DETECÇÃO EFICIENTE DE COLISÕES ENTRE ESTRUTURAS MARINHAS DELGADAS ATRAVÉS DO MÉTODO DE *BOUNDING BOXES* HIERÁRQUICOS

Pedro Henrique Rios Silveira

Eduardo Nobre Lages

pedro.silveira10@gmail.com

enl@lccv.ufal.br

Universidade Federal de Alagoas – UFAL

Centro de Tecnologia – CTEC

Laboratório de Computação Científica e Visualização – LCCV

Tabuleiro do Martins s/n, CEP 57072-970, Maceió/AL.

Resumo. *Neste trabalho, emprega-se o método de Bounding Boxes Hierárquicos para a detecção eficiente de colisões entre estruturas marítimas delgadas, notadamente risers e linhas de ancoragem. Apresenta-se uma implementação desenvolvida na linguagem de programação C++, empregando-se recursos de programação orientada a objetos e computação de alto desempenho, visando-se fácil integração com outros sistemas computacionais voltados à análise dinâmica de estruturas offshore. Detalhes da ferramenta são apresentados na forma de figuras, pseudocódigos e descrições de suas funções, bem como as funcionalidades, conceitos e recursos por trás do código. A implementação é validada através de testes comparativos com o método da força bruta (testes de distância elemento a elemento) para alguns arranjos selecionados, para os quais são utilizados históricos obtidos através de um software consolidado, em caráter de pós-processamento dos resultados. Tais testes mostraram-se bastante satisfatórios em termos de desempenho, complexidade e fidelidade das detecções.*

Palavras-chave: *Águas profundas, Alto desempenho, C++, Risers, Linhas de ancoragem.*

1 INTRODUÇÃO

Nos últimos anos, tem-se buscado alternativas para a exploração de petróleo em águas cada vez mais profundas, o que vem apresentando crescentes desafios tecnológicos. Uma das dificuldades encontradas neste contexto é a do projeto e análise de estruturas *offshore* delgadas. Entre estas, destacam-se os *risers* e as linhas de ancoragem (Chakrabarti, 2005; Rustad, 2007; He & Low, 2012).

Tal dificuldade deve-se ao fato de que a análise mecânica do comportamento dinâmico destas estruturas de linhas é extremamente complexa, notadamente devido à grande diversidade dos fenômenos associados à mesma, bem como à interação não linear entre eles. Entre tais fenômenos, destacam-se correntes, vórtices, a própria dinâmica do sistema flutuante onde essas se conectam, entre outros (Wu, 2003; Chakrabarti, 2005; Rustad, 2007; He & Low, 2012).

Este elevado grau de complexidade motivou o surgimento de diferentes sistemas computacionais, *softwares* e *frameworks* capazes de auxiliar o estudo e o projeto de sistemas de exploração e suas estruturas de linhas. Entre estes, destaca-se o DYNASIM (Nishimoto et al., 2001), um simulador utilizado com sucesso pela Petrobras.

Um importante problema encontrado dentro do âmbito das estruturas marinhas delgadas, principalmente no contexto de águas profundas, é a possibilidade da ocorrência de colisões (*clashing*) entre as linhas. Tais colisões, especialmente envolvendo *risers*, podem ser extremamente danosas, acarretando em problemas como deformações plásticas e danos por fadiga; desgaste da camada protetora empregada, acelerando processos de corrosão; entre outros (Wu, 2003; Bai & Bai, 2005; He & Low, 2012).

Com este fato em vista, a ocorrência de colisões entre linhas, especialmente envolvendo *risers*, é por vezes utilizada como critério de falha para o arranjo em questão. Quando as colisões são permitidas pelo projeto, guias de *design* de arranjos costumam exigir limitações em sua frequência, ou ainda a modelagem mecânica do seu efeito sobre as estruturas envolvidas (Chakrabarti, 2005; DNV, 2009).

É, portanto, de grande importância que os simuladores utilizados para auxiliar o projeto de arranjos de linhas para sistemas de exploração, principalmente em águas profundas, possuam ferramentas capazes de ao menos detectar colisões entre estas. Tais ferramentas de detecção e modelagem de efeitos de contato estão presentes em vários sistemas computacionais comerciais. Destaca-se, neste artigo, o *software* Orcaflex©, uma das principais referências no mercado (Orcina, 2015).

No momento, contudo, o DYNASIM não conta com nenhuma ferramenta destinada à detecção eficiente de colisões entre estruturas de linhas. Mais ainda, poucas referências são encontradas na literatura com relação a algoritmos eficientes de detecção de colisões adaptados ao contexto de estruturas *offshore* delgadas, nem tão pouco são encontradas bibliotecas de código aberto para tal função.

Visando preencher a lacuna entre a necessidade de implementações eficientes e particulares de detecção de colisão entre estruturas *offshore* delgadas e a falta de bibliotecas de código aberto documentadas que permitam fácil incorporação com sistemas de análise desenvolvidos por terceiros, este trabalho tem por objetivo apresentar uma estratégia eficiente de detecção, implementada na linguagem de programação *C++* (Stroustrup, 2014).

