João Nuno Delgado de Aguilar Botelho Rocha

Desenvolvimento de Algoritmos de Otimização para Métodos Sem Malha

João Nuno Delgado de Aguilar Botelho Rocha

Desenvolvimento de Algoritmos de Otimização para Métodos Sem Malha

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de Rui Pedro Ramos Cardoso, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro.

O júri / The jury

Presidente / President

Prof. Doutor Francisco José Malheiro Queirós de Melo
Professor Associado da Universidade de Aveiro

Vogais / Committee

Prof. Doutor Rui Pedro Ramos Cardoso
Professor Auxiliar da Universidade de Aveiro (orientador)

Prof. Doutor José Manuel de Almeida César de Sá
Professor Catedrático do Dep. Eng. Mecânica da Faculdade de Engenharia da Universidade do Porto

Agradecimentos / Acknowledgements

Não obstante do caracter científico que fez desta Tese um trabalho individual, só alguns contributos de natureza diversa foram capazes de dar a confiança e o apoio necessário à realização deste trabalho, os quais não posso deixar de agradecer.

Ao Professor Rui Pedro Ramos Cardoso, meu orientador, pela competência científica, pela disponibilidade e generosidade, assim como pelas críticas, correções e sugestões relevantes feitas durante a orientação.

À Ana Graça pelo incansável apoio, orientação e permanente disponibilidade. Agradeço o apoio, a partilha de saber, as valiosas contribuições para o trabalho e o tempo dispensado na orientação e leitura minuciosa da tese.

Sou muito grato a todos os meus familiares pelo incentivo recebido ao longo destes anos. Às minhas irmãs pela disponibilidade manifestada e pelo prestimoso apoio. Obrigado pelo amor, alegria, atenção sem reservas e pela paciência e compreensão reveladas ao longo destes anos.

A todos os amigos pela excelente amizade, acolhimento e apoio manifestados ao longo dos anos.

À Rádio Radar pela companhia.

Os meus mais sinceros agradecimentos.

Palavras-chave Métodos sem malha; EFG; NEM; kd-Tree; Diagramas de Voronoi

Resumo

Recentemente surgiu um novo tipo de métodos computacionais referenciados como métodos sem malha. Estes métodos têm sido amplamente usados no estudo e na resolução de problemas de engenharia, mais usualmente em problemas que contam com grandes deslocamentos, como a simulação de processos de conformação plástica e em problemas com elevadas taxas de deformação, com impacto e fratura estrutural. Devido ao caráter não interpolatório das funções de aproximação dos métodos sem malha, a definição das condições de fronteira é um dos maiores problemas destes métodos. Por outro lado, o facto de os nós poderem ser distribuídos aleatoriamente pelo domínio físico do problema em análise, sem ser necessário o recurso implícito a conetividades, torna os métodos sem malha bastante apelativos para problemas de contacto e impacto com fratura e desagregação de material. Este trabalho foca-se no desenvolvimento dos métodos sem malha, assim como na descrição e análise comparativa dos métodos sem malha mais relevantes nos dias de hoje e nos algoritmos e técnicas computacionais capazes de tornar estes métodos mais eficientes. São ainda propostos alguns algoritmos que resultam num substancial aumento de performance do método *EFG (Element Free Galerkin)* assim como uma implementação do método *NEM (Natural Element Method)* com base no algoritmo de Fortune e em funções de forma não Sibsonianas. Por fim conclui-se que o aumento de performance dos métodos sem malha pode ser conseguido recorrendo a técnicas e metodologias usadas com frequência nas ciências computacionais, tendo como desvantagem um aumento da complexidade de implementação.

Keywords Meshless Methods; EFG; NEM; kd-Tree; Voronoi Diagram

Abstract

Recently, a new type of computational methods, referenced as mesheless methods, has emerged. These methods have been widely used in studying and solving engineering problems, most commonly problems that deal with large displacements, such as the simulation of plastic forming processes and problems envolving high rates of deformation, fracture or structural impact. Due to the non-interpolatory nature of the approximation functions of the meshless methods, to define the boundary conditions poses a major problem when dealing with such methods. However, the fact that nodes can be distributed randomly within the physical domain of the problem under analysis, without requiring the use of implicit connectivities, sets meshless methods as quite appealing when dealing with contact and impact problems involving fracture or material breakdown. This dissertation focuses on the development of meshless methods, as well as on the description and comparative analysis of the most relevant meshless methods nowadays, and in the algorithms and computational techniques able to render these methods more efficient. Some algorithms that result in a substantial performance increase of the EFG method (Element Free Galerkin) are also suggested, as well as an implementation of the NEM method (Natural Element Method) based on the use of the Fortune algorithm and on non Sibsonian shape functions. Finally, it is concluded that the increase in performance of meshless methods can be achieved using techniques and methodologies frequently used in computational science, being the disadvantage an increase in the implementation complexity.

"If you're going to try, go all the way.
Otherwise, don't even start."

CHARLES BUKOWSKI

Conteúdo

Lista de Tabelas

Lista de Figuras

Parte I

Introdução

Enquadra-se o presente trabalho, apresentam-se os objetivos a serem concretizados e os desafios a serem transpostos. É ainda feito um sucinto resumo de cada segmento deste trabalho.

Capítulo 1

Introdução e Objetivos

O método dos elementos finitos (FEM) tem vindo a ser utilizado em diferentes áreas de Engenharia com resultados bastante satisfatórios. No entanto, existem situações onde este método revela inexatidão, podendo até não ser o mais apropriado para a aplicação em causa. Quando, por exemplo, um material é sujeito a uma elevada taxa de deformação, os elementos podem sofrer uma elevada distorção. A forma de ultrapassar este problema consiste na aplicação de uma nova malha (remeshing), o que acarreta custos mais elevados de computação. Nas últimas três décadas, novos métodos têm sido estudados para tentar dar resposta às situações onde o FEM falha. Por não terem uma forma explicita de encontrar a conetividade entre os nós do domínio, estes métodos são chamados de métodos sem malha (*meshless/meshfree methods*) e, em vez de elementos, o domínio é discretizado por uma nuvem de pontos.

Contudo, para se obterem bons resultados com estes novos métodos, são necessários algoritmos eficientes de colocação de pontos e procura de vizinhanças que garantam a solução das equações inerentes aos métodos.

Neste sentido, são objetivos deste trabalho:

- Identificação de algoritmos mais adequados para a colocação pontual e integração das equações do movimento nos métodos sem malha;
- Implementação desses mesmos algoritmos em códigos de simulação numérica baseados nos métodos sem malha;
- Avaliação dos resultados (eficiência computacional e precisão) por comparação de simulações numéricas provenientes da aplicação de *softwares* comerciais em problemas reais (através do FEM ou de métodos sem malha já existentes).

De seguida, encontra-se um pequeno resumo onde são fornecidas as ideias base e os tópicos mais importantes que documentam todo o trabalho realizado para dar resposta aos objetivos acima propostos.

Parte 1 Enquadra-se o presente trabalho, apresentam-se os objetivos a serem concretizados e os desafios a serem transpostos. É ainda feito um sucinto resumo de cada segmento deste trabalho.

Parte 2 É feita uma breve explicação acerca dos métodos sem malha, seguindo-se a apresentação da contextualização histórica do mesmos, expondo os factos históricos mais significativos para a estado atual desta tecnologia. De seguida são

apresentados em detalhe alguns métodos sem malha assim como todos os aspetos relevantes da implementação, integração e avaliação dos algoritmos responsáveis pelo aumento de eficiência computacional do *EFG* assim como da implementação do método sem malha *NEM*.

Parte 3 É feita uma apresentação dos problemas usados para testar o *NEM* e os algoritmos de otimização do *EFG*. São esclarecidas algumas condições de simulação. Apresentam-se os resultados obtidos para os problemas propostos. Compara-se o *EFG* com algoritmos de redução do custo computacional com o *EFG* original quanto à redução do tempo de computação e precisão dos resultados. Compara-se o *EFG* com o *NEM*.

Parte 4 Analisam-se os resultados obtidos sumariamente, sendo feita uma proposição final com algumas considerações acerca do trabalho. Por fim, são pospostos alguns caminhos a seguir na continuação deste trabalho.

Parte II

Métodos Sem Malha

É feita uma breve explicação acerca dos métodos sem malha, seguindo-se a apresentação da contextualização histórica dos mesmos, expondo os factos históricos mais significativos para a estado atual desta tecnologia. De seguida são apresentados em detalhe alguns métodos sem malha assim como todos os aspetos relevantes da implementação, integração e avaliação dos algoritmos responsáveis pelo aumento de eficiência computacional do *EFG* assim como da implementação do método sem malha *NEM*.

Capítulo 2

Revisão do Estado da Arte

O constante avanço da tecnologia tem permitido encontrar novas soluções para os desafios impostos na atual sociedade de consumo. Cada vez mais, os produtos vão tomando geometrias complexas, possuindo dimensões e peso reduzidos e exibindo um melhor desempenho sem que o custo de produção aumente. Para tal, a ciência e a indústria têm investido na pesquisa de novos materiais e de novas ferramentas que possam auxiliar o processo de desenvolvimento de produtos. Uma fase essencial desse processo é a fase de modelação, simulação e análise, onde é possível determinar a viabilidade do projeto do ponto de vista estrutural. Tendo como premissa que os fenómenos físicos podem ser traduzidos recorrendo ao uso de ferramentas matemáticas os princípios mecânicos poderão, também, ser descritos pelo cálculo diferencial e integral, através do uso de equações diferenciais parciais e ordinárias ou integrais. No entanto, a solução analítica só existe para alguns casos ou para geometrias e condições de fronteira mais simples. Uma vez que a solução numérica é de vital importância para a fiabilidade das estimativas obtidas em simulação, a comunidade científica tem feito esforços na procura e desenvolvimento de métodos numéricos capazes de encontrar soluções aproximadas para essas equações.

No século passado foram desenvolvidos vários métodos computacionais. O método das diferenças finitas (FDM), o método dos volumes finitos, o método dos elementos finitos (FEM), o método dos elementos discretos (DEM), o método dos elementos de fronteira (BEM) e os métodos sem malha são alguns exemplos desses métodos. Em 1943 Courant, para resolver um problema de torção, usou funções de interpolação lineares para elementos triangulares, tendo como base o princípio da energia potencial mínima. (1)

A *Boeing*, em 1950, necessitou estudar estruturas não retangulares para asas com enflechamento positivo porque, até essa altura, a análise estrutural estava restringida ao estudo de elementos retangulares ligados a dois nós no espaço (2). *Clough* e *Turner*, que se tinham juntado ao projeto da *Boeing* em 1956, publicaram um artigo com as soluções encontradas que, hoje em dia, é considerado o início dos *FEM* aplicados à análise estrutural.(3) Dadas as suas características, os *FEM* tornaram-se uma ferramenta de análise transversal em vários problemas de engenharia, sendo usados em áreas tão distintas como problemas relacionados com fluidos, sólidos ou eletromagnetismo, ou mesmo em áreas multidisciplicares como engenharia civil, mecânica, aeronáutica e biomecânica (4). Presentemente, os *FEM* enfrentam novos desafios e novos problemas como a propagação arbitrária de fendas, grandes deformações, penetração de projéteis e falha e degradação de material, com os quais ainda não consegue lidar, sobretudo porque a ma-

lha entra em conflito com a realidade física e impede que a computação dos resultados se realize, obrigando a que seja gerada uma nova malha a cada vez que a realidade física é alterada ou a cada etapa de simulação, sendo por vezes necessária interação manual (5) . De modo a tentar resolver este problema, técnicas de geração de malha automáticas têm sido aplicadas (6). Com o mesmo objetivo, e com o aumento da pressão e necessidade de analisar e simular problemas mais complexos e avançados em engenharia e ciência, foi potenciado o desenvolvimento de novos métodos onde a computação é feita sem recurso à malha e sem as limitações impostas pelo tipo de elemento usado. Esses métodos são denominados métodos sem malha (*Meshless ou Meshfree methods*) por não necessitarem de uma malha para definir o seu domínio.

Como foi dito anteriormente, estes métodos foram desenvolvidos para fazer frente a problemas com mudança de geometria, onde os métodos que recorrem ao uso de malha podem não convergir, como é o caso de materiais sujeitos a grandes deformações ou com iniciação e propagação arbitrária de fendas.

Os métodos sem malha são, fundamentalmente, uma classe de métodos numéricos para resolução de equações diferenciais de modo discreto, recorrendo unicamente a informação nodal. Num método sem malha ideal seria unicamente necessária informação acerca dos nós, sem que fosse necessário recorrer ao uso de uma malha durante toda a análise. Isto não acontece nos métodos atuais onde, em algumas metodologias, é feito o uso de uma malha para fins de integração, seja global ou localmente. No entanto, existem métodos que não recorrem ao uso de malha, o que normalmente leva à deteção de alguma instabilidade e falta de precisão.

A origem dos métodos sem malha remonta a 1977 com o *Smoth Particle Hydrodynamics (SPH)* sugerido por Lucy (7) e por Gingold e Monaghan (8). O *SPH* é baseado numa formulação Lagrangiana, onde cada partícula representa uma massa ou um volume e tinha como propósito o estudo de fenómenos astrofísicos como *Novae* e *Supernovas*. Um estudo detalhado acerca do *SPH* por Monaghan (9), em 1982, mostrou a metodologia e a aplicabilidade a problemas relevantes em astrofísica. Este método também se revelou útil, tendo sido obtidos bons resultados, em problemas onde ocorrem grandes deformações ou rotura. A versatilidade do método possibilitou, ainda, a sua aplicação em problemas de hidrodinâmica de fluidos (10), simulação de choques e impactos (11) (12), formação de protoestrelas e galáxias (13) (7), conformação de metais (14) e explosões submersas (15). Não obstante do sucesso nas mais variadas aplicações, o *SPH* continua a sofrer de algumas lacunas como as dificuldades em atribuir condições de fronteira e a presença de instabilidade nas aproximações. Assim, algumas alterações a este método têm sido propostas (12) (16).

Apesar de os primeiros conceitos relacionados com os métodos sem malha, e em específico com o método *SPH*, terem sido publicados há mais de três décadas, só nos anos 90 do século passado é que estes despertaram a atenção da comunidade científica, possibilitando avanços importantes na resolução de problemas de engenharia. Relativamente ao método *Moving Least-square (MLS)*, o seu desenvolvimento foi iniciado em 1981 por Lancaster e Salkauskas (17) e tinha por objetivo a construção de uma curva ou superfície que se ajustasse a um conjunto de pontos dispersos. Durante os dez anos que se seguiram o *MLS* serviu apenas como ferramenta de otimização para problemas topológicos. Mas, posteriormente, a aproximação *MLS* foi a chave para o desenvolvimento de diversos métodos sem malha, uma vez que esta providencia uma aproximação continua para as funções de campo em todo o domínio do problema, sendo hoje em dia globalmente

usada na construção de funções de forma para métodos sem malha. Nayroles et al., em 1991, usaram pela primeira vez a aproximação *MLS* para o desenvolvimento de um método ao qual foi chamado *Diffuse element method (DEM)* (18) (19). No seguimento do trabalho de Nayroles *et al.*, Belytschko *et al.*, em 1994, desenvolveram o método *element-free Galerkin method (EFG)* no qual a facilidade de construção de funções de aproximação do MLS foi combinada com a formulação de Galerkin na forma fraca para a resolução de problemas de mecânica dos sólidos (20). Nos anos que se seguiram apareceram diversas metodologias que serão apresentadas de seguida por ordem cronológica, classificando cada método pelo grau de dependência ao uso de malha. O *reproducing kernel particle method (RKPM)* desenvolvido por Liu *et al.* (21), em 1995, a partir do conceito do núcleo reprodutor, é o resultado de várias correções que foram efetuadas ao *SPH* combinadas com a introdução de uma função corretiva para as funções de peso. O *finite point method (FPM)* introduzido por Onate et al. (22), em 1996, baseia-se no *MLS* para construir as funções de forma, mas recorre a um procedimento de colocação que dispensa manter funções de aproximação e de integração. Melenk e Babuska (23) desenvolveram o conceito de aproximação pela partição de unidade *PU*.

Contemporaneamente, uma família de métodos intitulada *meshless local Petrov-Galerkin methods (MLPG)* foi proposta com base na aproximação *MLS* e na aplicação da formulação de Galerkin, localmente em subdomínios simples (esferas ou elipses) de forma de facilitar a integração *meshless local Petrov-Galerkin methods (MLPG)* (24) (25) (26).

Existem, no entanto, alguns métodos sem malha que definem aproximações cujas funções de forma são interpoladoras, por exemplo, o *natural element method (NEM)* (27) que se destaca por se basear na formulação de Galerkin e por usar o conceito de interpolação por vizinhos naturais na definição das funções tentativa e teste. Esta técnica, introduzida por Sibson (28), usa o diagrama de Voronoi, também conhecido por arranjo de Dirichlet para a construção geométrica. Outros exemplos são o *natural neighbour method (NNM)* (29) e o *meshless finite element method (MFEM)* (5). O *point interpolation method (PIM)* (30) (31) combina a forma fraca de Galerkin com funções de base radial, de suporte global e funções polinomiais básicas. Duarte e Oden (32) propuseram o *h-p cloud method* baseado na aproximação *PU* e *MLS*.

Em relação aos *FEM*, os métodos sem malha apresentados acima têm as seguintes vantagens:

- Só é necessária informação dos nós, logo é necessário menos tempo de geração de malha e facilita a integração com ferramentas CAD (*Computer-Aided Design*);
- Adapta-se facilmente a mudanças de geometria;
- O campo de tensões é mais suave que nos *FEM* porque é possível usar funções de aproximação com grau mais elevado, logo não necessita de técnicas de suavização no pós-processamento.

Estas vantagens permitem que os métodos sem malha sejam empregues em:

- Mecânica do impacto, penetração e fragmentação do material;
- Simulação de escoamentos não confinados, de mudanças de fase em transferências de calor, de cristalizações e crescimento dos cristais;

- Simulação de problemas com grandes deformações, como a análise de placas e cascas, estruturas finas e embutidura de metais.

Precedendo a aplicação ampla e robusta dos métodos sem malha, alguns problemas têm de ser equacionados sendo os mais relevantes:

- Custo computacional e complexidade das funções de forma;
- Inexistência de algoritmos eficientes na procura de vizinhanças nodais em tempo real para computação em paralelo;
- Tratamento das condições de fronteira.

De modo a evitar algumas das dificuldades apresentadas pelos métodos sem malha, alguns destes foram acoplados a métodos de elementos finitos, formando os métodos híbridos, que exploram as vantagens dos métodos sem malha e dos métodos de elementos finitos (33) (34) (35) (36) (37) (38) (39).

É usual afirmar-se que os métodos sem malha são, do ponto de vista computacional, mais dispendiosos que os *FEM*, mas Trobec et al. (40) mostraram que o custo computacional total, incluindo pré-processamento, do método *MLPG* comparado com o *FEM*, usando algoritmos paralelos bem construídos, é comparável quanto ao custo computacional para problemas de alto nível. Idelsohn e Onate (41) estabelecem que o importante numa escolha entre métodos com e sem malha é a capacidade de criar um algoritmo eficiente para a conetividade nodal que mantenha a análise a correr eficientemente e com sucesso. Mostram ainda que um algoritmo eficaz, independentemente de se basear num método com ou sem malha, em função do número de nós, terá um tempo de computação sempre da mesma ordem de grandeza.

Assim, as condicionantes técnicas que provocam um maior aumento do custo computacional estão relacionadas com a implementação das condições de fronteira essenciais e com a complexidade dos algoritmos de computação das funções de forma e suas derivadas. Estas condicionantes técnicas foram já alvo de vários trabalhos, nomeadamente, na implementação das condições de fronteira foram já propostos esquemas de resolução, como o método dos multiplicadores de Lagrange (42) e da colocação pontual (43), enquanto que a diminuição do custo computacional relativo à complexidade dos algoritmos de computação das funções de forma e suas derivadas, contou com a apresentação de alguns algoritmos como a integração analítica (21), o método recursivo (44), a computação paralela (45) e a computação com GPU's (46).

Existem diversos métodos sem malha, com metodologias e propriedades diferentes, mas com muito em comum. Daqui surge a necessidade de comparar os métodos mais relevantes, ao que se procederá na secção seguinte.

