em cada ponto do percurso, tanto para o efetuador final quanto para cada uma das juntas que o robô possui. São necessárias também informações sobre o material em que será realizada a tarefa e quais as ferramentas serão utilizadas.

O processo de planejamento de trajetórias é que permite que a estrutura do robô efetue uma tarefa, portanto, sem o planejamento de trajetórias, nenhuma tarefa é executada, e uma trajetória bem definida contribui para uma tarefa realizada com sucesso.

2.1 Robótica

Durante as últimas décadas, vários esforços foram realizados visando melhorar a qualidade das condições de trabalho e a eficiência no sistema de produção. A automação é um item muito importante nesse aspecto, pois permite melhorar a performance do processo industrial.

Segundo Groover [24], a automação industrial é definida como uma tecnologia que se ocupa do uso de sistemas mecânicos, eletrônicos e à base de computadores na operação e controle da produção. Algumas dessas tecnologias são as linhas de transferência de manufatura, máquinas de montagem mecanizadas, sistemas de controle de realimentação, máquinas operatrizes dotadas de comandos numéricos e robôs.

A automação é freqüentemente caracterizada por dois princípios importantes: a mecanização, na qual as máquinas são reguladas com os requisitos necessários para o processo de produção e a automação de processos contínuos, que trata do controle de grandezas que variam continuamente com o tempo.

Groover [24] comenta que existem três amplas classes de automação industrial: fixa, programável e flexível. A automação fixa é utilizada quando o volume de produção de um mesmo produto é muito elevado, pois, apesar da grande produtividade, ela apresenta pouca possibilidade de variedade. A automação programável é usada, quando o volume de produção é relativamente baixo, e há uma variedade de produtos a serem fabricados. Já a automação flexível é mais adequada para a produção de médio volume, possuindo características tanto da automação fixa quanto da programável, e deve ser programada para diferentes tipos de produto, sendo a sua variedade normalmente mais limitada do que a automação programável. Desses três tipos de automação, a automação programável é caracterizada pela aplicação da robótica. A característica de programação do robô permite que ele seja usado para uma variedade de diferentes operações industriais, muitas das quais envolvem a interação dele com outros equipamentos automatizados. A Figura 1, que foi adaptada de Groover [24], ilustra a relação entre essas três classes em função do volume de produção e da variedade de produtos.

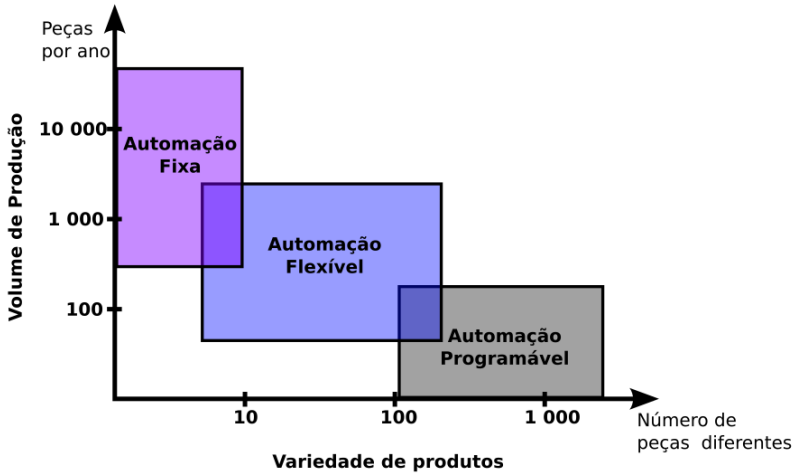


Figura 1: Relação entre automação fixa, programável e flexível em função do volume de produção e variedade de produtos [24].

A automação industrial aplica extensivamente robôs para a produção em massa, na qual as tarefas devem ser executadas repetidamente e exatamente da mesma forma. Existem várias definições para robôs industriais [38], dentre elas, a definição do *Robot Institute of America* (RIA): "um robô é um manipulador (ou mecanismo) reprogramável,

multi-funcional projetado para mover materiais, peças, ferramentas, ou dispositivos especiais através de movimentos variados programados para execução de uma variedade de tarefas". A *Japanese Industrial Robot Association* (JIRA) define mais detalhadamente, especificando em cinco níveis diferentes os robôs industriais:

- Manipuladores, que são operados diretamente por seres humanos;
- Seqüenciais, que se dividem em duas subcategorias, que são robôs de seqüência fixa ou variável;
- *Playback*, que executa instruções fixas ensinadas;
- Comando Numérico (NC), que executa informação carregada numericamente;
- Inteligentes, que têm seus próprios sistemas sensoriais, que ajudam os programas a tomar decisões em tempo real.

Robôs são considerados como representantes típicos de sistemas mecatrônicos, que integram aspectos de manipulação, sensoriamento, controle e comunicação. Esse conjunto de fatores permite com que os robôs possam ser programados para executar uma variedade de movimentos dentro das restrições exigidas pelo processo de produção.

O termo **mecatrônica** pode ser definido como uma combinação centrada na Engenharia Mecânica, Engenharia Elétrica e Engenharia de Software, que tornam possível a geração mais confiável e simples de sistemas.

Dessa forma, um robô industrial pode ser dividido em três principais componentes [51]: mecanismos, acionamento e sistema de controle (conforme mostrado na Figura 2):

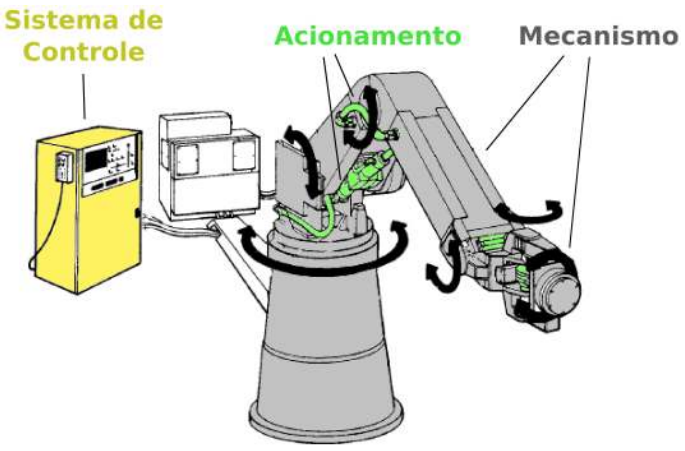


Figura 2: Principais componentes de um robô industrial.

O **mecanismo** é um sistema de corpos designado a converter movimentos de, e forças em, um ou mais corpos dentro das restrições de movimento de, e forças em, outros corpos [25]. O mecanismo é composto de elos interligados por juntas numa estrutura em cadeia cinemática aberta (manipulador serial) ou fechada (manipulador paralelo), ou ainda em uma estrutura mista (manipulador híbrido) [51].

O **acionamento** (ou atuador) é responsável pela aplicação da força ou torque necessária para movimentação adequada do mecanismo. O acionamento pode ser hidráulico, pneumático ou elétrico. Pode também incluir a unidade de potência e o sistema de transmissão [51].

O **sistema de controle** é um dispositivo de *hardware* e *software* no qual é implementado o esquema de controle projetado, incluindo os sensores, os elementos do modo de operação, o conjunto de circuitos e os elementos de saída para atuação [51].

Os robôs são úteis em uma variedade de aplicações na indústria, tais como manipulação de objetos, pintura, soldagem, inspeção e montagem, geralmente relacionados com a movimentação de uma estrutura articulada em um espaço de trabalho. O **espaço de trabalho** do robô é definido como o envelope ou espaço dentro do qual o robô move o efetuador, sendo dado em função do número de juntas do manipulador, do tamanho físico das juntas e do alcance das várias juntas [24].

Dentro do espaço de trabalho, o robô é designado a realizar **tarefas** que são definidas como uma ou mais seqüências de instruções tratadas por um programa de controle como um elemento de trabalho a ser executado.

Apesar da variedade de manipuladores e de suas aplicações, o uso mais comum dos robôs industriais ainda é caracterizado pela execução repetitiva de uma seqüência de movimentos preestabelecidos. De modo geral, pode ser afirmado que as aplicações de robôs manipuladores visam principalmente às tarefas tediosas, repetitivas e perigosas, ou que exigem perícia, força e destreza além da capacidade humana [13].

Do ponto de vista da interação com o ambiente, as aplicações podem ser divididas em duas categorias [13]:

- as que se caracterizam por não terem uma interação com o ambiente, sendo suficiente para elas controlar o movimento do efetuador final, fazendo-o seguir uma trajetória pré-especificada (caso típico da pintura, soldagem);
- as que apresentam interação com o ambiente (montagem e retificação), demandando,

além do controle do movimento, também o controle da força aplicada (sensores de força).

Em algumas aplicações robóticas é freqüentemente necessário que o manipulador siga o caminho planejado entre pontos objetivos, como no caso de transporte de objetos de um ponto a outro. Já em outras aplicações, como pintura e soldagem, é necessário que o manipulador siga a forma do objeto no qual está trabalhando, principalmente quando é preciso que o robô evite obstáculos entre pontos objetivos.

2.1.1 Planejamento de tarefas robóticas

Lozano-Peres *apud* Ranky [38] divide a função de planejamento de tarefas robóticas em três partes: modelagem do espaço de trabalho[1], especificação da tarefa e síntese de programação do manipulador. De fato, isso pode ser estruturado seqüencialmente conforme ilustra a Figura 3. O aspecto de modelagem do espaço de trabalho tenta encontrar a informação física do ambiente usando descrições matemáticas ou relacionais. As especificações das tarefas designam os estados de paradas ou movimentos para garantir que a tarefa seja executada completamente. A síntese de programação prescreve o comportamento do manipulador em relação à tarefa objetivo e aos sensores de entradas.

Figura 3: Planejamento robótico.

2.1.1.1 Modelagem do espaço de trabalho

No estágio de modelagem do espaço de trabalho do planejamento da tarefa, vários parâmetros são obtidos para definir os objetos que fazem parte do ambiente robótico e seus comportamentos. Dentre os objetos de interesse, são incluídos as superfícies das peças que sofrerão interação direta, os obstáculos a serem evitados e o manipulador em si. Esses parâmetros de definição de modelagem do espaço de trabalho podem ser descritos em cinco grupos:

[1]Alguns autores utilizam o termo: Modelagem do Mundo

- a descrição da geometria dos robôs e das peças no ambiente de tarefa, fornecendo informações relevantes ao estado do ambiente, pode ser obtida utilizando-se modelos de sistemas CAD, representações de folhas de engenharia ou outras técnicas;
- a descrição física de todas as peças e do ambiente de trabalho que leva em conta propriedades básicas como: massa, inércia, coloração, textura e estrutura do material;
- as descrições de cinemática de todas as relações entre as peças e robôs presentes no ambiente;
- as descrições de várias características robóticas, tais como limites das juntas e acelerações;
- as configurações das peças e robôs (estado do ambiente) e suas incertezas.

A modelagem do espaço de trabalho possibilita o uso de sistemas robóticos para operações no espaço de trabalho e fornece conhecimento do ambiente para o robô, o que é necessário para a realização da interação, ou seja, para que ele pegue um objeto ou evite colisões enquanto se move.

A maior desvantagem em controlar robôs, através de modelagem do espaço de trabalho, é que não existem modelos geométricos para muitos objetos. A geração desses modelos usando ferramentas de projeto CAD aumenta grandemente o tempo necessário para completar uma tarefa com o robô.

Os modelos do espaço de trabalho são tradicionalmente gerados usando ferramentas CAD. Mas geralmente os projetos não encaixam com as construções reais, ou seja, são necessárias modificações ou adições de características. Conseqüentemente, o modelador deve levar em consideração as distâncias ou localizações, ou então construir o modelo peça por peça. A modelagem CAD do espaço de trabalho é uma operação manual e consome tempo, podendo levar dias. Muitos dos detalhes como objetos com superfícies gastas, amassadas e modificadas são de difícil reconstrução manual.

Uma proposta de modelagem do espaço de trabalho é comentado por Little [29] em seu artigo. Ele desenvolve um procedimento para construir rapidamente modelos virtuais do espaço de trabalho do mundo real que permite o controle do robô em sua tarefa. Para isso, ele divide o processo nas seguintes etapas:

1. **Registro** – os sensores de coleta dos dados 3D necessitam ser calibrados para garantir a exatidão dos dados e saber onde estavam na hora da coleta dos dados;

2. **Coleta de dados 3D** – um sensor é usado para capturar os dados da superfície sobre uma área. Com informação suficiente sobre a escala do sensor, tais dados podem ser transformados em um sistema de coordenadas (x, y, z);

3. **Filtragem** – geralmente é necessária para a redução da quantidade de dados e realce da qualidade;

4. **Edição e Segmentação** – em geral, existem dados dentro da varredura que não são de interesse. Freqüentemente, dados falsos aparecem devido às limitações dos sensores com superfícies reflexivas, então, é geralmente vantajoso removê-los. A segmentação divide o espaço de trabalho em subseções, que podem ser manipuladas de forma independente;

5. **Triangularização** – os dados são processados para formar uma superfície triangularizada. Esse processo pode ser trivial quando os dados são coletados em uma grade uniforme. Para dados dispersos, outras técnicas precisam ser aplicadas, tais como a triangularização de Delaunay;

6. **Decimação** – devido ao grande número de dados de pontos de entrada, o resultado é uma grande lista de triângulos. Nesse passo, a técnica chamada de decimação reduz o número de triângulos pela remoção dos vértices;

7. **Construção da geometria primitiva** – um objeto de geometria simples pode ser utilizado como modelo colocando-se sobre ele os segmentos de dados espúrios (que diferem a geometria simples do objeto real). A geometria primitiva é formada a partir da composição desses objetos de geometria simples, tendo menos requisitos de dados do que as superfícies triangularizadas.

Assim, Little [29] conclui que, através de sua técnica, é possível rapidamente criar e registrar modelos de superfícies arbitrárias para incorporá-los dentro de um sistema robótico.

2.1.1.2 Especificação da tarefa

No processo de especificação da tarefa, é definida qual será executada e são analisados os requisitos para a sua execução. A seqüência executada pode permitir alguma liberdade de escolha da ordem de execução das subtarefas, incluindo a possibilidade de execuções simultâneas, sujeitas às várias condições ou eventos ocorridos no ambiente da tarefa [38].

Algumas das principais tarefas industriais são:

- transporte;
- soldagem;
- usinagem;
- retificação e polimento;
- pintura;
- montagem;
- inspeção.

Das tarefas citadas acima, todas podem ser executadas por robôs, no entanto, algumas, como retificação e polimento, apresentam dificuldades na sua realização devido à necessidade do robô aplicar uma força controlada no objeto de trabalho. O controle de força em robôs é considerado ainda uma questão de pesquisa em aberto.

A maioria dessas atividades industriais estão relacionadas com o contorno do produto ou com o deslocamento de um ponto objetivo a outro, dessa forma um manipulador precisa realizar um movimento para a execução da tarefa.

O movimento do efetuador final pode ser considerado caminho ponto-a-ponto ou caminho contínuo. No caminho ponto-a-ponto, o efetuador final tem que se mover de um ponto inicial a um ponto final em um intervalo de tempo pré-determinado. Nesse caso, o caminho seguido pelo efetuador final entre os pontos, inicial e final, não precisa estar precisamente definido, mas deve ser considerado a fim de evitar colisões no espaço de trabalho. Quando é necessário se definir precisamente os pontos intermediários, o caminho é chamado de caminho contínuo [9, 45].

Algumas das tarefas realizadas por robô exigem o movimento por caminho contínuo, em que o controle de velocidade e aceleração são importantes em todos os pontos do caminho. Esse é o caso de alguns processos como usinagem, soldagem à velocidade constante, pintura, retificação e polimento. Já em operações como transporte, montagem e inspeção, os pontos iniciais e finais são os que devem ser considerados na execução do movimento.

As tarefas que exigem o movimento por caminho contínuo necessitam de um planejamento mais detalhado, pois são necessários alguns parâmetros adicionais do processo, como a qualidade do acabamento, quando for o caso de usinagem, ou na pintura, em que existe a necessidade de recobrir uma superfície, de forma uniforme, pela tinta, sem regiões com acúmulo ou falta da mesma.

Por fim, é possível se concluir que existe uma característica comum em todas as tarefas robóticas que é a necessidade de execução de movimento. Tal movimento está relacionado com a peça e com o ambiente de trabalho, podendo a peça ser detalhada pelas superfícies que a compõem.

2.1.1.3 Programação do manipulador de robôs

A programação do robô pode ocorrer **em linha**[2] ou **fora de linha**[3]. A programação em linha é feita diretamente no robô, enquanto a programação fora de linha se faz fora da célula de trabalho para depois ser transferida para o robô em questão, sendo a vantagem potencial desse método que a programação possa ser realizada sem a retirada do robô da produção [14]. Com a programação fora de linha, é possível dar entrada em todo o programa em um computador para carregamento posterior no robô, o que aceleraria a mudança de um ciclo de trabalho para um novo ciclo sem parada da produção para a reprogramação.

Um programa de robô pode ser definido a partir da descrição de uma trajetória no espaço através da qual o manipulador realiza um movimento comandado [24]. Existem vários métodos usados para programar um robô (conforme Figura 4), dos quais duas categorias podem ser consideradas de maior importância: a programação por aprendizagem e a programação por linguagem textual.

Os métodos de programação por aprendizagem são considerados programação em linha, ou seja, é necessário que o robô pare de realizar sua função antiga enquanto é ensinada a nova função através das técnicas de aprendizagem. Já os métodos de programação textual são geralmente considerados programação fora de linha, sendo necessário parar o robô para realizar a transferência do código da nova tarefa.

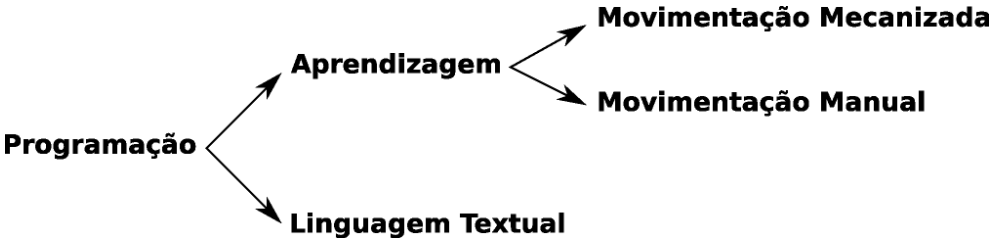


Figura 4: Métodos de programação de robôs.

A programação por aprendizagem consiste em o programador deslocar o braço do

[2]Termo utilizado em inglês: *on-line*
[3]Termo utilizado em inglês: *off-line*

robô na seqüência de movimentos requerida e registrar os movimentos na memória do controlador. Esse procedimento é geralmente utilizado para programar robôs de repetição e é conhecido como método de "ensino por demonstração".

Existem duas maneiras de se realizar a programação por aprendizagem: a movimentação mecanizada e a movimentação manual. O método de movimentação mecanizada consiste em usar um "*teach pendant*"[4], semelhante a um *"joystick"* [5], para controlar os movimentos das juntas do manipulador em uma série de pontos no espaço. Em seguida, cada ponto é registrado na memória para repetição subseqüente durante o ciclo de trabalho. O método de movimentação manual é mais usado em programação de caminho contínuo, que envolve caminhos curvilíneos suaves do manipulador. Nesse método, o programador segura o braço do robô, movimenta-o e segue o caminho desejado para o ciclo de trabalho. Caso o robô possua grandes dimensões e difícil movimentação física, ele é substituído por um aparelho programador especial que tenha a mesma geometria, mas que seja mais fácil de manipular durante a programação. Os pontos do movimento são registrados na memória do controlador para serem reproduzidos durante a execução do ciclo de trabalho[14].

As desvantagens que podem ser citadas sobre a programação por aprendizagem são:

- o robô não pode ser usado na produção enquanto estiver sendo programado;
- se houver um aumento na complexidade do caminho a ser percorrido pelo manipulador, torna-se mais difícil realizar a programação por aprendizagem usando os métodos disponíveis;
- incompatibilidade com sistemas CAD/CAM, redes de comunicação de dados e sistemas de informação de manufatura.

Na programação textual, utiliza-se uma linguagem de programação, semelhante às utilizadas no desenvolvimento de programas computacionais, que estabelece a lógica e a seqüência do ciclo de trabalho. Um terminal de computador é usado para dar entrada nas instruções de programa ao controlador. É necessário também pontos previamente definidos da trajetória a ser percorrida.

[4]Ferramenta de ensino.
[5]Dispositivo semelhante a um manche eletrônico.

2.1.2 Cinemática de robôs manipuladores

A cinemática de manipuladores trata dos movimentos do efetuador e de como realizá-los através dos movimentos coordenados das juntas. Os movimentos do efetuador são definidos no espaço denominado **operacional** ou espaço da **tarefa**. O espaço operacional pode ser descrito convenientemente por diferentes tipos de sistemas de coordenadas, tais como cartesiano, polar, esférico e cilíndrico. Por outro lado, o espaço das juntas representa aquele no qual o vetor das variáveis de juntas é definido. A essência do problema da cinemática de manipuladores é a coordenação dos movimentos individuais das juntas em seu espaço e o movimento do efetuador no espaço operacional [6].

Dois problemas comuns na cinemática de manipuladores são a cinemática **direta** e a cinemática **inversa** (Figura 5). Segundo Sciavicco [45], a cinemática direta permite que a orientação e a posição do efetuador final sejam expressas como uma função das variáveis de juntas da estrutura mecânica com respeito a um sistema de coordenadas. Já a cinemática inversa consiste na determinação das variáveis de juntas correspondentes a uma dada posição e orientação do efetuador final. A solução desse problema é de importância fundamental a fim de transformar as especificações do movimento, designadas para o efetuador final no espaço operacional em movimentos no espaço das juntas, permitindo a execução do movimento desejado.

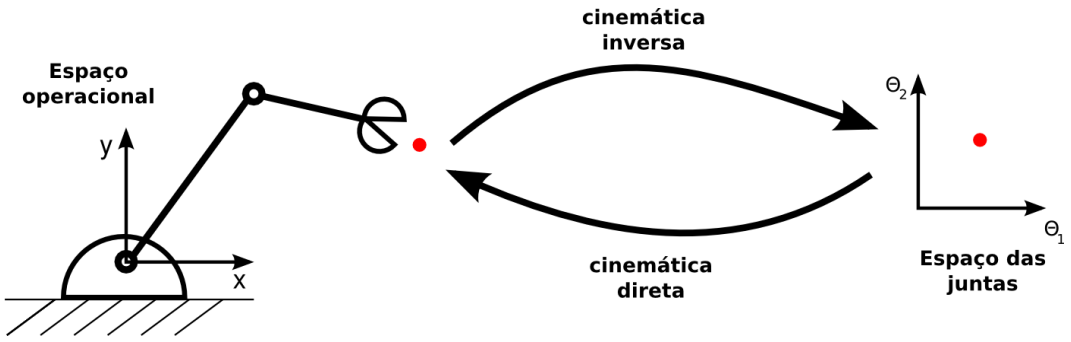


Figura 5: Cinemática direta e inversa.

Com respeito à cinemática direta, o cálculo da posição do efetuador final é obtido de maneira única, uma vez que são conhecidas as variáveis de juntas. Por outro lado, o problema da cinemática inversa é mais complexo, pois as equações que precisam ser resolvidas geralmente são não-lineares, podendo existir múltiplas, infinitas ou mesmo não existir solução para as equações.

2.2 Planejamento de trajetórias

A finalidade do planejamento da trajetória é controlar o percurso entre os pontos escolhidos de modo a produzir o deslocamento desejado, de maneira suave, de forma que seja necessário gerar dados de referência para o sistema de controle de movimento das juntas para assegurar que o manipulador execute as trajetórias planejadas.

O requisito mínimo desejado do efetuador de um robô é que ele tenha a capacidade de movimentar-se de uma posição inicial até a posição final desejada. Essa transição deve ser feita seguindo algumas restrições determinadas pelo sistema do robô e pelos processos de fabricação como as citadas a seguir [45, 59]:

- devem ser evitados regiões de singularidades no movimento das juntas do robô;
- o movimento do robô não pode ultrapassar os limites do intervalo (de operação) das juntas, suas velocidades, acelerações e torques/forças máximos;
- as trajetórias do suporte da ferramenta ou do efetuador final do robô precisam ser suaves, devendo ser levadas em consideração as continuidades de percurso, de velocidade e de aceleração;
- em algumas operações que exijam qualidade superficial (alguns processos de usinagem, por exemplo), é necessária uma precisão na trajetória levando em consideração a distância entre os caminhos;
- as trajetórias devem ser geradas no menor espaço de tempo possível.

Existe uma relação entre o planejamento da trajetória e o planejamento do caminho [34, 45], de forma que usualmente ambas as atividades precisam ser executadas:

Planejamento do caminho – denota o caminho geométrico dos pontos no espaço das juntas, ou no espaço operacional, que o manipulador deve seguir na execução do movimento. Nesse caso, o caminho é uma descrição puramente geométrica do movimento sem a informação tempo.

Planejamento da trajetória – é o caminho associado a uma lei temporal em cada ponto, por exemplo, em termos de velocidade e/ou aceleração em cada ponto, para o caminho geométrico especificado.

O operador deve discriminar os pontos da trajetória em um sistema de coordenadas (cartesianas, polares), no qual os resultados podem ser facilmente visualizados e medi-

dos. Porém, o único sistema de coordenadas diretamente compatível com o robô é o das juntas. Então, é necessário que o operador decida se o planejamento da trajetória ocorre em coordenadas cartesianas ou os pontos do caminho são convertidos para coordenadas de juntas, para posteriormente executar o planejamento da trajetória, associando a lei temporal a cada uma das juntas do robô.

O planejamento de trajetórias gera uma seqüência no tempo das variáveis que determina a posição e orientação do efetuador final (manipulador), respeitando as limitações impostas pelo robô. O controle do movimento é feito no espaço das juntas, e um algoritmo de cinemática inversa é utilizado para reconstruir uma seqüência temporal das variáveis das juntas, correspondendo à seqüência desejada no espaço operacional. Trabalhando-se a partir do espaço operacional, é possível prever quais as regiões do espaço de trabalho são livres de obstáculos e permitidas ao manipulador [45].

Quando é desejado que o efetuador siga um percurso geométrico específico no espaço operacional, é necessário planejar a execução da trajetória diretamente no espaço operacional, o que pode ser feito por interpolação de uma seqüência de pontos.

Existem muitos métodos que podem ser usados para encontrar ou interpolar os pontos gerando trajetórias das posições ou orientações do robô. Isso inclui a interpolação por segmentos de retas, curvas polinomiais, curvas cúbicas (por exemplo, splines cúbicas de Hermite, curvas de Bézier, B-Splines, B-Splines racional não uniforme) [57].

2.2.1 Técnicas de geração de caminho

Segundo Chen [7], existem basicamente três tipos de técnicas para a geração do caminho da ferramenta:

- **Método de geração do caminho baseado em *Automatically Programmed Tool* (APT)** – a ferramenta é movida em uma direção enquanto mantém contato com a superfície da peça e com a superfície especificada pelo usuário. Para cada passo, são feitas buscas de iteração numérica para encontrar a posição da ferramenta dentro de um limite de tolerância especificado. A desvantagem desse método é que os cálculos iterativos consomem muito tempo e não garantem que as iterações convirjam para todas as superfícies;

- **Método de geração do caminho no espaço de trabalho** – os caminhos da ferramenta são planejados no plano XY do sistema de coordenadas cartesianas. O

caminho de corte é a intersecção da superfície da peça e um plano perpendicular ao plano XY;

- **Método de geração do caminho no espaço paramétrico** (conhecido também como **iso-paramétrico**) – os caminhos da ferramenta são planejados no espaço paramétrico. A ferramenta é movida por pontos igualmente espaçados na direção de u e na direção de v da superfície (u e v são os parâmetros da superfície). Cada ponto selecionado do domínio $u - v$ corresponde a um ponto (x, y, z) na superfície da peça e pode ser usado no caminho da ferramenta. A desvantagem desse método é a dificuldade de controlar as distâncias entre os caminhos na superfície da peça baseada somente no intervalo de pontos do domínio $u - v$. Os pontos igualmente espaçados no espaço paramétrico, sem dúvida, resultam em pontos não igualmente espaçados na superfície da peça. Esse método é relativamente simples em termos computacionais.

Esses três métodos principais de geração do caminho da ferramenta têm uma característica em comum, quanto maior o nível de exatidão do caminho, em termos de qualidade dos pontos do caminho da ferramenta na superfície, tanto maior o requisito de memória.

Toledo [11] fez um estudo objetivando o planejamento, a análise e a programação de tarefas para robôs industriais com o uso de técnicas CAD/CAM. Para realizar tal tarefa, foram utilizados os dados da superfície de contorno do modelo geométrico gerado por um sistema CAD. Assim, foi possível o desenvolvimento de algoritmos para realizar diretamente da superfície do objeto CAD o planejamento das trajetórias paramétricas do efetuador final do manipulador. Um exemplo dos resultados obtidos por Toledo [11] pode ser observado na Figura 6, onde 6(a) é o caminho da ferramenta no espaço paramétrico, (b) é a superfície do objeto e (c) e (d) o caminho na superfície da peça.

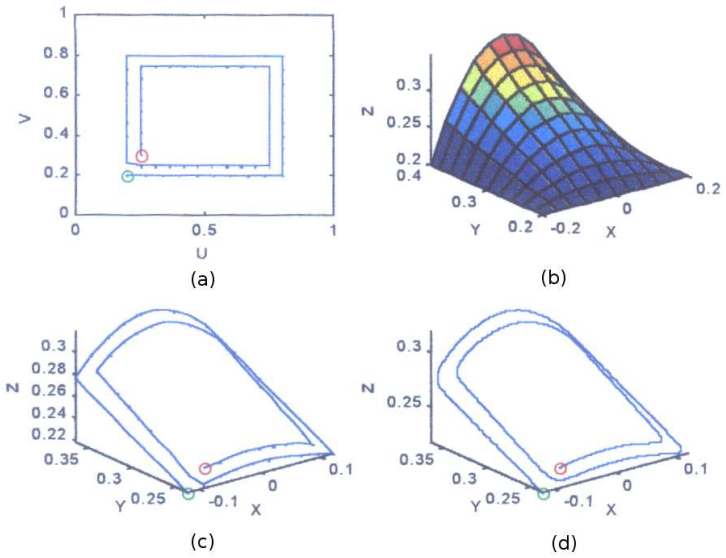


Figura 6: Ilustração demonstrativa do procedimento de geração de caminhos no espaço paramétrico [11].

Uma classificação mais específica dos métodos citados acima pode ser observada em Suresh [50], em que são considerados os métodos APT e iso-paramétrico propostos por Chen [7], descrevendo ainda três métodos de geração de caminho: iso-planar, *iso-offsets* e *iso-scallop*.

O método de planejamento de caminho **iso-planar** usa as curvas de intersecção entre a superfície e um plano como caminho da ferramenta. É, portanto, um método de geração de caminho no espaço de trabalho, que utiliza um plano como superfície definida pelo usuário, e é considerado um caso específico do APT. Tal método é caracterizado pela distância constante entre os planos que determinam caminhos adjacentes da ferramenta. Cada distância é determinada de acordo com a altura da rugosidade desejada na superfície de trabalho. Os caminhos da ferramenta iso-planar gerados, em geral, não são ótimos e a escolha de uma orientação do plano ótimo para uma dada superfície ainda é um problema em aberto.

A geração de caminho **iso-offsets** consiste em definir passos constantes do perfil da fronteira da superfície, que são muito eficientes somente quando ela não apresenta grandes variações de curvatura.

Suresh [50] desenvolve um método de geração de caminho **iso-scallop** propondo que a altura da rugosidade de processos de usinagem seja mantida constante durante a execução de uma tarefa. A usinagem com a altura da rugosidade constante consiste em gerar

caminhos paralelos para a ferramenta que mantenham a rugosidade constante com o caminho anterior. Os caminhos paralelos são obtidos de maneira que a distância entre dois caminhos consecutivos seja controlada e mantida constante de acordo com os parâmetros da tarefa (rugosidade). O caminho paralelo é definido pelo conjunto de pontos que estão a uma distância constante, medida na direção do vetor perpendicular ao vetor tangente da trajetória sobre a superfície, até um ponto pertencente à trajetória paralela adjacente. Para se calcular o caminho a ser percorrido, é preciso definir exatamente as propriedades da superfície em cada ponto. O método de geração de caminho *iso-scallop* proporciona um caminho da ferramenta com menor comprimento total quando comparado com os outros métodos.

Baseado na idéia de geração de caminho *iso-scallop* de Suresh [50], Sarma [43] implementa um algoritmo prático para geração de caminhos da ferramenta, que usa seções de varredura dessa ferramenta ao longo do caminho, para calcular os intervalos entre os passos da mesma. Existem ainda outros trabalhos que visam diminuir os erros numéricos e melhorar a qualidade superficial como os trabalhos de Feng [16] e Simas [48].

A Figura 7 ilustra os três métodos freqüentemente utilizados na geração do caminho da ferramenta: (a) iso-paramétrico, (b) iso-planar e o (c) *iso-scallop*.