2 ESTRATÉGIA DE DETECÇÃO

É comum nos sistemas computacionais destinados à análise de estruturas *offshore* de linhas, onde uma dimensão é bastante predominante em relação às demais, que as mesmas sejam analisadas como elementos unidimensionais, onde as informações extraídas de seus históricos dinâmicos são fornecidas como posições de nós de uma malha unidimensional, referentes à posição geométrica de pontos no eixo central das mesmas.

Desse modo, dado o formato comum de apresentação dos resultados, a maneira mais direta de detectar colisões entre partes dessas estruturas seria calcular a distância entre cada segmento de reta definido por dois nós sucessivos das linhas analisadas. As linhas e sua geometria essencialmente curva estariam, portanto, representadas por diversos segmentos de reta, aqui nomeados elementos, de forma bastante simplificada.

Contudo, mesmo com esta simplificação, testes elemento a elemento tornam-se extremamente custosos à medida que o número de linhas analisadas, bem como o número de elementos que as discretizam, é suficientemente grande (Ericson, 2005; Ketchel & Larochelle, 2005; Berg et al., 2008). Tais elevados valores, esperados em arranjos de exploração para águas profundas, tornam o método de análise elemento a elemento (também conhecido como método da força bruta) impraticável em tal cenário.

Visando-se reduzir o tempo despendido em tais detecções, o número de checagens de distância elemento a elemento precisa ser minimizado, sendo necessário utilizar algum método que permita 'filtrar' colisões impossíveis de forma rápida e precisa. Um desses métodos, no qual se baseia a implementação deste trabalho, faz uso dos chamados *Bouding Boxes*, que consiste em envolver os objetos que se deseja analisar, frequentemente de geometria complexa, tais como estruturas *offshore* em movimentos dinâmicos não lineares, por uma superfície simples, no caso particular desta implementação um paralelepípedo com as faces paralelas aos planos cartesianos. Tais paralelepípedos orientados com os eixos coordenados são chamados de *AABBs* (*Axis Aligned Bounding Boxes*) (Ericson, 2005; Ketchel & Larochelle, 2005; Berg et al., 2008).

Mais especificamente, a implementação utiliza tais *boxes* de maneira hierárquica da seguinte forma, como ilustrado na Fig. 1: cada linha é inicialmente envolvida por um único *box*. Em seguida, tais *boxes* iniciais são testados entre si, de maneira a detectar superposições entre os mesmos. Não sendo identificadas tais superposições, elimina-se a necessidade de mais testes. Uma vez detectada a superposição de dois *boxes*, contudo, as linhas envolvidas são fracionadas e cada porção é envolvida por um novo *box*, agora menor, uma vez que precisa englobar menos nós. Tais *boxes* são então testados entre si, para checar se ainda existem colisões entre eles, e o processo de refinamento hierárquico prossegue até que se chegue a um par de *boxes* que envolvam apenas um único elemento cada (ou seja: dois nós cada), onde finalmente se testa a colisão.

A Fig. 1 a) corresponde a um caso onde os *boxes* iniciais de cada linha não se superpõem, descartando-se de imediato as colisões; b), por sua vez, corresponde a um caso onde é necessário realizar um refinamento, passando cada linha a ser discretizada por 2 *boxes*, no que as colisões foram então descartadas novamente; por fim, em c), o processo recai em um teste de distância elemento a elemento entre os dois últimos elementos das linhas (neste exemplo em particular, cada linha é definida por apenas 3 nós, ou seja, 2 elementos, apenas em caráter ilustrativo).

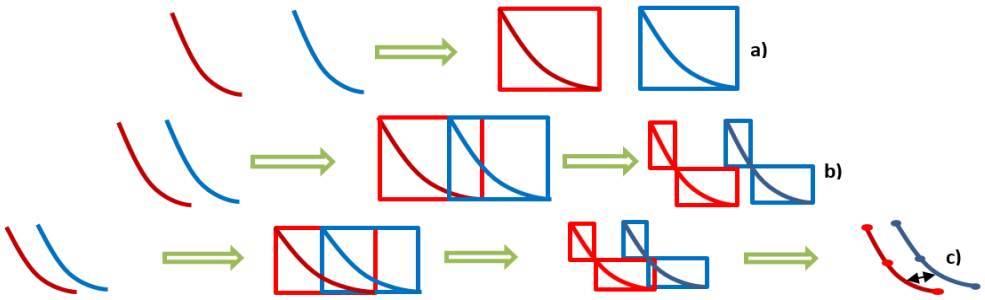

Figura 1. Método de *Bounding Boxes* Hierárquicos aplicados ao problema de *clashing* de linhas

O algoritmo proposto funciona bem por dois motivos: o teste de superposição de *boxes* alinhados com os eixos é muito menos custoso de que o teste de detecção de colisões entre elementos e, principalmente, a quantidade de testes necessários é também muito menor, especialmente para números elevados de linhas e de elementos.