Capítulo 3

Formulação dos Diferentes Métodos Sem Malha

3.1 *Smooth Particle Hydrodynamic - SPH*

O método *SPH*, pioneiro nos métodos sem malha, foi inicialmente desenvolvido por Lucy (7) e por Gingold e Monaghan (8) para resolução de simulações de fenómenos astrofísicos, como a formação e evolução de protoestrelas ou galáxias, tendo sido posteriormente adaptado para responder a problemas de mecânica dos sólidos, tais como fenómenos de grandes deformações e fenómenos de impacto e penetração (47) (48) (49). Atualmente, o *SPH* é usado na área da astrofísica em simulação de supernovas (50), colapso e formação de galáxias (51) (52) (53) (54), coalescência de buracos negros com estrelas de neutrões (55) e detonações em anões brancas (56). Na dinâmica de fluidos magnéticos é usado no estudo de ondas alfvenicas (57) e no estudo do desenvolvimento de ondas expansivas em nuvens magnéticas (58). Na mecânica do sólidos é aplicado a problemas de impacto (59) (60) (61). Na dinâmica de fluidos destacam-se os trabalhos sobre escoamentos multifase (62), condução de calor (63) e explosões submersas (15). O transporte e suspensão de sedimentos (64) também é uma matéria onde o método *SPH* é usado, assim como no estudo de ondas, rebentação de ondas e impacto destas em estruturas costeiras (65) (66) (67).

O *Smooth Particle Hydrodynamic* é baseado no seguinte integral:

$$u^h(\mathbf{x}) = \int \delta(\mathbf{x}-\mathbf{y})u(\mathbf{y})\, d\mathbf{y} \tag{3.1}$$

onde, $u(\mathbf{x})$ é a função a ser interpolada, e a ideia base é a de substituir a função de Dirac por um *kernel* positivo, par e compacto, obtendo-se:

$$u(\mathbf{x}) \simeq \tilde{n}^\rho(\mathbf{x}) := \int C_\rho \phi\left(\frac{\mathbf{x}-\mathbf{y}}{\rho}\right) u(\mathbf{y})\, d(\mathbf{y}) \tag{3.2}$$

onde ρ é o parâmetro de dilatação e caracteriza o raio de suporte da função *kernel*. C_ρ é a constante de normalização, assim:

$$\int C_\rho \phi\left(\frac{\mathbf{x}-\mathbf{y}}{\rho}\right)\, d(\mathbf{y}) = 1 \tag{3.3}$$

Quando ρ tende para zero, a função kernel aproxima-se da função de Dirac e consequentemente:

$$\lim_{\rho \to 0} \tilde{u}^{\rho}(\mathbf{x}) = u(\mathbf{x}) \tag{3.4}$$

As funções *kernel* mais comuns são:

- *Spline* cúbica

$$\phi(\xi) = 2 \begin{cases} \frac{2}{3} + 4(\xi - 1)\xi^2 & \text{se } \xi \leq 0.5 \\ \frac{4}{3}(1 - \xi)^3 & \text{se } 0.5 \leq \xi \leq 1 \\ 0 & \text{se } 1 \leq \xi \end{cases}, \quad \xi = \|x\| \tag{3.5}$$

- Gaussiana

$$\phi(\xi) = \begin{cases} \frac{e^{-9\xi^2} - e^{-9}}{1 - e^{-9}} & \text{se} \xi \leq 1 \\ 0 & \text{se} \xi \geq 1 \end{cases}, \quad \xi = \|x\| \tag{3.6}$$

De modo a desenvolver uma ferramenta computacional deve-se integrar a equação 3.2, da qual se obtém:

$$u(\mathbf{x}) \simeq \tilde{u}^{\rho} \simeq u^{\rho} := \sum_i V_i C_{\rho} \phi \left(\frac{\mathbf{x} - \mathbf{x}_i}{\rho} \right) u(\mathbf{x}_i) \tag{3.7}$$

Para cada partícula, a localização no espaço é dada por $\mathbf{x}_i$ à qual se associa um volume V_i. O contributo do volume de cada partícula $\mathbf{x}_i$, segundo Gingold and Monaghan, é dado por:

$$V_i^{-1} = \sum_j C_{\rho} \phi \frac{\mathbf{x}_i - \mathbf{x}_j}{\rho} \tag{3.8}$$

3.2 *Reproducing Kernel Particle Method - RKPM*

Em 1995, Liu propôs o *Reproducing kernel particle method(RKPM)* (68). Na tentativa de construir um procedimento que corrigisse a falta de consistência do método *SPH*, adicionou uma função de correção da aproximação do *kernel*, melhorando a precisão especialmente junto às fronteiras . O método *RKPM* foi usado com sucesso em fluidos, estruturas e acústica (69) (70). Lesoine e Kaila (71) usaram o *RKPM* para estudar os efeitos elásticos em aeronaves com grandes superfícies defletoras de controlo e Zhang, Wagner e Liu (72) mostraram que o *RKPM* é um método vantajoso em computação paralela na decomposição de domínios.

A função de peso foi inicialmente introduzida no método *SPH*. Para a função $u(\mathbf{x})$, a função de peso vem:

$$u^K(\mathbf{x}) = \int_{\Omega} \phi_a(\mathbf{y} - \mathbf{x}) u(\mathbf{y}) d\mathbf{y} \tag{3.9}$$

É de notar que se a função de peso $\phi(\mathbf{y} - \mathbf{x})$ for a função delta $\delta(\mathbf{y} - \mathbf{x})$, então, $u^K(\mathbf{x}) = \mathbf{u}(\mathbf{x})$. A computação de $u^K(\mathbf{x})$ num conjunto discreto de nós requer duas

aproximações: o integral contínuo é substituído pelo somatório nos nós e a função de peso deve ser escolhida de forma a que seja aproximada a uma função delta (*e.g.* *spline* ou Gaussiana) que possua consistência de ordem N e reproduza exatamente os polinómios de grau inferior ou igual a N. A função de peso $\phi_a(\mathbf{x})$ pode ser gerada a partir de $\phi(\mathbf{x})$ através do parâmetro a, que controla a amplitude da função de suporte:

$$\phi_a(\mathbf{x}) = \frac{1}{a}\phi\left(\frac{\mathbf{x}}{a}\right) \tag{3.10}$$

A função *spline* cúbica é um exemplo de uma função de peso uni-dimensional:

$$\phi(\xi) = 2\begin{cases} \frac{2}{3} + 4(\xi-1)\xi^2 & \text{se } \xi \leq 0.5 \\ \frac{4}{3}(1-\xi)^3 & \text{se } 0.5 \leq \xi \leq 1 \\ 0 & \text{se } 1 \leq \xi \end{cases}, \quad \xi = \|x\| \tag{3.11}$$

A forma discreta de 3.9 não assegura uma estimativa precisa de $u(\mathbf{x})$. Liu (21) , para impor condições de consistência de ordem superior, propôs a introdução de uma função de correção:

$$u^h(\mathbf{x}) = \sum_I C(\mathbf{x}; \mathbf{x}_I - \mathbf{x})\phi(\mathbf{x}_I - \mathbf{x})u(\mathbf{x}_I)\Delta V_I \tag{3.12}$$

Onde $u^h(\mathbf{x})$ é a versão *reproduzida* de $u(\mathbf{x})$. O somatório em 3.12 de nós discretos I e ΔV_I é o volume associado a cada nó I. O índice h representa a distância entre nós, e é normalmente escolhido de modo a ser proporcional a a em 3.10. Com h e a ambos zero, $u^h(\mathbf{x}) \to u(\mathbf{x})$. A função de correção $C(\mathbf{x}; \mathbf{x}_I - \mathbf{x})$ está subentendida em 3.12 uma vez que para uma dada consistência de ordem N, $u^h(\mathbf{x}) = u(\mathbf{x})$, se $u(\mathbf{x})$ for de ordem igual ou inferior a N. Assim C toma a forma:

$$C(\mathbf{x}; \mathbf{y} - \mathbf{x} = \mathbf{b}^T(\mathbf{x})\mathbf{P}(\mathbf{y} - \mathbf{x}) \tag{3.13}$$

Onde $\mathbf{P}(\mathbf{y} - \mathbf{x})$ é o vetor de polinómios de ordem inferior a N e $\mathbf{b}$ o vetor dos coeficientes a determinar:

$$\mathbf{P}^T(\mathbf{y} - \mathbf{x}) = [1, x_1 - y_1, x_2 - y_2, x_3 - y_3, (x_1 - y_1)^2, ..., (x_3 - y_3)^N]\mathbf{b}^T = [b_1(\mathbf{x}), b_2(\mathbf{x}), ...] \tag{3.14}$$

Os coeficientes $b_i(\mathbf{x})$ são determinados impondo a consistência de ordem N da aproximação. Substituindo 3.13 em 3.12, fica:

$$u^h(\mathbf{x}) = u(\mathbf{x}) = \sum_I \mathbf{b}^T(\mathbf{x})\mathbf{P}(\mathbf{x}_I - \mathbf{x})\phi_a(\mathbf{x}_I - \mathbf{x})u(\mathbf{x}_I)\Delta V_I \tag{3.15}$$

Quando $u(\mathbf{x})$ é um polinómio de ordem inferior a N, $u(\mathbf{x}_I)$ é representado pela série de Taylor:

$$u(\mathbf{x}_I) = u(\mathbf{x}) + (x_{I1} - x_1)\frac{\partial u(\mathbf{x})}{\partial x_1} + (x_{I2} - x_2)\frac{\partial u(\mathbf{x})}{\partial x_2} + (x_{I3} - x_3)\frac{\partial u(\mathbf{x})}{\partial x_2} + ... \tag{3.16a}$$

$$= \mathbf{P}^T(\mathbf{x}_I - \mathbf{x})\mathbf{w}(\mathbf{x}) \tag{3.16b}$$

onde,

$$\mathbf{w}(\mathbf{x}) \equiv \left[u(\mathbf{x}), \frac{\partial u(\mathbf{x})}{\partial x_1}, \frac{\partial u(\mathbf{x})}{\partial x_2}, \frac{\partial u(\mathbf{x})}{\partial x_3}, ...\right]^T \tag{3.17}$$

Substituindo 3.16b em 3.15 e resolvendo em ordem a $\mathbf{b}(\mathbf{x})$:

$$\mathbf{b}(\mathbf{x}) = \mathbf{M}^{-1}(\mathbf{x})\mathbf{P}(0) \tag{3.18}$$

onde,

$$\mathbf{M}(\mathbf{x}) \equiv \sum_I \mathbf{P}(\mathbf{x}_I - \mathbf{x})\mathbf{P}^T(\mathbf{x}_I - \mathbf{x})\phi_a(\mathbf{x}_1 - \mathbf{x})\Delta V_I \tag{3.19}$$

Substituindo 3.13 e 3.18 em 3.12, vem:

$$u^h(\mathbf{x}) = \sum_I \phi_I(\mathbf{x})u_I \tag{3.20}$$

onde $u_I = u(x_I)$, e a função de forma $\phi_I(x)$ é independente de $u(\mathbf{x})$ e é:

$$\phi_I(\mathbf{x}) = \mathbf{P}^T(0)\mathbf{M}^{-1}(x)\mathbf{P}^T(\mathbf{x}_I - \mathbf{x})\phi_a(\mathbf{x}_I - \mathbf{x})\Delta V_I \tag{3.21}$$

3.3 *Element Free Galerkin - EFG*

O método *Element Free Galerkin* foi inicialmente desenvolvido por Belytschko, em 1994 (20), com base nos trabalhos de Nayroles com o *Diffuse Element Method* (19). Para determinar as funções de forma, Belytshchko usou o *Moving Least Square Approximation* apresentado por Lancaster e Salkauskas em 1980 (17) (73). Em 1999, Belytshchko e Krysl desenvolveram o método *EFG* para placas finas e placas de Kirchhoff (74). Com este estudo foi possível encontrar a ordem de quadratura ótima e a dimensão do domínio de influência mais favorável à análise *EFG*. O método *Element Free Galerkin - EFG* foi usado com sucesso em problemas de placas e cascas (74) (75), mecânica da fratura e propagação de ondas (76) (77) (78) (79), *Nondestructive testing (NDT)* (80), campo eletromagnético (81) e transferência de calor (82) (83) (84).

A aproximação *Moving Least Squares (MLS)* foi descrita por Lancaster e Salkaukas (17) e é usada no método EFG para estimar a função $u(\mathbf{x})$, da seguinte forma:

$$u(\mathbf{x}) \approx u^h(\mathbf{x}) = \sum_j^m p_j(\mathbf{x})a_j(\mathbf{x}) \equiv \mathbf{p}^{\mathrm{T}}(\mathbf{x})\mathbf{a}(\mathbf{x}) \tag{3.22}$$

onde,

- m = ordem do polinómio,
- $\mathbf{p}^{\mathrm{T}}(\mathbf{x})$ = polinómio,
- $\mathbf{a}(\mathbf{x})$ = parâmetros desconhecidos,
- $p_j(\mathbf{x})$ = monómio de base funcional,
- $a_j(\mathbf{x})$ = coeficientes não constantes.

O polinómio deverá ser completo, o que para problemas a duas dimensões se traduz por:

$$\mathbf{p}^{\mathrm{T}}(x,y) = \{1, x, y, xy, x^2, y^2, ..., x^m, y^m\} \tag{3.23}$$

Os coeficientes $a_j(\mathbf{x})$ são calculados a partir da função quadrática $J(\mathbf{x})$ e o número de nós do domínio de influência tem a condição $n \geq m$. Logo:

$$\mathbf{J}(\mathbf{x}) = \sum_{l=i}^{n} \omega(\mathbf{x} - \mathbf{x}_l)\{\sum_{j=1} p_j(\mathbf{x}_l)a_j(\mathbf{x}) - u_1\}^2 \tag{3.24}$$

onde, $\omega(\mathbf{x} - \mathbf{x}_l)$ é a função de peso, positiva no domínio de influência, e n o número de nós no mesmo domínio de influência. Com:

$$\mathbf{a}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x})\mathbf{T} \tag{3.25}$$

e $\mathbf{A}$ e $\mathbf{B}$:

$$\mathbf{A} = \sum_{l=1}^{n} \omega(\mathbf{x} - \mathbf{x}_l)\mathbf{p}(\mathbf{x}_l)\mathbf{p}^T(\mathbf{x}_l) \tag{3.26}$$

$$\mathbf{B}(\mathbf{x}) = [\omega(\mathbf{x} - \mathbf{x}_1)\mathbf{p}(\mathbf{x}_1), ...\omega(\mathbf{x} - \mathbf{x}_n)\mathbf{p}(\mathbf{x}_n)] \tag{3.27}$$

substituindo 3.25 em 3.22, a função aproximação é dada por:

$$u^h(\mathbf{x}) = \sum_{l=1}^{n} \Phi(\mathbf{x})u_l = \mathbf{\Phi}(\mathbf{x})\mathbf{u} \tag{3.28}$$

a função de forma $\mathbf{\Phi}(\mathbf{x})$ é definida por:

$$\Phi_I(\mathbf{x}) = \sum_{j=0}^{m} p_j(\mathbf{x}) \left(\mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x})\right)_{jI} \tag{3.29}$$

As funções de peso podem ser escritas em função de um raio r normalizado pelo seguinte polinómio de quarta ordem (77):

$$\omega(\mathbf{x} - \mathbf{x}_I) = \omega(r) = \begin{cases} 1 - 6r^2 + 8r^3 - 3r^4, & 0 \leq r \leq 1 \\ 0, & r > 1 \end{cases} \tag{3.30}$$

pela função Gaussiana (85):

$$\omega(\mathbf{x} - \mathbf{x}_I) = \omega(r) = \begin{cases} e^{-(2.5r)^2}, & 0 \leq r \leq 1 \\ 0, & r > 1 \end{cases} \tag{3.31}$$

ou pela função *spline* cúbica (86):

$$\omega(\mathbf{x} - \mathbf{x}_I) = \omega(r) = \begin{cases} \frac{2}{3} - 4r^2 + 4r^3 & \text{se } r \leq 0.5 \\ \frac{4}{3} - 4r + 4r^2 - \frac{4}{3}r^3 & \text{se } 0.5 \leq r \leq 1 \\ 0 & \text{se } 1 \leq r \end{cases} \tag{3.32}$$

onde

$$r = \frac{d_i}{d_{mi}}, \;\; d_i = \|\mathbf{x} - \mathbf{x_i}\|, \;\; d_{mi} = d_{max}c_i \tag{3.33}$$

com d_{max} um parâmetro de escala, e a distância c_i escolhida de forma que a matriz A seja invertível.

Como as funções aproximação do método *EFG* não satisfazem as propriedades delta Kronecker $\Phi_I(x_J) \neq \delta_{IJ}$ é recomendado o uso de multiplicadores de Lagrange na imposição das condições de fronteira.

As equações discretas da formulação de Galerkin podem ser escritas da seguinte forma para o caso 2D (20):

$$\begin{bmatrix} \mathbf{K} & \mathbf{G} \\ \mathbf{G}^T & 0 \end{bmatrix} \left\{ \begin{array}{c} \mathbf{u} \\ \lambda \end{array} \right\} = \left\{ \begin{array}{c} \mathbf{f} \\ \mathbf{q} \end{array} \right\} \tag{3.34}$$

onde,

$$K_{IJ} = \int_{\Omega} \mathbf{B}_I^T \mathbf{D} \mathbf{B}_J \mathrm{d}\Omega \quad ; \quad \mathbf{B}_I = \begin{bmatrix} \Phi_{I,x} & 0 \\ 0 & \Phi_{I,y} \\ \Phi_{I,y} & \Phi_{I,x} \end{bmatrix} \tag{3.35}$$

$$\mathbf{G}_{IK} = -\int_{\Gamma_u} \Phi_I \mathrm{N}_K \mathrm{d}\Gamma \quad ; \quad \mathbf{f}_I = \int_{\Gamma_t} \Phi_I \bar{\mathbf{t}} \mathrm{d}\Gamma + \int_{\Omega} \Phi_I \mathbf{b} \mathrm{d}\Omega \quad ; \quad \mathbf{q}_K = -\int_{\Gamma_u} \mathrm{N}_K \bar{\mathbf{u}} \mathrm{d}\Gamma \tag{3.36}$$

Com:

- N_K = função de interpolação dos multiplicadores de Lagrange,
- $\mathbf{u}$ = vetor dos deslocamentos nodais,
- λ = multiplicadores de Lagrange,
- $\mathbf{D}$ = matriz de elasticidade ($\sigma = D\varepsilon$)

A matriz $\mathbf{D}$ para um material isotópico e para um estado plano de tensão é dada por:

$$\mathbf{D} = \frac{E}{(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{3.37}$$

e para um estado plano de deformação:

$$\mathbf{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1}{2}-\nu \end{bmatrix} \tag{3.38}$$

3.4 *Natural Element Method - NEM*

O *Natural Element Method* é, na sua essência, um método baseado na formulação de Galerkin que recorre à interpolação por vizinhos naturais na construção da função tentativa e teste (28). As funções de forma no *NEM* são estritamente interpoladoras, simplificando a questão da imposição das condições de fronteira essenciais, fazendo com que vários autores considerem o *NEM* como a generalização mais adequada de interpolações lineares

para duas ou mais dimensões (87). O *Natural Element Method* é um método sem malha bem estabelecido que mostrou o seu sucesso tanto do ponto de vista teórico (27) (88) (89) (90), como na resolução de problemas de hidráulica (91), mecânica dos sólidos (92), conformação de metais (93), dinâmica de fluidos (94) e eletromagnetismo (95)

A interpolação *NEM* é construída com base na construção de Voronoi que, por sua vez, tem a triangulação de Delaunay como seu dual topológico. No contexto do *natural neighbor interpolation*, um círculo que circunscreva um triângulo de Delaunay é conhecido por *natural neighbor circumcircle* ou vizinho natural (96).