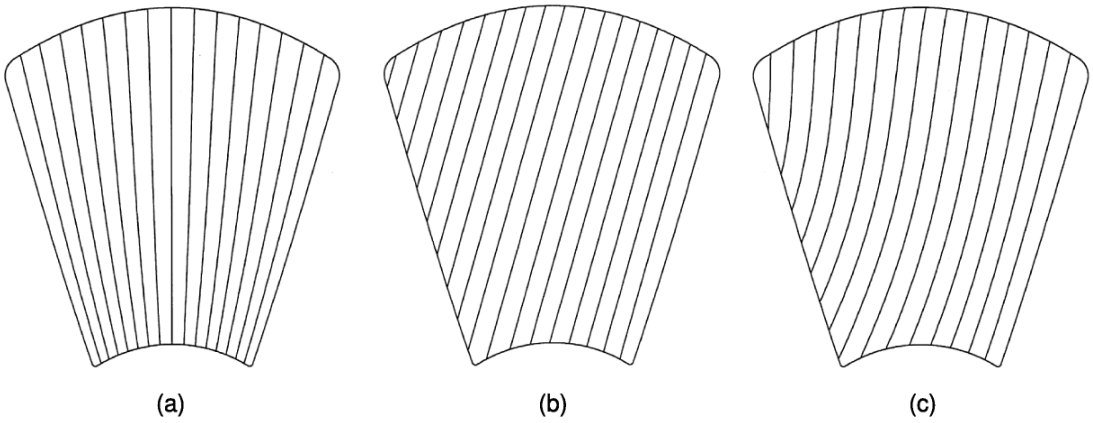


Figura 7: Métodos de geração de caminhos – (a) iso-paramétrico, (b) iso-planar e o (c) *iso-scallop* [16].

Os métodos de geração do caminho da ferramenta precisam da informação da superfície da peça para o cálculo dos caminhos. Em alguns métodos, como no APT, é necessário efetuar ainda cálculos de intersecções entre as superfícies da peça e a superfície especificada pelo usuário para determinar os caminhos.

2.2.2 Reconstrução de superfícies

A reconstrução de superfícies pode ser aplicada na superfície de trabalho que é designada como a região onde o robô deverá executar a tarefa especificada. Essas tarefas podem ser, por exemplo, recobrir erosões, através de processos de soldagem, ou executar recobrimento de superfícies pelo processo de pintura. Ao executar esse tipo de tarefa, é preciso ter modelos geométricos da superfície como relatam vários autores [11, 17, 48, 56, 60].

Segundo Gomes [21], a operação de obter o objeto no universo matemático a partir de sua representação discreta é chamada **reconstrução**. A Figura 8 ilustra as operações de representação e reconstrução. Assim, quando é desejado obter uma outra representação de um objeto, é possível reconstruir o objeto e representá-lo reconstruído.

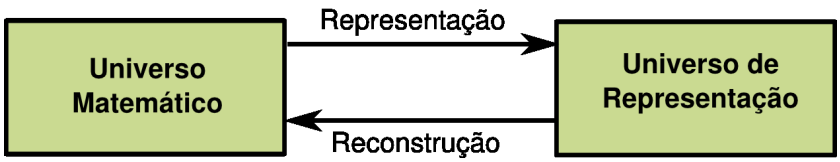


Figura 8: Representação X Reconstrução.

O processo de reconstrução de superfícies é constituído pelo reconhecimento de dados amostrados a partir da forma (representação) de um modelo real, tendo como objetivo a obtenção de uma descrição (matemática) exata dessa superfície. Ao trabalhar com o problema de reconstrução de superfícies, não são consideradas as suas propriedades adicionais, tais como cor, textura, material, mas somente a sua forma.

Gomes [21] comenta que a operação de reconstrução de superfícies é dependente da representação utilizada e, em geral, é muito difícil de ser calculada. Essa dificuldade é devido ao fato de a maioria dos métodos de reconstrução fornecerem aproximações dos objetos originais, sendo que o objetivo é sempre ter uma representação "idêntica" ao objeto original.

2.2.2.1 Categorias de representação de objetos reconstruídos

Foley [19] e Velho [52] descrevem algumas formas existentes de representação de superfícies reconstruídas, as quais podem ser categorizadas da seguinte maneira:

- Representação por conjunto de pontos – quanto maior o número de pontos, maior é a semelhança entre a representação e o objeto original (Figura 9a), no entanto

conjuntos grandes são difíceis de manipular;

- Representação por malhas
  - os pontos da representação são ligados através de segmentos de retas, originando uma visão facetada e em *wireframe*[6] (Figura 9b). Uma desvantagem da representação em *wireframe* é que ela fornece múltiplas interpretações para uma visualização, pois quando dois segmentos de retas reversas se sobrepõem é impossível determinar qual está mais próxima do observador;
  - os pontos da representação são ligados através de curvas (Figura 9c), originando também uma visão facetada e em *wireframe*. A forma suave proporciona uma representação bem próxima do objeto original;
- Representação por segmentos de superfícies – a visão deixa de ser *wireframe*, passando a possuir informação sobre os pontos superficiais, ou seja, os pontos da superfície que não estão sobre as arestas do *wireframe* também são representados. Essa representação é não-ambígua (Figura 9d);
- Representação por sólidos – possibilita a distinção das propriedades de dentro e fora do objeto, fornecendo também uma visão volumétrica do objeto (Figura 9e).

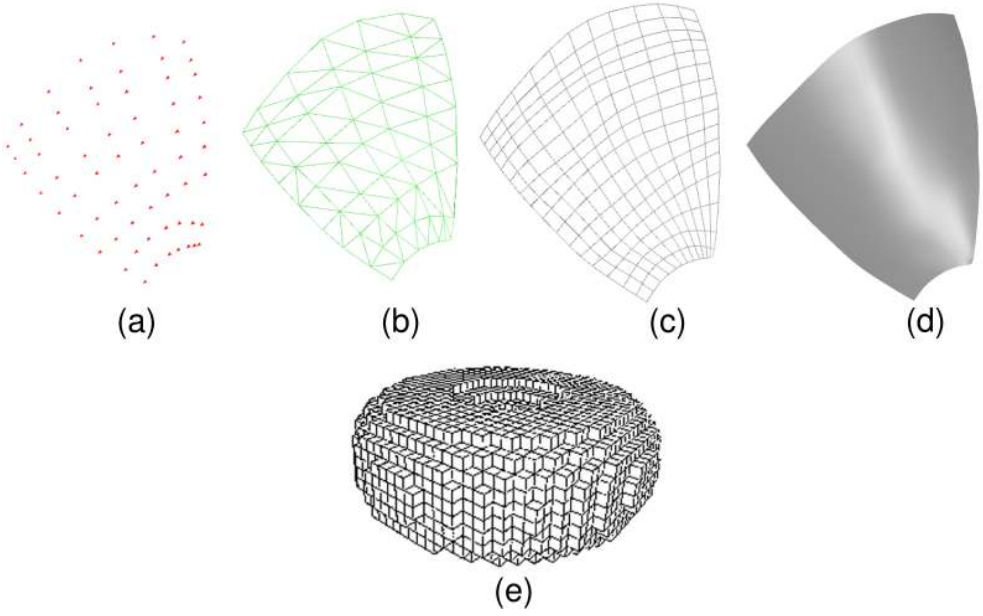


Figura 9: Exemplos de representação por – (a) Pontos, (b) Retas, (c) Curvas, (d) Segmentos de superfícies e (e) Sólidos [19].

[6]Estrutura de arames.

Cada representação evidencia alguns aspectos do objeto original, e quanto mais robusta é a forma de representação mais semelhante ao objeto original ela se torna. A Figura 9 exemplifica cada uma das categorias citadas anteriormente (sendo que 9a, 9b, 9c e 9d foram geradas pelo programa desenvolvido e 9e foi obtida em Foley [19]).

Para fazer a reconstrução de um objeto, os pontos da amostragem devem estar ordenados corretamente conforme comenta Gomes [21]. Esse fato ressalta um aspecto importante: a amostragem de um objeto gráfico precisa ser associada a uma estruturação das amostras de modo a obter a reconstrução corretamente. Nesse caso, a estruturação consiste em fazer uma ordenação. A Figura 10 (b) mostra uma representação por amostragem pontual do círculo (a); a Figura 10 (c) mostra a reconstrução com uma estruturação correta e (d) a reconstrução com uma estruturação incorreta, que não representa o objeto original (círculo).

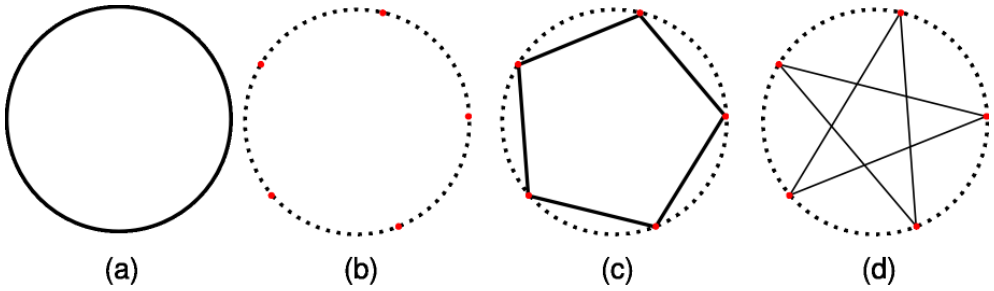


Figura 10: Ordenação de um conjunto de pontos representativos de um círculo – (a) Objeto original (círculo); (b) Amostragem pontual; (c) Reconstrução ordenada; (d) Reconstrução não-ordenada [21].

Dessa forma, uma superfície a ser reconstruída também precisa estar associada a uma ordenação correta do conjunto de pontos para que a reconstrução possa representar corretamente a superfície original. A ordenação do conjunto de pontos possibilita conhecer as relações topológicas dos pontos no espaço cartesiano 3D, indicando regiões de vizinhança de cada ponto.

2.2.2.2 Obtenção de informação para reconstrução de superfícies

As informações da superfície podem ser obtidas através de modelagem geométrica dos objetos em sistemas CAD/CAM quando ela tiver sido descrita por eles. Os objetos que não têm os dados organizados em sistemas CAD precisam de métodos para fazer a leitura da forma e conseqüentemente uma descrição de suas superfícies. Em muitas ocasiões, é preciso obtê-las a partir da captura de pontos utilizando-se técnicas de medição. As

técnicas de medição mais utilizados são a máquina de medição por coordenadas (CMM), a varredura a laser, as técnicas óticas utilizando câmeras e iluminação, sonar, radar [32].

Uma classificação das técnicas de captura da geometria dos objetos é descrita na Figura 11. Os métodos de aquisição de forma se dividem em duas categorias principais: os métodos que não entram em contato com a superfície do objeto e os métodos que entram em contato. Estes últimos podem ser subdivididos em métodos destrutivos, ou seja, que danificam o objeto impossibilitando a sua reutilização para novas medições, e em métodos não destrutivos. O conjunto de pontos, obtido através desses métodos de captura, são tratados como nuvem de pontos.

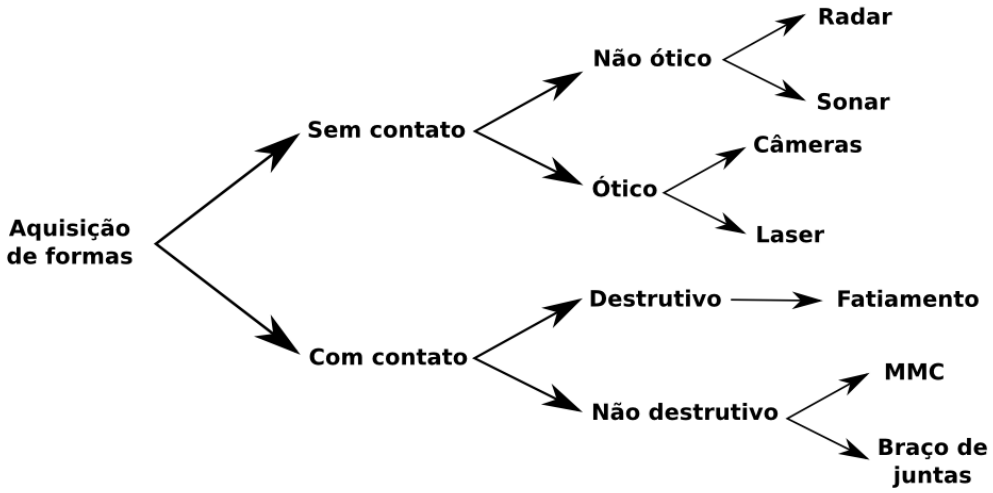


Figura 11: Tecnologias de digitalização 3D.

A nuvem de pontos é usada para identificar a superfície de trabalho ou mesmo formas de objetos nas quais o elemento robótico deve trabalhar. Uma primeira aproximação da forma da superfície pode ser obtida através da reconstrução de malhas. A interpolação de todos os pontos adquiridos dá uma forma aproximada da entidade, na qual a tarefa vai ser executada, visando garantir a ordenação dos pontos, pois, nesse processo, todas as relações de vizinhança são conhecidas.

2.2.2.3 Métodos de reconstruir superfícies

Mencl [31] descreve a existência de quatro classes de algoritmos de reconstrução, que têm como objetivo demonstrar os principais métodos disponíveis, para reconstruir até os casos mais críticos de modelos de superfícies:

- **Subdivisão espacial** – são caixas limitantes que subdividem o conjunto de pontos amostrados dentro de células disjuntas. Os mais aplicados são as grades regulares, *octrees*, malhas tetraedrais;
- **Deformação** – deforma uma superfície inicial qualquer até ter uma boa aproximação para o conjunto de pontos;
- **Funções distâncias ou poligonização implícita** – utiliza uma função distância para calcular os pontos do espaço mais próximos da superfície;
- **Crescimento incremental de superfícies** – acrescenta de forma incremental estruturas da superfície. Através de pontos próximos, traça-se uma aresta inicial, e a ela é adicionada os pontos próximos até que toda a superfície tenha sido reconstruída.

A classificação proposta por Mencl [31] é geral e trata da reconstrução no espaço 3D da superfície, sendo, portanto, técnica mais complexa e de difícil implementação. Boesel [3, 4] desenvolveu uma técnica de reconstrução que inclui a ordenação topológica dos pontos e satisfaz as propriedades necessárias para reconstruir a superfície onde será realizado o planejamento de trajetórias (detalhada na seção 2.2.2.4).

2.2.2.4 Reconstrução de superfícies utilizando diagrama de Voronoi

Boesel [3, 4] propõe uma forma de reconstrução de superfícies 3D pelo uso de um método geométrico baseado no diagrama de Voronoi. O estudo proposto foi organizar a estrutura de dados de entradas para que possa ser utilizada em planejamentos de trajetórias do efetuador de robôs.

O algoritmo constrói o diagrama de Voronoi a partir de projeções dos pontos no plano cartesiano, bem como a triangularização de Delaunay. O diagrama de Voronoi é feito em cada projeção dos três planos cartesianos, devendo-se, assim, escolher a triangularização de Delaunay que melhor representa a superfície para depois reconstruí-la no espaço a partir do plano selecionado.

A Figura 12 mostra o resultado obtido pelo algoritmo desenvolvido por Boesel [3, 4], para a geração de superfícies a partir de uma nuvem de pontos, utilizando o diagrama de Voronoi. Na Figura 12, (b) e (c) exibem a forma poligonizada da superfície real (a).

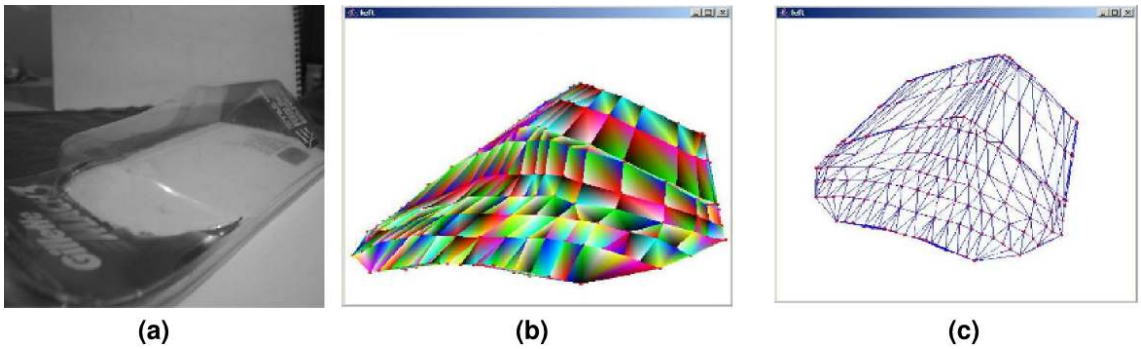


Figura 12: Reconstrução da superfície – (a) superfície original, (b) visualização sombreada e (c) visualização em *wireframe* [3, 4].

2.2.2.5 Reconstrução de superfícies utilizando sistemas CAD/CAM

Uma outra forma é a reconstrução da superfície a partir de modelagem em um sistema CAD/CAM. A superfície pode ser feita a partir das informações de projeto do objeto a ser trabalhado, no qual o projetista, pela disposição dos pontos, pode manipular e reconstruir a superfície (como na Figura 13). Muitos sistemas CAD/CAM oferecem também a possibilidade de reconstrução a partir da nuvem de pontos que é transformada em superfícies com o auxílio de aproximação por figuras geométricas (planos, cilindros, cones, esferas), malhas triangularizadas ou interpolação de pontos (através de métodos como Bézier, Hermite, B-Splines, etc). Tais métodos são muito aplicados em engenharia reversa nos sistemas CAD/CAM.

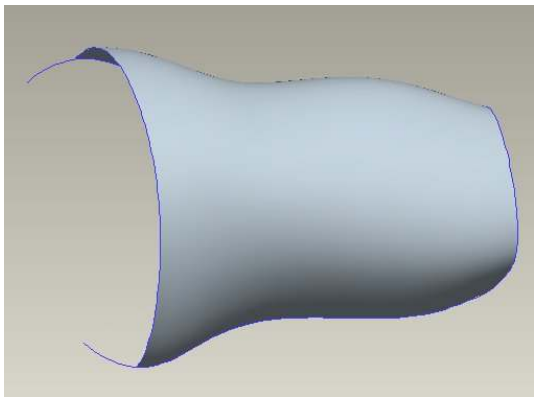


Figura 13: Superfície reconstruída através de modelagem em um sistema CAD.

Uma vez que o modelo CAD tenha sido construído, podem ser realizadas modificações no objeto original, a partir das ferramentas usuais oferecidas pelo sistema CAD, podendo-se tirar pontos espúrios ou mesmo trabalhar simetria e outros dados do projeto.

O processo de reconstrução de superfície é atualmente aplicado em muitos ramos da ciência: na medicina para a confecção de próteses personalizadas para seres humanos [8, 39, 28], na engenharia reversa para a reconstrução de produtos existentes ou não documentados [34, 42] e em operações de manutenção através de ambientes virtuais para a visualização do estado real, como em refinarias de petróleo, hidrelétricas, ambientes insalubres ou lugares de difícil acesso [33].

A reconstrução de superfície pode ser aplicada quando for necessário reproduzir mais precisamente superfícies originais não documentadas. O modelo reconstruído poderá ser modificado e manipulado em um sistema CAD a fim de satisfazer os requisitos de projeto no momento em que a peça for reproduzida posteriormente. Se for um problema de manutenção, geralmente a imagem obtida poderá servir para reconstruir a superfície original ou auxiliar na manutenção.

2.3 Considerações

Este capítulo apresentou uma revisão bibliográfica de conceitos relacionados à robótica, à reconstrução de superfícies e ao planejamento de trajetórias de robôs manipuladores. O planejamento de trajetória é um processo importante sempre que for necessário executar operações de manufatura utilizando equipamentos de controle de operações por comando numérico (NC) ou robôs.

Figura 14: Etapas do processo de planejamento de trajetórias.

Ao se realizar o processo de planejamento de trajetórias é preciso ter uma descrição sobre a superfície onde será realizada a operação de manufatura e detalhes sobre a operação a ser efetuada. A Figura 14 ilustra e resume as etapas necessárias ao processo de planejamento de trajetórias robóticas estudadas nesta revisão bibliográfica. Esse modelo será utilizado ao longo dos demais capítulos para sistematizar o processo de planejamento de tarefas robóticas contínuas.

3 Ferramentas Básicas

No desenvolvimento do método de reconstrução de superfícies a partir de uma nuvem de pontos, foram utilizadas as técnicas do diagrama de Voronoi e a triangularização de Delaunay, visando-se obter uma forma aproximada da superfície e uma visão topológica ordenada da nuvem de pontos. Com isso, é possível chegar-se a representação matemática mais precisa e melhor tratável do ponto de vista do planejamento de trajetórias através de um conjunto de segmentos de superfícies de forma livre.

Com a descrição da superfície, é possível se trabalhar no planejamento da tarefa. Duas abordagens são utilizadas para o planejamento de trajetórias em que a ferramenta descreve uma trajetória sobre uma superfície: uma definindo a trajetória no espaço paramétrico da superfície e outra em que é necessário definir precisamente o caminho a ser seguido pela ferramenta no espaço de trabalho do manipulador robótico.

Dessa forma, são abordados os seguintes recursos para o desenvolvimento dos algoritmos e para a formulação do método proposto: a teoria de diagrama de Voronoi, a triangularização de Delaunay, a técnica de recorte de Cyrus-Beck (para recorte de retas), o algoritmo de reconstrução de superfícies por B-Splines e a geração de caminhos sobre superfícies.

3.1 Diagrama de Voronoi

O diagrama de Voronoi consiste em uma decomposição do espaço em regiões convexas. Os matemáticos Dirichlet e Voronoi foram os primeiros a introduzir formalmente este conceito, em que Dirichlet estudou casos particulares em 1850 e Voronoi introduziu a definição formal para casos n dimensionais 1908 [53]. A estrutura resultante é também conhecida como tecelagem de Dirichlet ou diagrama de Voronoi [23, 46].

O diagrama de Voronoi surgiu para resolver o problema de como dividir uma cidade em áreas irregulares de forma que a área coberta por um carteiro vinculado a uma de-

terminada agência de correio fosse otimizada. Essa mesma concepção pode ser utilizada no processamento de imagem de um documento para separar em blocos as letras de uma palavra [27]; em sistemas de informações geográficas (GIS), como por exemplo, para o rastreamento de emergências a partir de dados coletados de localizações de telefones celulares [44]; na logística de transporte para o atendimento de todos os pontos de uma área de distribuição de materiais a fim de obter um conjunto de zonas para as quais está associado um veículo, no menor tempo e na menor distância possíveis [20] e na astronomia para a catalogação de conjuntos de galáxias [37]. Neste trabalho, o diagrama de Voronoi será utilizado, sobretudo, para ordenar topologicamente um conjunto de pontos. A ordenação topológica utilizando Voronoi é escolhida devido a vários fatores:

- a existência de algoritmos eficientes[1] para o cálculo do diagrama de Voronoi no plano;
- possibilidade de uso posterior para cálculo de colisões;
- fornece a relação de vizinhança entre os pontos de maneira direta;
- permite a geração de malhas de maneira eficiente.

Dado um conjunto de pontos no plano, o diagrama de Voronoi (ver Figura 15) divide o plano em regiões, de acordo com a regra do vizinho mais próximo, e cada ponto está associado com a região mais próxima no plano [1].

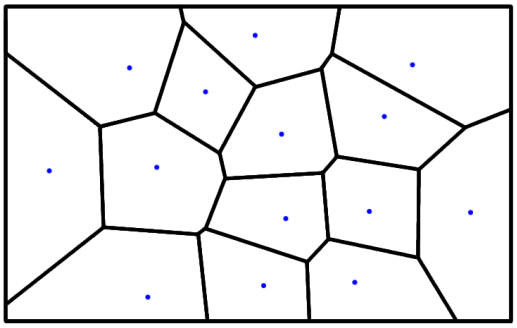


Figura 15: Diagrama de Voronoi.

Os seguintes componentes e nomenclaturas podem ser usados para descrever o diagrama de Voronoi [3]:

- **Malha** – é a rede formada pelo diagrama de Voronoi, a qual é reconhecida e denominada de diagrama em si;

[1] Com complexidade $n \log(n)$ [1].

- **Célula** – é a unidade básica do diagrama de Voronoi. É formada por um único e exclusivo vértice e uma ou mais retas ou arestas que formam o polígono que circunda um vértice;
- **Vértice** – é o ponto do diagrama que exerce influência e que define uma região de dominância ao seu redor. Sempre está associado a uma única célula;
- **Reta Limitadora** – é a reta que limita a região de dominância de dois vértices. Não é possível uma reta limitar a região de dominância de apenas um vértice ou mais de dois vértices;
- **Aresta** – é uma parcela da reta limitadora que define um lado do polígono ou contorno de uma célula;
- **Conjunto de Limites (Polígono ou Face)** – é o conjunto de segmentos de retas e/ou arestas de uma célula. O conjunto de limites pode formar um polígono fechado convexo ou pode ser um região de fronteira, onde o polígono é aberto.

Assim, considere S sendo um conjunto de $n \geq 3$ pontos locais no plano. Para dois pontos locais p e $q \in S$, a **região de dominância** (d) de p sobre q é definida como o subconjunto no plano que está mais próximo de p do que de q.

$$d(p,q) = \{x \in \mathbb{R}^2 \mid \delta(x,p) < \delta(x,q)\}, \tag{3.1}$$

em que δ denota a função distância Euclidiana ($\delta(x,p) = \sqrt{(p_1 - x_1)^2 + (p_2 - x_2)^2}$). Tem-se que $d(p,q)$ é um semi-plano fechado limitado pela **mediatriz** de p e q, dada por:

$$M(p,q) = \{x \in \mathbb{R}^2 \mid \delta(p,x) = \delta(q,x)\} \tag{3.2}$$

A mediatriz separa todos os pontos mais próximos de p dos que estão mais próximos de q e será chamada de **reta limitadora**. A **região de Voronoi(R_V)** de um local $p \in S$ é a porção do plano que está sobre a região de dominância local do ponto p em S.

$$R_V(p,S) = \bigcap_{q \in S,\ q \neq p} d(p,q) \tag{3.3}$$

Pela definição, cada região de Voronoi $R_V(p,S)$ é a intersecção de $n-1$ semi-planos abertos contendo o local p. Dessa forma, $R_V(p,S)$ é aberta e convexa. Regiões de Voronoi diferentes são desconexas [1].

Definição 1 *O* ***diagrama de Voronoi*** *de S é definido por:*

$$V(S) = \bigcup_{p,q \in S,\ p \neq q} \overline{R_V(p,S)} \cap \overline{R_V(q,S)} \tag{3.4}$$

A fronteira comum de duas regiões de Voronoi pertencentes ao $V(S)$ é chamada de **aresta** de Voronoi. Se a aresta de Voronoi (e) for fronteira da região de p e q, então $e \subset M(p,q)$. Os pontos finais das arestas de Voronoi são a fronteira comum de três ou mais regiões de Voronoi.

3.2 Triangularização de Delaunay

A triangularização de Delaunay, que foi definida por Boris Delaunay em 1934, é uma representação cujo dual é o diagrama de Voronoi, podendo ser obtida a partir dele. Para isso, basta que se una os vértices das células vizinhas do diagrama de Voronoi e a triangularização estará definida como pode ser visto na Figura 16.

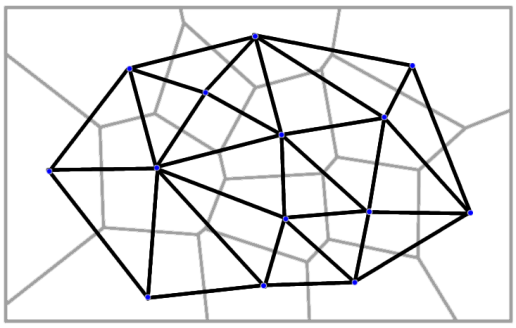


Figura 16: Triangularização de Delaunay obtida a partir do diagrama de Voronoi.

A triangularização de Delaunay pode também ser construída a partir de um conjunto de pontos, porém, antes, é necessário fazer algumas definições.

Para uma triangularização de Delaunay D, de um conjunto de vértices V no plano, devem ser consideradas as seguintes propriedades [30]:

1. qualquer círculo no plano é dito vazio caso não contenha nenhum vértice de V no seu interior;

2. vértices localizados sobre a circunferência não são considerados no interior do círculo, não invalidando, portanto, o critério anterior;

3. sejam m e n dois vértices quaisquer de V. Um círculo circunscrito da aresta mn é qualquer círculo que passe pelos pontos m e n e qualquer aresta possui infinitos círculos circunscritos.

Assim, pode-se dizer que a triangulação D respeita a seguinte regra:

Definição 2 *A aresta mn encontra-se na triangularização D se, e somente se, existe um círculo circunscrito vazio de mn. A aresta que satisfaz essa propriedade também é chamada de aresta de Delaunay.*

Neste trabalho, a triangularização de Delaunay é construída a partir do seu dual, o diagrama de Voronoi, sendo simples construí-la a partir dele, uma vez que ele já está implementando. O resultado final é idêntico ao obtido pela aplicação direta da definição, pois a triangularização de Delaunay é única para um dado conjunto de pontos.

3.3 Técnicas de recortes

O recorte é uma operação muito importante em computação gráfica devido à variedade de suas aplicações. O uso de técnicas de recorte no presente trabalho é importante em duas etapas, a primeira no processo de limitação da região de trabalho e a segunda no recorte de entidades que devem aparecer ou desaparecer da região focada.

Existem várias técnicas de recortes para as mais diversas aplicações. Dentre elas, as mais conhecidas e utilizadas são: algoritmo da Força Bruta, Cohen-Sutherland, Cyrus-Beck, Liang-Barsky (que seria um caso particular de Cyrus-Beck). Foi escolhida, neste trabalho, a adaptação do algoritmo de Cyrus-Beck por se tratar de um algoritmo eficaz e por trabalhar com polígonos em geral (não somente com retângulos).

3.3.1 Algoritmo paramétrico de Cyrus-Beck

O algoritmo de Cyrus-Beck, conforme detalhado por Foley [18, 19] e Gomes [21], permite o recorte de um segmento de reta por qualquer polígono convexo, em 2D, ou de um segmento de reta em 3D, por qualquer poliedro convexo.

Seja a representação paramétrica de um segmento de reta $\overline{AB}$ e um polígono (conforme Figura 17):

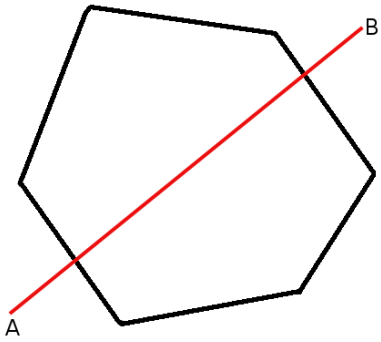


Figura 17: Polígono de recorte e segmento de reta.

Através da intersecção do segmento de reta $\overline{AB}$ com os lados/arestas do polígono, é possível calcular os valores do parâmetro t da reta nessas intersecções.

a) Intersecção entre segmentos de reta

Três propriedades devem ser satisfeitas para o cálculo da intersecção entre segmentos de reta:

1. considerar a equação paramétrica do segmento de reta

$$P(t) = A + t \cdot (B - A) \qquad 0 \leq t \leq 1 \tag{3.5}$$

2. o polígono de recorte é definido pela enumeração das suas arestas (ou vértices) percorrendo-as no sentido anti-horário;

3. para cada aresta é definida a normal (N_i), que aponta sempre para fora do polígono.

Assim, seja P_{e_i} um ponto qualquer pertencente a uma aresta do polígono de recorte. Conforme a Figura 18, são definidos três vetores que partem do ponto P_{e_i} até t_1, um ponto interior ao polígono e outro exterior ao polígono, mas pertencentes ao segmento de reta $\overline{AB}$.

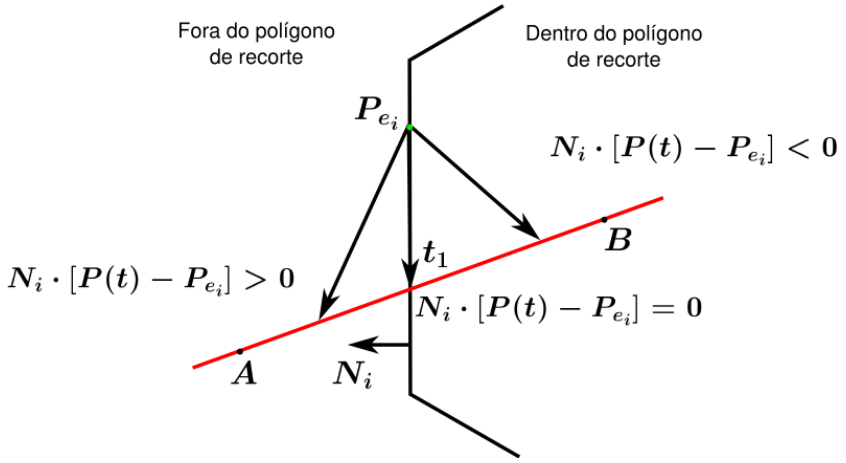


Figura 18: Produtos internos dos três vetores com a normal exterior ao polígono de recorte.

Se o produto interno $N_i \cdot [P(t) - P_{e_i}] = 0$, então $P(t)$ está sobre a aresta do polígono, pois os vetores são perpendiculares.

Assim, o produto interno da normal N_i com cada um destes três vetores é, respectivamente, negativo, nulo e positivo. Logo, o produto interno permite determinar se um ponto do segmento de reta é ponto interior, exterior ou se está sobre a aresta do polígono.

Para determinar o ponto de intersecção do segmento de reta $\overline{AB}$ com a aresta, é necessário obter o valor do parâmetro no ponto de intersecção (t_1).

Sabe-se que, no ponto de intersecção, o produto interno da normal à aresta com o vetor que une o ponto arbitrário P_{e_i} com o ponto de intersecção é nulo, ou seja,

$$N_i \cdot [P(t) - P_{e_i}] = 0 \tag{3.6}$$

Substituindo $P(t)$ pela equação paramétrica do segmento de reta 3.5, tem-se:

$$N_i \cdot [A + (B - A)t - P_{e_i}] = 0 \tag{3.7}$$

$$N_i \cdot (A - P_{e_i}) + N_i \cdot (B - A)t = 0 \tag{3.8}$$

$$t = \frac{N_i \cdot (A - P_{e_i})}{-N_i \cdot (B - A)} \tag{3.9}$$

em que $N_i \neq 0$ e $B \neq A$ e o produto escalar $N_i \cdot (B - A) \neq 0$ (ou seja, $\overline{AB}$ não deve ser

paralela à aresta do polígono).

A equação 3.9 pode ser usada para calcular os valores de t nas intersecções entre $\overline{AB}$ e cada aresta do polígono de recorte. Dados os possíveis valores de t para cada segmento de reta, o próximo passo é se determinar quais dos valores correspondem a intersecções internas do segmento de reta com as arestas do polígono de recorte. Como primeiro passo, pode ser descartado qualquer valor de t que esteja fora do intervalo [0,1], uma vez que eles estão fora do segmento de reta $\overline{AB}$. Para o próximo passo, é preciso se determinar qual intersecção está sobre a aresta do polígono de recorte (ver a Figura 19, que mostra segmentos de retas potenciais à realização de recortes).

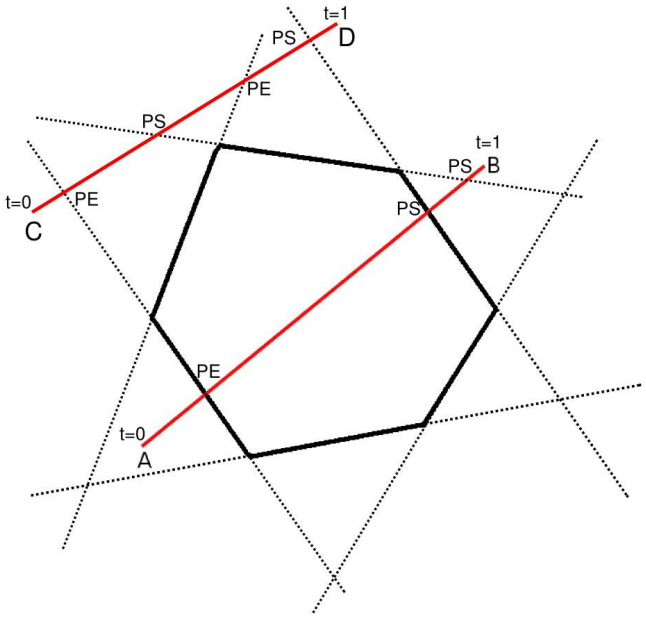


Figura 19: Segmentos de reta a serem recortados pelo polígono.

Porém, existe ainda a necessidade de se distinguir os pontos do segmento de reta que estão dentro do polígono de recorte. A Figura 19 ilustra o exemplo de um segmento de reta $\overline{CD}$, que possui intersecções com as retas que contêm as arestas do polígono de recorte, porém nenhuma das intersecções pertence ao conjunto de pontos da fronteira do polígono.

b) Classificação dos pontos paramétricos

As intersecções são caracterizadas como Potencialmente de Entrada (PE) e Potencialmente de Saída (PS) do polígono de recorte, como segue: Se o ponto A estiver do

lado de fora do polígono de recorte com relação a uma aresta do polígono, e o ponto B estiver dentro do polígono de recorte com relação a mesma aresta (que pode ser verificado através do vetor normal N_i), então o ponto $P(t)$ da intersecção entre $\overline{AB}$ e essa aresta é caracterizado como potencialmente de entrada (PE), conforme a Figura 18, caso contrário o ponto $P(t)$ é caracterizado como potencialmente de saída (PS).

Algebricamente, a caracterização de um ponto é obtida usando o sinal do produto interno entre a normal N_i e o segmento $\overline{AB}$. Se o produto interno $N_i \cdot (B - A) < 0$ (ângulo maior que 90°), o ponto $P(t)$ é PE, e se $N_i \cdot (B - A) > 0$ (ângulo menor que 90°), o ponto $P(t)$ é PS.

Logo, o segmento recortado é o segmento $\overline{PQ}$, onde P é o ponto PE, que possui o maior valor do parâmetro t, e Q é o ponto PS, que possui o menor valor do parâmetro t. Conforme a Figura 20:

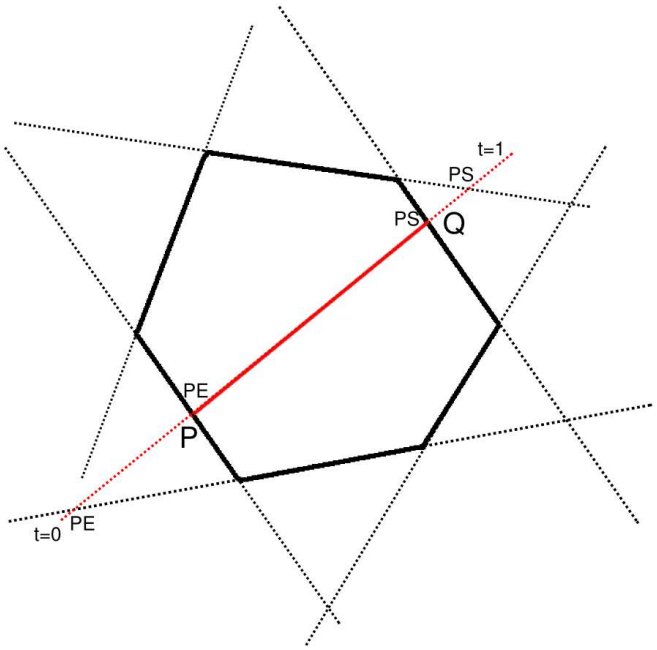


Figura 20: Segmento de reta recortado pelo polígono.

c) Pseudo-Algoritmo de Cyrus-Beck

Foley [18] propõe um pseudo-algoritmo para a técnica de recorte de Cyrus-Beck como pode ser visualizado em Algoritmo 1:

Algoritmo 1 Pseudo-Código do algoritmo de recorte de retas paramétricas de Cyrus-Beck

```
1: pré-calcular N_i e selecionar um P_Ei para cada aresta:
2: para cada segmento de reta a ser cortado
3:   se A = B então
4:     a reta é degenerada de tal forma que o corte é um ponto
5:   senão
6:     defina t_E = 0; t_S = 1;
7:     para cada intersecção candidata com a aresta de recorte
8:       se N_i · (B − A) ≠ 0 então
9:         ignorar arestas pararelas a reta
10:      fim se
11:      calcular t;
12:      usar sinal de N_i · (B − A) para categorizar PE ou PS;
13:      se PE então
14:        t_E = max(t_E, t);
15:      fim se
16:      se PS então
17:        t_S = min(t_S, t);
18:      fim se
19:    fim para
20:    se t_E > t_S então
21:      retornar nil
22:    senão
23:      retornar P(t_E) e P(t_S) como intersecções de recorte
24:    fim se
25:  fim se
26: fim para
```

O algoritmo de recorte é importante na implementação do diagrama de Voronoi. Todas as retas que formarão as arestas da célula são segmentadas durante a iteração do algoritmo de geração do diagrama de Voronoi utilizando o recorte de Cyrus-Beck. A implementação do algoritmo de Voronoi será comentada mais detalhadamente no capítulo 4, sendo implementado de forma incremental, ou seja, a construção do diagrama de Voronoi no plano é realizada pela adição de pontos na malha. Dessa forma, o algoritmo de Cyrus-Beck é aplicado, considerando como polígono de recorte a célula em que um novo vértice foi inserido, e a reta a ser recortada é a mediatriz entre o vértice da célula e o ponto adicionado.

3.4 Curvas e superfícies de forma livre

As curvas de um objeto podem ser obtidas pela digitalização de um modelo físico ou por um desenho ajustando uma curva matemática para os dados digitalizados. Existem vários métodos de obter um modelo que descreva a curva para os dados. Os métodos mais utilizados são as técnicas de interpolação de curvas que são caracterizadas pelo fato da

curva matemática resultante passar através de cada um dos pontos dados. É comum se usar na construção de objetos geométricos curvas splines interpolantes. Elas são curvas aproximadas, mas que possuem a garantia de ser interpolantes aos pontos dados. Métodos de modelagem usados em sistemas CAD/CAM usam freqüentemente curvas de Bézier, B-Splines, B-Splines racionais não uniformes (NURBS), para interpolar um conjunto de pontos.

Zeid [57] comenta que existem duas categorias de curvas que podem ser representadas parametricamente:

- As **curvas analíticas** são definidas como aquelas que podem ser descritas por equações analíticas, tais como linhas, circunferências, cônicas. Essas curvas fornecem uma forma compacta de representar formas e simplificar os cálculos das propriedades relacionadas, tais como áreas e volumes. Porém, essas curvas não são atrativas com relação à interatividade (exemplo de representação analítica de uma circunferência pode ser dada por: $x = x_c + R\cos(u)$, $y = y_c + R\text{sen}(u)$, $z = z_c$, onde $0 \leq u \leq 2\pi$).

- Nas **curvas sintéticas**, é conhecida a entidade, mas não se conhece a equação analítica que a define. São caracterizadas por serem de forma livre e podem ser descritas através de um conjunto de pontos dados (pontos de controle), tais como curvas splines e Bézier. As curvas sintéticas fornecem aos desenvolvedores uma grande flexibilidade e controle da forma da curva por mudança das posições dos pontos de controle. Algumas das representações fornecem controle global e outras permitem ainda um controle local.

As **curvas splines** paramétricas são definidas como um conjunto de curvas polinomiais segmentadas com continuidade $n - 1$ se a ordem da curva é n. A curva interpola os pontos dados, e sua forma é definida usando a informação dos pontos e suas tangentes nos pontos extremos. A Figura 21 mostra uma curva spline e seus vetores tangentes nos pontos extremos.

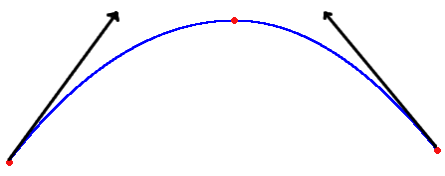


Figura 21: Curva spline.

A equação para um segmento de spline cúbica paramétrica é dado por:

$$P(u) = \sum_{i=1}^{4} B_i u^{i-1} \qquad 0 \leq u \leq 1, \tag{3.10}$$

em que u é o parâmetro da curva. $P(u) = [x(u)\ y(u)\ z(u)]$ é o vetor posição de qualquer ponto no segmento da spline cúbica. Os coeficientes constantes B_i são determinados pela especificação de quatro condições de contorno para o segmento de spline, que são usualmente tomadas como as posições e vetores tangentes no ponto inicial e final desse segmento.

O controle da forma das curvas splines cúbicas não é muito óbvio devido às características de controle global. Além disso, a ordem da curva é constante não importando o número de pontos dados.

A **curva de Bézier** é definida em termos da localização dos $n + 1$ pontos, que são chamados pontos de controle da curva. Os pontos de controle são os vértices do polígono característico da curva, e esse polígono fornece a forma da curva de Bézier. O número de vértices do polígono especifica diretamente o grau da curva gerada, e o aumento do número de vértices causa o aumento no grau da curva. A Figura 22 exemplifica uma curva de Bézier e seu polígono característico.

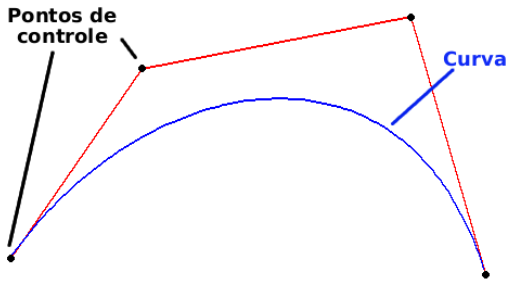


Figura 22: Curva Bézier.

Uma curva Bézier gerada através dos vértices do polígono $V_i (i = 0, 1, ..., n)$ é dependente de algum esquema de interpolação ou aproximação para estabelecer a relação entre a curva e o polígono. Tal esquema é fornecido pela escolha de funções bases. A função base de Bernstein (3.11) produz curvas de Bézier.

$$B_{n,i}(u) = C(n, i)u^i(1 - u)^{n-i} \tag{3.11}$$

em que $C(n, i)$ é o coeficiente binomial dado por:

$$C(n,i) = \frac{n!}{i!(n-1)!} \tag{3.12}$$

Então, a curva de Bézier é dada pela equação:

$$P(u) = \sum_{i=0}^{n} B_{n,i}(u)V_i \qquad 0 \leq u \leq 1 \tag{3.13}$$

em que u é o parâmetro.

As **curvas B-Splines** fornecem o controle local da curva, usando um conjunto especial de combinação de funções bases e a habilidade de adicionar pontos de controle sem aumentar o grau do polinômio, que é fornecido pelos pontos nós pertencentes à curva. Ao contrário da curva de Bézier, a teoria de curvas B-Splines separa o grau da curva gerado do número de pontos de controle. A Figura 23 mostra uma curva B-Spline, os pontos de controle e os pontos nós pertencentes à curva.

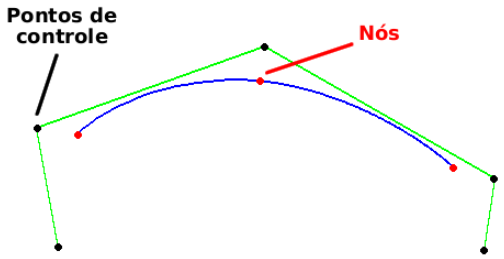


Figura 23: Curva B-Spline.

Assim, considerando-se $P(u)$ o vetor posição ao longo da curva em função do parâmetro u, uma curva B-Spline é dada por:

$$P(u) = \sum_{i=1}^{n+1} N_{k,i}(u)V_i \qquad 0 \leq u \leq 1, \qquad 2 \leq k \leq n+1 \tag{3.14}$$

em que $N_{i,k}$ são funções bases definidas pelas fórmulas de recursão de Boor e Cox (que podem ser encontradas em [36]), e V_i são vértices de controle que formam o polígono característico da curva B-Spline [41].

As **curvas NURBS** possuem uma característica relevante que é a adição de mais uma variável de controle na curva, em que é considerado também um peso associado a cada um dos vértices de controle. A curvas NURBS são consideradas uma generalização das curvas B-Splines e Bézier [36].

A teoria mais completa sobre curvas e superfícies de forma livre pode ser encontrada em [15, 19, 36, 41, 57].

Neste trabalho, foram implementadas as curvas e superfícies B-Splines uniformes, tanto para a reconstrução de superfícies quanto para a geração do caminho da ferramenta, sendo dada a elas uma atenção maior neste capítulo, prevendo-se sua utilização na formulação do algoritmo.

3.4.1 Detalhes da implementação do algoritmo para curvas e superfícies B-Splines

Considerando-se um conjunto de pontos representativos da superfície, as curvas e superfícies B-Splines podem ser geradas determinando-se os pontos nós e, através do algoritmo inverso, os vértices de controle que definem a superfície. Assim:

a) Curvas B-Splines Cúbicas

As curvas B-Splines cúbicas podem ser expressadas na forma matricial, na qual $0 \leqslant u \leqslant 1$ é o parâmetro da função (a dedução da igualdade abaixo encontra-se em [36]):

$$\begin{bmatrix} N_{4,0}(u) & N_{4,1}(u) & N_{4,2}(u) & N_{4,3}(u) \end{bmatrix} = U \cdot M_b$$

em que: $U = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix}$ e $M_b = \frac{1}{3!} \cdot \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$

Para um segmento de curva B-Spline cúbico $r_i(u)$, respectivo aos vértices de controle $V_{i+j} (i = 0, 1, \ldots, n \text{ e } j = 0, 1, 2, 3)$, é definido como:

$$r_i(u) = U \cdot M_b \cdot \begin{bmatrix} V_i \\ V_{i+1} \\ V_{i+2} \\ V_{i+3} \end{bmatrix} = U \cdot M_b \cdot V \tag{3.15}$$

A partir do desenvolvimento de curvas B-Splines, é possível estender o mesmo raciocínio para superfícies. Neste caso, os segmentos de curva tornam-se *patches*[2], que são

[2] *Patches* são designados em português por segmentos de superfícies.

considerados partes da superfície (conforme a Figura 24). É iniciado, portanto, o estudo de superfícies B-Splines a partir de segmentos de superfícies.

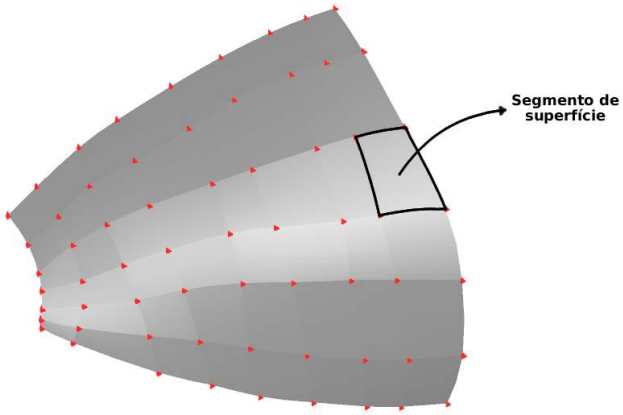


Figura 24: Segmento de uma superfície

b) Segmento de uma superfície B-Spline bi-cúbica

Considerando-se 16 vértices de controle $V_{i,j}(i = 0, 1, 2, 3 \text{ e } j = 0, 1, 2, 3)$ e representando-os ordenados em forma de matriz (4x4), tem-se:

$$V = \begin{bmatrix} V_{00} & V_{01} & V_{02} & V_{03} \\ V_{10} & V_{11} & V_{12} & V_{13} \\ V_{20} & V_{21} & V_{22} & V_{23} \\ V_{30} & V_{31} & V_{32} & V_{33} \end{bmatrix} \tag{3.16}$$

Cada linha da matriz de elementos de vértices de controle definem um segmento da curva B-Spline cúbica na direção do parâmetro $0 \leqslant u \leqslant 1$, e cada coluna representa um segmento na direção do parâmetro $0 \leqslant w \leqslant 1$. Assim, a representação de um segmento de superfície B-Spline é:

$$r_i(u, w) = U \cdot M_b \cdot V \cdot M_b^T \cdot W^T \tag{3.17}$$

em que $W = [1 \ w \ w^2 \ w^3]$

c) Algoritmo inverso para o cálculo dos pontos de controle

Durante o projeto de produto no qual se deseja criar superfícies, pode ser importante que alguns pontos estejam exatamente localizados no espaço, de forma que a geometria

da superfície precise se adequar a esses pontos. Então, a forma da superfície na maioria dos casos torna-se dependente dos pontos. Porém, a teoria de segmentos de superfícies B-Splines considera como parâmetro de entrada o conjunto de vértices de controle do poliedro característico e não os pontos que pertencem à superfície (que é geralmente a informação que se possui). Surge então a necessidade de se determinar os vértices de controle da superfície a partir dos pontos dados. Qiulin [36] propõe um algoritmo inverso para encontrar os vértices do polígono característico a partir dos pontos dados para em seguida aplicar o algoritmo para geração de superfícies B-Splines.

A curva B-Spline de interpolação cúbica pode ser construída usando-se $(n+1)$ pontos nós $r_i(i = 0, 1, \ldots, n)$ da curva e duas condições de contorno. Assumindo-se que $(n + 1)$ pontos nós consecutivos r_i são conhecidos, a fim de determinar os valores dos vetores dos vértices do polígono de controle $V_i(i = -1, 0, 1, \ldots, n + 1)$, são necessárias $(n + 3)$ equações.

De acordo com as propriedades das curvas B-Splines um ponto nó da curva B-Spline é determinado por três vértices consecutivos, tem-se, então, $n+1$ equações, como a seguinte, que relacionam os pontos nós da curva e os vértices do polígono de controle:

$$\frac{1}{6}(V_{i-1} + 4 \cdot V_i + V_{i+1}) = r_i \qquad (i = 0, 1, ..., n) \tag{3.18}$$

Para isso, duas relações adicionais são necessárias:

$$V_{-1} = V_0, \qquad V_{n+1} = V_n \tag{3.19}$$

Assim, 3.18 e 3.19 formam um sistema linear no qual são incluídas $(n + 3)$ equações. Fornecendo-se os nós da curva, é possível encontrar os vértices de controle da curva resolvendo esse sistema linear.

Para uma superfície, o algoritmo inverso significa que os vértices $V_{i,j}$ de um poliedro característico precisam ser calculados resolvendo um sistema dada a informação dos pontos da superfície.

Então, levando-se em consideração o algoritmo inverso da curva B-Spline com parâmetro u, considerando-se um conjunto de $(m+1)$ curvas, obtém-se os vértices do polígono característico, sendo esse procedimento repetido $(m+1)$ vezes, resultando no conjunto de vértices do poliedro de controle na direção do parâmetro u:

$$Q_{i,j} \quad \text{onde} \quad i = -1, 0, 1, \ldots, n, n+1 \quad j = 0, 1, \ldots, m \tag{3.20}$$

Dessa forma, $Q_{i,j}$ pode ser considerada como um conjunto de $(n+3)$ pontos, usando o algoritmo inverso da curva B-Spline com parâmetro w, sendo que esse procedimento pode ser repetido $(n+3)$ vezes, resultando nos vértices $V_{i,j}$ do poliedro de controle da superfície B-Spline bi-cúbica:

$$V_{i,j} \quad \text{onde} \quad i = -1, 0, 1, \ldots, n, n+1 \quad j = -1, 0, 1, \ldots, m, m+1 \tag{3.21}$$

Assim, o algoritmo inverso da superfície B-Spline é constituído pelo algoritmo inverso da curva B-Spline repetido $(m+1) \cdot (n+3)$ vezes.

3.5 Geração do caminho

O estudo de geração de caminhos é baseado no processo de planejamento de uma tarefa a ser executada em máquinas de controle por comando numérico. Esse caminho é equivalente àquele percorrido pela ponta do efetuador de robôs. No caso especial de usinagem, são levados em conta vários aspectos do processo como a geometria da ferramenta e a rugosidade desejada. Dessa forma, os parâmetros de entrada para a geração do caminho da ferramenta, que devem ser considerados, são a geometria da superfície do objeto/peça, os parâmetros da ferramenta e as condições de usinagem (como o passo e o avanço da ferramenta), que são usadas para executar as operações. A saída é o caminho da ferramenta que possui os dados de localização de corte.

O resultado final da geração de caminho costuma ser, portanto, o caminho percorrido pelo efetuador no espaço de trabalho. Esse caminho é geralmente independente da estrutura da máquina de comando numérico que deve ser programada para tal execução. No entanto, a programação em geral depende da estrutura da máquina devendo-se levar em conta os sistemas de coordenadas por ela utilizados para executar a tarefa.

O caminho a ser percorrido pela ferramenta nada mais é que um conjunto de curvas sobre a superfície da peça. É de conhecimento que uma curva pode ser aproximada por uma seqüência de segmentos lineares (aproximação por segmentos de retas) ou arcos circulares, uma vez que as máquinas de comando numérico aceitam somente interpolação linear ou circular entre um conjunto de pontos dados [36].

A Figura 25 mostra a ferramenta em contato com a superfície, onde P é a posição de contato com a superfície, C é a localização do centro, T é o ponto extremo e R é o raio da ferramenta. Esses dados são considerados a fim de representar a intersecção do efetuador do robô com a superfície da peça para o cálculo dos caminhos da ferramenta a serem gerados.

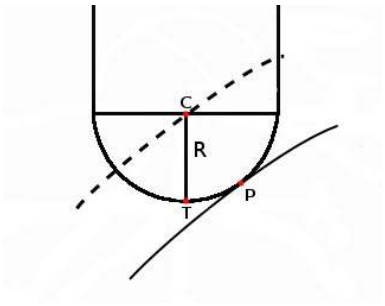


Figura 25: Características da ferramenta [36].

3.5.1 Recuo da superfície

A ponta da ferramenta é levada em consideração como ponto de referência para a programação do movimento durante o planejamento do caminho. Entretanto, ao planejar o movimento a ser percorrido pela ferramenta, o local de contato dela com a superfície da peça varia, e, por isso, torna-se difícil considerar esse ponto de contato como o ponto de referência.

Por outro lado, ao se levar em conta o centro da ferramenta para o planejamento do caminho, não existe alteração da posição de planejamento devido à mudança na posição de contato com relação à superfície de trabalho. Assim, o caminho a ser gerado terá um recuo com relação à superfície da peça, e esse recuo é exatamente do tamanho do raio R da ferramenta conforme visto na Figura 25.

3.5.2 Cálculo do avanço da ferramenta

Considerando-se o caminho como uma coleção ordenada de pontos, o avanço da ferramenta pode ser dado pela distância euclidiana entre os pontos adjacentes desse caminho [43] conforme a Figura 26.

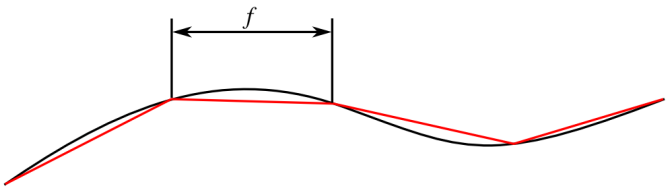


Figura 26: O caminho contínuo e o construído com avanço f.

O avanço da ferramenta é tido como um dos parâmetros de entrada para a determinação do caminho, indicando a amostragem da curva a fim de transformá-la em um conjunto de pontos discretos.

Durante a etapa de programação, esse conjunto de pontos é utilizado para determinação da trajetória do NC, que aproxima o caminho da ferramenta por um conjunto de segmentos de retas ou por um conjunto de arcos de circunferências, sendo esses os dois métodos tipicamente aceitos pela máquina.

A determinação do avanço da ferramenta depende da tolerância especificada para a superfície da peça, pois a aproximação posterior do caminho implica num erro ocorrido nele, que pode ser visto na Figura 26, como a diferença entre o caminho original (em preto) e o aproximado por segmentos de reta (em vermelho). O erro máximo de aproximação não pode ser maior que a tolerância exigida para a superfície da peça. Então, ao se considerar uma tolerância dada pelos requisitos de projeto, deve-se amostrar a curva adequadamente definindo o valor do avanço.

O processo de determinação do avanço depende da aproximação utilizada [36]:

a) Aproximação linear

Existem dois tipos de algoritmos para o cálculo do erro máximo de aproximação linear, o algoritmo direto e o algoritmo de curvatura.

Considerando-se um valor de incremento (Δu) no parâmetro da curva/caminho, é possível determinar a diferença máxima (erro de aproximação δ) entre a curva e a aproximação. Caso δ esteja fora da tolerância aceitada, redefine-se as escolha do incremento Δu até que a interpolação linear se aproxime da curva da forma desejada.

- **Algoritmo direto** – utilizando-se propriedades da **corda** do segmento de curva, o valor de δ é determinado (Figura 27)

$$\delta \simeq r\left(\frac{1}{2}\right) - \frac{1}{2}[r(1) + r(0)] \tag{3.22}$$

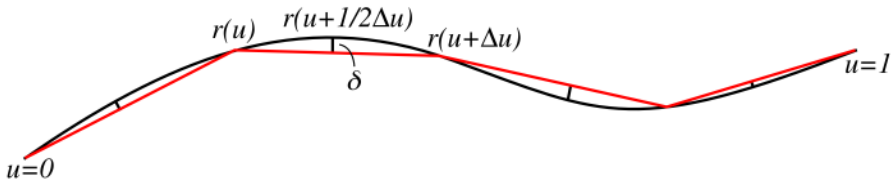


Figura 27: Interpretação do algoritmo direto [36].

- **Algoritmo da curvatura** – utilizando-se propriedades do **círculo osculante** descobre-se o valor de δ (Figura 28).

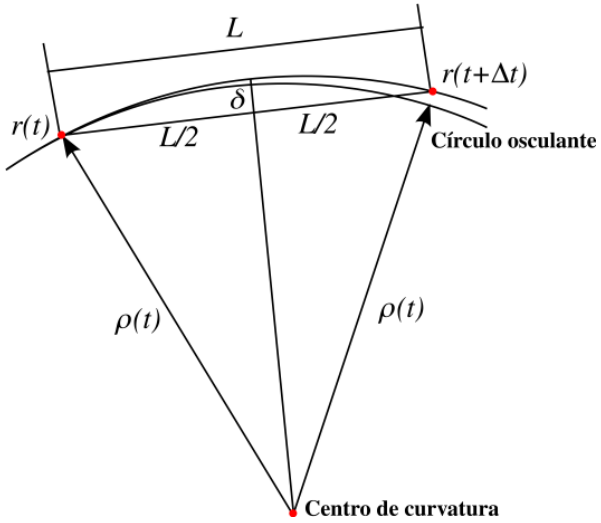


Figura 28: Interpretação do algoritmo da curvatura [48].

Aplicando-se o teorema de Pitágoras, é possível determinar o erro de aproximação δ:

$$\rho^2 = \left(\frac{L}{2}\right)^2 + (\rho - \delta)^2 \tag{3.23}$$

b) Aproximação por arcos

A curva é substituída por segmentos de arcos (como mostra a Figura 29), de forma que δ pode ser definido através dos seguintes passos:

- dados 3 pontos P_i, M, P_{i+1} e suas tangentes T_i, T_M, T_{i+1};
- calcular 2 arcos circulares de centro O_i e O_{i+1}, cujo raios de curvatura são ρ_i e ρ_{i+1};
- calcular a distância normal máxima (δ) entre o segmento verdadeiro e o arco circular.

Se a tolerância (δ) for satisfeita, o segmento de curva será trocado pelos dois arcos circulares; se não o segmento será dividido e o processo repetido.

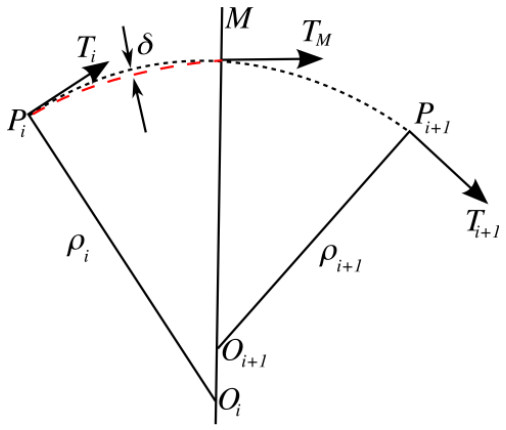


Figura 29: Interpretação do algoritmo de aproximação por arcos circulares [36].

A tabela 1 mostra uma comparação entre a aproximação linear e a aproximação por arcos.

Tabela 1: Aproximação linear X Aproximação por arcos

Propriedades	**Aprox. linear**	**Aprox. por arcos**
Complexidade	baixa	alta
Continuidade	C^0	C^1
Subdivisão	muita	pouca

Segundo Qiulin [36], é ideal utilizar a aproximação da curva por segmentos de retas quando ela não apresenta grande variações de curvatura, e a aproximação por arcos quando a curva apresentar curvaturas maiores.

3.5.3 Cálculo do passo da ferramenta

Para recobrir a superfície, é necessário que vários caminhos sejam percorridos e que caminhos adjacentes estejam a uma distância que satisfaça as condições desejadas do pro-

cesso. Essa distância entre dois caminhos adjacentes é chamada de passo da ferramenta.

Assumindo-se que a ponta da ferramenta de corte escolhida é esférica, a rugosidade da superfície dependerá do tamanho do passo dado pela ferramenta, quanto menor o passo menor será a rugosidade. A Figura 30 mostra o passo (L) e a altura (h) da rugosidade determinada pelo passo [36].

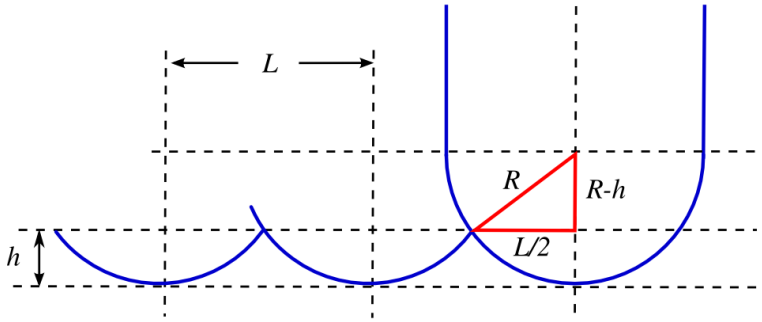


Figura 30: Passo da ferramenta [36].