Considerando um conjunto de nós $N = \{n_1, n_2, ..., n_M\}$ em $\Re^2$, o diagrama de Voronoi de primeira ordem do conjunto N consiste na subdivisão do plano em regiões T_I (células de Voronoi), dadas por:

$$T_I = \{\mathbf{x} \in \Re^2 \ : \ d(\mathbf{x}, \mathbf{x}_I) < d(\mathbf{x}, \mathbf{x}_J) \, \forall \, J \neq I\}, \tag{3.39}$$

onde, $d(\mathbf{x}_I, \mathbf{x}_J)$ é a distância Euclidiana entre $\mathbf{x}_I$ e $\mathbf{x}_J$. O diagrama de Voronoi para o conjunto de pontos N está representado na figura 3.2a com a introdução de um ponto $\mathbf{x}$. Se o $\mathbf{x}$ participar com o conjunto N na construção do diagrama de Voronoi, então, os seus vizinhos naturais serão os nós que formem um vértice de triângulo em $\mathbf{x}$. A função de forma do vizinho natural I é definida como o rácio entre a área sobreposta da sua célula de Voronoi e o total de área da célula de Voronoi $\mathbf{x}$, designadas funções de Sibson, Figura 3.2b:

$$\phi_I(\mathbf{x}) = \frac{A_I(\mathbf{x})}{A(\mathbf{x})}, \quad A(\mathbf{x}) = \sum_{J=1}^{n} A_J(\mathbf{x}) \tag{3.40}$$

onde, I está compreendido entre 1 e n. Se o ponto $\mathbf{x}$ coincidir com um nó ($\mathbf{x} = \mathbf{x}_I$), $\phi_I(\mathbf{x}) = 1$ e todas as restantes funções de forma são zero. As funções de forma são positivas e verificam a propriedade delta Kronecker, portanto são interpoladoras. Exemplificando para a figura 3.2b, a função de forma da célula denominada célula 3 de vértices {cdfgh} é:

$$\phi_3(\mathbf{x}) = \frac{A_{cdfgh}}{A_{abcde}} \tag{3.41}$$

Embora o cálculo das funções de forma seja trabalhoso, a sua construção é unicamente geométrica:

$$0 \leq \phi_I(\mathbf{x}) \leq 1, \quad \phi_I(\mathbf{x}_J) = \delta_{IJ}, \quad \sum_{I=1}^{n} \phi_I(\mathbf{x}) = 1 \; em \; \Omega \tag{3.42}$$

As funções de forma dos vizinhos naturais também satisfazem a propriedade das coordenadas locais (28):

$$\mathbf{x} = \sum_{I=1}^{n} \phi_I(\mathbf{x}) \mathbf{x}_I \tag{3.43}$$

As primeiras derivadas das funções de forma dos vizinhos naturais, usando a equação 3.40, podem ser escritas da seguinte forma:

$$\phi_{I,\alpha}(\mathbf{x}) = \frac{A_{I,\alpha} - \phi_I(\mathbf{x}) A_\alpha(\mathbf{x})}{A(\mathbf{x})} \ \ (\alpha = 1, 2) \tag{3.44}$$

As funções de forma são do tipo C^∞ em todo o espaço, exceto nos nós onde são C^0 (28). A implementação de Galerkin de uma interpolação de vizinhos naturais do tipo C^1 foi estudada por Sukumar e Moran (97). Neste estudo, o algoritmo geométrico proposto por Watson (96) é usado para processar as funções de forma dos vizinhos naturais e as suas derivadas.

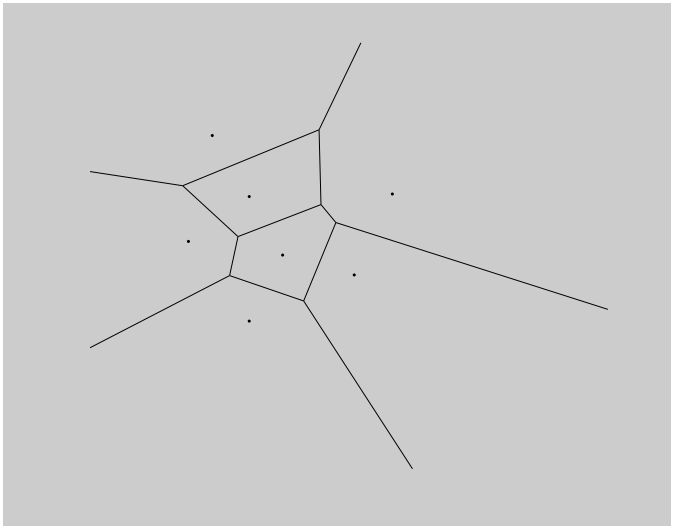


Figura 3.1: Diagrama de Voronoi.

Em alternativa às funções de forma acima mencionadas, tem-se o método de Laplace, também conhecido por método não Sibsoniano. Esta metodologia mostra-se como uma alternativa menos complexa e com uma implementação mais simples e eficiente, uma vez que usa comprimentos em vez de áreas para quantificar a sobreposição das células.

Na figura 3.3 $s_i(x)$ é o comprimento da aresta comum da célula de voronoi e da célula vizinha do nó i, e $h_i(x)$ é metade da distância euclidiana do ponto x ao i. Permitindo definir as funções de forma do seguinte modo (98):

$$\Phi_i(x) = \frac{\alpha_i(x)}{\sum_{j=1}^{n} \alpha_j(x)} \tag{3.45}$$

onde,

$$\alpha_j(x) = \frac{s_j(x)}{h_j(x)} \tag{3.46}$$

No método de Galerkin as funções básicas ($\mathbf{u}^h$) e as funções teste ($\mathbf{v}^h$) são:

$$\mathbf{u}^h(\mathbf{x}) = \sum_i \phi(\mathbf{x})\mathbf{u}_i \tag{3.47}$$

$$\mathbf{v}^h(\mathbf{x}) = \sum_i \phi(\mathbf{x})\mathbf{v}_i \tag{3.48}$$

E a forma fraca do problema discreto pode ser escrita como:

$$\int_{\Omega^b} \sigma(\mathbf{u}^h) : \varepsilon(\mathbf{v}^h) = \int_{\Omega^b} \mathbf{b} \cdot \mathbf{v} d\Omega + \int_{\Gamma_t^h} \bar{\mathbf{t}} \cdot \mathbf{v} d\Gamma \ \ \forall \mathbf{v^h} \in \mathbf{V}_0^h \subset \mathbf{V}_0 \tag{3.49}$$

As equações discretas finais obtêm-se por substituição das funções básicas e funções teste na equação 3.49 resultando o seguinte sistema final:

$$\mathbf{Kd} = \mathbf{f} \tag{3.50}$$

onde

$$\mathbf{K}_{ij} = \int_{\Omega^b} \mathbf{B}_i^t \mathbf{D} \mathbf{B}_j d\Omega \tag{3.51}$$

$$\mathbf{f}_i = \int_{\Gamma_t^h} \phi_i \bar{\mathbf{t}} d\Gamma + \int_{\Omega^h} \phi_i \mathbf{b} d\Omega + \int_{\Omega^b} \mathbf{B}_i^t \mathbf{D} \varepsilon^* d\Omega \tag{3.52}$$

$$\mathbf{B} = \begin{bmatrix} \phi_{i,x} & 0 \\ 0 & \phi_{i,y} \\ \phi_{i,x} & \phi_{i,y} \end{bmatrix} \tag{3.53}$$

E D a matriz constitutiva para um material isótropico linear elástico (equação 3.37 e 3.38).

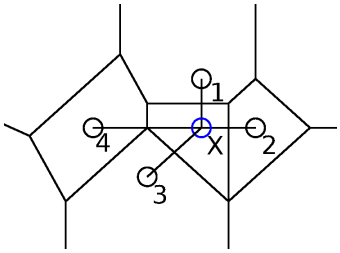
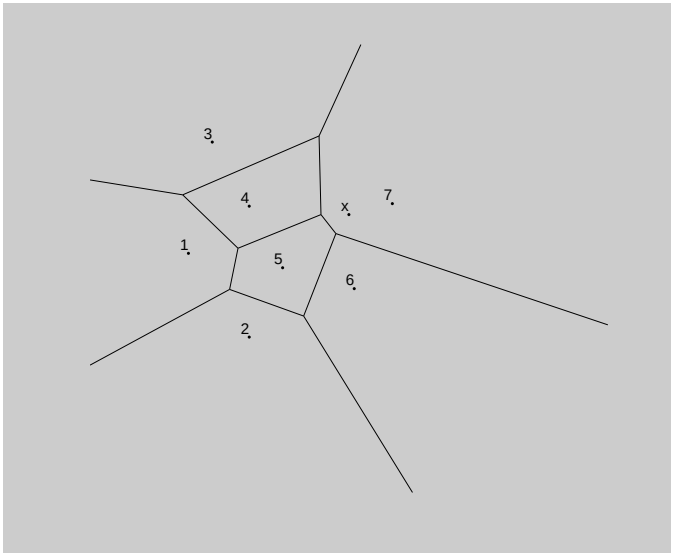
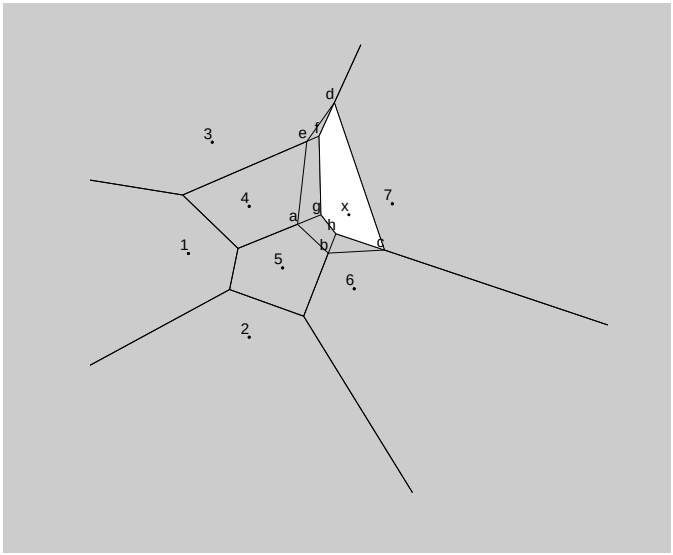


Figura 3.3: Funções de forma não Sibsonianas - *NEM*.



(a) Diagrama original com ponto x.



(b) Células de Voronoi em torno de x.

Figura 3.2: Construção de vizinhos naturais.

Capítulo 4

Algoritmos de Pesquisa e Estruturas de Dados

4.1 *Search Data Structures*

Muitas das operações das funções aproximação dos métodos sem malha são aplicadas à vizinhança local. Se as funções *kernel* usadas nas funções aproximação apresentarem suporte local, as funções valor no ponto x são influenciadas unicamente por um conjunto de vizinhos $N_r(x) = \{x_i : \|x_i - x\| \leq r\}$, onde r é distância máxima para um *kernel* diferente de zero.

Se for usada uma técnica baseada num algoritmo *Brute Force* (ou algoritmo de procura exaustiva, onde todas as alternativas possíveis são examinadas por forma a encontrar uma solução particular), é necessário um tempo muito elevado para encontrar o conjunto N_r . No entanto, este tempo pode ser reduzido se forem usadas estruturas de dados, como é o caso das estruturas do tipo *uniform grids*, *octrees*, *bounding volume hierarchies*, *kd-Tree*, *Spatial hashing* ou *Voronoi tree's*. Esta secção mostra o Estado da Arte acerca de estruturas de dados do tipo *kd-Tree* e *Spatial hashing*, as mais recomendadas para uso em métodos sem malha, ilustradas na figura 4.1 (99) (100).

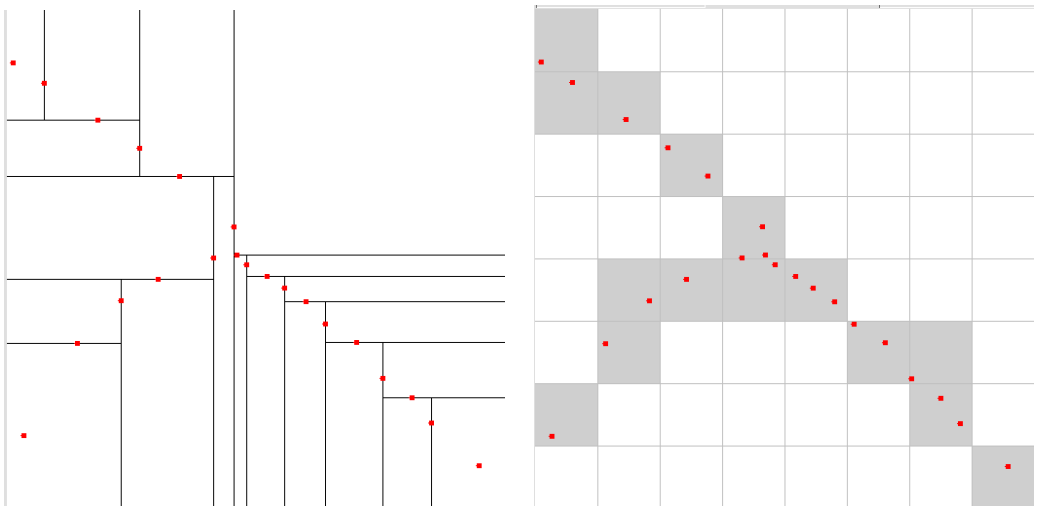
4.1.1 *kd-Tree*

Este tipo de estruturas tem o propósito de responder eficientemente a uma pesquisa, dado um conjunto de objetos já processados numa estrutura deste género.

Numa pesquisa por extensão, dado um conjunto de pontos P e uma região do espaço R com qualquer geometria, é pedido o sub-conjunto de P que está contido nessa região. Antes de se considerar o funcionamento desta pesquisa, assim como da construção da estrutura de dados para qualquer dimensão, deve-se considerar inicialmente para o caso a uma dimensão.

Considerando um conjunto de pontos $p = \{p_1, p_2, p_3, ..., p_n\}$ posicionados em linha, e um intervalo $[x_0, x_1]$, o objetivo é devolver os pontos contidos no intervalo.

Aparentemente a solução para resolver este problema seria ordenar os pontos por ordem crescente, e então, aplicar uma pesquisa para encontrar o primeiro ponto maior ou igual que x_0, o primeiro ponto menor ou igual que x_1 e listar os pontos entre estes extremos.



(a) *kd-Tree*, a árvore adapta-se à distribuição de pontos.

(b) *Hash grid*, a estrutura da grelha é guardada numa tabela.

Figura 4.1: Esquematização de algoritmos de *Search Data Structures*.

Não obstante de a implementação anterior devolver um resultado correto, apresenta-se de seguida outra implementação onde os pontos são ordenados por ordem crescente e armazenados nas folhas de uma árvore binária balanceada. Cada nó interno da árvore é referenciada pelo maior ponto contido nos sub-conjuntos dos níveis inferiores à esquerda, como representado na figura 4.2.

Para encontrar os pontos contidos no intervalo $[x_0, x_1]$. É feita uma pesquisa à árvore para encontrar a folha mais à esquerda cujo nó tenha uma referência maior ou igual que x_0 e a folha mais à direita b cuja referência do nó seja inferior ou igual a x_1. Todas as folhas entre a e b e possivelmente com a e b constituem os pontos contidos no intervalo. Se a referência da folha a for igual a x_0 esta é incluída. Assim como se a referência da folha b for igual a x_1, como exemplificado na Figura 4.2 .

Para devolver os pontos contidos no intervalo $[x_0, x_1]$ segue-se o caminho comum aos trajetos para chegar à folha a e à folha b. Assim que os caminhos divergirem e seguindo o caminho à esquerda para a, sempre que o caminho seguir para o nível inferior à esquerda de um nó, todos os pontos contidos nos níveis inferiores à direita desse nó são incluídos na resposta. Simetricamente, ao seguir o caminho à direita para a folha b, quando o caminho seguir para um sub-conjunto inferior à direita, todos os pontos contidos no sub-conjunto inferior à esquerda são adicionados. A questão de como organizar a árvore será respondida na generalização para dimensões superiores, a chamada *kd-Tree*.

Para implementar o princípio abordado anteriormente a duas ou três dimensões é necessário considerar as *kd-Tree*. Estas estruturas de dados são de fácil implementação e muito práticas, uma vez que é possível aplicar diferentes tipos de pesquisa. A estrutura foi desenvolvida por Jon Bentley (101) (102). Posteriormente, S. Omohundro recomendou este algoritmo como sendo uma possível técnica para aumentar a velocidade de aprendizagem de redes neuronais (103). O algoritmo *kd-Tree* é um caso especial de uma árvore binária de partição de espaço, com separação de planos axialmente alinhados (104).

A ideia base é a de extender o conceito aplicado anteriormente mas para cada nó o

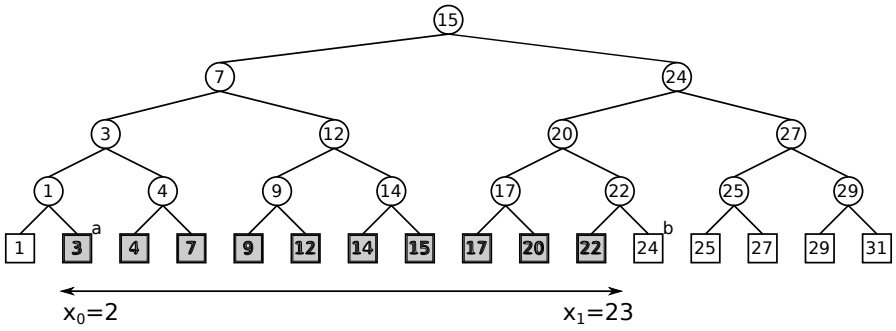


Figura 4.2: Pesquisa por extensão - 1D

espaço é subdividido ou segundo a coordenada x ou segundo a coordenada y do ponto, para um caso 2D. Cada nó interno (n) da *kd-Tree* tem associadas as seguintes referencias:

$n.divisão$ A dimensão de subdivisão (e.g. xx, yy, zz);

$n.valor$ O valor de corte;

$n.tamanho$ O número de pontos na sub-árvore.

Considerando o $n.valor = i$ e que o primeiro plano de divisão é o yy todos os pontos com coordenada y menor que $n.valor$ são guardados na sub-árvore esquerda, enquanto que os pontos com coordenada y maior que $n.valor$ são guardados na sub-árvore direita. Se a coordenada y do ponto for igual a $n.valor$ este ponto pode ir para qualquer um dos lados de modo a balancear a árvore, para que esta tenha o mesmo número de pontos de ambos os lados, como exemplificado na Figura 4.4.

O processo de criação da *kd-Tree* tem uma representação geométrica, em que cada nó da *kd-Tree* tem associado um espaço retangular chamado célula, representando uma decomposição hierárquica do espaço, como ilustrado na Figura 4.3.

Na construção de uma *kd-Tree* existem duas decisões fundamentais para o desempenho desta:

Escolha da dimensão da subdivisão A metodologia mais simplista é a de trocar a dimensão da subdivisão em ciclo (entre xx e yy), mas uma vez que pode formar células muito alongadas, diminuindo a performance, deve ser escolhida a dimensão com maior diferença entre a maior e menor coordenada.

Escolha da dimensão de corte O melhor método é que o $n.valor$ seja a mediana das coordenadas ao longo da dimensão de corte. Se existir um número par de pontos, deve-se optar pela mediana superior ou inferior, ou simplesmente pelo ponto médio.

O algoritmo 1 escrito em pseudocódigo é a base de construção do algoritmo *kd-tree*. Algoritmos mais sofisticados, baseados em análise estatística dos pontos de entrada, foram publicados em (105) (106).

Algorithm 1 Construção de uma kd-Tree.(107)

Seja S o conjunto de pontos, k a dimensão do espaço de incorporação tal que: $\mathbf{x} \in \Re^k \forall \mathbf{x} \in S$ e s o número máximo de pontos em cada ramo.

function SplitCell($axis, S$)
  if $|S| \leq s$ **then**
    return LeafNode (S)
  else
    $split = median_{\mathbf{x} \in S}\mathbf{x}(axis)$ ▷ Encontrar o plano de separação.
    $S_l = \{\mathbf{x} \in S : \mathbf{x}[axis] < split\}$ ▷ Selecionar nós para subespaços.
    $S_r = \frac{S}{S_l}$
    $newaxis = (axis + 1) mod k$ ▷ Ciclo das dimensões de divisão.
    return TreeNode(SplitCell(newaxis, S_l),
    SplitCell (newasix, S_r),
    split,axis)
  end if
end function
function MakeTree(S)
  return SplitCell $(0, S)$
end function

Pesquisa por extensão numa *kd-Tree*

Considerando R a extensão de procura, N o nó atual na *kd-Tree* e C a célula correspondente ao nó N. O algoritmo de pesquisa percorre a árvore recursivamente. Ao chegar a C, se N estiver contido em R, é contado. Se por outro lado N for um nó interno e se a célula C e o conjunto R forem disjuntos, nenhum ponto na sub-árvore de N é considerado. Se C é contida totalmente em R, todos os pontos da sub-árvore são contados. Se C está parcialmente contida em R, a sub-árvore será recursivamente analisada, procedendo de modo análogo para elemento do nível inferior. A Figura 4.3 e 4.4 mostram um exemplo de uma procura, onde os nós a branco são aqueles que foram visitados mas não foram considerados, a cinza claro não foram visitados por estarem completamente contidos em R e a cinza escuro estão os nós que não foram visitados por serem disjuntos de R.