O número de passos que a ferramenta deve dar ao longo da superfície, e o erro do acabamento máximo aceitável são dados pela seguinte relação:

$$R^2 = (R - h)^2 + \left(\frac{L}{2}\right)^2 \tag{3.24}$$

em que L é o passo da ferramenta, R o raio da ferramenta esférica dada e h altura do acabamento superficial.

Assim sendo, é possível escolher como parâmetro de entrada para o cálculo de caminhos o tamanho do passo da ferramenta sobre a superfície ou a altura da rugosidade máxima aceitável.

3.5.4 Cinemática inversa

Considerando-se o caminho da ferramenta sobre a superfície, a próxima etapa é converter as informações obtidas para o sistema de coordenadas aceito pela máquina NC. No caso de robôs manipuladores, os caminhos devem ser convertidos para o espaço das juntas do robô através do cálculo da cinemática inversa.

O processo de conversão para o espaço de coordenadas das juntas é não trivial e envolve a resolução de sistemas de equações não-lineares.

Toledo [11] propõe a geração da trajetória para as juntas de um robô Puma 560,

através da cinemática inversa, como é ilustrado nas Figuras 31, 32 e 33. A princípio, é feito o planejamento do caminho sobre uma superfície paramétrica (Figura 31), em seguida o cálculo da trajetória, que inclue velocidade e acelerações permitidas pelo efetuador (Figura 32), e, por fim, a geração da trajetória das seis juntas que o robô possui (Figura 33).

A Figura 31 ilustra o procedimento de geração de curvas cartesianas 3D, através de um caminho paramétrico (a), obtido através de pontos nos parâmetros de uma superfície de interesse (b), gerando um caminho de pontos cartesianos 3D (c) e uma aproximação com curvas B-Splines (d). Com isso, é possível gerar os dados para a trajetória cartesiana.

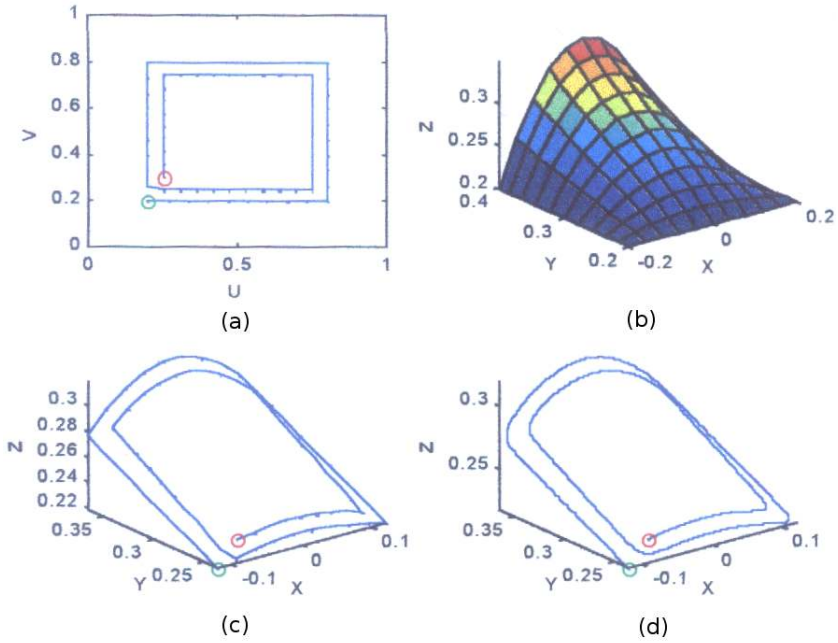


Figura 31: Visualização do caminho 3D da ferramenta [11].

A Figura 32 demonstra a saída gráfica da interface do programa, desenvolvido por Toledo [11], para a visualização da trajetória de posição, especificado pelo procedimento de geração de tarefas e ilustrado na Figura 31. Em 32, (a) é a visualização das curvas de posição, (b) a velocidade e (c) a aceleração no tempo do efetuador final do robô Puma 560 para a tarefa especificada.

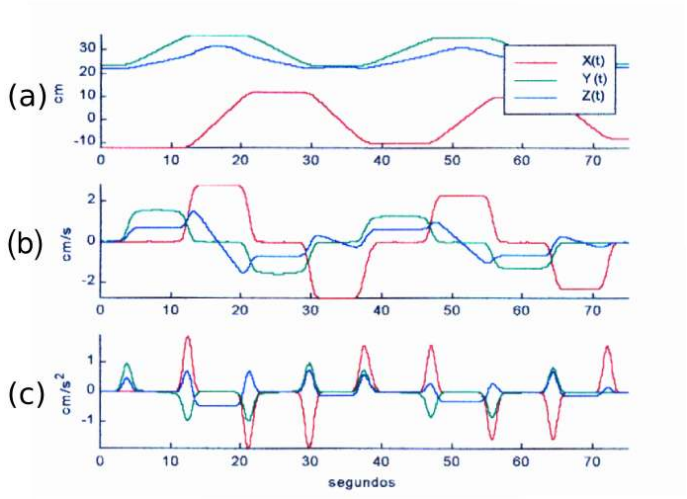


Figura 32: Visualização da trajetória do efetuador [11].

A trajetória no espaço das juntas de um manipulador tipo Puma 560, para uma tarefa como a da Figura 31, pode ser visualizada na Figura 33, em que (a) indica as posições das juntas, (b) a velocidade e (c) a aceleração.

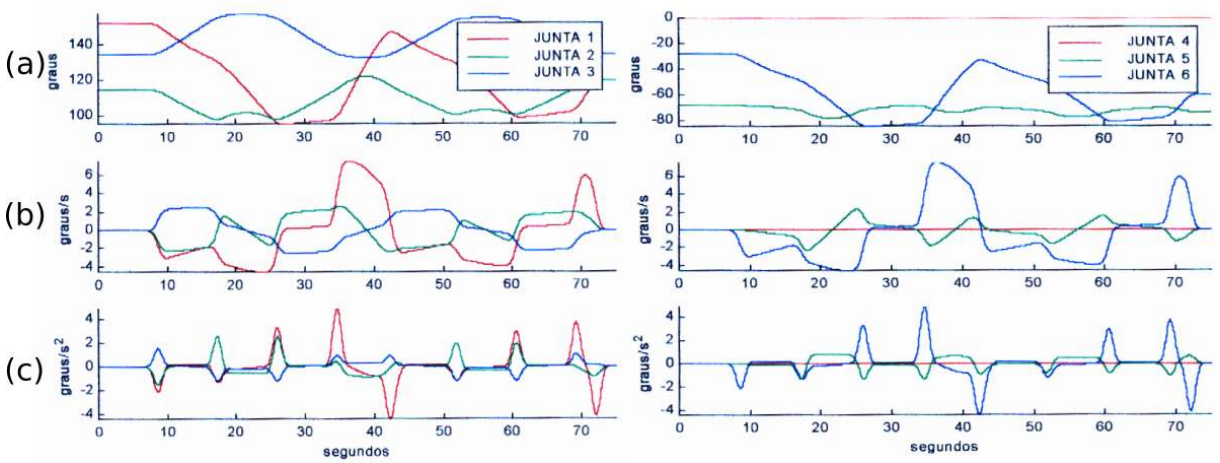


Figura 33: Visualização da trajetória no espaço das juntas no tempo respectivo às seis juntas [11].

3.6 Considerações

As ferramentas apresentadas serão utilizadas para a abordagem da proposta. Este capítulo ilustra algumas das várias possibilidades para o desenvolvimento do programa proposto, podendo ser aplicadas diferentes técnicas em outras abordagens.

O diagrama de Voronoi possibilita dividir o plano em regiões, de acordo com a regra do vizinho mais próximo, permitindo a ordenação topológica de um conjunto de pontos.

Uma reta pode ser segmentada por um polígono através do algoritmo de recorte de Cyrus-Beck. Esse procedimento é importante para o recorte de mediatrizes, segundo um polígono, durante a iteração do algoritmo construtor do diagrama de Voronoi.

Uma característica importante de curvas e superfícies de forma livre splines, Bézier e B-Splines, é que elas podem fornecer representações matemáticas que se ajustam ao conjunto de dados. Essas representações são utilizadas na reconstrução de superfícies, proporcionando uma visão facetada e aproximada da superfície original e fornecendo o conjunto de pontos sobre aquela que servirá como base para a geração do caminho da ferramenta.

4 O planejamento de trajetórias

Este capítulo apresenta uma proposta geral para o planejamento de trajetórias, visando à execução de tarefas de robôs manipuladores, e o desenvolvimento de métodos que busquem cumprir as etapas desse planejamento. O desenvolvimento dos algoritmos bem como os procedimentos utilizados são descritos de maneira seqüencial de acordo com a proposta geral. As aplicações, os resultados e a interface gerada serão apresentados no capítulo 5.

4.1 Uma proposta de modelo geral

Considerando-se as etapas necessárias ao processo de planejamento de trajetórias, conforme visto na seção 2.3 e as ferramentas básicas descritas no capítulo 3, é possível detalhar melhor a proposta de modelo geral. A Figura 34 categoriza e descreve cada uma das etapas do processo.

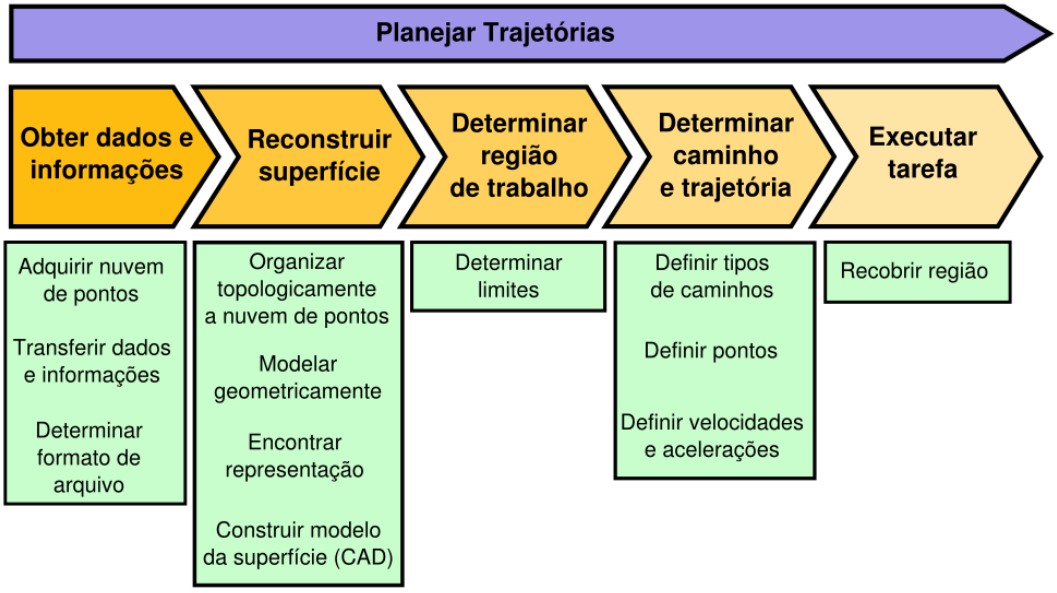


Figura 34: Proposta de modelo a ser estudada.

O processo de planejamento da trajetória é classificado em cinco etapas:

- obter dados e informações;
- reconstruir a superfície;
- determinar a região de trabalho;
- determinar o caminho e a trajetória;
- executar a tarefa.

Essas etapas serão detalhadas nas seções seguintes buscando-se descrever e ilustrar possíveis implementações para uma aplicação real.

4.1.1 Obter dados e informações

A etapa obter dados e informações (Figura 35) é a etapa inicial que foca a obtenção da informação sobre a região de interesse. Existem várias formas para se obter a informação da superfície de trabalho.

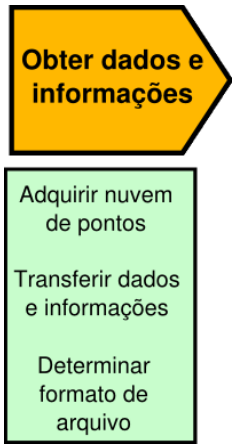


Figura 35: Obter dados e informações.

As formas as mais usadas atualmente são feitas, através de uma coleta, utilizando-se algum método de aquisição – máquina de medição de coordenadas (CMM), técnicas óticas com câmeras e iluminação especial, varredura a laser, entre outros –, gerando assim um conjunto representativo dos dados coletados (nuvem de pontos). Na Figura 36, são ilustrados três métodos de captura de dados: (a) Máquina de Medição por Coordenadas (CMM), (b) Câmeras e Iluminação e (c) Varredura a Laser.

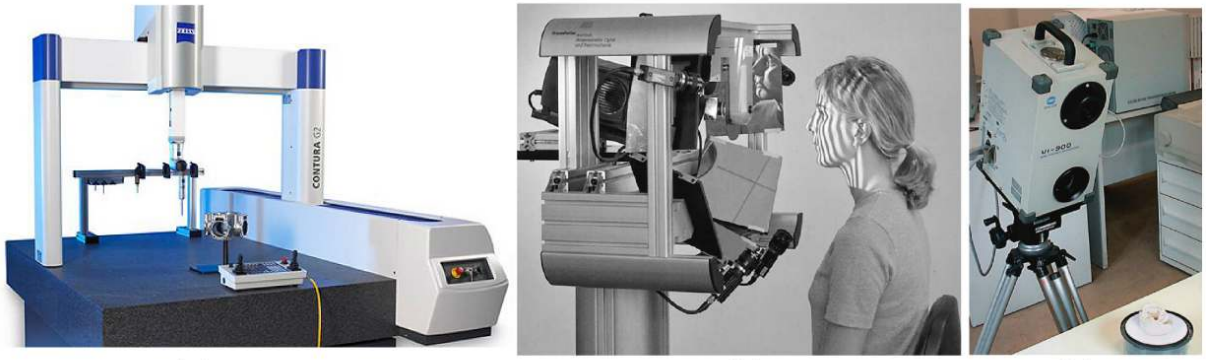


Figura 36: Máquinas de captura de dados – (a) Máquina de Medição de Coordenadas (CMM) [58], (b) Câmeras e Iluminação [39] e (c) Varredura a Laser [8].

Para que diversos sistemas possam compartilhar os dados obtidos, sem que as informações sejam perdidas ao serem manipuladas, eles devem ser armazenados de forma padronizada. Dessa forma, os dados podem ser compreendidos a partir da estrutura definida pelo padrão. Para isso, existem padrões universais de armazenagem de dados, sendo os mais conhecidos: IGES, STEP, ASCII [11].

É necessário que esses padrões suportem os dados a ser transferidos para que ocorra a recomposição do objeto de interesse no sistema a ser manipulado. Caso alguma informação não seja suportada pelo formato, ao se transferir esse arquivo de dados, ele não corresponderá aos dados originais, e assim haverá uma perda de informação (incompatibilidade com o objeto de interesse original). Isso também poderá ocorrer se não for utilizado algum padrão para se arquivar os dados. A incompatibilidade entre sistemas pode implicar na perda de informação ou em algum erro ao tentar transferir os dados de sistema para sistema.

4.1.2 Reconstrução da superfície

Na etapa de reconstrução (Figura 37), os dados são trabalhados de forma a gerar o modelo virtual, que possa ser manipulado.

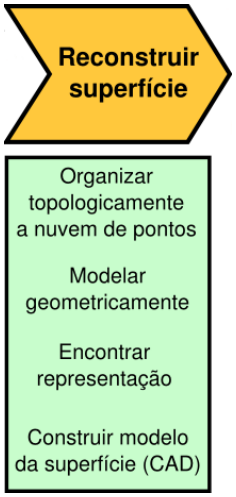


Figura 37: Reconstruir a superfície.

Alguns sistemas de medição de coordenadas conseguem fornecer o conjunto de pontos capturados da superfície de forma ordenada, reduzindo o processo de sua construção através de uma representação, de forma livre, que interpole os pontos ordenados e forneça o conjunto de pontos necessário para a geração do caminho sobre essa superfície.

Se as informações são fornecidas em forma de um conjunto de dados sem a ordenação previamente determinada, então, é necessário que se obtenha uma representação topológica da região, fornecendo assim a informação sobre a localização dos pontos no espaço, determinando quais estão mais próximos entre si. Essa etapa é realizada buscando-se obter um modelo que se aproxime ao máximo da região original. Uma representação através de malhas seria uma opção para organizar topologicamente o conjunto representativo de dados coletados. Caso os pontos adquiridos já possuam alguma regra de ordenação, o processo de geração de malhas pode ser efetuado facilmente, utilizando a regra de ordenação conhecida para criar uma representação topológica.

A Figura 38 ilustra a reconstrução de uma superfície da estrutura de um osso: (a) a partir de um conjunto de pontos representativos, (b) através da representação por malhas triangularizadas e (c) usando a sua reconstrução por B-Splines [34].

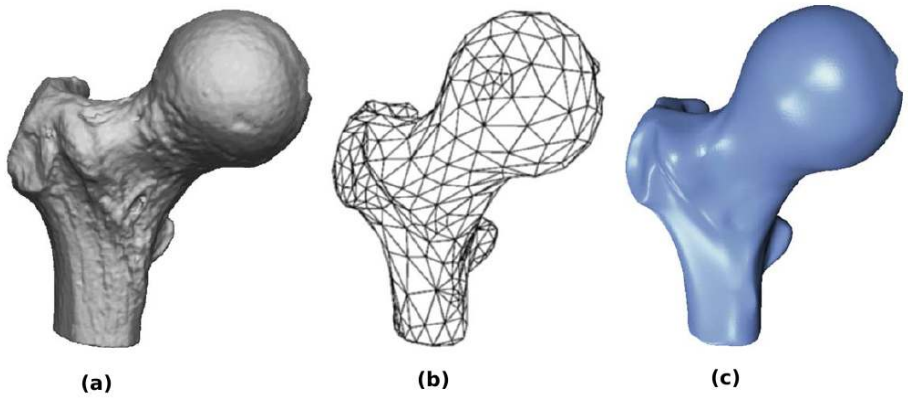


Figura 38: Reconstrução de uma superfície através de um conjunto de pontos – (a) Nuvem de pontos, (b) malha triangularizada e (c) superfície reconstruída por segmentos de superfícies B-Splines [34].

As técnicas, como o diagrama de Voronoi e triangularização de Delaunay, fornecem uma ordenação do conjunto de pontos e também uma representação topológica da superfície, que podem inclusive ser usadas como representações ilustrativas da superfície.

A representação topológica da superfície precisa ainda ser transformada em uma que permita manipulações. Para isso, uma representação matemática sintética seria ideal devido à sua facilidade de modificação.

Uma possibilidade para uma representação matemática sintética seria a utilização de superfícies splines, Bézier ou B-Splines, ou seja, superfícies de forma livre que possam ser representadas através de equações. As vantagens oferecidas por essas representações são: a interpolação pelo conjunto de pontos dados, o controle da suavidade da superfície e a facilidade de manipulações (como o cálculo de intersecções entre superfícies).

A reconstrução de superfícies também pode ser feita através de sistemas CAD, nos quais o modelo é desenhado pelo projetista a partir de informações de projetos e modificações por ele determinadas. A Figura 39 apresenta um sistema de modelagem CAD e a reconstrução de uma superfície.

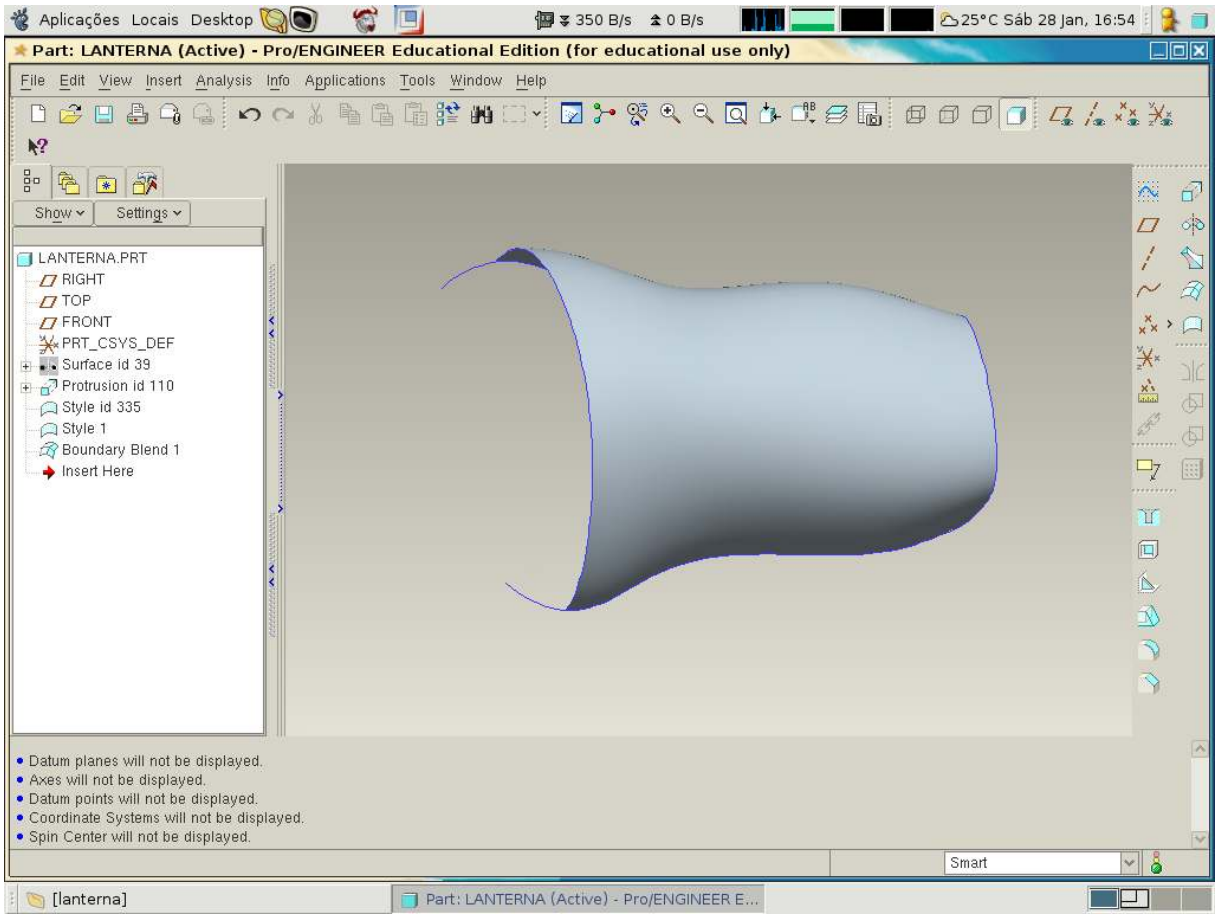


Figura 39: Sistema de modelagem CAD.

4.1.3 Determinar a região de trabalho

A região de trabalho é considerada a parte da superfície em que será planejada e executada a tarefa, podendo em alguns casos compreender toda essa superfície.

Quando a coleta de dados é realizada, ela possivelmente inclui uma margem de segurança, medindo regiões circundantes à região de interesse e regiões que possam apresentar características importantes. Essa margem de segurança é acrescentada de forma a garantir que toda a região esteja incluída na coleta, mesmo que ocorram pequenos deslocamentos.

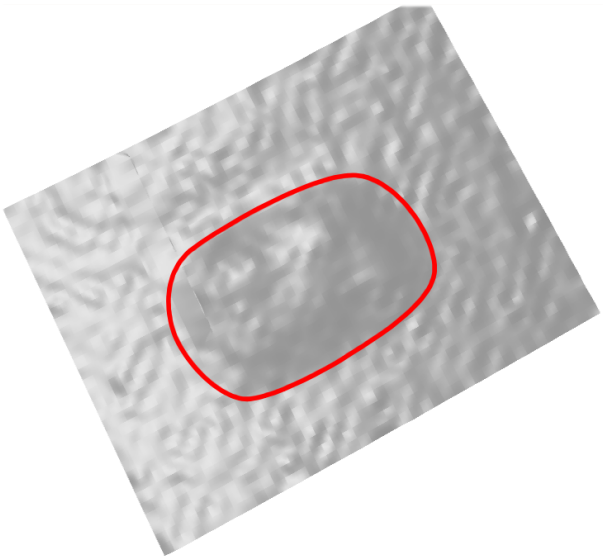


Figura 40: Região de trabalho.

Dessa forma, nem sempre toda a região coletada é a de interesse. Por isso, é necessário defini-la, como na Figura 40, em que a região interna ao contorno vermelho indica a de interesse. Nessa etapa (conforme Figura 41), é preciso selecionar/recortar apenas o que de fato constitui a região de interesse.

Figura 41: Determinar a região de trabalho.

A seleção da região de interesse pode ser obtida pelo cálculo de intersecções entre superfícies, onde é necessário possuir uma superfície original (fornecida pelos parâmetros de projeto) e a superfície em que estaria contida a região de interesse. A superfície resultante do cálculo de intersecção determina a diferença entre a superfície original e a superfície adquirida por métodos de captura de dados. Essa diferença é que constitui a região de interesse a ser manipulada posteriormente.

4.1.4 Determinar a trajetória

Durante a determinação da trajetória (Figura 42), é definido o caminho que o manipulador do robô deve percorrer para recobrir a região, passando pelos pontos pré-selecionados.

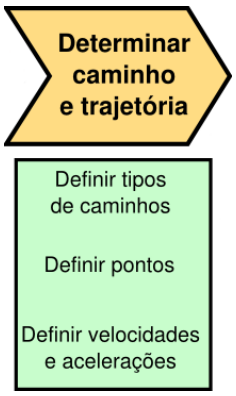


Figura 42: Determinar o caminho e a trajetória.

Os pontos do caminho a ser gerado podem ser encontrados utilizando-se técnicas como: baseado em APT, no espaço de trabalho (*iso-planar*, *iso-offset*, *iso-scallop*) e no espaço paramétrico (iso-paramétrico) conforme comentado no capítulo 2 (seção 2.2.1). A forma de recobrimento pode ser zig-zag, espiral ou movimento livre, passando pelos pontos já determinados. A Figura 43 ilustra a geração de um caminho no espaço paramétrico em zig-zag sobre uma superfície.

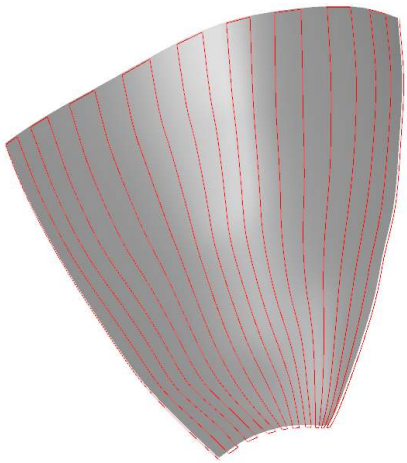


Figura 43: Caminho sobre uma superfície.

De acordo com o capítulo 3 (seção 3.5), as distâncias de avanço e os passos da ferra-

menta devem ser calculados a fim de satisfazer os requisitos de projeto, e o caminho deve ser planejado sobre a região de interesse.

Além das posições do efetuador final do manipulador, também podem ser necessárias as definições de velocidade e aceleração, pois algumas operações de manufatura exigem características especiais para a execução da tarefa (por exemplo, operações de soldagem à velocidade constante). Surge ainda a necessidade de executar o planejamento da trajetória no espaço das juntas do manipulador do robô (através do cálculo da cinemática inversa), quando ele for utilizado para a execução da tarefa.

4.1.5 Executar a tarefa

A etapa de execução da tarefa (Figura 44) consiste em se passar as informações e preparar a máquina. Além da trajetória, podem ser necessárias informações adicionais que completem a etapa de programação antes de se comandar a máquina para executar o programa.

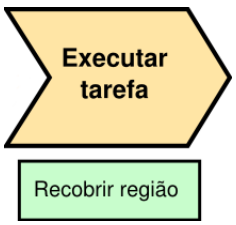


Figura 44: Executar a tarefa.

Todas as etapas do processo de planejamento da tarefa devem estar bem definidas e corretas, porque a falha em algum dos procedimentos pode ocasionar perdas do equipamento ou erros durante a execução. Por isso, muitas vezes é executada uma simulação para garantir a ausência de falhas. A Figura 45 exemplifica uma tarefa sendo executada por um robô.

Todos esses elementos aparecem de alguma forma durante o processo de planejamento do trajetórias. Em alguns casos, certas etapas podem ter prioridade sobre outras, no entanto, em algum momento do processo, todas elas tiveram que ter sido consideradas.

Figura 45: Roboturb-UFSC executando uma tarefa [40].

4.2 Desenvolvimento do método proposto

O desenvolvimento do método foi divido em quatro etapas descritas no diagrama de atividades exibido na Figura 46. Essas etapas foram baseadas no modelo geral proposto na seção 4.1. Através dele, é desenvolvido o algoritmo que reconstrói a superfície do objeto de interesse e que propõe um caminho sobre a superfície com o objetivo de verificar o modelo proposto.

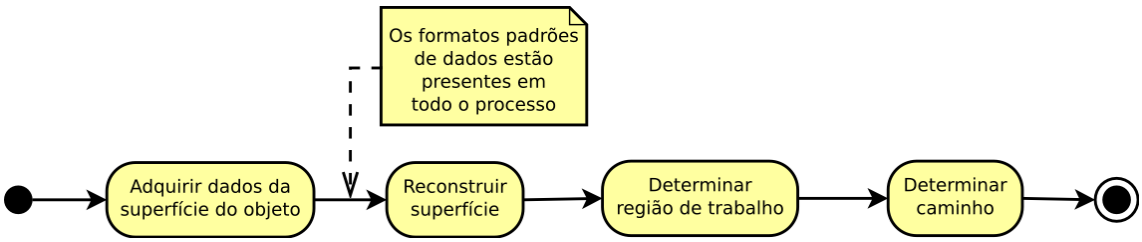


Figura 46: Diagrama de atividades do planejamento da trajetória.

O diagrama geral de atividades, do algoritmo desenvolvido, é ilustrado na Figura 46. Ele é composto das seguintes etapas:

- adquirir dados da superfície do objeto;
- reconstruir a superfície;
- determinar a região de trabalho;
- determinar o caminho.

Cada uma dessas etapas será detalhada nas próximas seções, descrevendo-se as estratégias utilizadas para a implementação. O formato padrão de arquivo de dados está presente em todo o processo.

4.2.1 A estrutura de dados e a aquisição de informação da superfície

A primeira etapa é a aquisição de informação da superfície de interesse. As informações seriam obtidas através de algum método de aquisição e estariam armazenadas em um arquivo. No método proposto, o formato do arquivo contendo as informações da superfície é ASCII com uma estrutura que armazena as coordenadas cartesianas na ordem `(x, y, z)` de cada ponto:

```
-59.522 34.097 -33.453
-59.620 27.820 -33.017
-59.617  8.991 -33.369
-59.590  2.816 -33.612
-59.590 -3.388 -33.713
-59.670 -9.698 -33.383
```

O diagrama de atividades para adquirir dados da superfície de projeto pode ser visualizado na Figura 47:

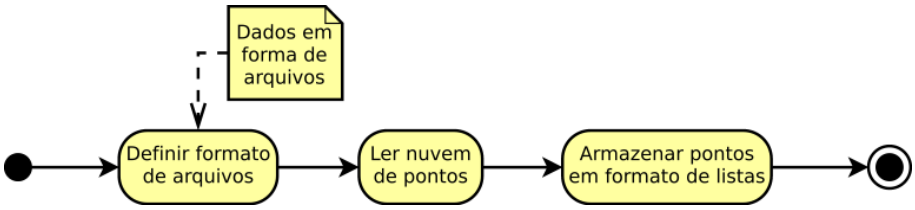


Figura 47: Diagrama de atividades para acesso e manipulação do arquivo de pontos.

Conhecido o formato do arquivo, é possível se processar a informação contida nele e armazená-la em formato de listas que serão utilizadas na etapa de reconstrução de superfícies.

4.2.2 Reconstrução da superfície

Com os dados da superfície armazenado em formato de listas, o próximo passo seria reconstruir a superfície. O diagrama de atividades da Figura 48 simboliza 3 etapas para esse processo.

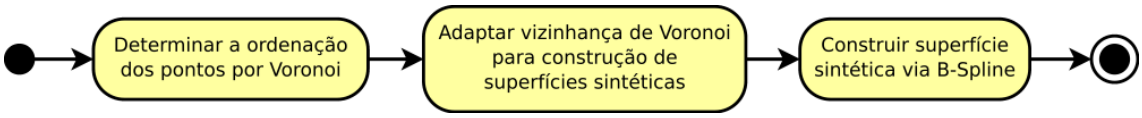


Figura 48: Diagrama de atividades para reconstrução da superfície.

Cada etapa será descrita com mais detalhes, incluindo a organização da estrutura de dados que deve ser levada em consideração em todo o processo de geração do diagrama de Voronoi. Os dados serão passados de uma etapa para a outra, por isso é importante que se dê uma atenção especial à forma com que eles serão trabalhados e armazenados.