Uma versão do pseudocódigo deste processo está descrito no algoritmo 2. Esta operação é uma modificação ao algoritmo de procura do vizinho mais próximo e é possível que o custo computacional continue a ser o mesmo (108).

Implementação

Para a implementação da *kd-Tree*, foi escolhido o *Software Matlab*, pela compatibilidade com o restante trabalho já efetuado no Departamento de Engenharia Mecânica da Universidade de Aveiro assim como pelo amplo uso desta ferramenta por toda a comunidade académica, pela robustez das *toolboxes* e pela facilidade de modificação do trabalho no futuro.

De início a opção foi implementar o algoritmo de construção das *kd-Tree* assim como o de pesquisa por extensão e por número de elementos (*k-nearest neighbor search*), esta implementação foi realizada em *Matlab* seguindo a metodologia apresentada anteriormente, mas ainda que funcional, comparativamente aos algoritmos nativos deste software para o

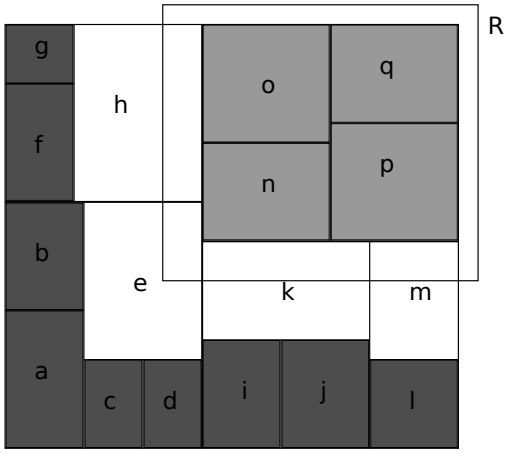


Figura 4.3: Divisão de uma *kd-Tree*.

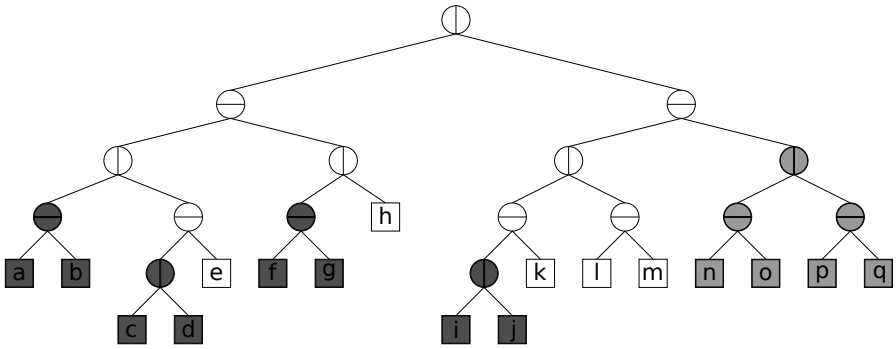


Figura 4.4: Estrutura de uma *kd-Tree*.

mesmo fim, mostrou-se muito pouco eficiente comprometendo o objetivo primário deste trabalho. Tendo-se adotado os algoritmos nativos do *Matlab* no restante trabalho e nas simulações realizadas.

De seguida o rumo tomado passou por modificar uma implementação do método *EFG* já existente no Departamento de Engenharia Mecânica da Universidade de Aveiro de modo a diminuir o tempo de computação. A implementação da *kd-Tree* para cada um dos problemas do *EFG* foi a seguinte:

Problema do domínio de suporte É construída uma *kd-Tree* de todos os nós, aplicando-se a essa *kd-Tree* uma procura pelo número de vizinhos mais próximos (*k-nearest neighbor search*), número esse dependente da ordem do polinómio usado. Sendo guardado numa estrutura de dados do tipo *struct* o valor da distância do vizinho mais distante encontrado. Esta procura é feita para todos os nós e multiplicando o d_{max} pela distância ao nó mais distante encontra-se o domínio de suporte de cada nó.

Problema da procura dos nós Sabendo o domínio de influência de cada nó, constrói-se uma estrutura do tipo *stuct* com uma entrada para cada ponto de Gauss. De seguida é construída uma *kd-Tree* com os pontos de Gauss, e faz-se uma pesquisa por extensão em cada nó com o raio do seu domínio de influência, obtendo os

Algorithm 2 Pesquisa por extensão numa kd-Tree.(107)

Pesquisando numa kd-Tree T no ponto $\mathbf{x}$, para encontrar todos os vizinhos em que $N_r(\mathbf{x}) = \mathbf{x}_i : \|\mathbf{x}_i - \mathbf{x}\| < r$

function GATHERSAMPLES($T, \mathbf{x}, r$)
  if isLeaf(T) **then**
    return $\mathbf{x}_i \in T.S \|\mathbf{x} - \mathbf{x}_i\| < r$
  else
    $S = 0$
    if IntersectsSphere($T.left, \mathbf{x}, r$) **then**
      $S \leftarrow S \cup$ GatherSamples ($T.left, \mathbf{x}, r$)
    end if
    if IntersectsSphere($T.right, \mathbf{x}, r$) **then**
      $S \leftarrow S \cup$ GatherSamples ($T.right, \mathbf{x}, r$)
    end if
    return S
  end if
end function
function TREEQUERY($T, \mathbf{x}, r$) ▷ Encontrar o ultimo nó contido na extensão de pesquisa.
  whithe not isLeaf(T) **do**
    if ConstainsSphere ($T.left, \mathbf{x}, r$) **then**
      $T = T.left$
    else if ConstainsSphere ($T.right, \mathbf{x}, r$) **then**
      $T = T.right$
    else
      break
    end if
  end while
  return GatherSamples ($T, \mathbf{x}, r$)
end function

pontos de Gauss nesse domínio. Adicionando ao campo de introdução de cada ponto de Gauss na estrutura criada, a informação de que o nó em que foi feita a pesquisa o continha no seu domínio de influência. Ficando no fim com uma estrutura de dados que relaciona cada ponto de Gauss com os nós cujo domínio de influência o inclui.

De modo a validar esta implementação, propõem-se alguns problemas, fazendo sempre que possível a comparação do custo computacional entre esta implementação e a implementação original sendo também avaliadas a precisão e a convergência de ambas. O número de nós usado nas simulações ultrapassa o número de nós de convergência uma vez que com com um maior número de nós os ganhos relativos ao uso da *kd-Tree* de dados é mais percetível.

Para o cálculo do tempo de computação foi usada a função *cputime* do *Software Matlab* (109), com o objetivo de minimizar o efeito de variáveis influentes no processamento, uma vez que esta função mede o tempo dedicado pela unidade central de processamento

(CPU) numa determinada tarefa. Também foram tomados cuidados de modo a realizar todas as simulações em ambientes, tanto físicos como virtuais, o mais similares possível. Como o tempo de computação varia minimamente a cada nova simulação, foram feitas várias simulações e foi usado o valor médio com uma quantificação da incerteza realizada para um nível de confiança não inferior a 95 %.

Todas as simulações foram realizadas com uma unidade central de processamento (CPU) Intel® Core™ 2 Duo CPU P9400 @ 2.40GHz ×2 e uma memória interna de 3.8 GB.

4.1.2 Diagramas de Voronoi

Os diagramas de Voronoi são uma das mais importantes estruturas da geometria computacional. Basicamente um diagrama de Voronoi fornece informação sobre vizinhança e prioridade de pontos. Sendo $P = \{p_1, p_2, ..., p_n\}$ um conjunto de pontos, a célula de Voronoi $V(pi)$ é o conjunto de pontos q que está mais próximo de p_i do que de qualquer outro ponto, sendo definida por:

$$V(pi) = \{q \mid |p_i q| < |p_j q|, \forall j \neq i\}. \tag{4.1}$$

Propriedades dos diagramas de Voronoi

As seguintes observações acerca da estrutura dos diagramas de Voronoi são essenciais para o entendimento do funcionamento dos mesmos.

- **Segmentos do diagrama de Voronoi:** Cada ponto de um segmento do diagrama de Voronoi está equidistante dos dois nós mais próximos de si. Um círculo que passe por esses três pontos não pode conter mais nenhum nó no seu interior;
- **Vértices do diagrama de Voronoi:** A interseção de três células de Voronoi forma um vértice, este está à mesma distância de qualquer um dos nós e o círculo formado por esses três nós tem como centro o vértice e não pode conter outro nó no seu interior;
- **Grau:** Assumindo que nenhuns quatro nós no diagrama formem um círculo, então os vértices têm grau três;
- **Tamanho:** O diagrama de Voronoi segue a fórmula de Euler. Para um diagrama com n nós, o número de vértices num diagrama de Voronoi é de $2n - 5$ e de segmentos $3n - 6$.

Implementação

Existem vários algoritmos que permitem construir um diagrama de Voronoi. Para um espaço M e um conjunto S de nós p:

- **Construção incremental:** Este algoritmo inicialmente estudado por Green e Sibson (110) baseia-se na ideia de construir o diagrama de Voronoi por *incremental insertion*, obter $V(S)$ a partir de $V(S\backslash\{p\})$ inserindo o ponto p, ou seja, considerando que o diagrama de Voronoi para $V(S\backslash\{p\})$ já é conhecido, adiciona-se o

ponto P e atualiza-se o diagrama com as operações necessárias. Como a região de p pode ter $n-1$ segmentos, para $n = |S|$, o tempo de execução será de $O(n^2)$. Outros autores tentaram diminuir o tempo de execução (111) (112);

- **Dividir para conquistar:** O primeiro algoritmo determinístico *worst-case optimal* de construção do diagrama de Voronoi foi apresentado por Shamos e Hoey (113). Neste algoritmo o conjunto de pontos S é dividido por uma linha nos subconjuntos L e R com tamanhos similares. Os diagramas $V(R)$ e $V(L)$ são construídos recursivamente e $V(S)$ é obtido juntando $V(R)$ e $V(L)$ pela linha de divisão. Se estas operações tiverem um tempo de execução de ordem $O(n)$, o tempo total de execução será da ordem $O(n \log n)$.

- **Sweep Line:** Bentley e Ottman (114) desenvolveram um algoritmo em que o varrimento de uma linha ao longo do plano permitia listar todas as interseções num conjunto de segmentos de reta.

  Como a implementação dos algoritmos apresentados anteriormente tem aspetos menos vantajosos (e.g. a construção incremental tem um tempo de execução superior e o algoritmo dividir para conquistar é complexo e de certo modo de implementação difícil), Steve Fortune (115) criou um algoritmo de construção de diagramas de Voronoi recorrendo à abordagem *sweep line*. É este o algoritmo que será apresentado de seguida.

4.1.3 Algoritmo de Fortune

O segredo de qualquer algoritmo do tipo *sweep line* consiste em considerar todos os eventos sucessores de forma eficiente. No caso do diagrama de Voronoi a dificuldade é acrescida uma vez que é necessário prever quando e onde o próximo evento irá ocorrer. Como está exemplificado na Figura 4.5 o vértice A é um evento antecipado, uma vez que é provocado pelo nó que está a sua direita.

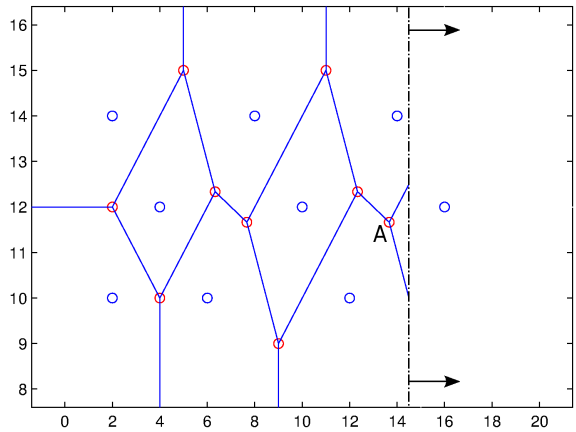


Figura 4.5: Linha de varrimento num diagrama de Voronoi com um evento antecipado.

Beach Line

De modo a evitar o problema falado anteriormente, só pode ser construída a parte do diagrama que não pode ser afetada por nenhum nó à direita da linha de varrimento. Para o conseguir, cria-se uma linha formada por um conjunto de pontos que estão equidistantes do nó mais próximo e da linha de varrimento, conjunto de pontos esse chamado *Beach Line*, geometricamente traduzido por uma parábola.

O algoritmo de Fortune consiste no crescimento da *Beach Line* à medida que a linha de varrimento se desloca para a direita. Assim, a evolução da interseção das varias parábolas cria os segmentos do diagrama de Voronoi.

Estado da linha de varrimento

O algoritmo avança a linha de varrimento de nó em nó ao longo do xx e regista a estrutura topológica da *Beach line* de cima para baixo, pelos nós que dão origem a cada uma das parábolas.

Eventos

Existem dois tipos de eventos:

- **Site events:** Quando a linha de varrimento passa por um novo nó, um novo arco é inserido na *Beach line*;
- **Vertex event:** Quando a linha de varrimento atinge o ponto mais afastado de um círculo, um arco é eliminado da *Beach line* e um novo vértice do diagrama é criado.

O algoritmo consiste em processar estes dois tipos de eventos, avançando a linha de varrimento até ao ponto mais afastado do último círculo, registando sempre os vértices e os segmentos de ligação dos vértices à medida que estes vão sendo processados.

Site events

Quando a linha de varrimento toca num novo nó, o arco parabólico associado a esse nó degenera numa linha reta que interceta a *Beach line*. Assim que a linha de varrimento avança, este novo arco é introduzido na *Beach line* dividindo o arco que foi intercetado.

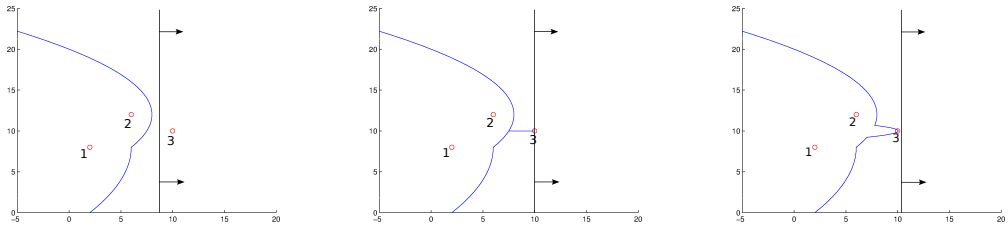
Como é mostrado na Figura 4.6a a sequencia dos arcos da *Beach line* é:

$$\langle 2; 1 \rangle$$

A *Beach line* avança para o nó 3, e é traçada uma reta horizontal que interceta o arco formado pelo nó 2 (Figura 4.6b). O arco 2 é dividido nos arcos $2'$ e $2''$ para introduzir o arco 3, formando a seguinte sequência (Figura 4.6c):

$$\langle 2'; 3; 2''; 1 \rangle$$

É de notar que os arcos $2'$, 3 e $2''$ não podem gerar um evento pois os arcos $2'$ e $2''$ provêm do mesmo nó, sendo necessários três nós distintos. Também é de notar que esta é a única forma de introduzir arcos na *Beach line*.



(a) (b) (c)

Figura 4.6: *Site events.*

Vertex events

Contrariamente aos *Site events*, que são conhecidos à priori, os Vertex events são gerados dinamicamente à medida que o algoritmo vai processando os nós.

Considerando três nós distintos, cujos arcos são vizinhos na *Beach line*, pode-se passar um círculo por esses três nós. Se este círculo ultrapassar a linha de varrimento e se no seu interior não houver nenhum nó, está-se na presença de um evento válido. Este evento será processado quando a linha de varrimento for tangente ao ponto mais à direita do círculo e se a estrutura morfológica da *Beach line*, no que diz respeito aos arcos que deram origem ao evento, não tiver sofrido alterações.

No instante em que a linha de varrimento é tangente ao ponto mais afastado do círculo, o contributo de um dos arcos para a *Beach line* é nulo, logo, deve ser removido da *Beach line*.

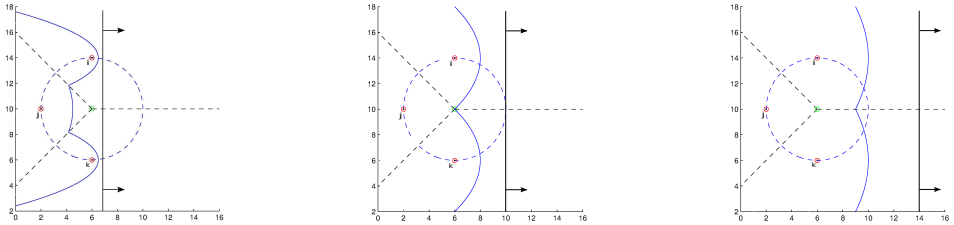
Sejam i, j e k os nós que provocaram este evento (Figura 4.7a), a *Beach line* tem a seguinte estrutura:

$$\langle i; j; k \rangle$$

Como não existe nenhum nó no interior do círculo, e a *Beach line* permaneceu sem alterações, quando a linha de varrimento for tangente ao ponto mais à direita do círculo (Figura 4.7b) o arco j é retirado da *Beach line* (Figura 4.7c). Ficando a seguinte estrutura para a *Beach line*:

$$\langle i; k \rangle$$

Todos os eventos relacionados com o arco j têm de ser eliminados, assim como o evento que acabou de ser processado, sendo necessário considerar novos eventos para a nova estrutura da *Beach line*.



(a) (b) (c)

Figura 4.7: *Vertex events.*

O algoritmo procede como todo os algoritmos *Sweep line*, os eventos estão ordenados pela sua abcissa. Os eventos vão sendo adicionados e removidos, e quando um é processado pode resultar em modificações ao diagrama de Voronoi e à *Beach line* e a criação ou remoção de eventos.

4.1.4 *NEM* com Algoritmo de Fortune

O *Natural Elements Method* tem como requisito a construção do diagrama de Voronoi e a consequente triangulação de Delaunay. Este requisito obrigou à criação de uma função capaz de gerar estas decomposições do espaço. A escolha do algoritmo de Fortune deve-se ao facto de combinar as seguintes vantagens:

- Computacionalmente ótimo ($O(N \log N)$ para N nós);
- Implementação relativamente simples por evitar passos complexos como os associados à sincronização e junção de dados (por exemplo o algoritmo de dividir para conquistar).

A implementação do algoritmo de Fortune foi realizada no *Software Matlab*, pelas mesmas razões que as explicadas anteriormente acerca da implementação da *kd-Tree*.

Concetualmente o algoritmo de Fortune é uma linha que passa pelo conjunto de nós, construindo o diagrama de Voronoi na retaguarda dessa linha. Esta técnica é usada noutros algoritmos, chamada de técnica de varrimento, pelo que os algoritmos que adotam esta técnica, são chamados de algoritmos de varrimento.

Existem duas estruturas de dados base para qualquer algoritmo de varrimento, a lista de eventos (*Event queue*) e a sequência da linha de varrimento (*sweep table*). A lista de eventos tem a informação ordenada acerca de todos os pontos por onde a linha de varrimento tem de passar e é atualizada dinamicamente. Enquanto, como já foi dito anteriormente, a sequência da linha de varrimento informa acerca do estado da *Beachline*, ou seja, como estão ordenados os arcos e quais são esses arcos, e esta também é atualizada dinamicamente.

As estruturas de dados usadas não foram construídas em específico para este algoritmo, sendo neste caso funções do *Software Matlab*. Esta abordagem abstrata traz o benefício de permitir o aumento de eficiência global do algoritmo, pela simples troca das estruturas usadas por estruturas mais evoluídas. Neste trabalho são usadas as seguintes estruturas:

Lista de Eventos Uma vez que os nós são previamente conhecidos, é possível criar um vetor ordenado com essa informação. Os restantes pontos, *Site Events* e *Vertex Events*, são colocados num outro vetor e, sempre que necessário, os vetores são assemblados, procedendo-se à ordenação deste novo vetor para determinar o próximo ponto a ser considerado pela linha de varrimento. De seguida este ponto é dado como obsoleto.