4.2.2.1 Determinar a ordenação dos pontos por Voronoi

Antes de construir o diagrama de Voronoi, é necessário determinar como as suas entidades geométricas deverão estar estruturadas durante o desenvolvimento do algoritmo. Para isso, foi escolhida uma estrutura em forma de listas conforme proposto por Gomes[21].

a) Estrutura de dados

A estrutura de dados é composta de um conjunto de listas: para vértices, para arestas e para as células. Os vértices foram organizados através de duas listas: uma com os vértices centrais das células (que são os pontos obtidos pela aquisição da nuvem de pontos) e outra com os vértices extremos das arestas (que são calculados pelo algoritmo durante a construção do diagrama de Voronoi). Cada lista contém uma codificação (chave) de acesso a outras listas. Desse modo, define-se as células por referência às arestas e aos vértices centrais: as arestas fazem referências aos seus vértices extremos, e as listas de vértices contêm as coordenadas cartesianas de todos os vértices.

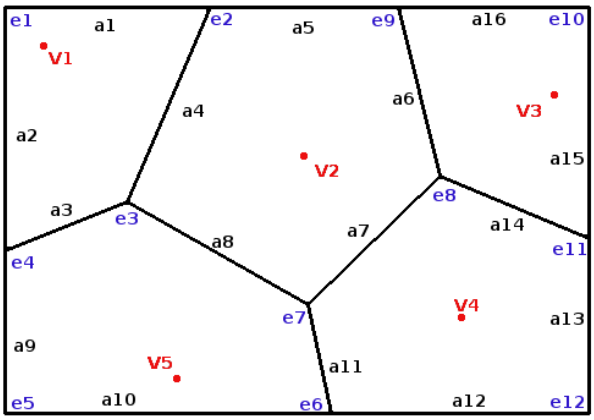


Figura 49: Diagrama de Voronoi para cinco vértices.

Sendo assim, as quatro listas do diagrama de Voronoi da Figura 49 podem ser encontradas nas tabelas 2, 3, 4 e 5.

As tabelas 2 e 3 são semelhantes, uma exemplifica a codificação para os vértices centrais de cada célula, e a outra exemplifica a codificação para os vértices extremos de cada aresta das células do diagrama de Voronoi, respectivamente. Cada ponto extremo da aresta recebe um código que o identifica. Se as coordenadas cartesianas desse ponto forem mudadas por um evento, durante alguma das etapas do algoritmo, todas as arestas que contêm esse ponto serão automaticamente atualizadas, devido à utilização do código de acesso, que permanece o igual, mesmo que suas coordenadas cartesianas mudem. Da mesma forma, é feito com relação aos vértices centrais da célula.

Tabela 2: Lista de vértices centrais.

Lista de vértices centrais
$\mathbf{v_1} = (x_1, y_1, z_1)$
$\mathbf{v_2} = (x_2, y_2, z_2)$
$\mathbf{v_3} = (x_3, y_3, z_3)$
$\mathbf{v_4} = (x_4, y_4, z_4)$
$\mathbf{v_5} = (x_5, y_5, z_5)$

Tabela 3: Lista de vértices extremos.

Lista de vértices extremos	
$\mathbf{e_1} = (x_{e1}, y_{e1}, z_{e1})$	$\mathbf{e_7} = (x_{e7}, y_{e7}, z_{e7})$
$\mathbf{e_2} = (x_{e2}, y_{e2}, z_{e2})$	$\mathbf{e_8} = (x_{e8}, y_{e8}, z_{e8})$
$\mathbf{e_3} = (x_{e3}, y_{e3}, z_{e3})$	$\mathbf{e_9} = (x_{e9}, y_{e9}, z_{e9})$
$\mathbf{e_4} = (x_{e4}, y_{e4}, z_{e4})$	$\mathbf{e_{10}} = (x_{e10}, y_{e10}, z_{e10})$
$\mathbf{e_5} = (x_{e5}, y_{e5}, z_{e5})$	$\mathbf{e_{11}} = (x_{e11}, y_{e11}, z_{e11})$
$\mathbf{e_6} = (x_{e6}, y_{e6}, z_{e6})$	$\mathbf{e_{12}} = (x_{e12}, y_{e12}, z_{e12})$

As arestas, além de fazer referência aos vértices extremos, guardam a informação dos vértices centrais vizinhos como pode ser visto na Figura 50. Então, cada aresta possui a informação dos códigos dos vértices extremos e dos dois códigos dos vértices centrais, que são divididos pela aresta (tabela 4). Quando a aresta é fronteira do diagrama, ela possui somente a informação de um vértice central.

Tabela 4: Lista de arestas.

Lista de arestas	
$\mathbf{a_1} \rightarrow e_1, e_2, v_1$	$\mathbf{a_9} \rightarrow e_4, e_5, v_5$
$\mathbf{a_2} \rightarrow e_1, e_4, v_1$	$\mathbf{a_{10}} \rightarrow e_5, e_6, v_5$
$\mathbf{a_3} \rightarrow e_4, e_3, v_1, v_5$	$\mathbf{a_{11}} \rightarrow e_7, e_6, v_5, v_4$
$\mathbf{a_4} \rightarrow e_3, e_2, v_1, v_2$	$\mathbf{a_{12}} \rightarrow e_6, e_{12}, v_4$
$\mathbf{a_5} \rightarrow e_2, e_9, v_2$	$\mathbf{a_{13}} \rightarrow e_{11}, e_{12}, v_4$
$\mathbf{a_6} \rightarrow e_8, e_9, v_2, v_3$	$\mathbf{a_{14}} \rightarrow e_8, e_{11}, v_3, v_4$
$\mathbf{a_7} \rightarrow e_7, e_8, v_2, v_4$	$\mathbf{a_{15}} \rightarrow e_{10}, e_{11}, v_3$
$\mathbf{a_8} \rightarrow e_7, e_3, v_2, v_5$	$\mathbf{a_{16}} \rightarrow e_9, e_{10}, v_3$

As células do diagrama de Voronoi têm seu nome associado à chave do vértice central (como pode ser visualizado na tabela 5). As células são, então, representadas pela chave dos vértices centrais, sendo o vértice da célula característica única de cada uma, não existindo dois vértices iguais. Dessa forma, os vértices centrais possuem propriedades ideais para uma chave de acesso à célula. Cada célula recebe como informação o código das arestas que a determinam.

Tabela 5: Lista de células.

Lista de células
$\mathbf{v_1} \longrightarrow a_1, a_2, a_3, a_4$
$\mathbf{v_2} \longrightarrow a_4, a_5, a_6, a_7, a_8$
$\mathbf{v_3} \longrightarrow a_6, a_{14}, a_{15}, a_{16}$
$\mathbf{v_4} \longrightarrow a_7, a_{11}, a_{12}, a_{13}, a_{14}$
$\mathbf{v_5} \longrightarrow a_3, a_8, a_9, a_{10}, a_{11}$

A Figura 50 ilustra como as entidades do diagrama de Voronoi estão relacionadas. Nela, é possível perceber que o diagrama de Voronoi é composto por células e que cada célula possui apenas um vértice e a informação das arestas. As arestas armazenam a relação de vizinhança e possuem a informação dos seus vértices extremos.

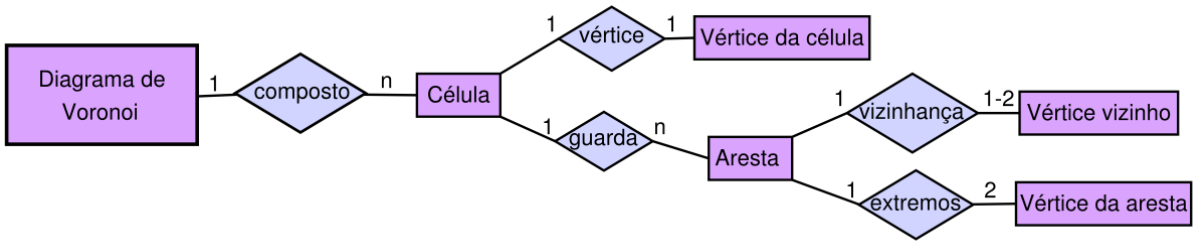


Figura 50: Diagrama de relações da composição do diagrama de Voronoi.

Essa codificação proporciona uma economia de espaço, pois cada vértice é armazenado uma única vez e acessado através de sua chave. Além disso, ao alterar as coordenadas cartesianas de um vértice ou uma aresta, todas as células que os contêm são alteradas de modo automático, facilitando a manipulação interativa da superfície e a consistência dos dados. Caso múltiplas cópias fossem armazenadas, todas precisariam ser modificadas ao se realizar uma operação.

b) Limitação do conjunto de pontos

Para evitar de se gerar o diagrama de Voronoi no espaço 2D, o conjunto de pontos é limitado, ou seja, a menor região que contenha todos os pontos é utilizada, sendo que o próprio conjunto de pontos no espaço já estabelece essa região. Para isso, são encontrados

os pontos extremos, que fazem parte da fronteira do conjunto de pontos no plano (os pontos que possuem coordenadas cartesianas máximas e mínimas), delimitando assim um retângulo (região), que contém todos os pontos a ser utilizados na reconstrução.

4.2.2.2 Construção do diagrama de Voronoi no plano

O algoritmo construtor do diagrama de Voronoi é implementado de forma incremental, ou seja, ao inserir um novo vértice, o algoritmo reconstrói a malha apenas na região em que esse novo vértice é inserido, fazendo uma atualização local dessa malha.

O sistema de coordenadas cartesianas é utilizado para o conjunto de pontos adquirido, porém o algoritmo foi implementado para o espaço paramétrico. O diagrama de Voronoi é construído no plano a partir de duas coordenadas cartesianas dos pontos, e, em seguida, é feito o mapeamento para o espaço (utilizando-se a terceira coordenada cartesiana) do seu dual, a triangularização de Delaunay, ou do próprio diagrama de Voronoi.

Dado um conjunto de n pontos distintos ($n > 1$) no espaço, é possível ordená-los e verificar se eles formam uma superfície suave. Para ordenar o conjunto de pontos, foi implementado o seguinte pseudo-algoritmo 2 gerador de diagramas de Voronoi.

Algoritmo 2 Pseudo-algoritmo do algoritmo do Diagrama de Voronoi no plano

```
1: Limitar o conjunto de pontos.
2: para cada ponto na lista de pontos:
3:     Criar uma célula para o novo ponto (vértice) e inseri-la na lista.
4:     se for a primeira célula então
5:         transformar a célula para compreender toda a região limitada.
6:     senão
7:         Verificar em qual célula o novo vértice foi inserido.
8:         Calcular a mediatriz entre o novo vértice e o vértice da célula na qual foi inserido.
9:         Fazer o recorte da mediatriz.
10:        Determinar as arestas da nova célula para o novo vértice.
11:        Verificar quais células são vizinhas à nova célula.
12:        Verificar quais células vizinhas foram afetadas pela nova célula.
13:        Corrigir células vizinhas afetadas.
14:        se a correção das células vizinhas afetar outras células então
15:            executar a correção para elas também.
16:        fim se
17:    fim se
18: fim para
19: Após criar células para todos os pontos, retornar a lista de células e finalizar a execução.
```

O diagrama de atividades que representa o pseudo-algoritmo pode ser visto na Figura 51:

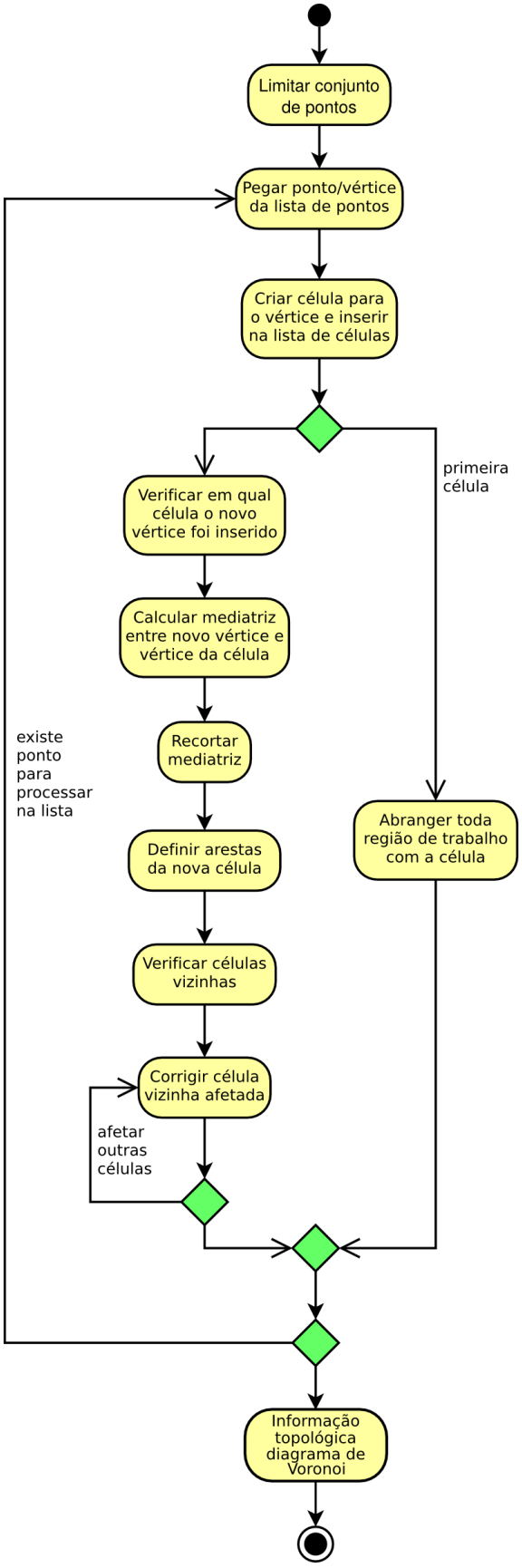


Figura 51: Diagrama de atividades de construção do diagrama de Voronoi no plano.

Cada aresta de uma célula do diagrama de Voronoi é dada pela mediatriz entre dois vértices. As mediatrizes são retas que precisam ser segmentadas para formar o polígono e assim as arestas das células. Para segmentar as retas, é utilizado o algoritmo de Cyrus-Beck.

O algoritmo de Cyrus-Beck executa o recorte da mediatriz (entre o novo vértice e aquele pertencente à célula na qual foi inserido). O polígono de recorte é, então, considerado a célula na qual o novo vértice é inserido. O procedimento de recorte é executado toda vez que um novo ponto é adicionado ao diagrama de Voronoi.

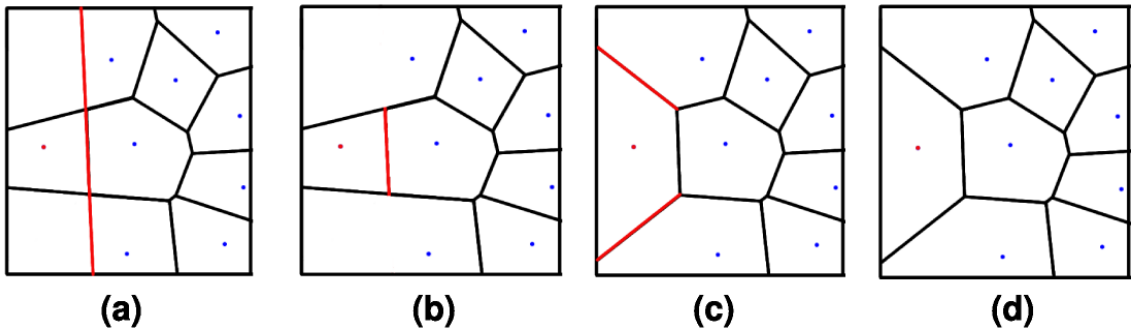


Figura 52: Procedimento de construção do diagrama de Voronoi no plano.

A Figura 52 ilustra geometricamente a adição de um novo ponto na malha já existente do diagrama de Voronoi. Na Figura 52 (a), é determinado em qual célula o novo vértice foi inserido e calculado a mediatriz entre ele e o vértice da célula. Em (b), é executado o recorte da mediatriz utilizando-se o algoritmo de Cyrus-Beck. Em seguida, é necessário determinar as arestas para a nova célula, mas, para isso, é preciso determinar quais são as células vizinhas ao novo vértice, calcular a mediatriz entre os vértices vizinhos e executar o recorte (Figura 52 (c)). Ao se fazer o recorte da mediatriz com os vértices vizinhos, pode ocorrer a danificação das células vizinhas, sendo necessário fazer uma verificação se o recorte não as danificou, terminando assim o procedimento para esse vértice (d). Ao se adicionar um novo vértice, é necessário reiniciar essas etapas.

Dessa forma, é construído o diagrama de Voronoi no plano para todos os pontos do conjunto de forma eficiente. Através disso, é possível construir a superfície que representa a superfície original cujo processo será descrito a seguir.

4.2.2.3 Reconstrução da superfície no espaço 3D

A forma da superfície pode ser aproximada de várias maneiras. Dentre elas, três maneiras utilizando a representação topológica obtida anteriormente são apresentadas. Uma delas aproxima a superfície a partir do diagrama de Voronoi, outra através da triangularização de Delaunay e a terceira através da técnica de interpolação de B-Spline. Elas fornecem uma visão aproximada da superfície original.

a) Reconstrução da superfície a partir do diagrama de Voronoi

O próprio diagrama de Voronoi, mapeado no espaço 3D, é uma representação da forma da superfície da peça. No entanto, esse diagrama é gerado a partir das coordenadas cartesianas 2D da nuvem de pontos. Como ele é calculado nos pontos de coordenadas cartesianas dos vértices da célula, será preciso interpolar as coordenadas cartesianas dos vértices do polígono da célula 2D para o espaço 3D. Para se obter o valor da coordenada cartesiana, é considerado um plano passando pelos vértices vizinhos V_1, V_2 e V_3 no espaço 3D e uma reta perpendicular ao plano (x,y) (onde foi gerado o diagrama de Voronoi) passando pelo vértice extremo da aresta a ser determinado. O valor da coordenada cartesiana do vértice extremo da aresta será o ponto de intersecção entre o plano formado por V_1, V_2 e V_3 e a reta conforme pode ser visualizado na Figura 53.

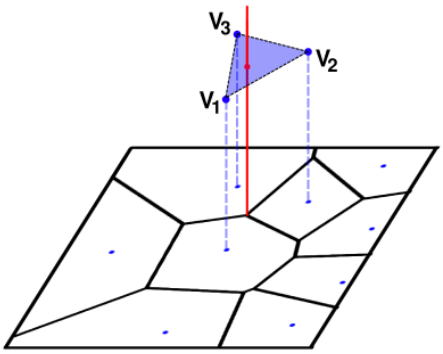


Figura 53: Cálculo da terceira coordenada z.

A Figura 54 exemplifica a superfície reconstruída através da construção das células do diagrama de Voronoi do plano para o espaço 3D. Em (a), é a representação em *wireframe* da superfície, e em (b) a representação sombreada. A construção do diagrama de Voronoi, mapeado no espaço, possibilita uma forma de visualização aproximada da superfície original. No entanto, o objetivo principal da utilização do diagrama de Voronoi neste trabalho

é a ordenação topológica do conjunto de pontos a qual será utilizada posteriormente para se obter uma representação matemática como superfícies B-Splines.

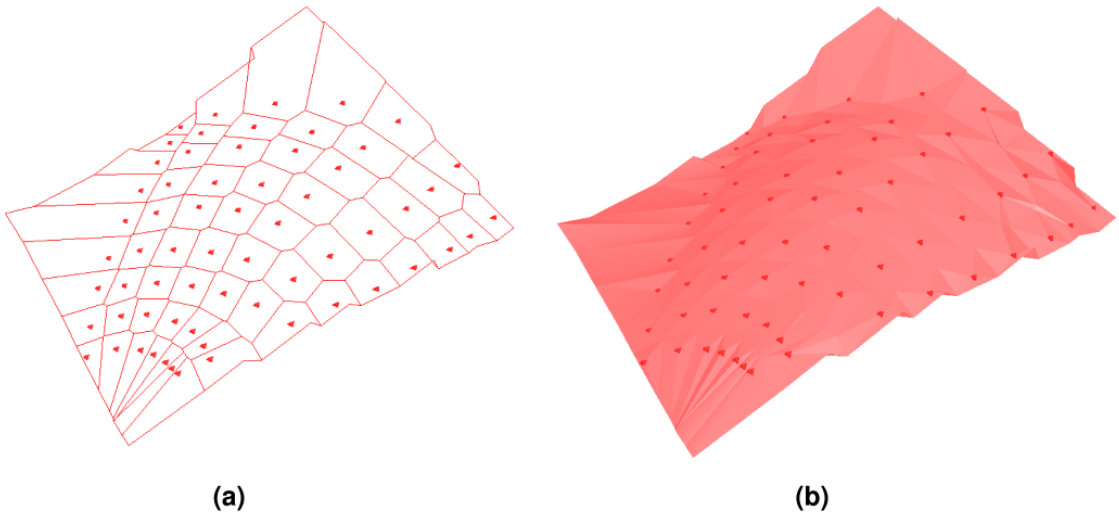


Figura 54: Superfície reconstruída a partir do diagrama de Voronoi – (a) *Wireframe* e (b) Sombreada.

b) Reconstrução da superfície através da triangularização de Delaunay

Considerando-se o diagrama de Voronoi construído no plano, é possível obter, a partir do seu dual, a triangularização de Delaunay no plano. Para isso, basta se unir os vértices das células vizinhas por um segmento de reta. Como a estrutura de dados já armazena a informação sobre os dois vizinhos, encapsulada nas arestas, o processo é bastante direto.

Tendo-se a triangularização de Delaunay no plano, o próximo passo é trabalhar com a terceira coordenada cartesiana que indica a profundidade da superfície. Assim, tem-se a superfície triangularizada, aproximada e reconstruída, conforme pode ser visto na Figura 55, onde (a) é a representação em *wireframe* da superfície e (b) a representação sombreada da superfície obtida a partir da triangularização de Delaunay.

A triangularização de Delaunay fornece uma forma da superfície facetada dada pelos triângulos obtidos pelo método.

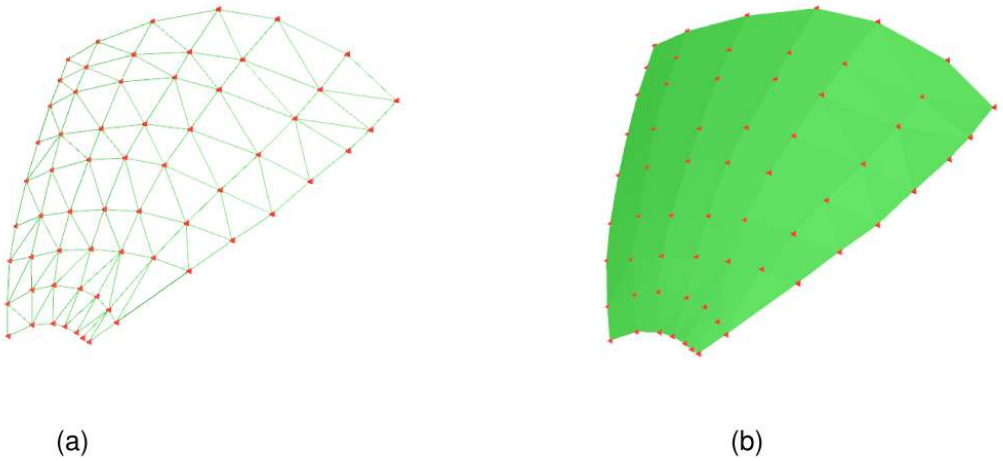


Figura 55: Superfície reconstruída a partir da triangularização de Delaunay – (a) *Wire-frame* e (b) Sombreada.

Essas duas representações da superfície reconstruída oferecem uma visão topológica da superfície, porém não oferecem nenhuma representação matemática para possíveis manipulações posteriores. No entanto, tal tipo de informação pode ser usado para detectar interferências rapidamente ou para acelerar a detecção de colisão entre a superfície e elemento robótico.

c) Reconstrução da superfície através de B-Spline

A reconstrução da superfície através de B-Spline ilustra o processo de reconstrução de superfícies de forma livre e oferece uma representação matemática sintética. A seguir, será descrito detalhadamente o processo de reconstrução por B-Spline, considerando-se o diagrama de Voronoi construído no plano. *A priori*, é necessário que se reveja a estrutura gerada pelo diagrama de Voronoi, que será descrita na subseção 4.2.2.4, e que descreve a próxima etapa do processo de reconstrução de superfícies (Figura 48).

4.2.2.4 Adaptar a vizinhança de Voronoi para a construção de superfícies sintéticas

A ordenação da nuvem de pontos, através do diagrama de Voronoi, no plano, é usada para obter a representação sintética da superfície. O diagrama de atividades, mostrado

na Figura 56, ilustra o processo realizado para adaptar o arquivo de dados que contém a relação de vizinhança, fornecida pelo diagrama de Voronoi, para a reconstrução de superfícies sintéticas.

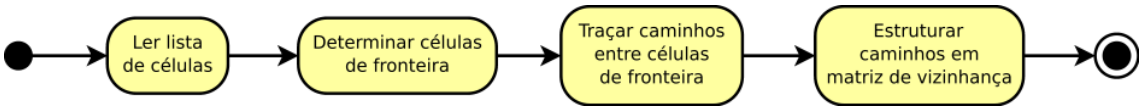


Figura 56: Diagrama de atividades para adaptar a vizinhança de Voronoi para a reconstrução de superfícies.

O arquivo com o conjunto de pontos ordenados pelo diagrama de Voronoi contém a informação necessária para a reconstrução da superfície B-Spline, mas é preciso estruturá-lo para se poder aplicar o algoritmo que constrói a superfície B-Spline. A princípio, são encontradas as células que fazem parte da fronteira do diagrama de Voronoi. As células de fronteira são caracterizadas por possuir área infinita na definição do diagrama de Voronoi, todavia é preciso lembrar que no algoritmo proposto o conjunto de pontos é limitado por uma região retangular (como comentado no final da subseção 4.2.2.1).

Os pontos que fazem parte da fronteira e os quatro cantos da superfície são os elementos que compõem a primeira e última linha e a primeira e última coluna da matriz. Os pontos intermediários são determinados pela relação de vizinhança fornecida pelo diagrama de Voronoi e o caminho que liga as colunas e/ou linhas da matriz.

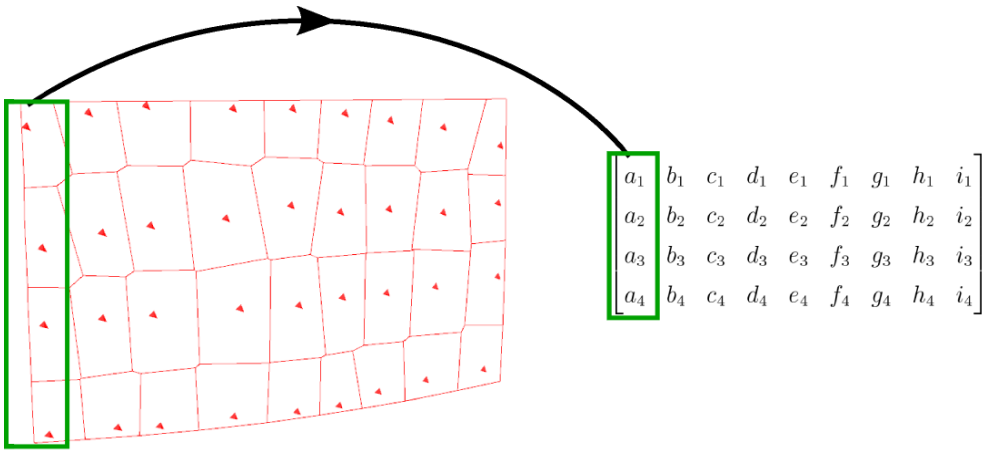


Figura 57: Diagrama de Voronoi no plano e matriz com o conjunto de pontos ordenados.

A Figura 57 ilustra a relação entre o diagrama de Voronoi no plano e a matriz que será

utilizada para a reconstrução da superfície B-Spline (é necessário levar em consideração que o número de elementos nas linhas/colunas da matriz deve ser igual).

Para determinar os elementos que fazem parte da fronteira do conjunto de pontos do diagrama de Voronoi, foi utilizado o seguinte critério: determinar as células que possuem arestas que não separam dois vizinhos, ou seja, a aresta que faz divisão com um vizinho nulo. Assim, é possível se descobrir todas as células que fazem parte da fronteira do diagrama de Voronoi.

Os vértices da fronteira são representados na "borda" da matriz de vizinhança que será utilizada posteriormente para o cálculo da superfície B-Spline.

Definindo-se os caminhos que ligam as células de fronteira, passando pelas células internas, é possível complementar a matriz de vizinhança. Esses caminhos são obtidos utilizando-se informações topológicas fornecidas pelo diagrama de Voronoi.

4.2.2.5 Construir superfície sintética via B-Spline

Com os pontos ordenados e estruturados pelo diagrama de Voronoi, a próxima etapa para haver uma representação da superfície B-Spline é se utilizar alguma das representações de superfícies de forma livre comentadas no capítulo 3. O algoritmo proposto utiliza a teoria de superfícies B-Splines uniforme para a reconstrução de superfícies.

A representação por B-Spline foi escolhida por ter algumas vantagens sobre as outras representações, como controle local da forma da superfície, interpolação dos pontos dados (obtidas através do algoritmo inverso), flexibilidade do grau da curva, ou da superfície, e por se tratar de uma generalização de outras representações.

O diagrama de atividades, ilustrado na Figura 58, descreve as etapas do algoritmo desenvolvido para a reconstrução da superfície B-Spline.

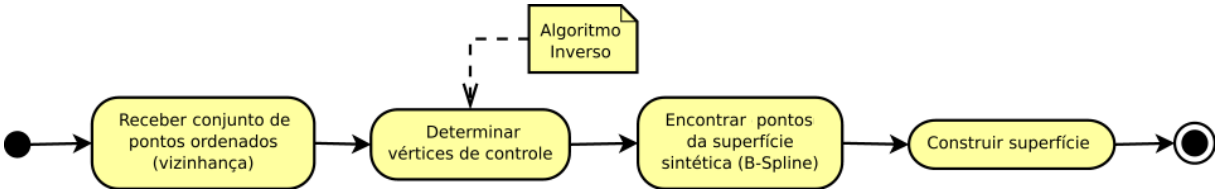


Figura 58: Diagrama de atividades para representação sintética através de B-Spline.

Com a matriz de vértices já preenchida com todos os elementos, conforme seção ante-

rior (4.2.2.4), é possível ser aplicado o algoritmo inverso de B-Spline, detalhado na secção 3.4.1 do capítulo 3, e, em seguida, o algoritmo construtor de B-Spline conforme proposto por Qiulin [36]. A Figura 59 exemplifica uma reconstrução de superfície por B-Spline.

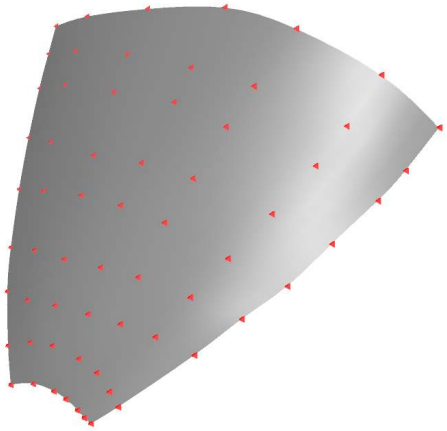


Figura 59: Superfície B-Spline reconstruída.

O algoritmo inverso fornece os vértices de controle da superfície B-Spline, a partir da matriz ordenada que contém os pontos nós representativos, possibilitando que a superfície reconstruída passe pelos pontos ordenados a partir do algoritmo construtor do diagrama de Voronoi. A seguir, pode ser aplicado o algoritmo construtor de superfícies B-Splines que recebe como parâmetro de entrada os vértices de controle e retorna os pontos da superfície B-Spline.

Esse tipo de representação fornece uma forma mais suave da superfície. A superfície B-Spline é necessária para calcular de forma mais exata as propriedades em operações de manufatura, como o planejamento do caminho e trajetória, operações de recobrimentos, cálculo e simulação de propriedades mecânicas como curvatura, volume e análise de tensão.

4.2.3 Determinar a região de trabalho

Não foram implementados algoritmos particulares que determinassem a região de trabalho, pelo fato de ela ser fortemente dependente do tipo de aplicação no qual o processo de planejamento de trajetória será utilizado.

No algoritmo proposto, é assumido que a região de trabalho é toda a superfície obtida através do conjunto de pontos amostrados. Porém, se a região de trabalho não compreende toda a superfície, é fácil se adaptar o algoritmo para as regiões de interesse.

Uma solução primária é a aplicação novamente do algoritmo somente para os pontos

selecionados que pertencem à região de trabalho, que criará uma nova superfície que compreenderá toda essa região de trabalho, podendo-se por exemplo, utilizar algum método, como cálculo de intersecção entre superfícies, para determinar os pontos selecionados.

4.2.4 Determinar o caminho sobre a superfície

Determinada a região que a ferramenta deve percorrer, o próximo passo seria determinar o caminho a ser percorrido pelo efetuador final do robô. Para isso, foram implementados três algoritmos para a geração do caminho da ferramenta: no espaço paramétrico (iso-paramétrico) e no espaço de trabalho (iso-planar e *iso-scallop*).