Sequência da Linha de Varrimento A sequência da linha de varrimento é construída com recurso à estrutura do *Software Matlab* lista ligada, uma vez que permite conetar os arcos da *Beachline* de forma simples, assim como saber de forma rápida e simples os arcos adjacentes e subjacentes a um determinado arco. A eliminação de um arco também é uma função existente e muito usada neste algoritmo, que

depois de eliminar o arco restabelece a ligação entre os arcos adjacentes e subjacentes ao arco eliminado. Esta estrutura de dados permite ainda a identificação inequívoca de cada arco, reconhecendo quando dois arcos provêm do mesmo nó.

O programa segue o algoritmo de Fortune como explicado anteriormente, é validado e armazenado um novo vértice e com ele, três semi-segmentos do diagrama de Voronoi, assim como os nós que deram origem ao vértice, nós esses que vão compor a triangulação de Delaunay. Após o varrimento de todos os pontos são processados os eventos remanescentes, e assim é concluído o Diagrama de Voronoi e a triangulação de Delaunay.

Para o *Natural Elements Method* é necessário conhecer muito bem a topologia do diagrama de Voronoi, ou seja, é necessário conhecer as células que o compõem, uma vez que até agora o programa só obteve os vértices e os semissegmentos do mesmo, estes têm de ser processados. Partindo do princípio que cada dois semi-segmentos no mesmo sentido devem ser ligados por um ponto que não é um vértice, consegue-se listar o dobro dos segmentos que ligam os vértices. Uma vez que o mesmo segmento é comum a duas células, e se essas células forem percorridas no mesmo sentido, os segmentos têm orientação contrária. Partindo desta ideia, basta percorrer os segmentos sempre no mesmo sentido. Ao voltar ao vértice inicial, o conjunto do segmentos percorridos é associado a uma célula. Aplicando esta ideia a todos os segmentos encontram-se todas as células. Para cada célula encontrada é feita uma pesquisa de modo a saber qual o nó no seu interior.

Com esta informação, acerca do diagrama de Voronoi, é possível partir para a construção do NEM. Para o conjunto de pontos inicial, é calculada a triangulação de *Delaunay* e os consequentes pontos de Gauss. O diagrama de Voronoi de segunda ordem é feito para o conjunto dos pontos iniciais com três pontos de Gauss. Para cada ponto de Gauss é feita uma pesquisa pelas células vizinhas e são calculadas as funções de forma não sibsonianas e as suas derivadas parciais. Sendo por fim aplicado o método de Galerkin para obter os resultados.

4.1.5 *Spatial Hashing*

As grelhas de dispersão são grelhas regulares cujo conteúdo é armazenado em tabelas de dispersão. O espaço de incorporação é discretizado em células e a cada célula é atribuído um índice. Os índices das células são processados por uma função de dispersão, resultando num índice da tabela de dispersão. Todos os dados associados a cada célula, como o número de pontos que essa célula contém, são armazenados numa entrada da tabela de dispersão. As grelhas de dispersão fornecem um rápido acesso à informação armazenada e têm uma manutenção muito acessível. O mapeamento destas grelhas numa comparação com grelhas habituais possui vantagens, uma vez que a grelha de dispersão está definida em todo o espaço (116).

Para construir uma grelha de dispersão, o espaço de pesquisa é dividido numa grelha finita e regular. Define-se uma grelha regular, alinhada com os eixos, com lado d e uma célula de canto na origem, para qualquer ponto $x = [x, y, z]$, formando um índice de tuplas $I = (i, j, k)$, que identifica unicamente a célula que contenha:

$$I(x) = ([\frac{x}{d}, \frac{y}{d}, \frac{z}{d}]), \tag{4.2}$$

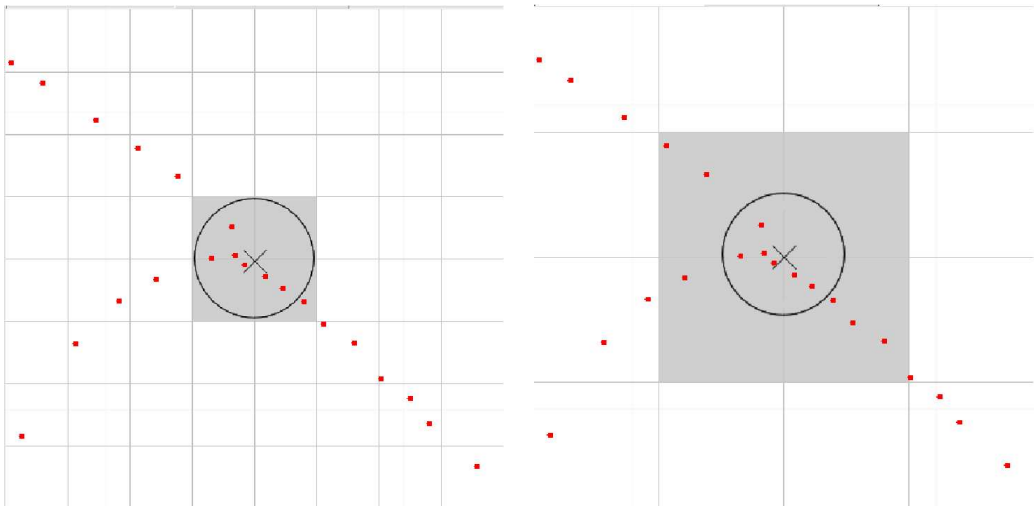
Também é necessária uma função que transforme os índices das células em entradas de uma tabela de dispersão. Uma função de dispersão para índices de tuplas como a equação 4.3 foi proposta por Teschner *et al.* (117)

$$H(i,j,k) = (ip_1 \; xor \; jp_2 \; xor \; kp_3) mod \; s, \tag{4.3}$$

Onde $p_{1,2,3}$ são números primos e s é o tamanho da tabela de dispersão. De modo a minimizar a probabilidade de duas ou mais células partilharem o mesmo índice (fenómeno conhecido por colisão), s deve ser um número primo ou primo com $p_{1,2,3}$. Funções de dispersão mais evoluídas podem retornar menos colisões, mas têm um custo computacional superior, enquanto que o custo computacional de uma colisão é baixo. As melhores performances são conseguidas com uma tabela de dispersão em que o tamanho seja um número primo (118).

Dado o espaço S, pontos de P são sequencialmente adicionados à grelha de dispersão e é-lhes atribuído um índice correspondente à entrada na tabela de dispersão. Adicionar, remover ou mover pontos numa grelha de dispersão tem um impacto pouco significativo no custo computacional. O pseudocódigo apresentado no algoritmo 3 resume a construção de uma grelha de dispersão.

Ao fazer uma pesquisa na tabela de dispersão, é encontrada a célula (i,j,k) que contém o ponto x pesquisado. Todas as células intercetadas por uma esfera de raio r centrada em x são identificadas e os seus pontos testados quanto à distância. Se $d = r$ um total de 3^k células têm de ser verificadas quanto à distância, enquanto que se $d = 2r$ só as 2^k células mais próximas de x são verificadas. O algoritmo 4 mostra um pseudocódigo para esta pesquisa.



(a) Pesquisa numa grelha de dispersão com $d = r$.

(b) Pesquisa numa grelha de dispersão com $d = 2r$.

Figura 4.8: Pesquisa numa grelha de dispersão.

Algorithm 3 Construção de uma grelha de dispersão.

Armazena um ponto pertencente a S com $S \subset \Re^k$ numa tabela de dispersão de tamanho s.

function HASH(($\mathbf{I}$))
  Data: on array of prime numbers $\mathbf{p}$
  $H = 0$
  for $i \in 0...k-1$ **do**
    $H \leftarrow H + \mathbf{p}[i]\mathbf{x}[i]$
  end for
  return H
end function
function MAKEHASHGRID(S, d)
  $table$ = Array (s)
  for all $x \in Sdo$ **do** ▷ Processar o índice da célula.
    for $i \in 0...k-1do$ **do**
      $\mathbf{I}[i] = [\mathbf{x}[i]/d]$
      $hash = Hash(\mathbf{I})$ ▷ Adicionar à tabela de dispersão.
      $table[hash].S \leftarrow table[hash].S \cup \mathbf{x}$
    end for
  end for
  return table
end function

Algorithm 4 Pesquisa numa grelha de dispersão.

Para uma grelha de dispersão de dimensão k com espaçamento $d = r$.

function HASHGRIDQUERY($table, \mathbf{x}$) ▷ Processar o índice e a direção da célula vizinha.
  for $i \in 0...k-1$ **do**
    $\mathbf{I}[i] = [\mathbf{x}[i]/d$
    if $\mathbf{x}[i] - \mathbf{I}[i]d < 2$ **then**
      $\mathbf{dI}[i] = -1$
    else
      $\mathbf{dI}[i] = 1$
    end if
  end for
  $C = 0$ ▷ Procurar os pontos nas células.
  for $\mathbf{c} = BinaryCount(0, 2^k - 1)$ **do**
    for $i \in o...k-1$ **do** ▷ Processar as células com índice modificado.
      $\mathbf{J}[i] = \mathbf{I}[i] + \mathbf{c}[i]\mathbf{dI}[i]$
      $C \leftarrow C \cup table[Hash(\mathbf{J})]S$
    end for
  end for
  return $\mathbf{x}_i \in C : \|\mathbf{x}_i - \mathbf{x}\| < r$
end function

Parte III

Resultados e Discussão

É feita uma apresentação dos problemas usados para testar o *NEM* e os algoritmos de otimização do *EFG*. São esclarecidas algumas condições de simulação. Apresentam-se os resultados obtidos para os problemas propostos. Compara-se o *EFG* com algoritmos de redução do custo computacional com o *EFG* original quanto à redução do tempo de computação e precisão dos resultados. Compara-se o *EFG* com o *NEM*.

Capítulo 5

Resultados Numéricos

5.1 Viga de Timoshenko

A viga de Timoshenko é um problema clássico de teste que é caracterizado por:

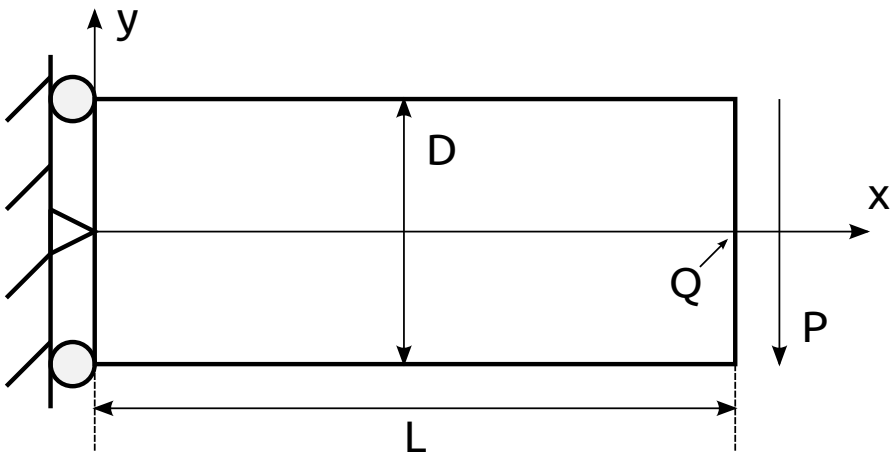


Figura 5.1: Viga de Timoshenko - esquema

Uma viga de comprimento L e largura D sujeita a um esforço parabólico de tração na extremidade livre, como ilustrado na Figura 5.1. Assumindo um estado plano de tensão para a viga, a solução analítica dada por Timoshenko e Goodier (119) é:

$$u_x = -\frac{Py}{6EI}\left[(6L - 3x)\,x + (2+\nu)\left(y^2 - \frac{D^2}{4}\right)\right] \tag{5.1a}$$

$$u_y = \frac{P}{6EI}\left[3\nu y^2\,(L-x) + (4+5\nu)\,\frac{D^2 x}{4} + (3L - x)\,x^2\right] \tag{5.1b}$$

Com o momento de inércia I dado por:

$$I = \frac{D^3}{12} \tag{5.2}$$

Nas simulações foram usadas as seguintes condições para a viga de Timoshenko;

Tabela 5.1: Condições usadas na simulação da viga de Timoshenko.

Campo	**Valor**
E	$3.0 \times 10^7 Pa$
ν	0.3
D	$12m$
L	$48m$
P	$1000N$

Para a resolução do problema da viga de Timoshenko foram usadas células de integração do tipo quadrilátero com 4 pontos de integração. A solução foi obtida recorrendo ao uso de um polinómio linear com uma função de peso do tipo *spline* cúbica e um $d_{max} = 3.5$.

Um exemplo da malha regular de pontos usada, está representada na Figura 5.2.

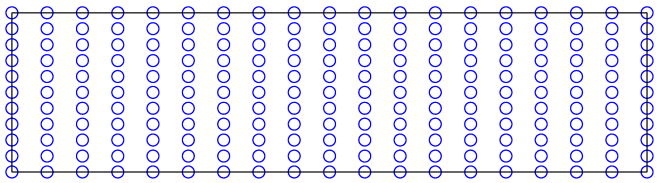


Figura 5.2: Discretização nodal uniforme para o modelo viga de Timoshenko - 209 nós (*EFG*).

Para efeitos de comparação foram usadas distribuições aleatórias com o mesmo número de nós que as distribuições uniformes, como as representadas nas Figuras 5.3, 5.4 e 5.5.

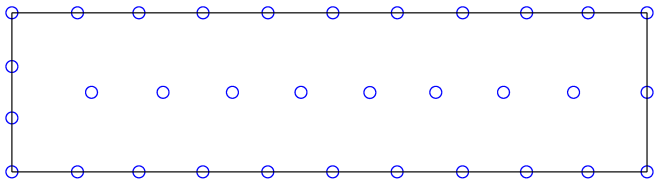


Figura 5.3: Discretização nodal aleatória para o modelo viga de Timoshenko - 33 nós (*EFG*).

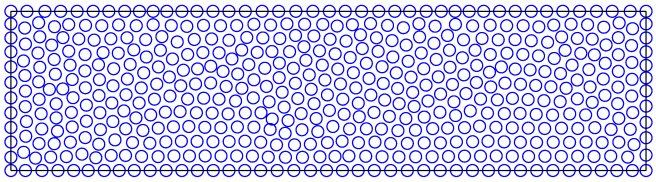


Figura 5.4: Discretização nodal aleatória para o modelo viga de Timoshenko - 506 nós (*EFG*).

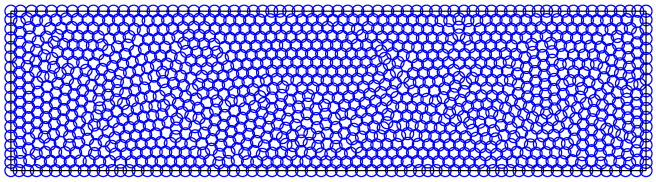


Figura 5.5: Discretização nodal aleatória para o modelo viga de Timoshenko - 1258 nós (*EFG*).

Para as simulações com o metodologia *NEM*, foram usados diagramas de *Voronoi* processados a partir de distribuições aleatórias, como é exemplo o diagrama apresentado na Figura 5.15.

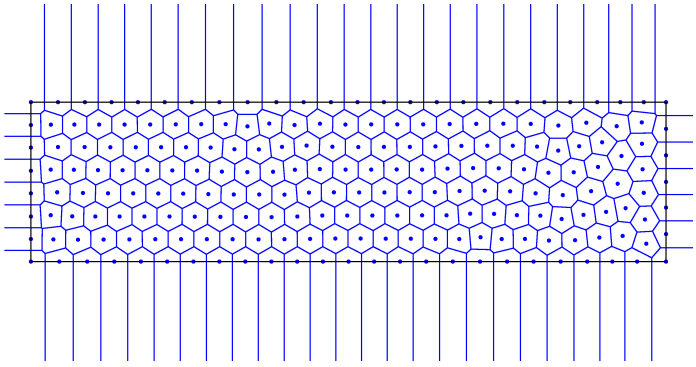


Figura 5.6: Discretização nodal para o modelo viga de Timoshenko - 209 nós (*NEM*).

O gráfico da Figura 5.7 compara o tempo de computação afeto à unidade central de processamento (*CPU*) em função do número de nós para a simulação do problema

descrito pelo método *EFG* com e sem *kd-Tree*. Como era esperado, a influência da *kd-Tree* na redução do custo computacional aumentou com o incremento do número de nós, mostrando para os casos realizados uma redução já algo significativa. Esta redução é maior para simulações com *kd-Tree*, uma vez que sem o uso desta estrutura de dados as diferenças temporais são pequenas.

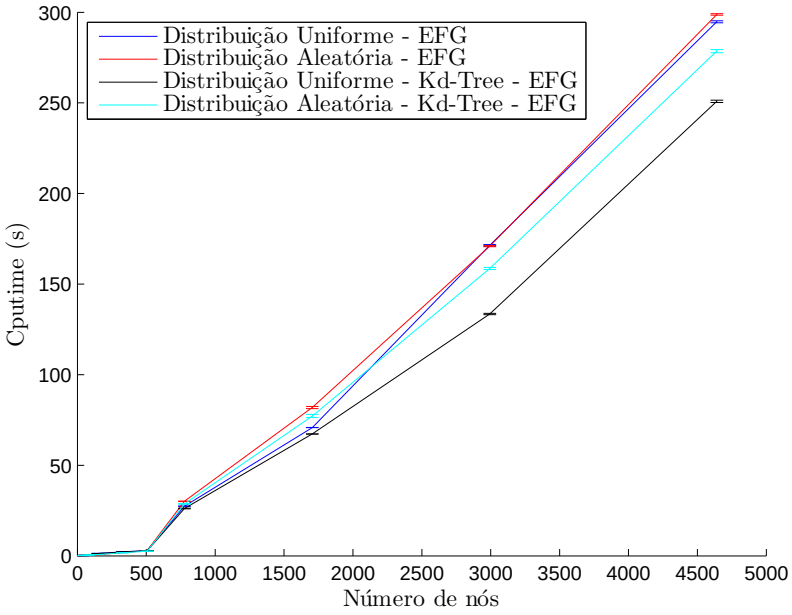


Figura 5.7: Comparativo do custo computacional - viga de Timoshenko.

Na Figura 5.9 encontra-se a comparação entre o cálculo do deslocamento vertical do ponto Q (Figura 5.1) pelo método *EFG* com e sem *kd-Tree* para distribuições nodais uniformes e aleatórias, pelo método *NEM* assim como a solução analítica. A Figura 5.8 é uma ilustração do deslocamento da simulação com o maior número de nós, recorrendo ao uso de um fator de 250 na força para representar o deslocamento ocorrido.

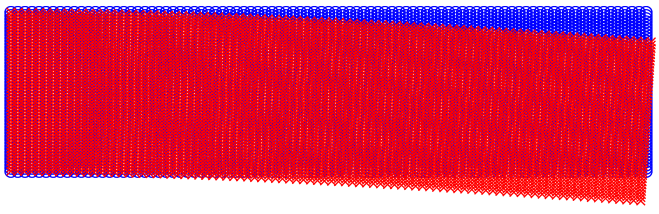


Figura 5.8: *EFG* com *kd-Tree* e distribuição uniforme - Resultado - 4641 nós (viga de Timoshenko) - $F \times 250$.

A Figura 5.9 demonstra uma rápida convergência entre os valores calculados e a

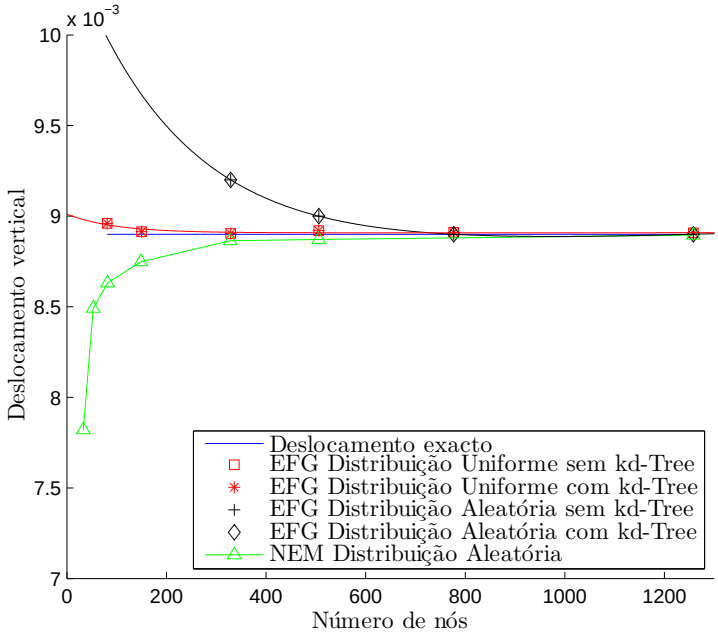


Figura 5.9: Comparativo do deslocamento - viga de Timoshenko.

solução, além de possibilitar a verificação da preservação dos resultados por parte da *kd-Tree*. Ainda assim, é de notar que o método *EFG* para distribuições uniformes é o mais rápido a convergir, sendo também este para distribuições nodais aleatórias o mais demorado. Nas distribuições aleatórias, o *NEM* para qualquer quantidade de nós obteve sempre soluções mais próximas da solução exata, convergindo também mais rapidamente, mostrando assim que consegue resultados melhores que o *EFG* para distribuições aleatórias.

5.2 Membrana de Cook

A membrana de Cook é um painel trapezoidal de espessura unitária. A extremidade esquerda está encastrada e a extremidade direita está sujeita a uma força distribuída F, como ilustrado na Figura 5.10. Este tipo de problema é usualmente aplicado com o objetivo de avaliar a convergência de um método computacional. Este é um conhecido problema do tipo membrana de Cook (120). A solução exata deste problema não é conhecida para as condições especificadas na tabela 5.2.

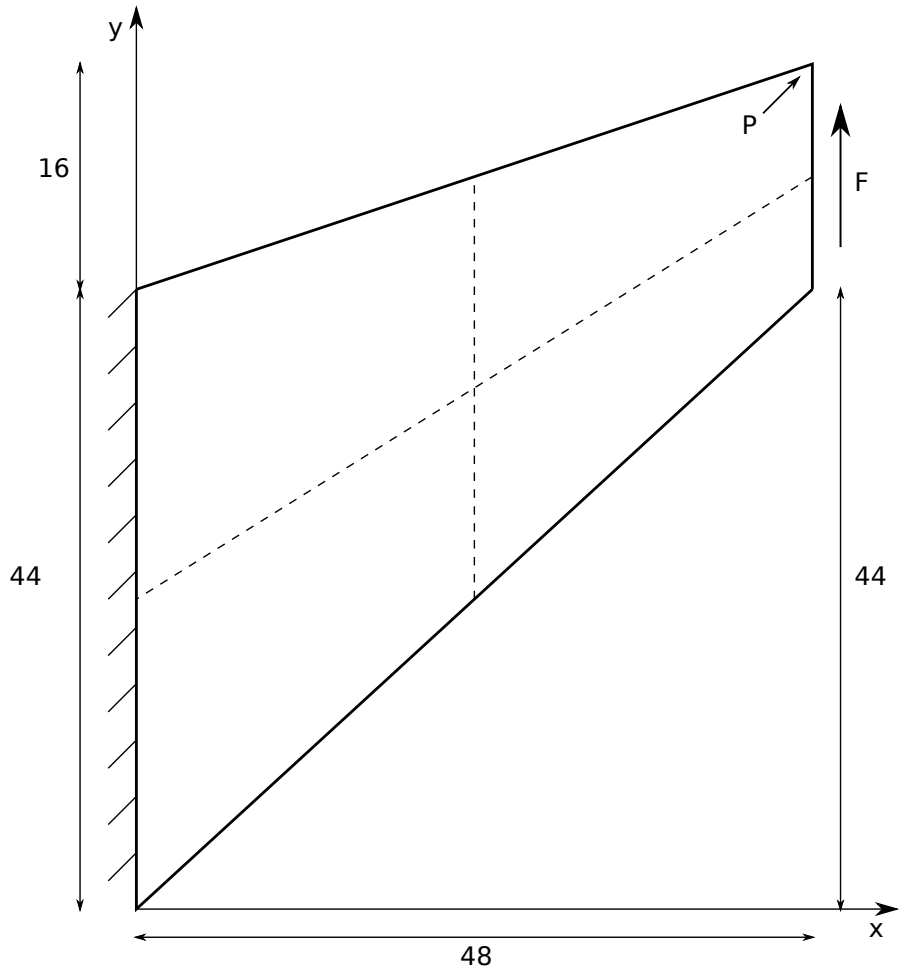


Figura 5.10: Membrana de Cook - esquema.

Tabela 5.2: Condições usadas na simulação da membrana de Cook.

Campo	**Valor**
E	$240,565$
ν	0.3

O problema da membrana de Cook foi resolvido contando com células de integração quadriláteras com um $d_{max} = 1.8$, uma função de peso do tipo *spline* cúbica, uma integração 2×2 e um polinómio de ordem 3 para um estado plano de deformação.

Foram usadas malhas regulares de nós como a ilustrada na Figura 5.11 nas simulações com a metodologia *EFG*.

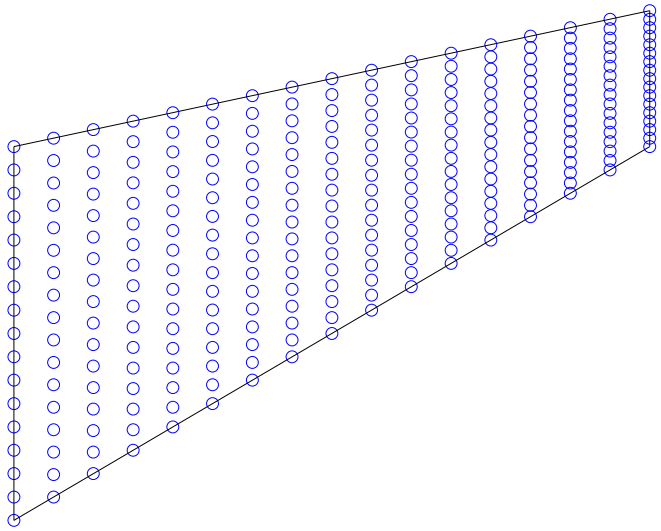


Figura 5.11: Discretização nodal uniforme para o modelo membrana de Cook - 289 nós.

De modo a avaliar a influência do uso de uma estrutura de dados do tipo *kd-Tree* em distribuições aleatórias, foram usadas distribuições como as representadas nas Figuras 5.12, 5.13 e 5.14.

A membrana de Cook serviu também de modelo de simulação para o *NEM*, para o qual foram construídos diagramas de Voronoi a partir de distribuições aleatórias, um exemplo desses diagramas é o apresentado na Figura 5.15.

A Figura 5.16 compara o custo computacional da simulação do problema da membrana de Cook com e sem estruturas de dados do tipo *kd-Tree* para distribuições aleatórias e uniformes de nós. Estes resultados mostram uma redução significativa do tempo de computação com uso da *kd-Tree*, crescente com o número de nós, sendo que sem *kd-Tree* os tempos de computação são muito próximos. É também de referir que o modelo com mais vantagem é com uso de *kd-Tree* e distribuição uniforme. A Figura 5.17 mostra a deformada da membrana de Cook.

Uma vez que este é um exemplo típico de avaliação de convergência, o deslocamento do ponto P (Figura 5.10) foi apurado para as simulações com e sem recurso a *kd-Tree* com o *EFG* e para a metodologia *NEM*, resultando no gráfico da Figura 5.18. Estes resultados não mostram quaisquer divergências entre os resultados usando *kd-Tree* e sem esta. Comparando as distribuições aleatórias, neste caso, o *EFG* atingiu a convergência com maior rapidez, mas para quantidades de nós inferiores, o *NEM* mostrou-se sempre mais próximo do valor de convergência. O *EFG* convergiu para ambas as distribuições

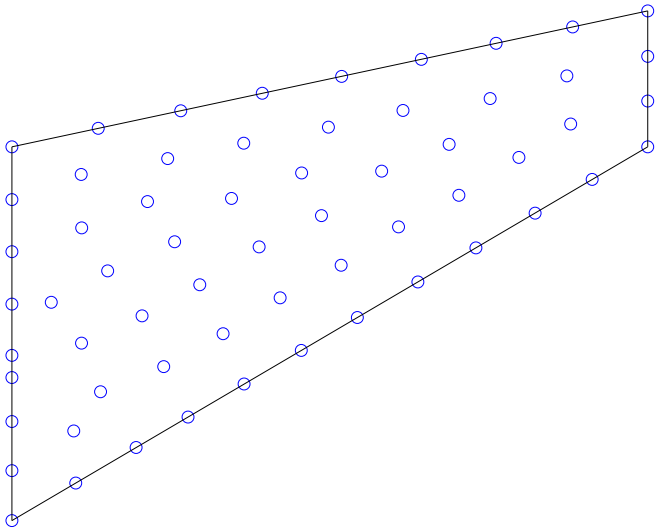


Figura 5.12: Discretização nodal aleatória para o modelo membrana de Cook - 61 nós (*EFG*).

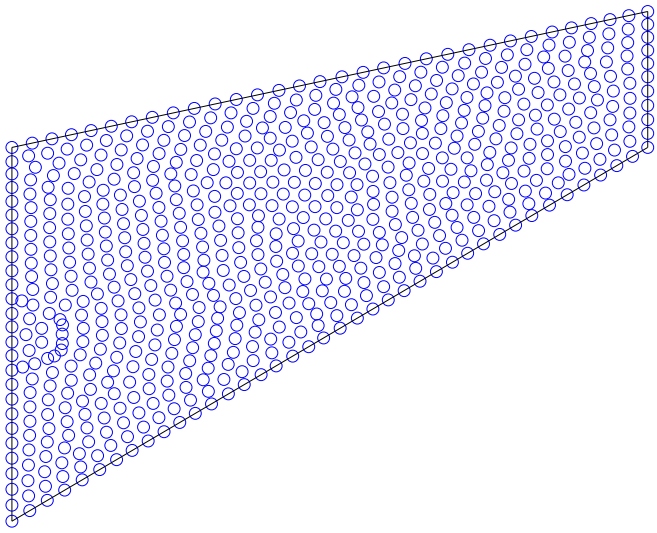


Figura 5.13: Discretização nodal aleatória para o modelo membrana de Cook - 684 nós (*EFG*).

muito rapidamente e sensivelmente ao mesmo tempo. Ambas as distribuições aleatórias convergiram para um valor algo superior ao da distribuição uniforme.

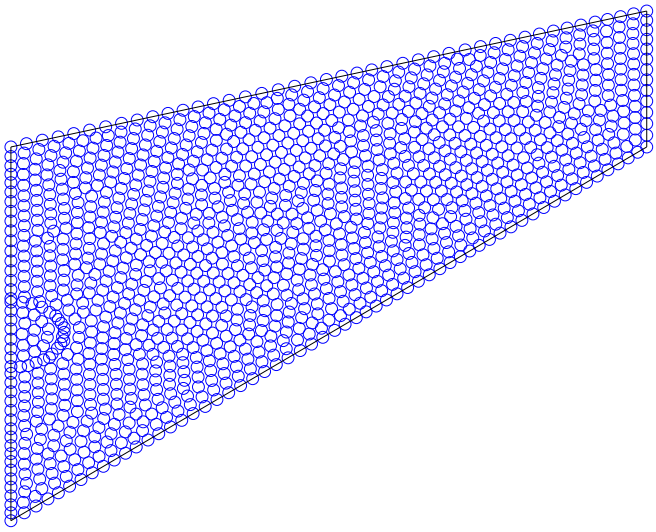


Figura 5.14: Discretização nodal aleatória para o modelo membrana de Cook - 1345 nós (*EFG*).

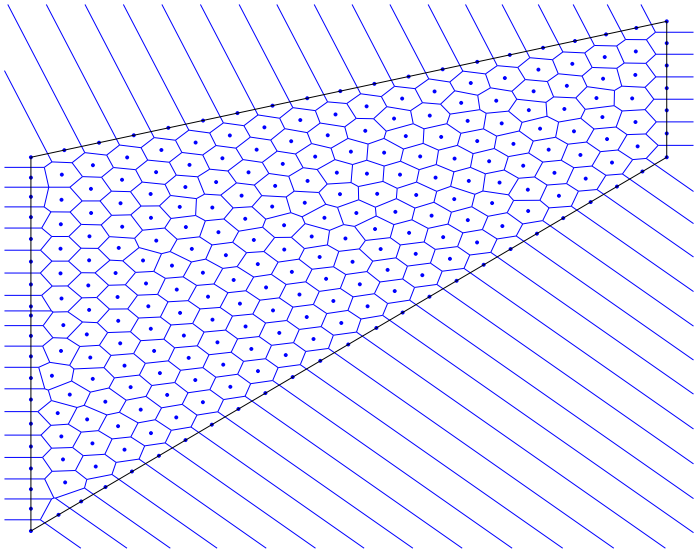


Figura 5.15: Discretização nodal para o modelo Membrana de Cook - 264 nós (*NEM*).

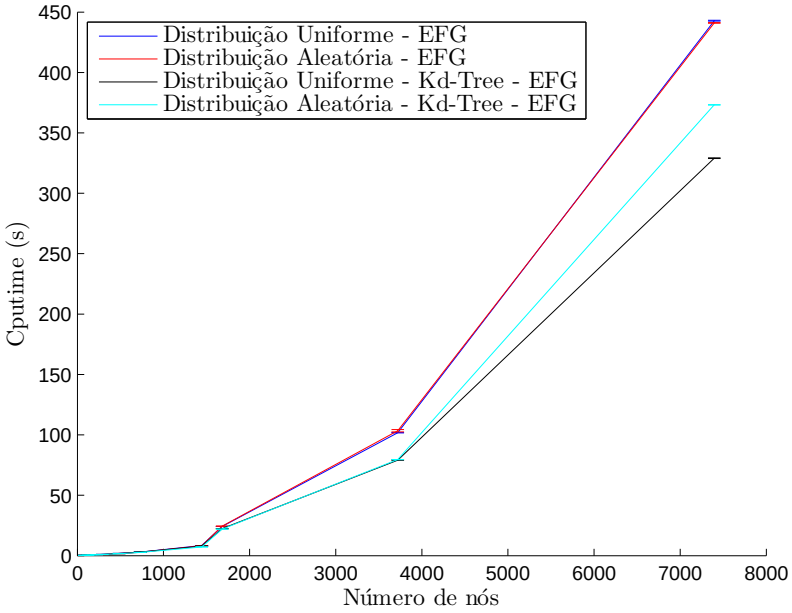


Figura 5.16: Comparativo do custo computacional - membrana de Cook.

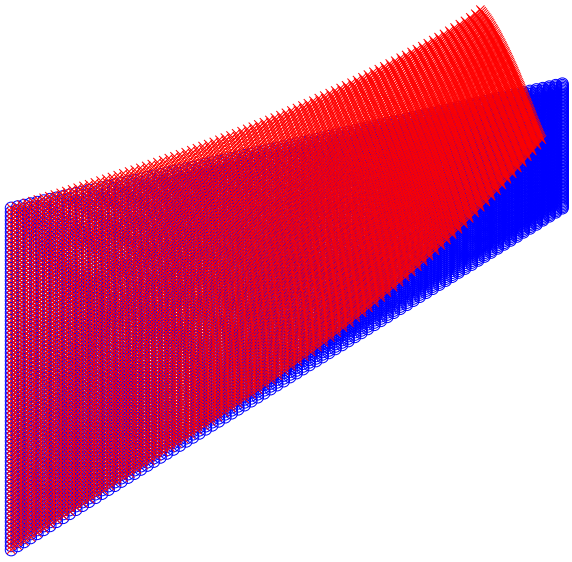


Figura 5.17: *EFG* com *kd-Tree* e distribuição uniforme - Resultado - 7396 nós (membrana de Cook).

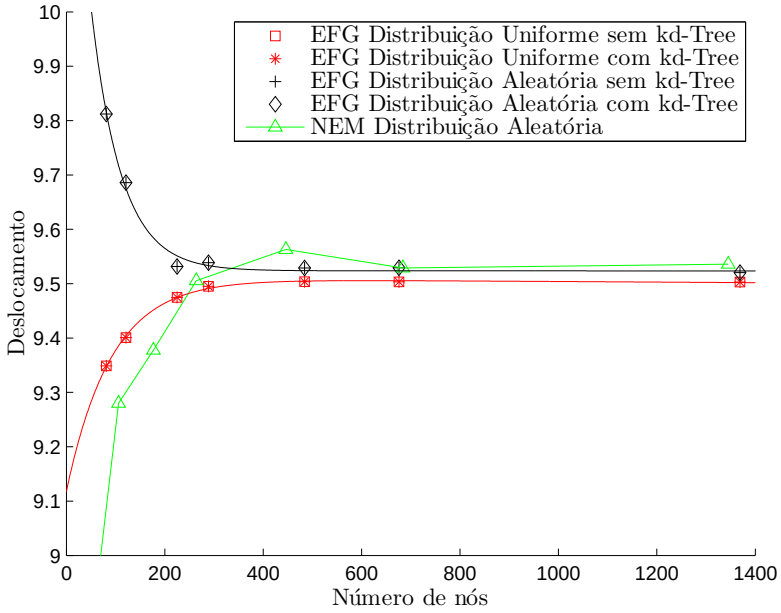
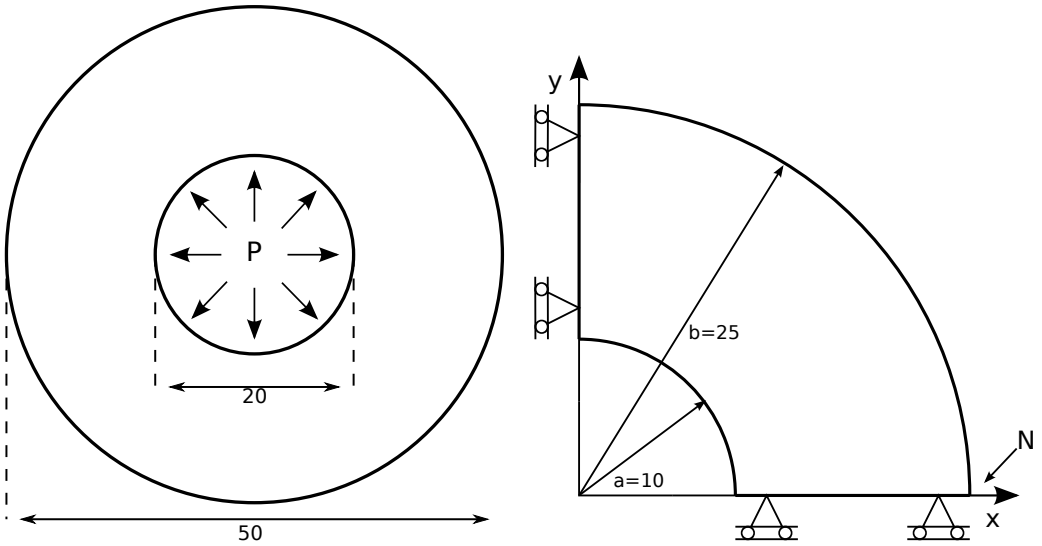


Figura 5.18: Comparativo do deslocamento - membrana de Cook.

5.3 Cilindro oco pressurizado internamente

Um cilindro sujeito a pressão interna é também simulado como problema de referência. Dada a simetria do problema só é necessário modelar 1/4 do cilindro. Como mostra a Figura 5.19, considerou-se um cilindro com um raio interno de $a = 10$, raio externo de $b = 25$ e uma pressão interna $p = 100$. Considerando um estado plano de deformação para as condições expressas na tabela 5.3, a solução analítica pode ser escrita da seguinte forma (119):

$$u_r = \frac{pa^2}{E\left(b^2 - a^2\right) r}\left[(1-\nu)\, r^2 + (1+\nu)\, b^2\right] \tag{5.3}$$



(a) Cilindro oco sob pressão interna.

(b) Esquematização das condições de simulação de um cilindro oco sob pressão interna.

Figura 5.19: Cilindro oco pressurizado internamente - esquema.

Tabela 5.3: Condições usadas na simulação do cilindro oco pressurizado internamente.

Campo	**Valor**
Go	8000
E	$2 \times Go \times (1 + nu)$
ν	0.3

Nesta simulação foram usadas células de integração com quatro nós, polinómio de ordem 3, integração 2×2 e $d_{max} = 2.5$.

Foram aplicadas distribuições nodais regulares ao *EFG* para avaliar o seu comportamento, como a distribuição ilustrada na Figura 5.20.

Foram também usadas distribuições nodais aleatórias como as exemplificadas nas Figuras 5.21, 5.22 e 5.23 para as condições referidas anteriormente.