4.2.4.1 Geração do caminho da ferramenta no espaço paramétrico

O algoritmo que determina o caminho no espaço paramétrico foi implementado levando-se em consideração os segmentos da superfície B-Spline. Dado o segmento inicial e o avanço da ferramenta desejado, o caminho é determinado, ao passar pelos segmentos, seguindo uma das direções até o segmento final. Em seguida, o caminho de volta da ferramenta é gerado, quando é somado, o valor do passo da ferramenta ao parâmetro referente a outra direção do caminho, obtendo assim a malha de caminhos no espaço paramétrico que, após, é transferida para a superfície como é ilustrado na Figura 60.

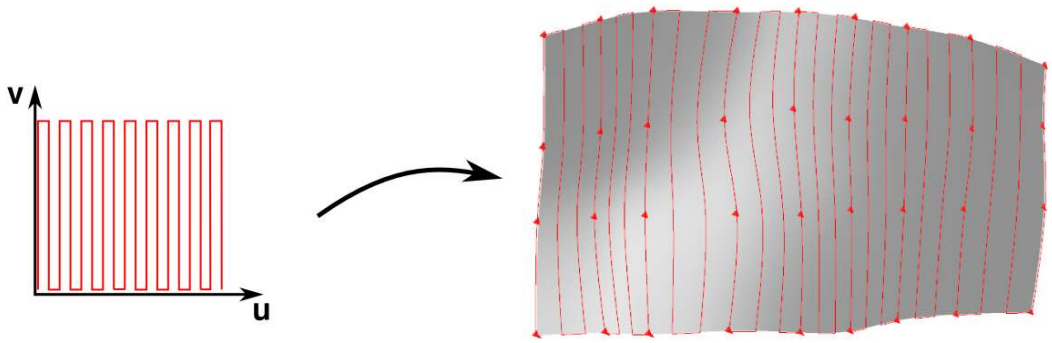


Figura 60: Caminho no espaço paramétrico.

4.2.4.2 Geração do caminho da ferramenta no espaço de trabalho

A geração do caminho da ferramenta **iso-planar** é determinada considerando-se a superfície reconstruída B-Spline e um plano. A curva gerada pela intersecção do plano

com a superfície B-Spline estabelece um caminho sobre a superfície. A intersecção de planos transladados em uma dada direção a uma distância estabelecida pelo passo da ferramenta fornece a malha de caminhos sobre a superfície, como pode ser visualizado na Figura 61, onde (a) e (b) ilustram duas possibilidades de direção do plano.

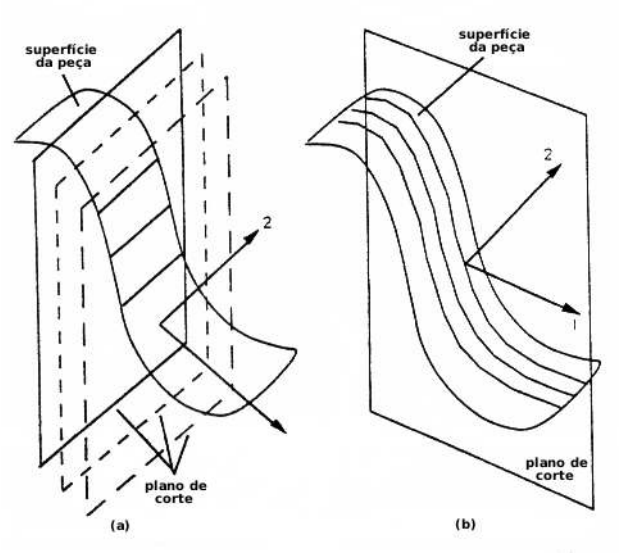


Figura 61: Superfície e plano.

A direção do plano de corte influencia nos resultados da geração do caminho da ferramenta. No entanto, a decisão da escolha da direção de um plano ótimo é um problema de difícil resolução [50]. No algoritmo proposto, a direção do plano é fixa e é paralela ao plano XZ do sistema de coordenadas de trabalho.

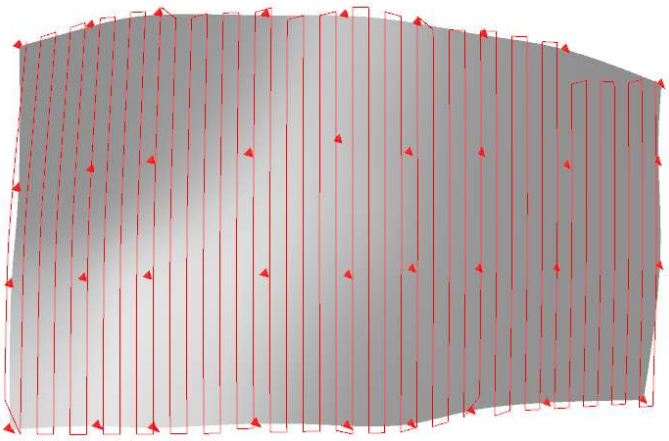


Figura 62: Caminho iso-planar gerado sobre o espaço de trabalho.

A Figura 62 ilustra o resultado obtido para o procedimento de geração do caminho

iso-planar no espaço de trabalho.

A geração do caminho da ferramenta ***iso-scallop*** é realizada considerando-se a superfície reconstruída B-Spline tendo como objetivo manter a distância constante da altura da rugosidade em uma operação de usinagem. O primeiro caminho é gerado a partir de uma das fronteiras da superfície, e os caminhos subseqüentes são paralelos (mantêm distância constante ao caminho anterior) de forma a manter a rugosidade constante. A abordagem utilizada para o cálculo dos caminhos paralelos foi baseada no método proposto por Suresh[50].

Considerando-se o primeiro caminho dado, para cada ponto desse caminho são encontrados os pontos que estão a uma distância L, dada pelo passo da ferramenta desejada, gerando um conjunto de pontos paralelos ao primeiro caminho. A união desses pontos paralelos originará uma curva paralela ao caminho anterior. Repetindo-se recursivamente essa operação, é possível construir todos os caminhos paralelos, de forma que o recobrimento da superfície possa ser dado por caminhos que estão a uma distância constante entre si. A Figura 63 ilustra a geração do caminho *iso-scallop* da ferramenta.

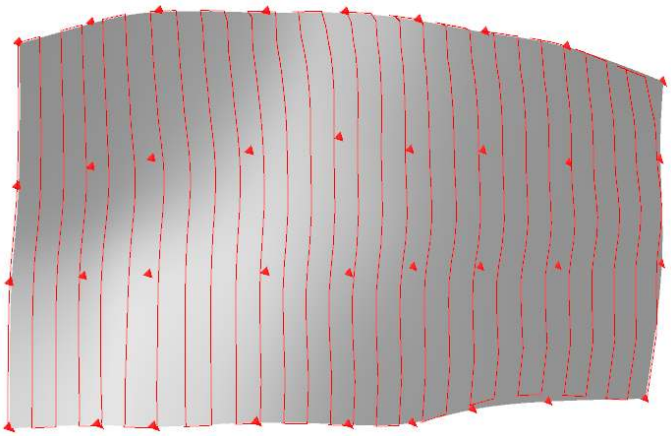


Figura 63: Caminho *iso-scallop* gerado sobre o espaço de trabalho.

4.3 Resultados e discussão

O algoritmo foi implementado utilizando-se a linguagem de programação Java [12, 49]. Essa linguagem foi escolhida por oferecer principalmente portabilidade, pela variedade de ferramentas e pela facilidade para posterior uso em outros projetos.

O valor de entrada para o algoritmo do Voronoi é apenas os pontos amostrados. O resultado da aplicação do diagrama de Voronoi é capaz de representar topologicamente a

superfície.

As superfícies obtidas a partir do diagrama de Voronoi e da triangularização de Delaunay são representações geométricas da superfície original, possibilitando um visualização aproximada da superfície, sem oferecer uma representação matemática como as geradas através de segmentos de superfícies B-Splines.

O algoritmo implementado para a geração do caminho da ferramenta iso-paramétrico é mais simples, porém os resultados não são tão bons, quando a curvatura da superfície apresenta grandes variações. A distância entre dois caminhos adjacentes no espaço paramétrico é constante, e, ao ser transformado para o espaço de trabalho, essa distância pode variar consideravelmente.

O algoritmo implementado para a geração do caminho iso-planar oferece maior controle na forma do caminho, sendo que o plano poderá futuramente ser substituído por outras superfícies, como cilindros, parabolóides, dependendo da forma dos caminhos necessários para a execução da tarefa. Já a geração do caminho *iso-scallop* garante que a distância entre os caminhos seja constante, o que pode ser muito importante em alguns processos de usinagem.

Os algoritmos propostos apresentam uma visão completa da superfície do objeto de trabalho, fornecendo informações topológicas e estruturais necessárias para o cálculo de geração do caminho para a execução da tarefa.

5 Aplicação do Método

Neste capítulo, são apresentados detalhes do protótipo desenvolvido, comentários de instalação e modo de execução. É realizada a aplicação do programa para alguns conjuntos de pontos (nuvem de pontos), visando-se a reconstrução da superfície e a geração de caminhos para a ferramenta sobre a superfície reconstruída. Por fim, são comentados os resultados obtidos, a divisão geral da estrutura do programa e suas limitações.

5.1 Abrangência do programa no processo de planejamento de trajetórias

De acordo com a proposta de sistematização para o planejamento de trajetórias de robôs manipuladores, foi desenvolvido um protótipo que abrange várias etapas da proposta. Comparando-se o modelo com o protótipo desenvolvido, as etapas da sistematização que o protótipo realiza são descritas nos itens a seguir:

- processar os dados obtidos através de algum método de aquisição no formato de arquivo ASCII com estrutura definida;
- reconstruir a superfície geometricamente e encontrar uma representação matemática sintética;
- considerar a região de trabalho utilizando todo o conjunto de dados obtidos;
- determinar o caminho da ferramenta sobre a superfície.

A Figura 64 destaca (em vermelho) as etapas do modelo proposto que o protótipo executa.

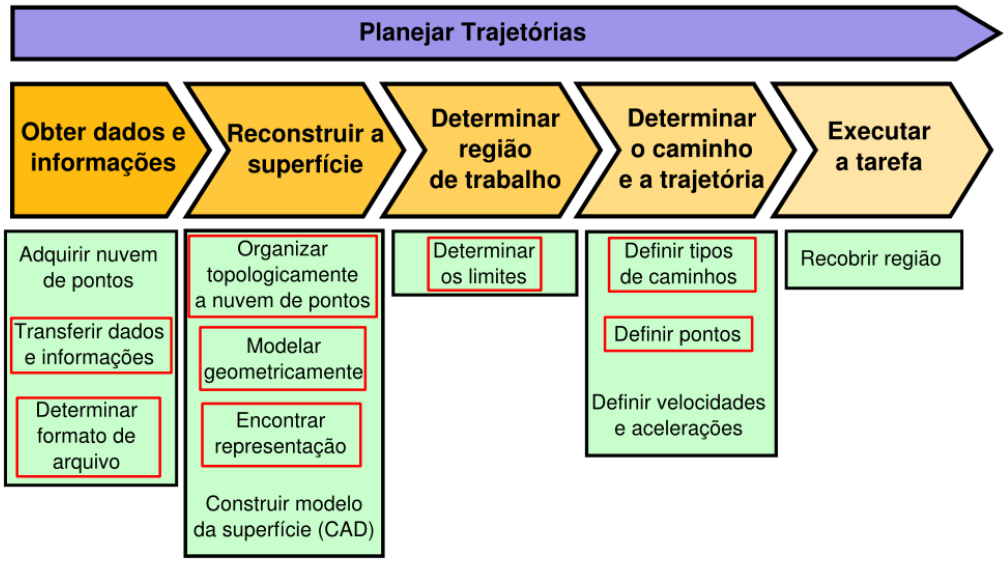


Figura 64: Etapas do planejamento de trajetórias.

De todas as cinco etapas descritas, quatro delas estão presentes durante a execução do programa. A etapa **executar tarefa** cabe à estrutura mecânica realizar, devendo o programa somente salvar os dados corretamente a fim de os transferir para a máquina que irá executar a tarefa. De maneira geral, as etapas que não puderam ser satisfeitas pelo protótipo correspondem àquelas que dependem diretamente da estrutura física específica de cada aplicação, como as máquinas que realizam a aquisição de dados e o comando numérico que irá realizar a trajetória.

Na etapa **determinar caminho e trajetória**, foi desenvolvido apenas o algoritmo que constrói o caminho sobre a superfície, e o cálculo das velocidades e acelerações não foram implementadas por serem dependentes dos parâmetros da tarefa a ser executada e da estrutura mecânica que irá realizá-la.

5.2 Utilização do programa

O algoritmo foi implementado utilizando-se a linguagem de programação JAVA (`Sun - Java(TM) Development Kit (JDK) 5.0`) [12, 49, 47]. Foi necessário utilizar algumas bibliotecas extras disponíveis de forma livre na internet, sendo uma chamada `Jama-1.0.2` [26], que adiciona funcionalidades relativas à manipulação e operação de matrizes, e outra chamada `java3d-1.4.0.01` [49], utilizada para visualização e manipulação dos objetos 3D e a biblioteca `CyberVRML97` para salvar os resultados do aplicativo no formato de arquivo

VRML [10]. O protótipo desenvolvido possui as seguintes capacidades:

- entrar com dados do conjunto de pontos no formato de arquivos ASCII com estrutura definida;
- reconstruir e visualizar superfícies a partir de três formas diferentes (diagrama de Voronoi, triangularização de Delaunay e B-Splines);
- permitir a geração de caminhos da ferramenta no espaço paramétrico e no espaço de trabalho, com entrada de valores para o avanço e passo da ferramenta;
- visualizar as superfícies em *wireframe* e sombreadas;
- salvar e abrir arquivos em formatos compatíveis com outros programas;
- permitir a manipulação da visualização utilizando a interface padrão via **mouse** (controle de *zoom*, rotação e translação).

5.2.1 Instalação e modo de execução

Para a instalação e execução do programa, é necessário que o usuário tenha instalado em sua máquina o ambiente de execução JAVA (*Java Runtime Environment* – JRE Versão 5.0), da SUN, disponível na página `http://www.java.com/`.

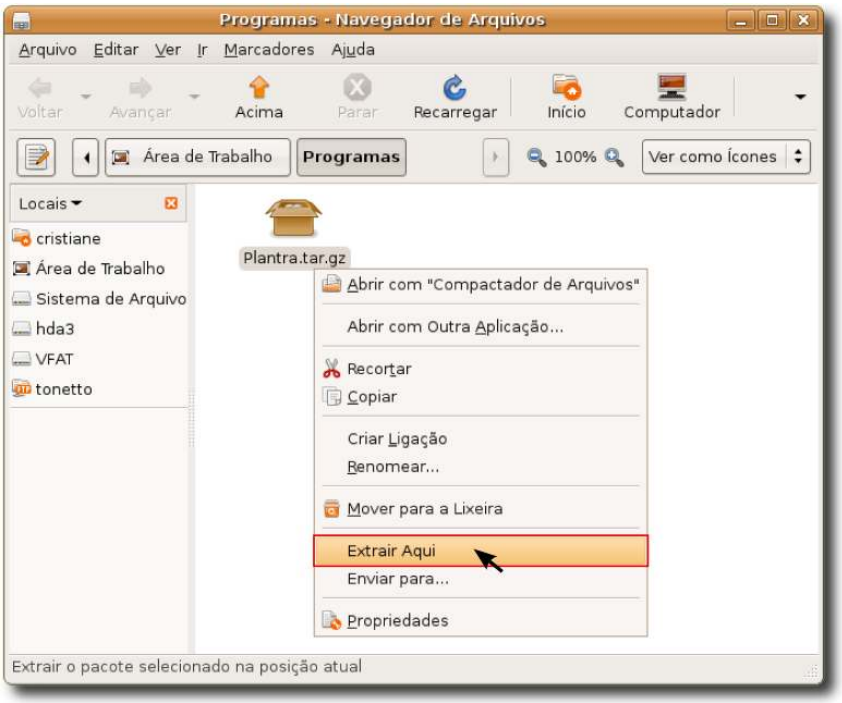


Figura 65: Descompactando o arquivo para instalação.

O protótipo é distribuído em dois possíveis pacotes com o mesmo conteúdo: `Plantra.tar.gz` e `Plantra.zip`. Com um dos pacotes no computador, o usuário, para instalar o programa, deve descompactá-lo (Figura 65). Dentro da pasta extraída estão todos os arquivos necessários para a execução desse programa.

Para executar o programa, o usuário pode dar um duplo clique no ícone `plantra_linux` para usuários GNU/LINUX ou `plantra_MSWindows.bat` para usuários MS Windows (Figura 66), e o protótipo irá iniciar.

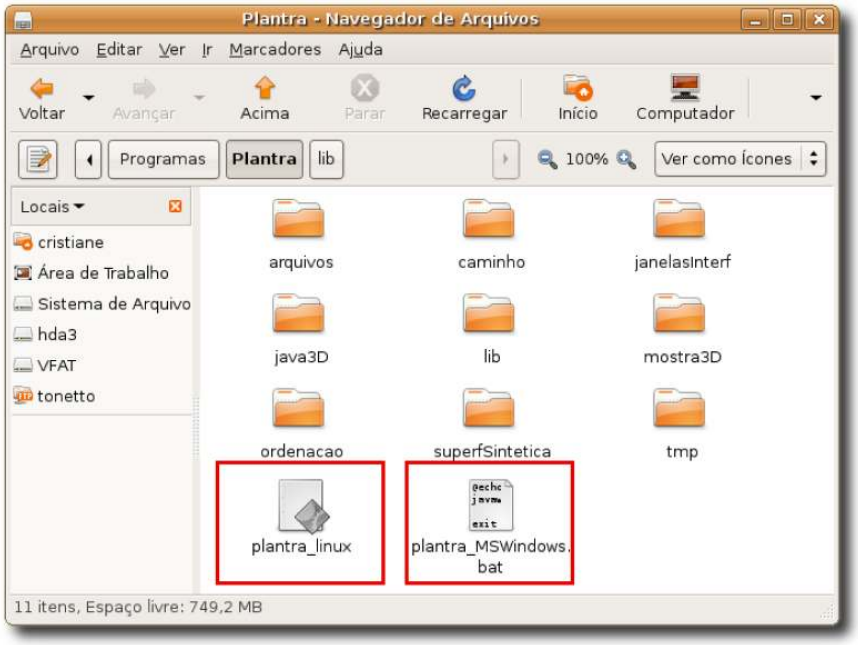


Figura 66: Ícones de execução do programa.

O protótipo está disponível na página:

`http://www.labcadcam.ufsc.br/~cris.tonetto/`

Junto com o arquivo estão disponíveis todos os códigos-fontes necessários para a execução do programa distribuído sob a licença GPL [22](conforme o anexo B).

5.2.2 Interface do protótipo

A interface é composta por duas janelas interativas principais (conforme Figura 67), uma para escolha das opções de trabalho e outra para manipulação e visualização do objeto. A escolha por duas janelas se deu porque sem os menus no ambiente de visualização, este último pode ocupar toda a tela ampliando assim a área de visualização.

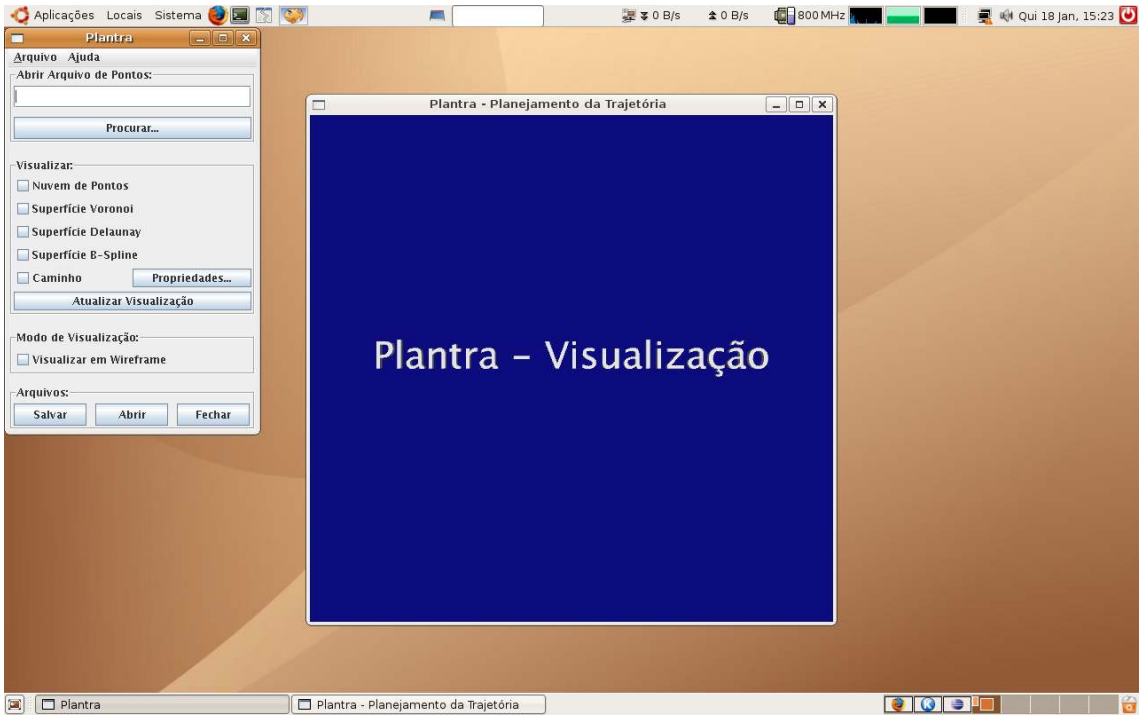


Figura 67: Interface do programa desenvolvido.

A janela com menus é composta de quatro áreas como indicado na Figura 68. A área indicada pelo número 1 possibilita ao usuário selecionar o arquivo com os dados da superfície a ser reconstruída.

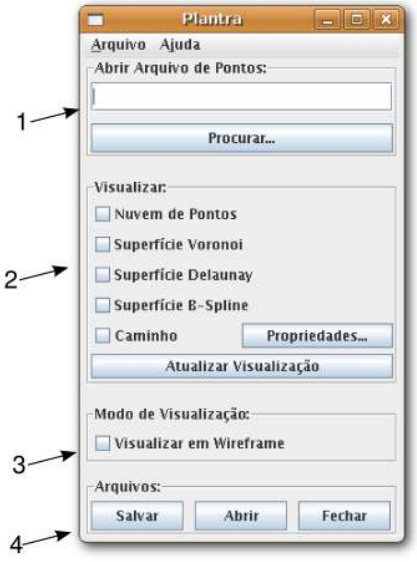


Figura 68: Janela de opções.

Na área indicada pelo número 2, estão disponíveis cinco itens que podem ser selecionados individual ou mutuamente. Esses itens são descritos a seguir:

- **Nuvem de pontos** – mostra o conjunto de pontos lido do arquivo selecionado na área 1;
- **Superfície de Voronoi** – reconstrói a superfície utilizando o diagrama de Voronoi, apresentado no capítulo 4, que aproxima a construção das células no espaço;
- **Superfície de Delaunay** – reconstrói a superfície a partir da triangularização de Delaunay, gerada do seu dual, diagrama de Voronoi;
- **Superfície B-Spline** – reconstrói a superfície utilizando superfícies B-Splines;
- **Caminho** – gera o caminho sobre a superfície utilizando as informações fornecidas pelo usuário e pela superfície reconstruída.

Quando selecionado quaisquer um desses itens e for apertado o botão **Atualizar Visualização**, a janela da Figura 69 mostrará as imagens 3D dos itens selecionados. Essas imagens também podem ser vistas em *wireframe* quando o campo **Visualizar em *Wireframe*** (da área 3) estiver marcado.

Figura 69: Janela de visualização 3D.

A Figura 70 exemplifica a visualização 3D de itens selecionados, (a) ilustra a superfície 3D, em *wireframe*, reconstruída através do diagrama de Voronoi, (b) a superfície

3D reconstruída através da triangularização de Delaunay e o seu conjunto de pontos representativos em vermelho e (c) a superfície 3D reconstruída por B-Spline e o caminho paramétrico sobre a superfície.

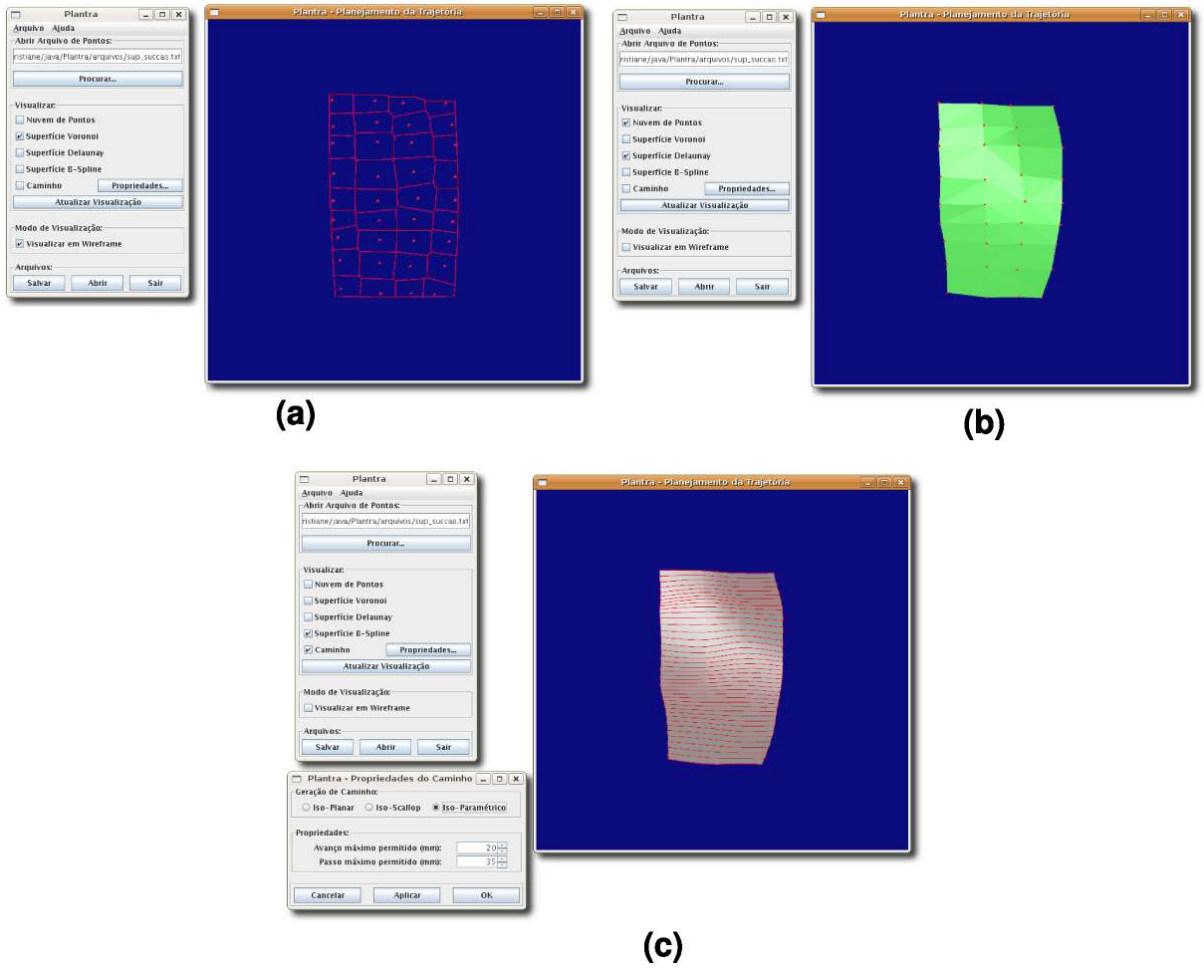


Figura 70: Exemplos de utilização do aplicativo.

Clicando-se no botão **Propriedades...** (do item **Caminho** na área 2), é aberto uma janela como a da Figura 71, onde é possível que se escolha se a geração do caminho da ferramenta é iso-planar, *iso-scallop* ou iso-paramétrico e que se determina os valores para o avanço e passo da ferramenta sobre a superfície. O valor do avanço e do passo para geração do caminho no espaço paramétrico são aproximados, o usuário determina esses valores e o algoritmo calcula os parâmetros (geralmente convencionados por Δu e Δv) de acordo com os dados da superfície e os valores escolhidos pelo usuário.

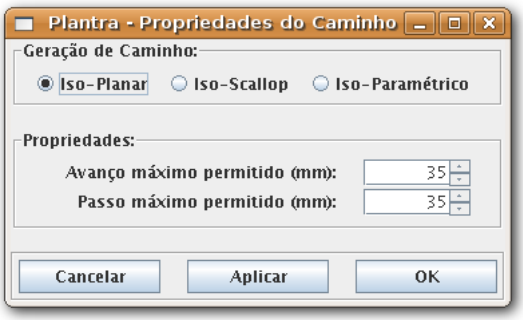


Figura 71: Janela para propriedades do caminho.

A área 4 (Figura 68) é composta de três botões: **Salvar**, **Abrir** e **Fechar**. O botão **Salvar** armazena em um arquivo as opções escolhidas pelo usuário e executadas pelo programa, o conjunto de pontos utilizado e a estrutura da superfície reconstruída e do caminho gerado. Essas informações são salvas em um arquivo compactado em formato **zip**, que contém todos os arquivos de geometria em formato OBJ e VRML, e as opções em formato texto.

O OBJ é um formato de arquivo aberto, que define a geometria e atualmente é adotado por vários desenvolvedores de aplicação 3D, podendo ser importado/exportado para Blender, Autodesk's Maya, e-Frontier's Poser, 3D Studio Max, Hexagon, Newtek Lightwave, Art of Illusion, etc.

O formato de arquivo OBJ é um formato de dados simples, que representa somente a geometria 3D, ou seja, a posição, a coordenada de textura e a normal, associadas a cada vértice e às faces que compõem cada polígono [54].

O formato *Virtual Reality Modeling Language* (VRML) é um formato padrão (ISO/IEC 14772), em arquivo texto, para representar vetores gráficos interativos 3D, nos quais os vértices e arestas de um polígono 3D podem ser especificados, assim como a cor da superfície, texturas de imagens mapeadas e transparência. Os arquivos VRML são comumente chamados de mundos VRML e têm a extensão .wrl. Muitos programas de modelagem 3D podem salvar objetos e cenas representados utilizando modelos VRML [55].

As formatos de arquivos OBJ e VRML foram adotados por serem simples de implementar, fornecerem resultados satisfatórios e serem compatíveis com outros programas.

O botão **Abrir** acessa os arquivos com as características que foram salvas pelo usuário anteriormente. E o botão **Fechar** encerra todo o aplicativo.

5.2.3 Aplicação do protótipo para conjuntos de pontos reais

O programa foi aplicado com vários conjuntos de pontos, com até 1500 pontos, e os resultados, que foram satisfatórios, serão comentados agora. Os pontos foram fornecidos pelo Projeto Roboturb [40], que é desenvolvido na Universidade Federal de Santa Catarina (UFSC). Um exemplo desses conjuntos de pontos e um cenário de aplicação do programa estão disponíveis no apêndice A.

A seguir, são ilustrados vários resultados obtidos a partir da execução do programa. Como dado de entrada foi fornecido o conjunto de pontos. Durante a execução, foram determinadas as mesmas opções fornecidas pela interface para todos os conjuntos de pontos, tendo sido escolhido visualizar o conjunto de pontos, a superfície B-Spline e o caminho. A unidade dos conjuntos de pontos fornecidas pelo projeto Roboturb estão em milímetros (mm).

A superfície B-Spline da Figura 72 (a) foi reconstruída a partir de um conjunto que contém 16 pontos gerados manualmente para a realização de alguns testes iniciais, (b) mostra o resultado obtido da geração do caminho da ferramenta iso-paramétrico e (c) iso-planar. Em propriedades do caminho, foi escolhido o valor 10mm para avanço e 10mm para passo da ferramenta, sendo que as dimensões da base da superfície é de 300mm por 300mm.

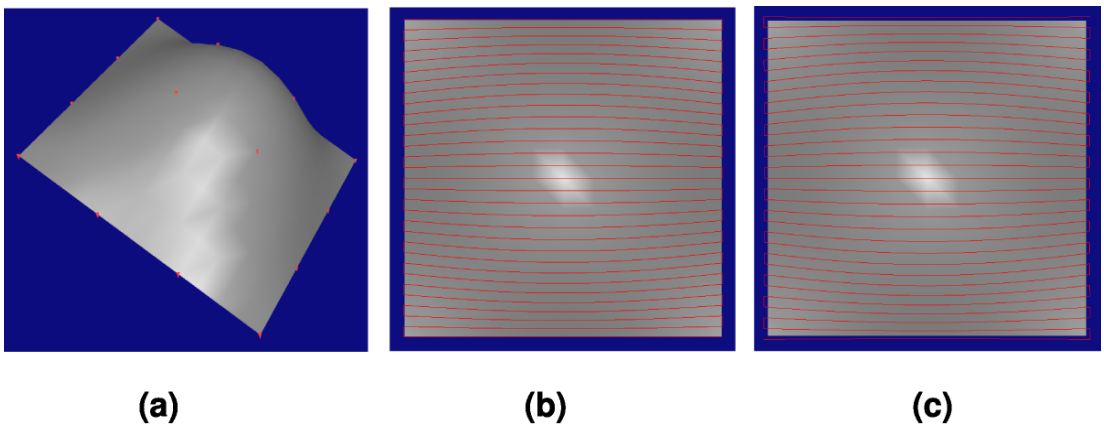


Figura 72: Resultados obtidos com conjunto de 16 pontos gerado manualmente, (a) visualização da superfície (b) caminho iso-paramétrico gerado e (c) caminho iso-planar gerado.