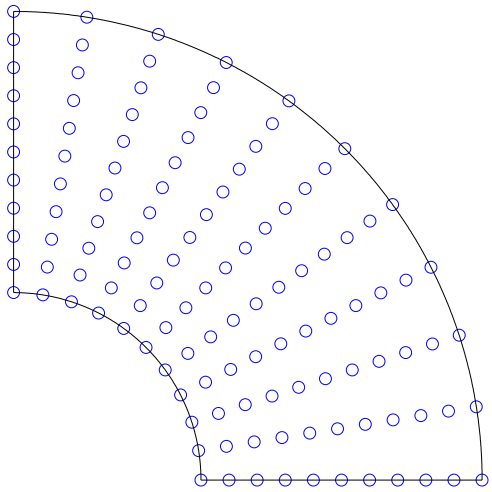


Figura 5.20: Discretização nodal uniforme para o modelo cilindro com pressão interna - 121 nós.

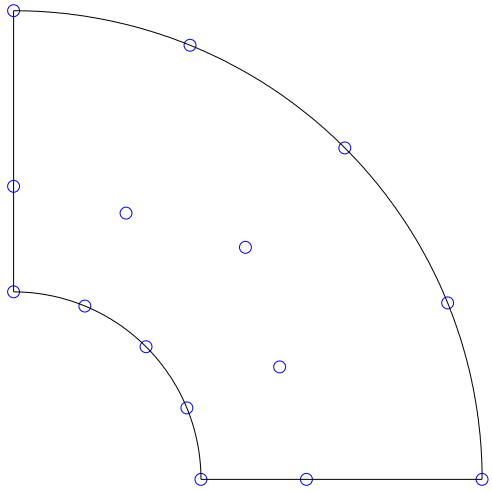


Figura 5.21: Discretização nodal aleatória para o modelo cilindro com pressão interna - 15 nós.

As simulações com o *NEM* usaram diagramas de *Voronoi* como o exemplificado na figura 5.24,

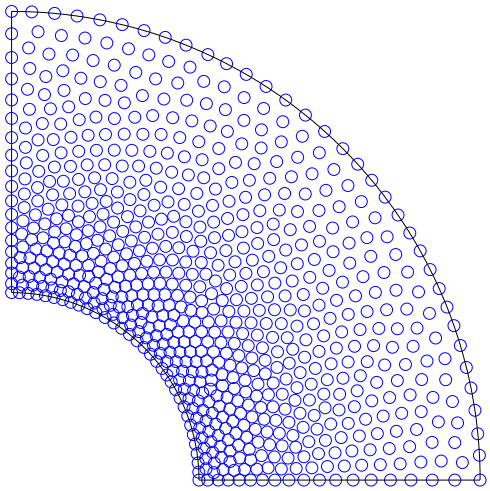


Figura 5.22: Discretização nodal aleatória para o modelo cilindro com pressão interna - 710 nós.

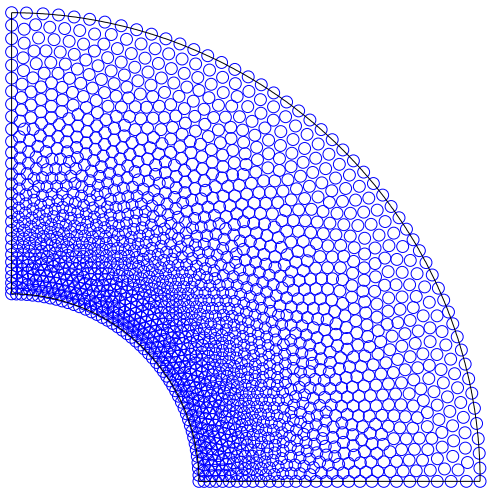


Figura 5.23: Discretização nodal aleatória para o modelo cilindro com pressão interna - 1760 nós.

Os efeitos provocados pelo uso da *Kd-tree*, avaliados pelo tempo de *CPU* requerido para completar a simulação para um número crescente de nós, estão representados na

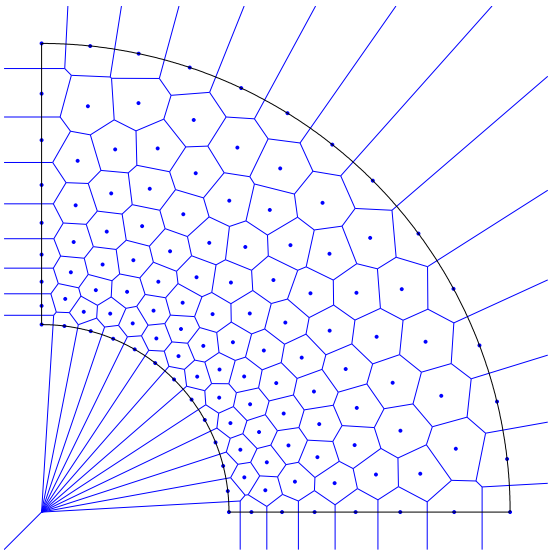


Figura 5.24: Discretização nodal para o modelo cilindro com pressão interna - 710 nós.

Figura 5.25. Vindo confirmar novamente a tendência observada nas simulações anteriores, ainda que para este caso e para números de nós mais baixos, o uso da *kd-Tree* seja prejudicial. Para o cilindro com pressão interna os tempos de simulação sem recorrer ao uso de *kd-Tree* são muito próximos, assim como os tempos das simulações que recorrem ao uso de *kd-Tree*.

A figura 5.26 mostra a deformação do cilindro para uma pressão interior maior que a usada nas simulações ($p \times 50$) para evidenciar essa deformação. Enquanto que o gráfico da Figura 5.27 compara o deslocamento obtido para um conjunto de nós crescente nas simulações com o *EFG* com e sem *kd-Tree* para distribuições uniformes e aleatórias, para o *NEM* com distribuições aleatórias, em comparação com o valor exato obtido pela equação 5.3.

O gráfico da Figura 5.27, no que diz respeito ao uso da *kd-Tree*, mostrou mais uma vez não ter qualquer influência sobre o resultado. Quanto à convergência o *EFG*, tanto com distribuições uniformes como aleatórias, convergiu prontamente e com rapidez sensivelmente semelhante para ambas as distribuições. Ainda assim, o *NEM* foi o mais rápido a convergir, tendo apresentado, mesmo em quantidades pequenas de nós, valores mais próximos do valor teórico. Os resultados obtidos com o *EFG* com distribuições aleatórias convergiram para um valor distanciado do valor teórico sensivelmente igual que as restantes simulações, ainda que em valor oposto.

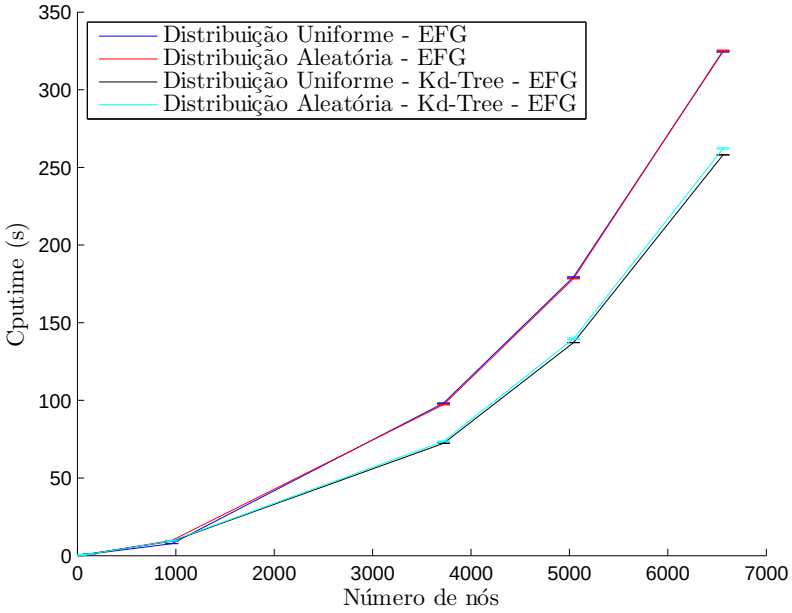


Figura 5.25: Comparativo do custo computacional.

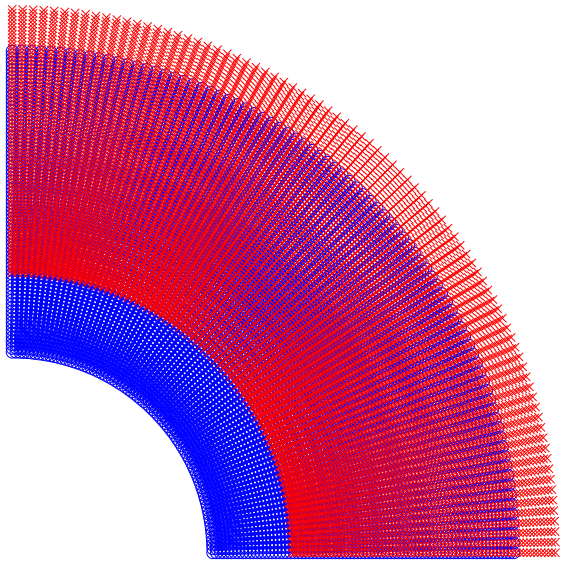


Figura 5.26: *EFG* com *kd-Tree* e distribuição uniforme - Resultado - 6561 nós (cilindro com pressão interna) - $p \times 50$.

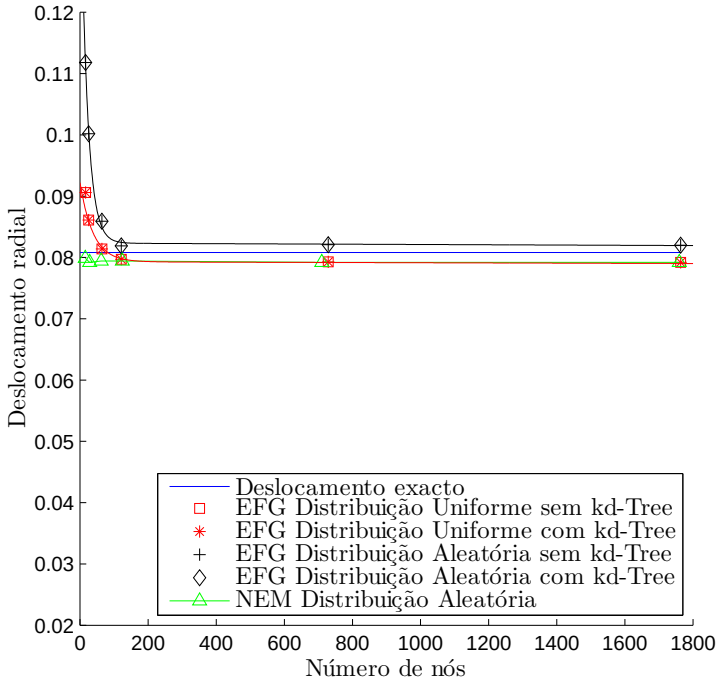


Figura 5.27: Comparativo do deslocamento.

Parte IV

Epílogo

Analisam-se os resultados obtidos sumariamente, sendo feita uma proposição final com algumas considerações acerca do trabalho. Por fim, são pospostos alguns caminhos a seguir na continuação deste trabalho.

Capítulo 6

Conclusões

As conclusões foram sendo expressas à medida que as simulações e os resultados foram apresentados, pretendendo-se aqui apenas evidenciar essas conclusões, apresentando as mais significativas.

A principal conclusão a tirar do trabalho apresentado, e a mais clara, é que o uso de estruturas de dados do tipo *kd-Tree* têm grande potencial na diminuição do custo computacional do método *EFG*, tendo sido neste caso aplicado uma função de procura *k-nearest neighbor searching* ao problema do domínio de influência e uma função de pesquisa por extensão ao problema da procura dos nós. A influência da *kd-Tree* foi mais notada com o aumento do número de nós usados.

Entretanto, esta implementação mostrou ser bastante rigorosa e robusta quanto aos resultados, em comparação com a implementação sem *kd-Tree*, que mesmo em simulações com mais nós que os necessários para encontrar convergência, devolveu os mesmos resultados.

Fazendo uma comparação para os três problemas quanto à diferença entre a distribuição nodal uniforme e aleatória. Consistentemente os melhores resultados foram obtidos com o uso de *kd-Tree* em distribuições uniformes, as simulações sem uso de *kd-Tree* mostraram valores muito próximos para ambas as distribuições. Esta conclusão deve-se ao facto de que o *EFG* sem *kd-Tree* recorre ao uso de *Brute Force*, ou seja, independentemente da localização dos nós, para o mesmo número de nós são feitas as mesmas operações com o mesmo custo computacional. O algoritmo da *kd-Tree* aplicado ao *EFG* faz uso de planos de divisão paralelos aos planos $xx = 0$, $yy = 0$ e $zz = 0$, e uma vez que as distribuições uniformes são efetuadas por divisões em x e y, à exceção do modelo do cilindro com pressão interna que é radial, consegue árvores melhor balanceadas e com menos folhas que nas distribuições aleatórias. Razão que explica também o porquê de no modelo do cilindro com pressão interna os ganhos do uso da *kd-Tree* para ambas as distribuições serem sensivelmente os mesmos. Durante as simulações foi ainda notado que o custo computacional do *NEM* é consideravelmente superior.

Em todas as simulações foi verificado que os modelos convergiram e para o problema da viga de *Timoshenko* e para o problema do cilindro com pressão interna, onde a solução teórica é conhecida, as diferenças entre os valores teóricos e os valores para os quais os modelos convergiram são relativamente pequenas. Comparando as distribuições aleatórias aplicadas ao *NEM* e ao *EFG*, notou-se que o *NEM* obteve sempre valores mais próximos do valor de convergência, diferenças mais evidentes em conjuntos de nós pequenos, sendo dissimuladas com o aumento do número de nós usados na simulação.

O *NEM* também foi a metodologia que mais rápido convergiu em dois dos problemas, mostrando-se assim, um método mais preciso para pequenos grupos de nós. Ainda é de notar que consistentemente o *EFG* com distribuição uniforme converge mais rapidamente que o *EFG* com distribuição aleatória.

No *NEM* a interpolação é baseada no diagrama de Voronoi, permitindo que o *NEM* seja sensível quanto a densidade e posição dos nós. As funções teste e tentativa também são construídas com base na interpolação por vizinhos naturais (n-n), enquanto que no *EFG* embora a função tentativa tenha consistência linear não é interpoladora, sendo necessário recorrer por exemplo aos multiplicadores de Lagrange para satisfazer as condições de fronteira. Estas diferenças podem explicar a vantagem que o *NEM* mostrou nos testes de convergência, uma vez que no *NEM* as células usadas para construir as funções de forma são geralmente as mesmas que as usadas na integração para evitar construir uma malha de suporte, enquanto que no *EFG* as células de integração e o domínio de suporte das funções de forma podem não coincidir.

Fazendo agora uma análise global ao trabalho desenvolvido, este foi de desenvolvimento, análise e teste de algoritmos no *Software Matlab* e comparação dos resultados dos mesmos com soluções teóricas sempre que possível. Estas simulações são fundamentais, uma vez que só assim se ganha a confiança exigida para que seja feita a inclusão destes algoritmos numa versão definitiva do software já desenvolvido.

Capítulo 7

Trabalho Futuro

Para que um método computacional seja plenamente aceite na comunidade científica e seja o padrão industrial tem de possuir algumas qualidades base, quanto aos métodos sem malha a performance será a que está menos desenvolvida. No caminho para esse fim, o custo computacional deve ser diminuído.

Os mais recentes avanços tecnológicos nas unidades de processamento central dos computadores (CPU) têm sido quanto ao número de núcleos de cada processador e não quanto à velocidade de relógio desse processador. De modo a aproveitar os novos processadores multinúcleo, é necessário que as aplicações tenham a capacidade de dividir o esforço computacional entre os vários núcleos. Como trabalho futuro é sugerida a transformação dos algoritmos em *Matlab* atualmente usados, em algoritmos capazes de utilizar todo o poder de processamento, assim como também deve ser explorado o potencial de processamento dos GPU, uma vez que este *software* tem vindo a disponibilizar ferramentas nesse sentido (121) (122).

O uso destas novas tecnologias foi já documentado, onde é evidente as vantagens, como na resolução de equações diferenciais parciais implementadas em sistemas de computação paralela para diferentes aplicações (123) (124) (125) (126). O uso de um sistema de computação paralela, com quatro processadores de dois núcleos, aplicado ao método *Element Free Galerkin* (127) revelou um tempo de computação 7,5 vezes inferior e 89% mais eficiente que num sistema de núcleo único para um problema com mais de 2000 equações. Também a aplicação a um problema de simulação 3D de propagação de uma fenda foi resolvido por um conjunto de 64 processadores com uma eficiência de 60% (128) (129) A implementação de computação paralela na aplicação do método *RKPM* a um problema em 3D apresentou uma eficiência de 60% num *cluster* de 64 processadores. Fica assim mostrado que o uso de computação paralela nos métodos sem malha pode diminuir significativamente o tempo de computação.

Existem também já trabalhos que concluem acerca das vantagens do uso de *GPU's* como é o caso do típico problema *N-body* (130), que usando como ambiente de processamento uma *NVIDIA CUDA (Compute Unified Device Architecture)*, foi atingida uma performance acima de 200 GFlops com um único GPU (131) (132) (133). Vários investigadores tentaram usar esta tecnologia para diminuir o tempo de computação, como é o caso de Schive *et al.* que usou 32 GPUs para acelerar o cálculo direto de iterações gravitacionais (134) e de Stone *el at.* que recorreu ao uso de 6 GPUs para o cálculo do método *particle mesh Ewald (PME)* (135) num problema de dinâmica molecular (136). Stock e Gharakhani implementaram um algoritmo para diminuir o tempo de processamento de

um problema usando o *Vortex method* (137), com o qual conseguiram diminuir em 17 vezes o tempo do cálculo direto, enquanto que com o uso de um GPU o cálculo direto seria 127 vezes mais rápido que o cálculo direto com CPU, para $N = 500000$ partículas. Similarmente, Gumerov & Duraiswami calcularam iterações de *Coulomb* usando o algoritmo *Fast Multipole Method (FMM)*, aumentando a velocidade do cálculo 72 vezes, quando o cálculo direto num GPU seria 855 mais rápido, para $N = 1048576$ partículas (138).

Neste trabalho foram usadas distribuições não uniformes de nós, estas foram realizadas dividindo o domínio em subespaços e distribuindo aleatoriamente um nó em cada subespaço para garantir uma distribuição balanceada. De modo a tornar a distribuição de nós mais flexível deve-se implementar soluções como a abordagem probabilística (139).

Existem outros processos que influenciam bastante a performance dos métodos sem malha, dos mais importantes é o cálculo das funções de forma e suas derivadas, uma vez quem envolvem cálculo matricial como a multiplicação e inversão de matrizes. Assim propõe-se encontrar alternativas para este tipo de cálculos, que usualmente é feito de forma direta, implementando soluções, por exemplo, com base na já implementada no MLS (140).

Bibliografia

[1] Courant R. Variational methods for the solution of problems of equilibrium and vibrations. Bulletin of American Mathematical Society. 1943.

[2] Clough RW, Wilson E. Early finite element research at Berkeley. In: Presentation on the Fifth U.S. National Conference on Computational Mechanics; 4 a 6 de Agosto de 1999.

[3] Turner MJ, Clough RW, Martin RW, , Topp LJ. Stiffness and deflection analysis of complex structures. Journal of the Aeronautical Sciences. 1956.

[4] Zheng Y. Enabling technologies for large-scale multidisciplinary simulations and semi-plenary speech. In: 9th World Congress on Computational Mechanics and 4th Asian Pacific Congress on Computational Mechanics; 2010.

[5] Idelsohn SR, Onate E, Calvo N, , Pin FD. The meshless finite element method. International Journal for Numerical Methods in Engineering. 2003.

[6] Remacle JF, Geuzaine C, Compere G, , Marchandise E. High-quality surface remeshing using harmonic maps. International Journal for Numerical Methods in Engineering. 2010.

[7] Lucy LB. Numerical approach to testing of fission hypothesis. Astronomical Journal. 1977.

[8] Gingold RA, Monaghan JJ. Smoothed particle hydrodynamics - theory and application to non-spherical stars. Monthly Notices of the Royal Astronomical Society. 1977.