As Figuras 73, 74 e 75 são conjuntos de pontos obtidos da digitalização, utilizando um braço de medição, de um canal de turbina hidrelétrica de grande porte. Os três conjuntos

representam medições de regiões subjacentes que unidas formam uma parte da superfície da turbina.

A superfície da Figura 73 é denominada superfície de concordância entre a coroa e a superfície de sucção e contém 21 pontos. O caminho gerado sobre a superfície possui 8mm para o avanço e 20mm para o passo da ferramenta.

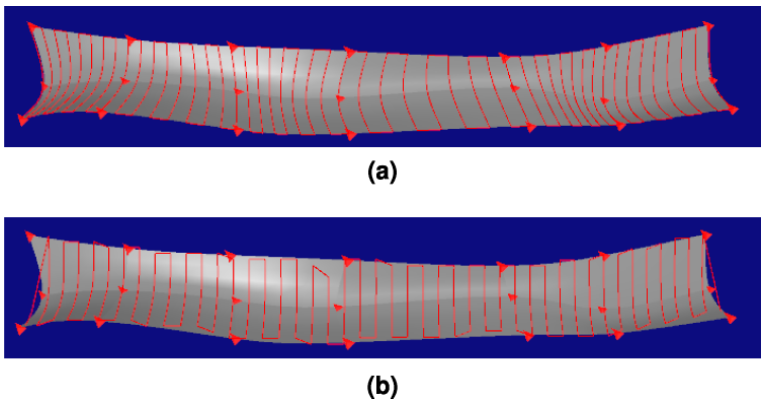


Figura 73: Resultados obtidos com conjunto de 21 pontos fornecido pelo projeto Roboturb - UFSC, (a) caminho iso-paramétrico e (b) caminho iso-planar.

A superfície da Figura 74 é denominada superfície de pressão e possui 63 pontos. O caminho gerado sobre essa superfície possui 15mm para o avanço e 30mm para o passo da ferramenta.

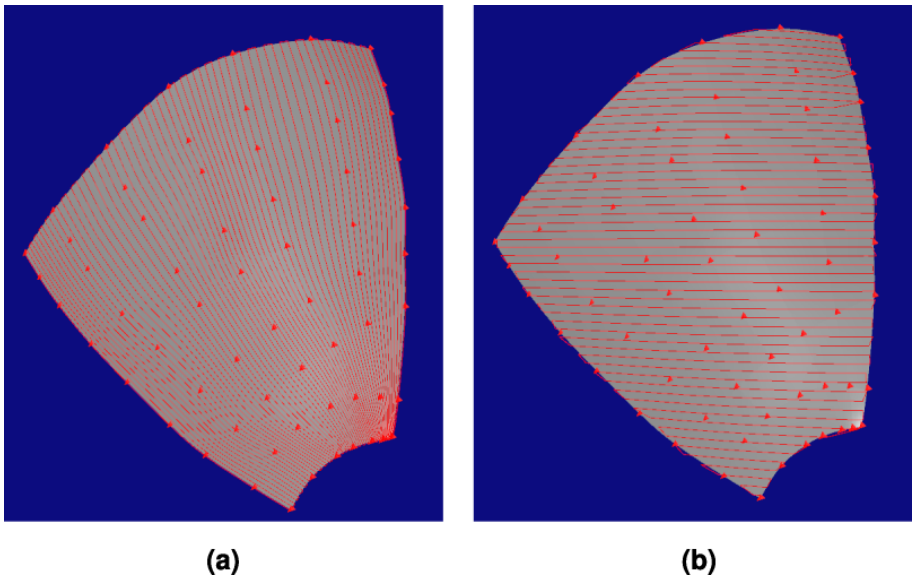


Figura 74: Resultados obtidos com conjunto de 63 pontos fornecido pelo projeto Roboturb - UFSC, (a) caminho iso-paramétrico e (b) caminho iso-planar.

A superfície da Figura 75 é denominada superfície de sucção, contendo 36 pontos. O caminho gerado sobre a superfície possui 8mm para o avanço e 36mm para o passo da ferramenta. Foi gerado o caminho *iso-scallop* visando ilustrar os três caminhos sobre essa superfície.

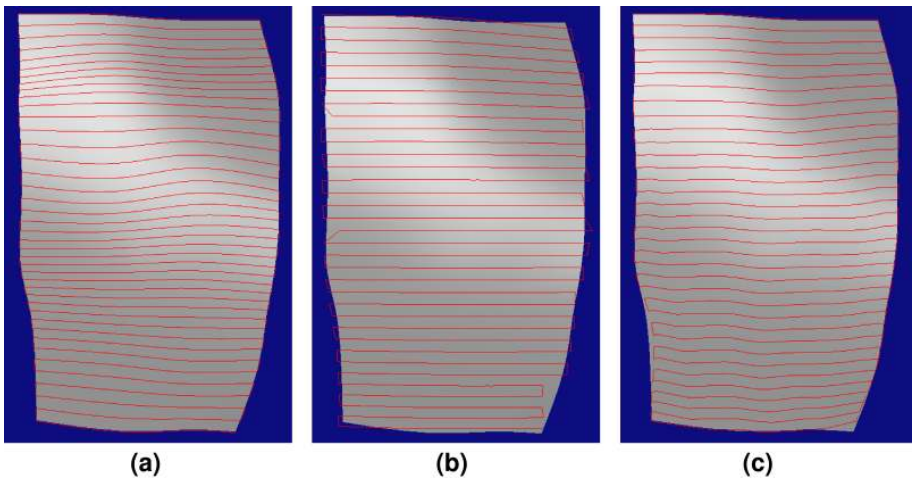


Figura 75: Resultados obtidos com conjunto de 36 pontos fornecido pelo projeto Roboturb - UFSC, (a) caminho iso-paramétrico, (b) caminho iso-planar (c) caminho *iso-scallop*.

Foram realizados testes com conjuntos de pontos gerados aleatoriamente em adição aos testes realizados com pontos obtidos a partir de medições. A Figura 76 ilustra a reconstrução, a partir de Delaunay, de uma superfície com 900 pontos gerados aleatoriamente.

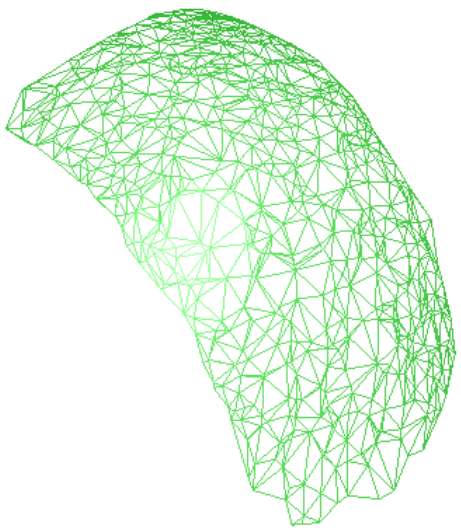


Figura 76: Representação em *wireframe* de uma superfície reconstruída a partir de pontos gerados aleatoriamente.

5.2.4 Compatibilidade com outros programas

O protótipo desenvolvido permite salvar os resultados obtidos em dois formatos compatíveis com outros programas (OBJ e VRML). Dois programas auxiliares foram utilizados para a realização de alguns testes preliminares: Blender [2] e o Pro/ENGINEER [35]. O Blender permite importar arquivos no formato OBJ, e o Pro/ENGINEER no formato VRML. Os resultados obtidos serão apresentados a seguir.

A Figura 77 (a) ilustra a geometria de um corpo de prova, a qual foi definida com base em um molde de argila conformado a partir de uma superfície danificada de uma das pás da turbina da hidrelétrica. As dimensões máximas da cavidade a ser preenchida por soldagem são de 160mm de comprimento, 120mm de largura e 8,50mm de profundidade, enquanto que as dimensões do corpo de prova são: 300mm de comprimento, 250mm de largura e 33mm de espessura máxima [5]. As superfícies da Figura 77 (b) e (c) foram obtidas a partir da execução do programa, tendo como entrada somente o arquivo de dados com 1504 pontos, sendo (b) a imagem renderizada pelo Blender, que utiliza o formato OBJ, e (c) a imagem da superfície salva no formato VRML e importada para o Pro/ENGINEER [35].

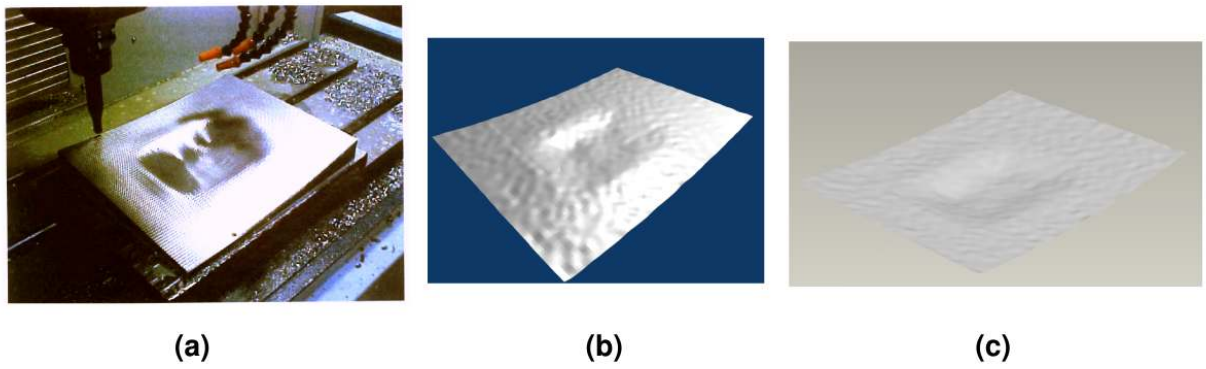


Figura 77: Reconstrução de superfícies – (a) Superfície original [5], (b) Superfície reconstruída por B-Spline, (c) Superfície importada para o Pro/ENGINEER.

A Figura 78 ilustra uma superfície reconstruída pelo aplicativo, a partir de um conjunto contendo 100 pontos, representando uma soma de senos, gerados, utilizando a equação:

$$z = \text{sen}\left(\frac{\pi}{4}x\right) + \text{sen}\left(\frac{\pi}{4}y\right) \tag{5.1}$$

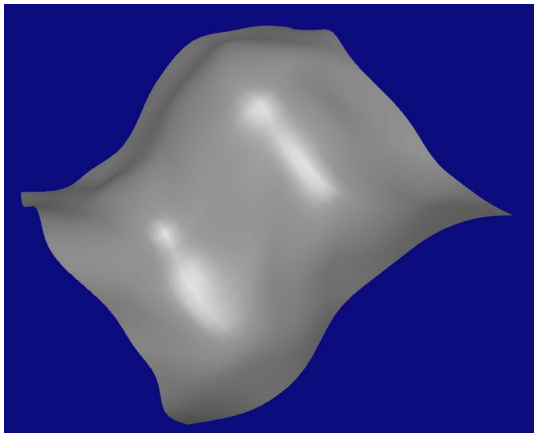


Figura 78: Superfície reconstruída pelo aplicativo.

A superfície foi salva no formato OBJ e VRML. A Figura 79 ilustra a superfície importada para o Blender, onde foi realizada a renderização, sendo utilizado o algoritmo de sombreamento de alta qualidade oferecida pelo software.

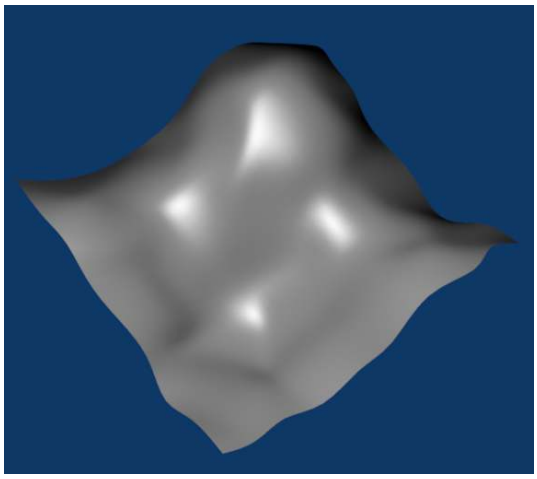


Figura 79: Superfície importada para o Blender.

A Figura 80 mostra a superfície importada para o Pro/ENGINEER, sendo efetuada sobre ela a análise de sua curvatura, exemplificando-se a possibilidade de utilização das funcionalidades oferecidas por esse programa numa superfície reconstruída pelo aplicativo desenvolvido.

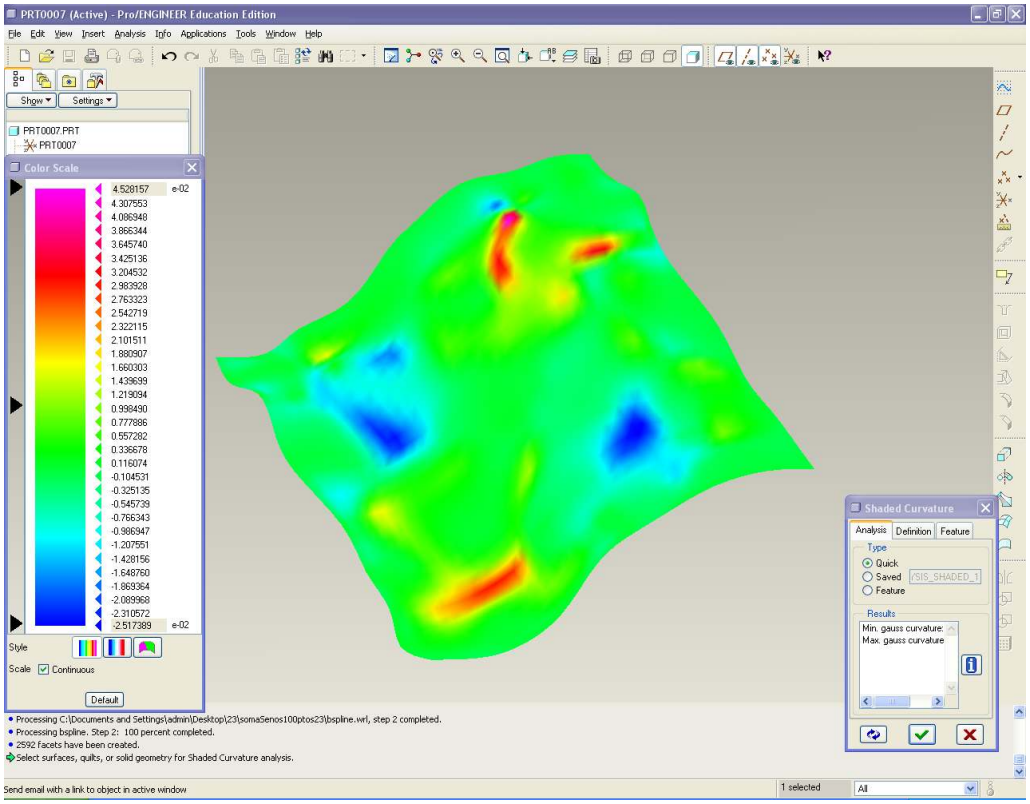


Figura 80: Superfície importada para o Pro/ENGINEER.

A aplicação do programa para esses conjuntos de pontos demonstrou resultados satisfatórios, e as superfícies reconstruídas apresentam visualizações muito próximas das superfícies reais, embora o erro de reconstrução não tenha sido avaliado. A Figura 81 ilustra o tempo de execução do algoritmo de reconstrução de superfícies para alguns conjuntos de pontos. Porém, testes e comparações mais rigorosos quanto ao desempenho do programa devem ser realizados futuramente.

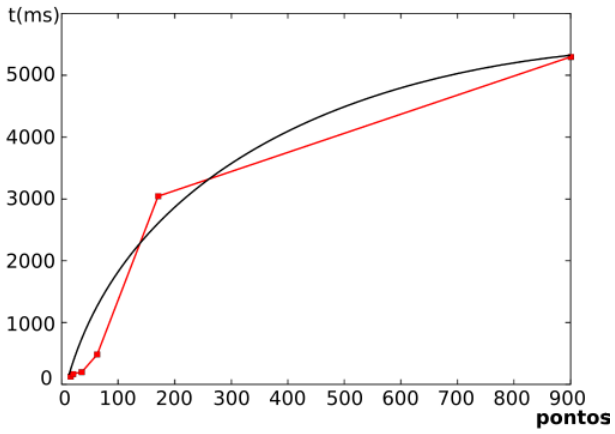


Figura 81: Gráfico de desempenho relacionando o número de pontos X tempo.

5.3 Detalhes de programação

O programa desenvolvido está dividido em 5 pacotes principais (conforme Figura 82), que dividem as classes implementadas de acordo com suas funcionalidades. O apêndice C descreve em mais detalhes as principais classes da arquitetura utilizando Linguagem de Modelagem Unificada (UML).

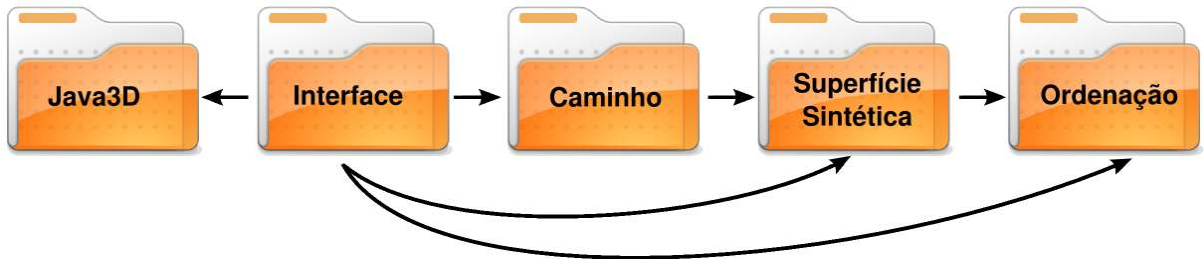


Figura 82: Divisão do algoritmo em pacotes.

A descrição de cada pacote é comentada a seguir:

- **Java3D** – esse pacote contém as classes que proporcionam a visualização 3D das superfícies. Nele, estão incluídas as classes que fornecerão a aparência, a intensidade de luz, a câmera e o universo de visualização do objeto;
- **Interface** – define as duas janelas de interação com o usuário: a janela principal e a janela de propriedades do caminho;
- **Caminho** – contém as classes que geram os caminhos no espaço de trabalho (iso-scallop, iso-planar) e no espaço paramétrico (iso-paramétrico);
- **Superfície sintética** – nesse pacote, está a implementação das curvas e superfícies B-Splines, o algoritmo inverso que calcula os vértices de controle da superfície, fornecendo um conjunto de pontos, e o método de Newton adaptado para o cálculo de intersecção entre o plano e a superfície. Há também duas classes que foram implementadas que contêm os algoritmos para a geração de curvas splines e curvas Bézier. Esses algoritmos não foram utilizados em nenhuma das etapas do protótipo, mas poderão ser utilizados futuramente para a geração de superfícies splines ou Bézier;
- **Ordenação** – esse é um dos pacotes com maior número de linhas de código. Nele, estão todas as classes para a geração do diagrama de Voronoi, a triangularização

de Delaunay, a técnica de recorte de Cyrus-Beck, as definições de ponto, aresta, segmento, célula e também as classes que geram as listas de vértices, células e arestas.

5.4 Considerações

Considera-se que a proposta de sistematização do processo de planejamento de trajetórias pôde ser concretizada através do aplicativo desenvolvido. É possível se utilizar a estrutura do modelo em aplicações reais como foi mostrado através da execução do aplicativo para alguns conjuntos de pontos. No decorrer da implementação do algoritmo, existiu um certo cuidado para que o algoritmo pudesse ser adaptado facilmente para casos específicos através de mudanças simples em futuras modificações do programa geral.

6 Conclusão

Este trabalho apresenta uma proposta de sistematização para o processo de planejamento de trajetórias para execução de tarefas de robôs manipuladores, além de um aplicativo que permite a reconstrução de superfícies e o planejamento do caminho da ferramenta a partir de um conjunto de pontos.

Apesar de existir uma grande quantidade de ferramentas que auxiliam e desenvolvem o planejamento de tarefas robóticas, poucas delas oferecem um estudo que contemplem todas as etapas do processo. Isso se deve ao fato de existirem muitos métodos desenvolvidos, sendo cada caso de aplicação basicamente específico, tornando difícil uma generalização. Para tentar minimizar esse problema, o estudo realizado apresenta uma proposta que contemple o processo todo de forma generalizada e sistemática.

O estudo de cada etapa e o desenvolvimento do programa possibilitaram o conhecimento de informações gerais sobre o processo de planejamento de trajetórias. Na proposta apresentada e no protótipo desenvolvido, é feito o uso de métodos de reconstrução de superfícies visando à reconstrução a partir de um conjunto de pontos adquiridos da própria superfície, possibilitando o planejamento e a geração de dados para o desenvolvimento de tarefas de robôs manipuladores no espaço de trabalho.

O programa desenvolvido tem como objetivo complementar a proposta, tentando integrá-la com a aplicação para casos reais. Em atividades industriais, a metodologia visa contribuir na redução de tempo durante as etapas de planejamento e desenvolvimento de tarefas robóticas e na simplificação de tarefas complexas. O aplicativo também possibilita a avaliação e testes, durante a etapa de planejamento e anteriores à execução da tarefa, que garantam que o processo seja completado satisfatoriamente.

Apesar deste estudo ter como objetivo a apresentação de uma proposta de sistematização, ele ainda possibilita a aplicação e o desenvolvimento de novos trabalhos para casos mais específicos, podendo ser uma referência geral para outros estudos e para possíveis complementações, pois variações do procedimento podem rapidamente ser aplicados em

estudos com caráter prático.

6.1 Limitações do protótipo

No decorrer da aplicação do programa, foram observadas possíveis limitações que ocorrem durante a sua execução, as quais são:

- o protótipo foi aplicado com conjuntos de pontos de até 1500 pontos, e é possível que o programa possa perder desempenho com conjuntos maiores;
- se os pontos estiverem expostos irregularmente no espaço[1], o resultado da superfície B-Spline pode apresentar algumas ondulações, como ilustra a Figura 83, o que ocorreu com testes realizados em alguns conjunto de pontos. Possivelmente, uma subdivisão do conjunto de pontos possa reduzir essas ondulações;

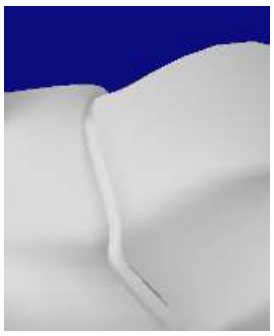


Figura 83: Distorções causadas por pontos expostos irregularmente.

- durante o desenvolvimento do algoritmo, foram utilizados os mesmos conjuntos de pontos, totalizando uma amostra de aproximadamente 10 conjuntos distintos (deles, quatro foram fornecidos pelo projeto Roboturb - UFSC). Acredita-se que essa amostra possa ser pequena, pois apesar desses conjuntos serem considerados representativos, é possível que eles não contemplem toda a gama de aplicações possível;
- apesar de suportar o planejamento no espaço de trabalho para regiões quaisquer, a interface desenvolvida só permite que o usuário faça o planejamento do caminho iso-planar sobre espaço de trabalho em forma de retângulos;
- não foram implementadas soluções para o cálculo de intersecção quando o método de Newton não converge. O método de Newton é utilizado no cálculo de intersecção

[1]Pontos distribuídos não homogeneamente, contendo aglomerações de pontos.

entre o plano e a superfície na geração do caminho iso-planar no espaço de trabalho e a Figura 84 ilustra um pequeno problema ocasionado pela não convergência do método de Newton;

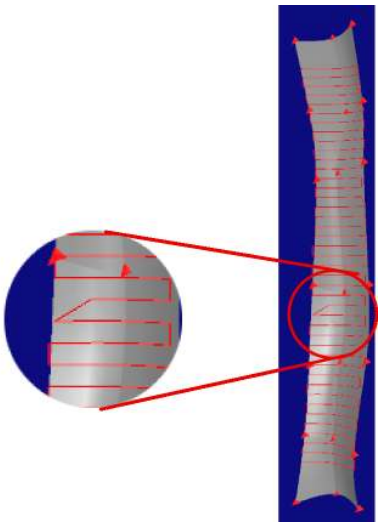


Figura 84: Problema detectado pela não convergência do método de Newton.

- as superfícies que não são consideradas funções não podem ser reconstruídas pelo programa, a não ser que sejam decompostas em duas ou mais superfícies em forma de função. Por exemplo, para reconstruir uma esfera é necessário dividir o conjunto de pontos em dois subconjuntos com relação aos que fazem parte do hemisfério norte e do hemisfério sul.

Essas limitações podem ser trabalhadas posteriormente, uma vez que ocorrem para casos particulares.

6.2 Trabalhos futuros

Algumas recomendações e propostas para trabalhos futuros na área de planejamento de tarefa para robôs industriais podem ser citadas. Entre elas:

1. otimização dos algoritmos desenvolvidos a fim de tornar o processamento mais rápido, eficiente, seguro, possibilitando a aplicação ser executada com conjuntos de pontos maiores;

2. desenvolvimento e implementação de técnicas que limitem regiões para trabalhar somente com a região de interesse, utilizando-se, por exemplo, algoritmos que calculem

intersecções entre superfícies;

3. propiciar a integração do sistema, desenvolvido com outros sistemas CAD/CAM, afim de proporcionar a comunicação entre eles e facilitar o cálculo de intersecção entre uma superfície de projeto e a superfície reconstruída, a qual foi obtida as informações através de algum método de aquisição;

4. desenvolvimento de algoritmos para o cálculo de posição, velocidade e aceleração no espaço das juntas do robô, introduzindo-se o cálculo de cinemática inversa para diferentes robôs, usando o caminho planejado;

5. utilizar um formato de transferência de dados padrão que inclua informações sobre outras etapas do processo produtivo como IGES ou STEP;

6. aplicar o processo em um caso real que compreenda todas as etapas, desde a aquisição de dados até a execução da tarefa, incluindo-se a utilização de equipamentos comercialmente disponíveis;

7. implementar métodos de geração de caminhos da ferramenta que incluam qualquer forma de região de interesse;

8. implementar novas tarefas, fazendo-se variar o caminho para diferentes tipos de ferramentas – desbaste, soldagem, usinagem, acabamento e pintura;

9. avaliar o erro gerado pela abordagem de reconstrução de superfícies a partir de um conjunto de pontos.

Referências

[1] Aurenhammer, F. Voronoi diagrams - a survey of a fundamental data structure. In **ACM Computing Surveys** (Austria, September 1991), vol. 23, pp. 345–405. Institute fur Informationsverarbeitung Technische Universitat Graz.

[2] Blender. Disponível em: http://www.blender.org/. Acesso em: 25/02/2007.

[3] Boesel, D., Dias, A. Reconstrução de superfícies no planejamento de trajetórias de tarefas de robôs através de diagrama de Voronoi. In **7 Congresso Iberoamericano de Ingenieria Mecanica** (México D.F., Outubro 2005).

[4] Boesel, D., Simas, H., Dias, A. An approach to explore path planning task by using Voronoi diagram. In **Flexible Automation and Intelligent Manufacturing (FAIM)** (Bilbao-Spain, 2005), pp. 288–295.

[5] Bonacorso, N. G. **Automatização dos processos de medição de superfícies e de deposição por soldagem visando a recuperação de rotores de turbinas hidráulicas de grande porte**. Florianópolis-SC, Agosto 2004. Universidade Federal de Santa Catarina.

[6] Bonilla, A. A. C. **Cinemática diferencial de manipuladores empregando cadeias virtuais**. Florianópolis-SC, Março 2004. Universidade Federal de Santa Catarina.

[7] Chen, Y. D., Ni, J., Wu, S. Real-time cnc tool path generation for maching iges surfaces. **Journal of Engineering for Industry 115** (november 1993), 480–486.

[8] Ciocca, L., Scotti, R. CAD-CAM generated ear cast by means of a laser scanner and rapid prototyping machine. **J Prosthet Dent**, 92 (2004), 591 – 595.

[9] Craig, J. J. **Introduction to Robotics Mechanics and Control**, second ed. Addison Wesley Publishing Company, Inc, New York, 1989.

[10] CyberVRML97. Disponível em:
http://www.cybergarage.org/vrml/cv97/cv97java/index.html.
Acesso em: 28/02/2007.

[11] de Toledo, L. B. **Uma interface CAD/CAM para a programação fora de linha de robôs industriais**. Florianópolis-SC, Fevereiro 2000. Universidade Federal de Santa Catarina.

[12] Deitel, H. M., Deitel, P. J. **JAVA Como Programar**, quarta ed. Bookman, Porto Alegre-RS, 2003.

[13] do Amaral, S. **Controle a estrutura variável de robôs manipuladores interagindo com ambientes passivos**. Florianópolis-SC, 2000. Universidade Federal de Santa Catarina.

[14] Dourado, A. O. **Cinemática de robôs cooperativos**. Florianópolis, Abril 2005. Universidade Federal de Santa Catarina.

[15] Farin, G. **Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide**, third ed. Academic Press, Inc, United States of America, 1993.

[16] Feng, H.-Y., Li, H. Constant scallop-height tool path generation for three-axis sculptured surface machining. **Computer-Aided Design**, 34 (2001), 647 – 654.

[17] Feng, H.-Y., Teng, Z. Iso-planar piecewise linear nc tool path generation from discrete measured data points. In **Computer-Aided Design**, vol. 37. Elsevier Ltd, 2005, pp. 55–64.

[18] Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F. **Computer Graphics: Priciples and Practice**, second ed. The systen programing series. Addison-Wesley Publishing Company, New York, 1990.

[19] Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F., Phillips, R. L. **Introduction to computer graphics**. Addison-Wesley Publishing Company, New York, 1993.

[20] Galvão, L. C. **Dimensionamento de sistemas de distribuição através do diagrama multiplicativo de Voronoi com pesos**. Florianópolis-SC, setembro 2003.

[21] Gomes, J., Velho, L. **Fundamentos da computação gráfica**, 1 ed. Série de Computação e Matemática. Instituto Nacional de Matemática Pura e Aplicada - IMPA, Rio de Janeiro-RJ, 2003.

[22] GPL. Disponível em: http://www.gnu.org/licenses/. Acesso em: 25/01/2007.

[23] Graham, R., Yao, F. A whirlwind tour of computacional geometry. **The American Mathematical Monthly 97**, 8 (October 1990), 687–701.

[24] Groover, M. P., Weis, M., Nagel, R. N., Odrey, N. G. **Robótica: Tecnologia e programação**. São Paulo, 1988.

[25] IFToMM. Iftomm terminology/english 1.1. In **Mechanism and Machine Theory**, vol. 38. 2003, pp. 767–776.

[26] Jama-Matrix. Disponível em: http://math.nist.gov/javanumerics/jama/. Acesso em: 24/01/2007.

[27] Kise, K. be - a block extraction program based on the area voronoi diagram. Tech. rep., Osaka Prefecture University, October 1999. Dept. of Computer & Systems Sciences.

[28] Lin, Y.-P., Wang, C.-T., Dai, K.-R. Reverse engineering in CAD model reconstruction of customized artificial joint. **Medical Engineering and Physics**, 27 (2005), 189 – 193.

[29] Little, C. Q., Wilson, C. W. Rapid world modelling for robotics. In **Robotic and Manufacturing Systems** (Montpellier, France, may 1996), M. Jamshidi F. Pin, Eds., vol. 3, TSI Press Series, pp. 441–446. Proceedings of the World Automation Congress (WAC '96).

[30] Maliska Junior, C. R. **Geração de malhas para domínios 2,5 dimensionais usando triangularização de Delaunay Restrita**. Florianópolis, Fevereiro 2001. Universidade Federal de Santa Catarina.

[31] Mencl, R., Müller, H. Interpolation and approximation of surfaces from three-dimensional scattered data points. In **Informatik VII (Computer Graphics)** (Germany, 1998), University of Dortmund.

[32] Montenegro, A. A., Peixoto, A., Sá, A., Soares, E. **Fotografia 3D**. Impa, Rio de Janeiro-RJ, 2005. Colóquio Brasileiro de Matemática.

[33] Pot, J., Thibault, G., Levesque, P. Techniques for CAD reconstruction of 'as-built' environments and application to preparing for dismantling of plants. **Nuclear Engineering and Design**, 178 (1997), 135 – 143.

[34] Pottmann, H., Leopoldseder, S., Hofer, M., Steiner, T., Wang, W. Industrial geometry: recent advances and applications in cad. **Computer-Aided Design**, 37 (2005), 751 – 766.

[35] PTC. Disponível em: http://www.ptc.com. Acesso em: 24/02/2007.

[36] Qiulin, D., Davies, B. J. **Surface engineering geometry for computer-aided desing and manufacture**. Ellis Horwood limited, Chichester - United Kingdom, 1987.

[37] Ramella, M., Boschin, W., Fadda, D., Nonino1, M. Finding galaxy clusters using voronoi tessellations. **Astronomy e astrophysics**, 386 (December 2000), 776–786.

[38] Ránky, P. G., Ho, C. Y. **Robot Modelling: Control and Applications with Software**. IFS (Publications) Ltd. UK, Springer-Verlag Berlin, 1985.

[39] Reitemeier, B., Notni, G., Heinze, M., Schone, C., Schmidt, A., Fichtner, D. Optical modeling of extraoral defects. **J Prosthet Dent**, 91 (2004), 80 – 84.

[40] RoboTurb. Disponível em: http://www.roboturb.ufsc.br. Acesso em: 23/10/2005.

[41] Rogers, D. F., Adams, J. A. **Mathematical Elements for Computer Graphics**, second ed. McGraw-Hill Publishing Company, Singapore, 1990.

[42] Sansoni, G., Docchio, F. Three-dimensional optical measurements and reverse engineering for automotive applications. **Robotics and Computer-Integrated Manufacturing**, 20 (2004), 359 – 367.

[43] Sarma, R., Dutta, D. The geometry and generation of nc tool paths. **Journal of Mechanical Design 119** (June 1997), 253 – 258.

[44] Schoenharl, T., Bravo, R., Madey, G. Wiper: Leveraging the cell phone network for emergency response. Tech. rep., University of Notre Dame, September 2006. Dept of Computer Science and Engineering.

[45] Sciavicco, L., Siciliano, B. **Modelling and Control of Robot Manipulators**, 2 ed. Springer, New York, 2004.

[46] Senechal, M. Spatial tessellations: Concepts and applications of voronoi diagrams. **The College Mathematics Journal 26**, 1 (January 1995), 79–81.

[47] Silva, O. J. **Programando em Java 2: Interfaces Gráficas e Aplicações Práticas com AWT e Swing**, 1 ed. Editora Érica Ltda, São Paulo - SP, 2004.

[48] Simas, H. **Planejamento de Trajetória de Soldagem para Robôs Redundantes Operando em Ambientes Confinados**. Florianópolis, março 2005. Universidade Federal de Santa Catarina.

[49] Sun-JAVA. Disponível em: http://java.sun.com/. Acesso em: 27/06/2006.

[50] Suresh, K., Yang, D. C. H. Constant scallop-height machining of free-form surfaces. **Journal of Engineering for Industry 116** (May 1994), 253 – 259.

[51] Valdiero, A. C. **Controle de robôs hidráulicos com compensação de atrito**. Florianópolis-SC, Fevereiro 2005. Universidade Federal de Santa Catarina.

[52] Velho, L., Gomes, J. **Sistemas Graficos 3D**, 1 ed. Série de Computação e Matemática. Instituto Nacional de Matemática Pura e Aplicada - IMPA, Rio de Janeiro-RJ, 2001.

[53] Wikipedia. Disponível em: http://en.wikipedia.org/wiki/dirichlet_tessellation. Acesso em: 27/03/2007.

[54] Wikipedia. Disponível em: http://en.wikipedia.org/wiki/obj. Acesso em: 20/11/2006.

[55] Wikipedia. Disponível em: http://en.wikipedia.org/wiki/vrml. Acesso em: 20/11/2006.

[56] Yin, Z. Rough and finish tool-path generation for nc machining of freeform surfaces based on a multiresolution method. **Computer-Aided Design - Elsevier**, 36 (January 2004), 1231 – 1239.

[57] Zeid, I. **CAD/CAM Theory and Practice**. Series in Mechanical Engineering. McGraw-Hill, Inc., New York, 1991.

[58] Zeiss. Disponível em: www.zeiss.com. Acesso em: 14/01/2007.

[59] Zha, X. F. Optimal pose trajectory planning for robot manipulators. **Mechanism and machine Theory**, 37 (may 2002), 1063–1086.

[60] Zhongwei, Y., Shouwei, J. Iso-phote based adaptive surface fitting to digitized points and its applications in region-based tool path generation, slicing and surface triangulation. **Computers in Industry**, 55 (January 2004), 15 – 28.

APÊNDICE A – Um cenário para o aplicativo

A.1 Conjuntos de pontos

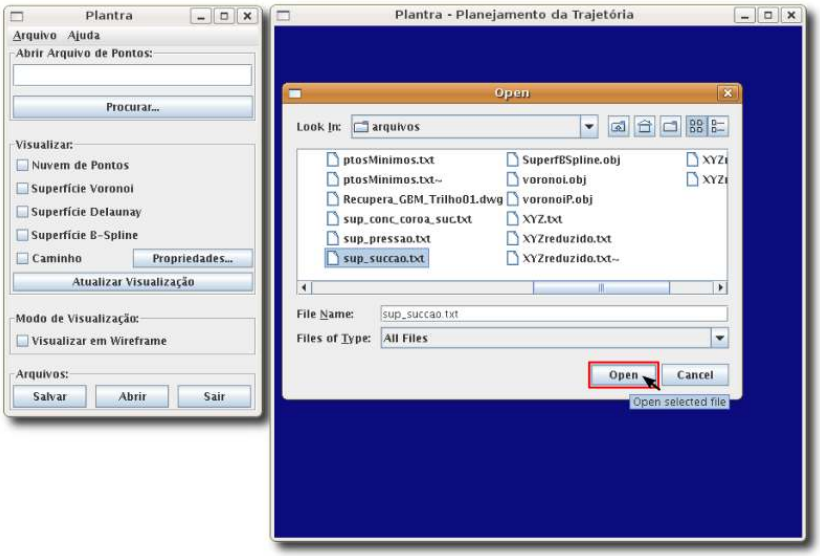
Foi selecionado o seguinte conjunto de 36 pontos, fornecido pelo projeto ROBOTURB - UFSC, para o desenvolvimento de um cenário típico para a utilização do protótipo.

```
-0116.375 -1064.384 -0443.579
+0152.972 -1093.705 -0425.393
+0360.112 -1090.259 -0424.186
+0516.616 -1099.828 -0440.424
-0110.077 -0857.685 -0345.318
+0150.977 -0873.364 -0320.482
+0348.830 -0893.997 -0324.146
+0572.466 -0892.318 -0325.584
-0118.334 -0706.360 -0289.376
+0151.605 -0718.939 -0259.079
+0369.173 -0723.833 -0257.812
+0593.372 -0729.568 -0267.866
-0139.385 -0590.620 -0256.284
+0152.272 -0591.354 -0222.111
+0369.128 -0585.934 -0221.050
+0612.243 -0602.838 -0239.145
-0138.088 -0470.229 -0221.545
+0141.379 -0472.587 -0198.802
+0390.730 -0455.103 -0201.503
+0625.122 -0466.508 -0225.832
```

```
-0129.532 -0302.335 -0187.892
+0144.058 -0320.692 -0179.566
+0364.953 -0291.333 -0191.541
+0626.525 -0319.717 -0232.339
-0135.250 -0116.468 -0170.916
+0140.858 -0107.694 -0170.421
+0351.791 -0110.114 -0201.428
+0634.181 -0114.036 -0278.411
-0130.675 -0015.008 -0166.110
+0136.900 +0011.907 -0176.120
+0336.883 +0004.032 -0215.820
+0618.178 +0022.449 -0322.899
-0137.612 +0147.215 -0169.351
+0124.796 +0149.036 -0192.553
+0301.748 +0145.257 -0236.967
+0584.703 +0171.862 -0369.513
```

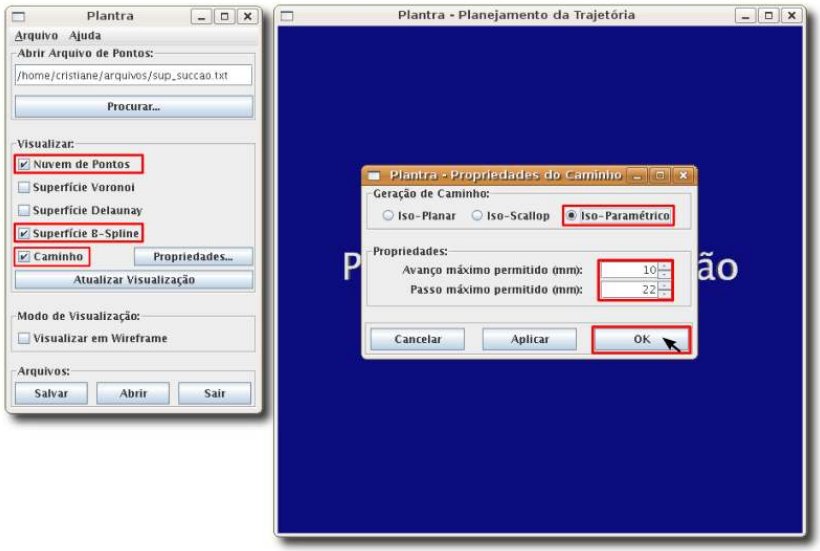
A.2 Cenário típico de aplicação do programa

A seguir, é ilustrado um cenário típico de utilização do programa desenvolvido. Esse cenário é composto de 6 passos em que o usuário abre um conjunto de pontos, escolhe as opções fornecidas na área 2 – Visualizar (Figura 68), atualiza a janela de visualização e salva os dados obtidos pelo programa.



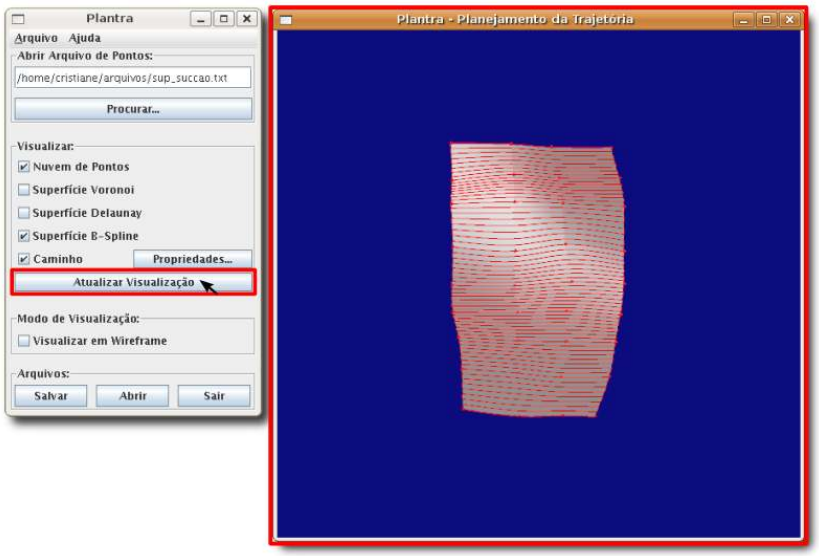
1. O usuário seleciona o arquivo de dados.

Figura 85: Escolhendo arquivo de pontos.



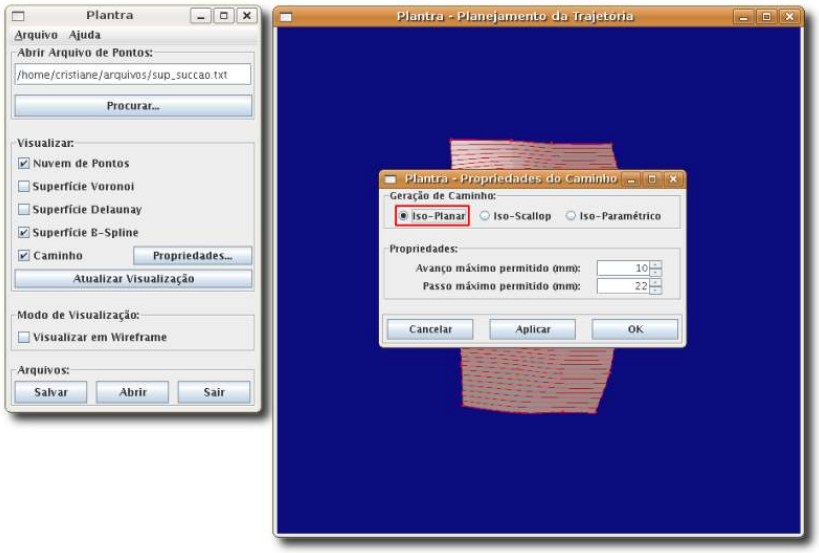
2. O usuário seleciona a opção visualizar nuvem de pontos, superfície B-Spline e caminho. Em propriedades, do caminho escolhe a opção iso-paramétrico com valores de 10mm para avanço e 22mm para passo da ferramenta.

Figura 86: Selecionando características.



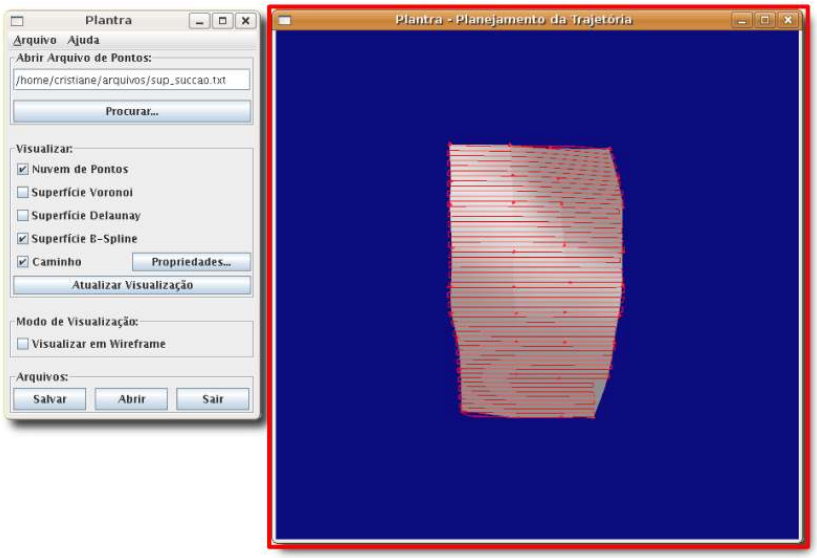
3. O usuário atualiza a janela de visualização 3D apertando o botão **Atualizar Visualização** e os botões **Aplicar** ou **OK** da janela de propriedades do caminho.

Figura 87: Atualização da janela de visualização 3D.



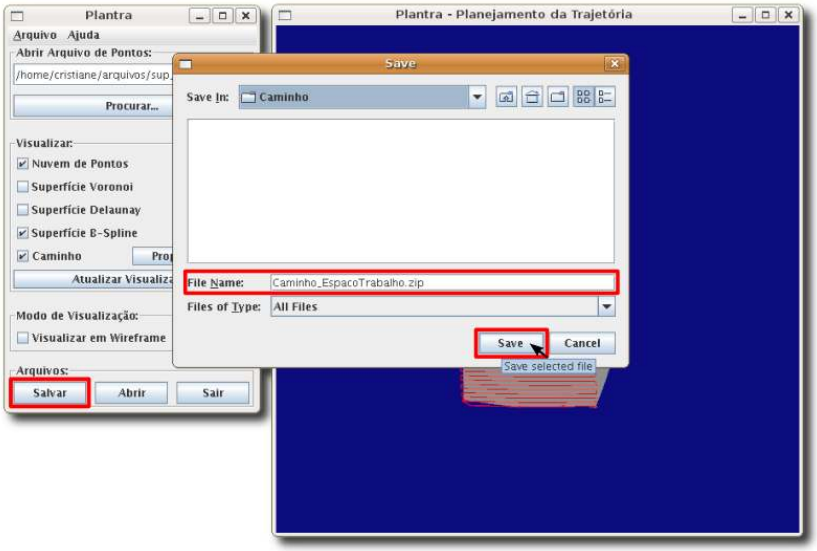
4. O usuário muda as opções anteriores selecionando geração do caminho no iso-planar.

Figura 88: Selecionando geração do caminho no espaço de trabalho.



5. Atualização da janela de visualização 3D ocasionada pela mudança feita anteriormente.

Figura 89: Atualização da janela de visualização 3D.



6. O usuário salva os resultados obtidos num arquivo e pode abrir posteriormente para possíveis modificações ou manipulações.

Figura 90: Salvando os resultados.

Esse cenário ilustra uma possível utilização do programa para um caso típico de reconstrução de superfície. Esse cenário compreende as ações de abrir um conjunto de pontos, escolher opções de visualização e salvar as informações obtidas.

APÊNDICE B – The GNU General Public License

Version 2, June 1991

51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA

Abstract

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU General Public License

Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system,

which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

End of Terms and Conditions

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>

> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) <year> <name of author>
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items—whatever suits your program.

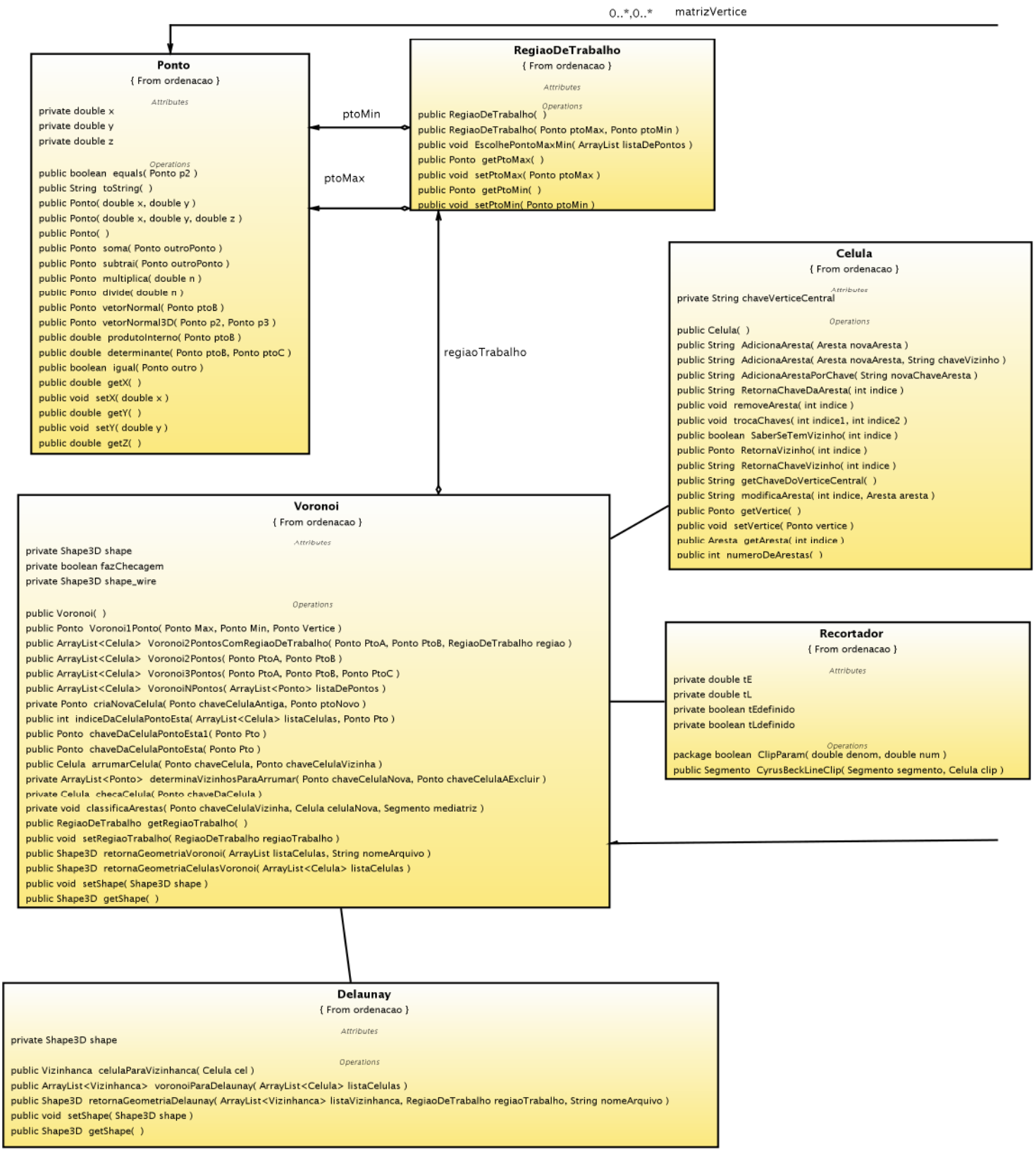
You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

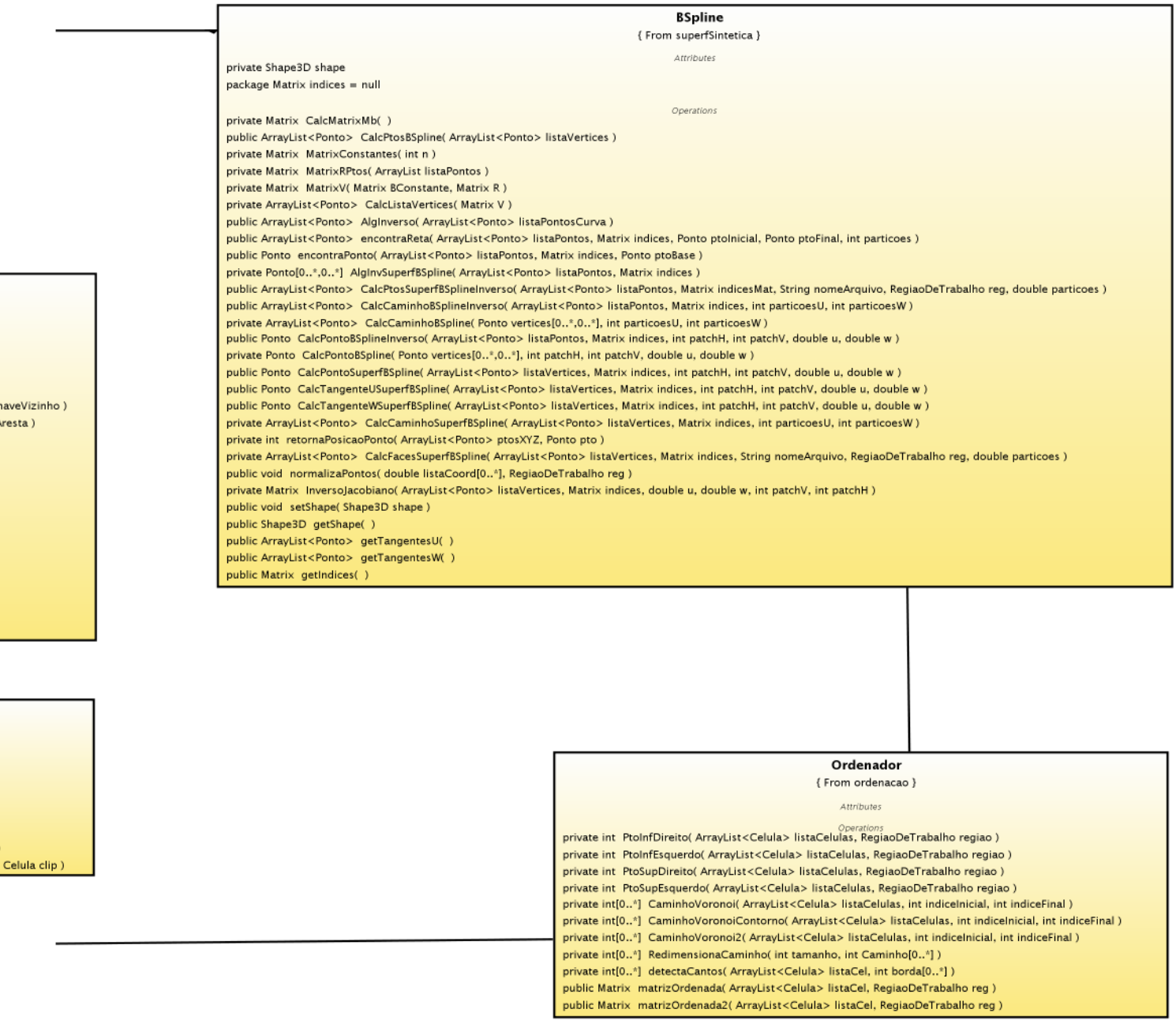
> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> <signature of Ty Coon>, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

APÊNDICE C – Diagrama de classes

A página seguinte contém o diagrama de classes simplificado das principais classes implementadas.


0..*,0..*
matrizVertice
Ponto
{ From ordenacao }
Attributes
private double x
private double y
private double z
Operations
public boolean equals(Ponto p2)
public String toString( )
public Ponto(double x, double y)
public Ponto(double x, double y, double z)
public Ponto( )
public Ponto soma(Ponto outroPonto)
public Ponto subtrai(Ponto outroPonto)
public Ponto multiplica(double n)
public Ponto divide(double n)
public Ponto vetorNormal(Ponto ptoB)
public Ponto vetorNormal3D(Ponto p2, Ponto p3)
public double produtoInterno(Ponto ptoB)
public double determinante(Ponto ptoB, Ponto ptoC)
public boolean igual(Ponto outro)
public double getX( )
public void setX(double x)
public double getY( )
public void setY(double y)
public double getZ( )
ptoMin
ptoMax
RegiaoDeTrabalho
{ From ordenacao }
Attributes
Operations
public RegiaoDeTrabalho( )
public RegiaoDeTrabalho(Ponto ptoMax, Ponto ptoMin)
public void EscolhePontoMaxMin(ArrayList listaDePontos)
public Ponto getPtoMax( )
public void setPtoMax(Ponto ptoMax)
public Ponto getPtoMin( )
public void setPtoMin(Ponto ptoMin)
regiaoTrabalho
Celula
{ From ordenacao }
Attributes
private String chaveVerticeCentral
Operations
public Celula( )
public String AdicionaAresta(Aresta novaAresta)
public String AdicionaAresta(Aresta novaAresta, String chaveVizinho)
public String AdicionaArestaPorChave(String novaChaveAresta)
public String RetornaChaveDaAresta(int indice)
public void removeAresta(int indice)
public void trocaChaves(int indice1, int indice2)
public boolean SaberSeTemVizinho(int indice)
public Ponto RetornaVizinho(int indice)
public String RetornaChaveVizinho(int indice)
public String getChaveDoVerticeCentral( )
public String modificaAresta(int indice, Aresta aresta)
public Ponto getVertice( )
public void setVertice(Ponto vertice)
public Aresta getAresta(int indice)
public int numeroDeArestas( )
Voronoi
{ From ordenacao }
Attributes
private Shape3D shape
private boolean fazChecagem
private Shape3D shape_wire
Operations
public Voronoi( )
public Ponto Voronoi1Ponto(Ponto Max, Ponto Min, Ponto Vertice)
public ArrayList<Celula> Voronoi2PontosComRegiaoDeTrabalho(Ponto PtoA, Ponto PtoB, RegiaoDeTrabalho regiao)
public ArrayList<Celula> Voronoi2Pontos(Ponto PtoA, Ponto PtoB)
public ArrayList<Celula> Voronoi3Pontos(Ponto PtoA, Ponto PtoB, Ponto PtoC)
public ArrayList<Celula> VoronoiNPontos(ArrayList<Ponto> listaDePontos)
private Ponto criaNovaCelula(Ponto chaveCelulaAntiga, Ponto ptoNovo)
public int indiceDaCelulaPontoEsta(ArrayList<Celula> listaCelulas, Ponto Pto)
public Ponto chaveDaCelulaPontoEsta1(Ponto Pto)
public Ponto chaveDaCelulaPontoEsta(Ponto Pto)
public Celula arrumarCelula(Ponto chaveCelula, Ponto chaveCelulaVizinha)
private ArrayList<Ponto> determinaVizinhosParaArrumar(Ponto chaveCelulaNova, Ponto chaveCelulaAExcluir)
private Celula checaCelula(Ponto chaveDaCelula)
private void classificaArestas(Ponto chaveCelulaVizinha, Celula celulaNova, Segmento mediatriz)
public RegiaoDeTrabalho getRegiaoTrabalho( )
public void setRegiaoTrabalho(RegiaoDeTrabalho regiaoTrabalho)
public Shape3D retornaGeometriaVoronoi(ArrayList listaCelulas, String nomeArquivo)
public Shape3D retornaGeometriaCelulasVoronoi(ArrayList<Celula> listaCelulas)
public void setShape(Shape3D shape)
public Shape3D getShape( )
Recortador
{ From ordenacao }
Attributes
private double tE
private double tL
private boolean tEdefinido
private boolean tLdefinido
Operations
package boolean ClipParam(double denom, double num)
public Segmento CyrusBeckLineClip(Segmento segmento, Celula clip)
Delaunay
{ From ordenacao }
Attributes
private Shape3D shape
Operations
public Vizinhanca celulaParaVizinhanca(Celula cel)
public ArrayList<Vizinhanca> voronoiParaDelaunay(ArrayList<Celula> listaCelulas)
public Shape3D retornaGeometriaDelaunay(ArrayList<Vizinhanca> listaVizinhanca, RegiaoDeTrabalho regiaoTrabalho, String nomeArquivo)
public void setShape(Shape3D shape)
public Shape3D getShape( )


BSpline
{ From superfSintetica }
Attributes
private Shape3D shape
package Matrix indices = null
Operations
private Matrix CalcMatrixMb( )
public ArrayList<Ponto> CalcPtosBSpline(ArrayList<Ponto> listaVertices)
private Matrix MatrixConstantes(int n)
private Matrix MatrixRPtos(ArrayList listaPontos)
private Matrix MatrixV(Matrix BConstante, Matrix R)
private ArrayList<Ponto> CalcListaVertices(Matrix V)
public ArrayList<Ponto> AlgInverso(ArrayList<Ponto> listaPontosCurva)
public ArrayList<Ponto> encontraReta(ArrayList<Ponto> listaPontos, Matrix indices, Ponto ptoInicial, Ponto ptoFinal, int particoes)
public Ponto encontraPonto(ArrayList<Ponto> listaPontos, Matrix indices, Ponto ptoBase)
private Ponto[0..*,0..*] AlgInvSuperfBSpline(ArrayList<Ponto> listaPontos, Matrix indices)
public ArrayList<Ponto> CalcPtosSuperfBSplineInverso(ArrayList<Ponto> listaPontos, Matrix indicesMat, String nomeArquivo, RegiaoDeTrabalho reg, double particoes)
public ArrayList<Ponto> CalcCaminhoBSplineInverso(ArrayList<Ponto> listaPontos, Matrix indices, int particoesU, int particoesW)
private ArrayList<Ponto> CalcCaminhoBSpline(Ponto vertices[0..*,0..*], int particoesU, int particoesW)
public Ponto CalcPontoBSplineInverso(ArrayList<Ponto> listaPontos, Matrix indices, int patchH, int patchV, double u, double w)
private Ponto CalcPontoBSpline(Ponto vertices[0..*,0..*], int patchH, int patchV, double u, double w)
public Ponto CalcPontoSuperfBSpline(ArrayList<Ponto> listaVertices, Matrix indices, int patchH, int patchV, double u, double w)
public Ponto CalcTangenteUSuperfBSpline(ArrayList<Ponto> listaVertices, Matrix indices, int patchH, int patchV, double u, double w)
public Ponto CalcTangenteWSuperfBSpline(ArrayList<Ponto> listaVertices, Matrix indices, int patchH, int patchV, double u, double w)
private ArrayList<Ponto> CalcCaminhoSuperfBSpline(ArrayList<Ponto> listaVertices, Matrix indices, int particoesU, int particoesW)
private int retornaPosicaoPonto(ArrayList<Ponto> ptosXYZ, Ponto pto)
private ArrayList<Ponto> CalcFacesSuperfBSpline(ArrayList<Ponto> listaVertices, Matrix indices, String nomeArquivo, RegiaoDeTrabalho reg, double particoes)
public void normalizaPontos(double listaCoord[0..*], RegiaoDeTrabalho reg)
private Matrix InversoJacobiano(ArrayList<Ponto> listaVertices, Matrix indices, double u, double w, int patchV, int patchH)
public void setShape(Shape3D shape)
public Shape3D getShape( )
public ArrayList<Ponto> getTangentesU( )
public ArrayList<Ponto> getTangentesW( )
public Matrix getIndices( )
haveVizinho)
aresta)
Celula clip)
Ordenador
{ From ordenacao }
Attributes
Operations
private int PtoInfDireito(ArrayList<Celula> listaCelulas, RegiaoDeTrabalho regiao)
private int PtoInfEsquerdo(ArrayList<Celula> listaCelulas, RegiaoDeTrabalho regiao)
private int PtoSupDireito(ArrayList<Celula> listaCelulas, RegiaoDeTrabalho regiao)
private int PtoSupEsquerdo(ArrayList<Celula> listaCelulas, RegiaoDeTrabalho regiao)
private int[0..*] CaminhoVoronoi(ArrayList<Celula> listaCelulas, int indiceInicial, int indiceFinal)
private int[0..*] CaminhoVoronoiContorno(ArrayList<Celula> listaCelulas, int indiceInicial, int indiceFinal)
private int[0..*] CaminhoVoronoi2(ArrayList<Celula> listaCelulas, int indiceInicial, int indiceFinal)
private int[0..*] RedimensionaCaminho(int tamanho, int Caminho[0..*])
private int[0..*] detectaCantos(ArrayList<Celula> listaCel, int borda[0..*])
public Matrix matrizOrdenada(ArrayList<Celula> listaCel, RegiaoDeTrabalho reg)
public Matrix matrizOrdenada2(ArrayList<Celula> listaCel, RegiaoDeTrabalho reg)