[9] Monaghan JJ. Why particle methods work. SIAM Journal on Scientific and Statistical Computing. 1982.

[10] Gingold RA, Monaghan JJ. Kernel estimates as a basis for general particle methods in hydrodynamics. Journal of Computational Physics. 1982.

[11] Monaghan JJ, Gingold RA. Shock simulation by the paticle method (SPH). Journal of Computational Physics. 1983.

[12] Johnson GR, Beissel SR. Normalized smoothing functions for SPH impact computations. nternational Journal for Numerical Methods in Engi- neering. 1996.

[13] Federrath C, Banerjee R, Clark PC, , Klessen RS. Modeling collapse and accretion in turbulent gas clouds: implementation and comparison of sink particles in AMR and SPH. Astrophysical Journal. 2010.

[14] Bonet J, Kulasegaram S. Correction and stabilization of smooth particle hydrodynamics methods with applications in metal forming simulations. International Journal for Numerical Methods in Engineering. 2000.

[15] Swegle JW, Attaway SW. On the feasibility of using smoothed hydrodynamics paticle methods in hydrodynamics. Computaional Mechanics. 1995.

[16] Swegle JW, Hicks DL, , Attaway SW. Smoothed particle hydrodynamics stability analysis. Journal of Computational Physics. 1995.

[17] Lancaster P, Salkauskas K. Surfaces generated by moving least-squares methods. Mathematics of Computation. 1981.

[18] Nayroles B, Touzot G, , Villon P. The diffuse elements method. Comptes Rendus de LAcademie des Sciences Serie II. 1991.

[19] Nayroles B, Touzot G, , Villon P. Generalizing the finite element method: diffuse approximation and diffuse elements. Computational Mechanics. 1992.

[20] Belytschko T, Lu YY, , Gu L. Element-free Galerkin methods. International Journal for Numerical Methods in Engineering. 1994.

[21] Liu WK, Jun S, Zhang YF. Reproducing kernel particle methods. International Journal for Numerical Methods in Fluids. 1995.

[22] Onate E, Idelsohn S, Zienkiewicz OC, , Taylor RL. A finite point method in computational mechanics. applications to convective transport and fluid flow. International Journal for Numerical Methods in Engineering. 1996.

[23] Melenk M, Babuska I. The partition of unity finite element method: Basic theory and applications. Computer Methods in Applied Mechanics and Engineering. 1996.

[24] Zhu TL, Zhang JD, Atluri SN. A meshless numerical method based on the local boundary integral equation (LBIE) to solve linear and non-linear boundary value problems. Engineering Analysis With Boundary Elements. 1999.

[25] Atluri SN, Kim HG, , Cho JY. A critical assessment of the truly meshless local Petrov-Galerkin (MLPG) and Local Boundary Integral Equation (LBIE) methods. Computational Mechanics. 1999.

[26] Atluri SN, Zhu T. A new meshless local Petrov-Galerkin (MLPG) approach in computational mechanics. Computational Mechanics. 1998.

[27] Braun J, Sambridge M. A numerical method for solving partial differential equations on highly irregular evolving grids. Nature. 1995.

[28] Sibson R. A vector identity for the dirichlet tesselation. Mathematical Proceedings of the Cambridge Philosophical Society. 1980.

[29] Sukumar N, Moran B, Semenov AY, Belikov VV. Natural neighbour Galerkin methods. International Journal for Numerical Methods in Engineering. 2001.

[30] Liu GR, Gu YT. A point interpolation method for two-dimensional solids. International Journal for Numerical Methods in Engineering. 2001.

[31] Wang JG, Liu GR. A point interpolation meshless method based on radial basis functions. International Journal for Numerical Methods in Engineering.

[32] Duarte CA, Oden JT. H-p clouds an h-p meshless method. Numerical methods for Partial Diffrential Equations. 1996.

[33] Belytschko T, Organ D, Krongauz Y. A coupled finite element - element - free Galerkin method. Computational Mechanics. 1995.

[34] Mendez SF, Diez P, Huerta A. Convergence of finite elements enriched with meshless methods. Numerische Mathematik. 2003.

[35] Mendez SF, Huerta A. Imposing essential boundary conditions in mesh-free methods. Computer Methods in Applied Mechanics and Engineering. 2004.

[36] Huerta A, Fernandez-Mendez S. Enrichment and coupling of the finite element and meshless methods. International Journal for Numerical Methods in Engineering.

[37] Huerta A, Fernandez-Mendez S, Liu WK. A comparison of two formulations to blend finite elements and mesh-free methods. Computer Methods in Applied Mechanics and Engineering. 2004.

[38] Idelsohn SR, Onate E, Pin FD. The particle finite element method: a powerful tool to solve incompressible flows with free surfaces and breaking waves. International Journal for Numerical Methods in Engineering. 2004.

[39] Rabczuk T, Xiao SP, Sauer M. Coupling of meshfree methods with finite elements: Basic concepts and test results. Communications in Numerical Methods in Engineering. 2006.

[40] Trobec R, Sterk M, , Robic B. Computational complexity and parallelization of the meshless local Petrov-Galerkin method. Computers & Structures. 2009.

[41] Idelsohn SR, Onate E. To mesh or not to mesh. That is the question... Computer Methods in Applied Mechanics and Engineering. 2006.

[42] Lu Y, Belytschko T, Gu L. A new implementation of the element-free Galerkin method. Computational Methods in Applied Mechanics and Engineering. 1994.

[43] Wagner G, Liu W. Application of essential boundary conditions in mesh-free methods: a corrected collocation method. International Journal for Numerical Methods in Engineering. 2000.

[44] Breitkopf P, Rassineux A, Touzot G, Villon P. Explicit form and e cient computation of MLS shape functions and their derivatives. International Journal for Numerical Methods in Engineering. 2000.

[45] Danielson KT, Hao S, Liu WK, Uras RA, Li S. Parallel computation of meshless methods for explicit dynamic analysis. International Journal for Numerical Methods in Engineering. 2000.

[46] Schive HY, Chien CH, Wong SK, Tsai YC, Chiueh T. Graphic-card cluster for astrophysics (GraCCA). Performance tests New Astronomy. 2008.

[47] Libersky LD, Petscheck AG. Smooth particle hydrodynamics with strengh of materials. Advanced in the Free Lagrange Method and Lecture Notes in Physics. 1990.

[48] Randles P, Libersky LD. Smoothed particle hydrodynamics: some recent improvements and applications. Computer Methods in Applied Mechanics and Engineering. 1996.

[49] Stellingwerf RF, Wingate CA. Impact modelling with smoothed particle hydrodynamics. International Journal Impact Engineering. 1993.

[50] Hultman J, Pharayn A. Hierarchical and dissipative formation of elliptical galaxies: Is thermal instability the key mechanism? Hydrodynamical simulations including supernova feedback multi-phase gas and metal enrichment in CDM: Structure and dynamics of ellip-tical galaxies. Astronomy & Astrophysics. 1999.

[51] Monaghan JJ, Lattanzio JC. A simulation of the collapse and fragmentation of cooling molecular clouds. Astrophysical Journal. 1991.

[52] Berczik P, Kolesnik I. Smoothed particle hydrodynamics and its application to astrophysical problems. Kinematics and Physics of Celestial Bodies. 1993.

[53] Berczik P, Kolesnik I. Gasdynamical model of the triaxial protogalaxy collapse. Kinematics and Physics of Celestial Bodies. 1998.

[54] Berczik P. Modeling the star formation in galaxies using the chemo-dynamical sph code. Astronomy & Astrophysics. 2000.

[55] Lee W. Newtonian hydrodynamics of the coalescence of black holes with neutron stars ii. tidally locked binaries with a soft equation of state. Monthly Notices of the Royal Astronomical Society. 1998.

[56] Garcia-Senz D, Bravo E, Woosley SE. Single and multiple detonations in white dwarfs. Astronomy & Astrophysics. 1999.

[57] Phillips GJ, Monaghan JJ. MNRAS. 1985.

[58] Stellingwerf RF, Peterkin RE. Smooth Particle Magnetohydrodynamics. Albuquerque: Mission Ren Corp. 1990.

[59] Johnson GR, Stryk RA. SPH for high velocity impact computations. Comput Methods Appl Mech Engineering. 1996.

[60] Libersky LD, Petscheck AG. Smoothed Particle Hydrodynamics with strength oif materials. In: H, J F, Crowley, V WS, editors. Proceedings of the Next Free Lagrange Conference; 1991.

[61] Libersky LD, Petscheck AG. High strain Lagrangian hydrodynamics- a three-dimensional SPH code for dynamic material response. J Comput Phys. 1993.

[62] Monaghan JJ, Kocharyan A. SPH simulation of multi-phase flow. Computer Physics Communication. 1995.

[63] Chen JK, Beraun JE, Carney TC. A corrective Smoothed Particle Method for boundary value problems in heat conduction. Computer Methods in Applied Mechanics and Engineering. 1999.

[64] Zou S, Dalrymple RA. Sediment suspension simulation under oscillatory flow with SPH-SPS method. In Proc 30th International Conference on Coastal Engineering. 2006.

[65] Guzzo AD, Panizzo A. Application of SPH-NSWE to simulate landslide generated waves. SPHERIC and Second International Workshop. 2007.

[66] Dalrymple RA, Rogers B. Numerical modeling of water waves with the SPH method. Coastal Engineering. 2006.

[67] Crespo AJC, Gomez-Gesteira M, Dalrymple RA. 3D SPH Simulation of large waves mitigation with a dike. Journal of Hydraulic Research. 2007.

[68] Liu WK, Jun S, Li S, Jonathan A, Belytschko T. Reproducing kernel particle methods for structural dynamics. International Journal for Numerical Methods in Engineering. 1995.

[69] Liu GR. Mesh Free Methods: Moving Beyond the Finite Element Method. CRC Press; 2003.

[70] Gunther FC, Liu WK. Implementation of boundary conditions for meshless methods. Computaional Methods Applied to Mechanical Engineering. 1998.

[71] Lesoinne M, Kaila V. Meshless aeroelastic simulations of aircraft with large control surface deflections. In: AIAA 43rd Aerospace Sciences Meeting and Exhibit; 2005.

[72] Zhang LT, Wagner GJ, , Liu WK. A parallelized meshfree method with boundary enrichment for large-scale cfd. Journal of Computational Physics. 2002.

[73] Lancaster P, Salkauskas K. Curve and surface fitting: an introduction. Academic Press. 1986.

[74] Krysl P, Belytschko T. Analysis of thin plates using Element Free Galerkin Method. 1999.

[75] Krysl P, Belytschko T. Analysis of thin shells by element-free Galerkin method. International Journal Solids Struct. 1996.

[76] Belytschko T, Gu L, Lu Y. Fracture and crack growth by element-free Galerkin methods. Modelling and Simulation in Materials Science and Engineering. 1994.

[77] Belytschko T, Tabbara M. Dynamic fracture using element-free Galerkin methods. International Journal for Numerical Methods in Engineering. 1996.

[78] Lu YY, Belytschko T, Tabbara M. Element-free Galerkin methods for wave propagation and dynamic fracture. Computer Methods in Applied Mechanics and Engineering. 1995.

[79] Sukumar N, Moran B, Black T, Belytschko T. An element-free Galerkin method for three-dimensional fracture mechanics. Computational Mechanics. 1997.

[80] Xuan L, Zeng Z, Shanker B, Udpa L. Meshless element-free Galerkin method in NDT applications. In: 28th Annu. Rev. of Progress in Quantitative and Nondestructive Evaluation. USA; 2001.

[81] Cingoski V, Miyamoto N, Yamashita H. Element-free Galerkin method for electromagnetic field computations. IEEE Transactions on Magnetics. 1998.

[82] Singh IV, Sandeep K, Prakash R. The element-free Galerkin method in three-dimensional steady state heat conduction. International Journal of Computational Engineering Science. 2002.

[83] Singh IV, Sandeep K, Prakash R. Heat transfer analysis of two-dimensional fins using meshless element-free Galerkin method. Numerical Heat Transfer. 2003.

[84] Singh IV, Sandeep K, Prakash R. Meshless EFG method in transient heat conduction problems. International Journal of Heat and Technology. 2003.

[85] Belytschko T, Organ D, Krongauz Y. A coupled finite element-element-free Galerkin method. Computational Mechanics. 1995.

[86] Dolbow J, Belytschko T. An introduction to programming the meshless Element F reeGalerkin method. Archives of Computational Methods in Engineering. 1998. 5(3):207–241.

[87] Farin G. Curves and surfaces for CAGD. Kaufmann M, editor; 2002.

[88] Cueto E, Doblar M, Gracia L. Imposing essential boundary conditions in the Natural e Element Method by means of density-scaled α-shapes. International Journal for Numerical Methods in Engineering. 2000.

[89] Cueto E, Sukumar N, Calvo B, Martinez MA, Cegoino J, Doblar M. Overview and recent advances in Natural Neighbour Galerkin methods. Archives of Computational Methods in Engineering. 2003.

[90] Yvonnet J, Ryckelynck D, Lorong P, Chinesta F. A new extension of the Natural Element method for non convex and discontnuous problems: the Constrained Natural Element method. International Journal for Numerical Methods in Enginering. 2004.

[91] Traversoni L. Natural Neighbour Finite Elements. Computational Mechanics publications. 1994. Intl. Conference on Hydraulic Engineering Software. Hydrosoft Proceedings.

[92] Sukumar N, Moran B, e T Belytchko. The Natural Element Method in Solid Mechanics. International Journal for Numerical Methods in Engineering. 1998.

[93] Alfaro I, Yvonnet J, Cueto E, Chinesta F, Doblare M. Meshless methods with application to metal forming. Comput Methods Appl Mech Engrg. 2004.

[94] Gonzalez D, Cueto E, Chinesta F, , Doblare M. A natural element updated Lagrangian strategy for free-surface fluid dynamics. Journal of Computational Physics. 2007.

[95] Illoul L, Yvonnet J, Chinesta F, , Clenet S. Application of the natural-element method to model moving electromagnetic devices. IEEE Transactions on Magnetics. 2006.

[96] Watson DF. Contouring: A Guide to the Analysis and Display of Spatial Data. Oxford, editor. Pergamon Press; 1992.

[97] Sukumar N, Moran B. C1 natural neighbor interpolant for partial differential equations. Numerical Methods for Partial Diferential Equations. 1999.

[98] Belikov, V V, Ivanov, D V, Kontorovich, K V, et al. The non-Sibsonian interpolation: a new method of interpolation of the values of a function on an arbitrary set of points. Computational Mathematics and Mathematical Physics. 1997.

[99] Bentley JL, Friedman JH. Data structures for range searching. ACM Computing Surveys 11. 1979.

[100] Amet SH. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann; 2005.

[101] Bentley JL. Multidimensional Divide and Conquer. Communications of the ACM. 1980.

[102] Friedman JH, Bentley JL, Finkel RA. An Algorithm for Finding Best Matches in Logarithmic Expected Time. ACM Trans on Mathematical Software. 1977.

[103] Omohundro SM. Efficient Algorithms with Neural Network Behaviour. Journal of Complex Systems. 1987.

[104] Bentley JL. Multidimensional binary search tree used for associative searching. Communications of the ACM 18. 1975.

[105] Quinlan JR. Learning efficient classification procedures and their applications to chess endgames. R M, J C, T M, editors; 1983.

[106] Wess S, Althoff KD, Derwand G. Using k-d trees to improve the retrieval step in case-based reasoning. Selected papers from the European Workshop on Case Based Reasoning. 1994.

[107] Rya A, Ount M. Algorithms for fast vector quantization. In: Proceedings of the IEEE Data Compression Conference; 1993.

[108] Preparata FP, Shamos M. Computational Geometry. Springer-Verlag. 1985.

[109] MathWorks T, Inc. Elapsed CPU time - MATLAB; 2012. Acedido em 23/08/2012. Available from: `http://www.mathworks.com/help/techdoc/ref/cputime.html`.

[110] Green PJ, Sibson RR. Computing Dirichlet tessellations in the plane. The Computer Journal. 1978. 21:168–173.

[111] Ohya T, Iri M, Murota K. Improvements of the incremental method for the Voronoi diagram with computational comparison of various algorithms. Journal of the Operational Research Society. 1984. 24.

[112] Sugihara K, Iri M. Construction of the Voronoi diagram for one million generators in single-precision arithmetic. Proceedings of the IEEE. 1992.

[113] Shamos MI, Hoey D. Closest-point problems. IEEE Symposium on Foundations of Computer Science. 1975.

[114] Bentley JL, Ottmann TA. Algorithms for reporting and counting geometric intersections. IEEE Transactions on Computers. 1979.

[115] Fortune SJ. A sweepline algorithm for Voronoi diagrams. Algorithmica. 1987.

[116] Heckbert PS. Fast surface particle repulsion. CMU Computer Science. 1997.

[117] Teschner M, Heidelberger B, Muller M, Pomeranerts D, e M Gross. Optimized spatial hashing for collision detection of deformable objects. Proceedings of Vision and Modeling and Visualization VMV. 2003.

[118] Cormen TH, Leiserson CE, , Rivest RL. Introduction to Algorithms. MIT Press/McGraw-Hill. 1990.

[119] Timoshenko SP, Goodier JN. Theory of Elasticity. McGraw Hill; 1970.

[120] Cook R. Improved two-dimensional finite element. Journal Structural Division Proceedings of the American Society of Civil Engineers. 1974.

[121] MathWorks T, Inc. Parallel Computing Toolbox; 2012. Acedido em 19/09/2012. Available from: `http://www.mathworks.com/products/parallel-computing/`.

[122] MathWorks T, Inc. MATLAB GPU Computing Support for NVIDIA CUDA-Enabled GPUs; 2012. Acedido em 19/08/2012. Available from: `http://www.mathworks.com/discovery/matlab-gpu.html`.

[123] Golub G, Ortega JM. Scientific Computing - An Introduction with Parallel Computing. Academic Press Inc. 1993.

[124] Foster I. Designing and building parallel programs. Addison-Wesley. 1996.

[125] Urban B, Janezic D. Symplectic molecular dynamics simulations on specially designed parallel computers. Journal of Chemical Information and Modelling. 2005.

[126] Paik SH, Moon JJ, Kim SJ, , Lee M. Parallel performance of large scale impact simulations on linux cluster super computer. Computers and Structures. 2006.

[127] Singh V. Parallel implementation of the EFG method for heat transfer and fluid flow problems. Computational Mechanics. 2004.

[128] Rabczuk T, Bordas S, Zi G. A three-dimensional meshfree method for continuous crack initiation and nucleation and propagation in statics and dynamics. Computational Mechanics. 2007.

[129] Bordas S, Rabczuk T, Zi G. Three-dimensional crack initiation and propagation and branching and junction in non-linear materials by an extended meshfree method without asymptotic enrichment. Engineering Fracture Mechanics. 2007.

[130] von Hoerner S. Die numerische Integration des n-Korper-Problemes fur Sternhaufen. I. Zeitschrift fur Astrophysik. 1960.

[131] Nyland L, Harris M, Prins J. GPU Gems 3. Nguyen H, editor. Addison-Wesley; 2007.

[132] Belleman RG, Bedorf J, Zwart SFP. New Astronomy 13. 2008.

[133] Hamada T, Iitaka T. ArXiv Astrophysics e-prints Astro - ph. 2007.

[134] Schive HY, Chien CH, Wong SK, Tsai YC, Chiueh T. ArXiv Astrophysics e - prints astro - ph. 2007.

[135] Ewald P. Die Berechnung optischer und elektrostatischer Gitterpotentiale. Annals of Physics. 1921.

[136] Stone JE, Phillips JC, Freddolino PL, Hardy DJ, Trabuco LG, Schulten K. Accelerating molecular modeling applications with graphics processors. Journal of Computational Chemistry. 2007.

[137] Stock MJ, Gharakhani A. 3-D Vortex Simulation of Flow Over A Circular Disk At An Angle of Attack. In: 46th AIAA Aerospace Sciences Meeting and Exhibit; 2008.

[138] Gumerov NA, Duraiswami R, J. Fast multipole methods on graphics processors. Journal of Computational Physics. 2008.

[139] Du Q, Gunzburger M, Ju L. Meshfree and probabilistic determination of point sets and support regions for meshless computing. Computer methods in applied mechanics and engineering. 2002.

[140] Breitkopf P, Rassineux A, Touzot G, Villon P. Explicit form and efficient computation of mls shape functions and their derivatives. International Journal for Numerical Methods in Engineering. 2000.