3 IMPLEMENTAÇÃO COMPUTACIONAL

Uma vez definida a estratégia de detecção, parte-se para a implementação computacional. Opta-se pela linguagem de programação *C++* (Stroustrup, 2014). Tal escolha é motivada, principalmente, pelo uso recorrente da mesma em grandes sistemas computacionais voltados a análise de alto desempenho de problemas mecânicos, notadamente no próprio DYNASIM e seus módulos acoplados. Um importante recurso fornecido pela linguagem é o paradigma de programação orientada a objetos.

Seguindo os preceitos deste paradigma e boas práticas de desenvolvimento, inicia-se por criar uma classe, denominada **Box**, responsável por fornecer, de forma encapsulada, a ferramenta eficiente de detecção de colisões proposta. Tal classe é então dividida em diversos métodos (funções dentro da classe), sendo cada um tão curto quanto possível e responsável por realizar uma única tarefa coesa, garantindo legibilidade. Juntamente com os comentários feitos no código fonte e a documentação fornecida para a implementação, a ferramenta torna-se otimizável, extensível e de fácil utilização e compreensão.

Detalhes sobre a classe **Box** e alguns de seus métodos são apresentados a seguir na forma de figuras, pseudocódigos e descrições de seus métodos.

3.1 A classe *Box*

Box necessita de três parâmetros para sua inicialização. Tais parâmetros são: o número de linhas analisadas, representado por um inteiro; o número de elementos por linha, na forma de um endereço de memória para um *array* de inteiros; e, por fim, a tolerância para detecção (distância adicionada ao eixo central da linha durante o processo de detecção, podendo ser o raio da linha ou algum valor maior, caso se deseje), também na forma de um endereço, porém para um *array* de *doubles*. Dessa forma, o construtor da classe, em *C++*, é escrito como:

```
Box(int num_of_lines, int* num_of_knots, double* tolerance)
```

O construtor é responsável por alocar e inicializar as variáveis utilizadas de forma apropriada para que as mesmas sejam posteriormente utilizadas dentro da classe.

Em seguida, uma vez finalizada a inicialização, para executar os testes de superposição, basta chamar o método **int test_collisions(double** coordinates, vector<int>& storing_vec)**, que possui como parâmetros de chamada um ponteiro para o endereço do bloco de memória onde o histórico de linhas a ser analisado está armazenado, e que é responsável por executar os testes de superposição de *boxes*, seus refinamentos e hierarquização, até a detecção elemento a elemento, e uma referência para um *vector* de inteiros onde será feito o armazenando das colisões que eventualmente forem detectadas. O ponteiro duplo apresentado é específico para o formato de armazenamento do histórico de linhas utilizado nos testes feitos para a implementação, podendo ser facilmente modificado pelo usuário. O *vector*, por sua vez, é um *conteiner* fornecido pela biblioteca padrão da linguagem, tratando-se, basicamente, de um *array* dinâmico que pode se expandir com tempo amortizado de realocação de memória e ferramentas apropriadas de controle. Já o inteiro retornado pelo método refere-se ao número de colisões detectadas pela implementação.

O método **test_collisions** é ferramenta de acesso para dois outros métodos, estes privados, que realizam os procedimentos necessários para a estratégia de detecção utilizada. São eles: **first_superposition_test(void)** e **carry_on_superposition_tests(void)**, responsáveis, respectivamente, por definir o tamanho dos *boxes* iniciais que englobam as linhas inteiras, bem como realizar o primeiro teste de superposição, para o qual se faz a análise linha a linha, e, com as superposições detectadas pelo primeiro método, carregar o processo de divisão, refinamento e testes de superposição até a detecção de colisões elemento a elemento. Pseudocódigos para ambos os métodos, de forma bastante simplificada, são apresentados na Fig. 2.

```
void first_superposition_test(void){
// Definir o tamanho dos boxes iniciais de cada linha
    for(/*cada box inicial*/){
        for(/*cada outro*/){
            // Testar a superposição entre o par de boxes e armazenar casos positivos
        }
    }
}

void carry_on_superposition_tests(void){
    while(/*houver boxes para testar*/){
        for(/*cada box envolvido em superposições*/){
            for(/*cada outro box superposto a ele*/)
                // Dividir ambos os boxes e redefinir seus tamanhos e os elementos neles contidos
                // Testar a superposição dos boxes recém-divididos e salvar casos positivos
                // Se um par de boxes contiver apenas um elemento, testar colisão e armazenar casos positivos
            }
        }
    }
}
```

Figura 2. Pseudocódigos par dois dos métodos da implementação

Dentro destes dois métodos, vários outros são chamados para desempenhar as tarefas apresentadas (dividir os *boxes*, registrar as colisões, definir seus tamanhos, etc.), visando assim modularizar o código, garantindo legibilidade. Tais métodos não serão destrinchados em prol da brevidade. Vale salientar, contudo, que durante o processo de divisão e testes de superposição, busca-se armazenar e acessar as variáveis utilizadas de forma direta e linear, garantindo assim a performance da implementação.

Para alguns métodos selecionados, opta-se por utilizar a *flag inline*. O *inline* é uma sugestão para que o compilador tente 'copiar' o código do método em questão para o local onde ele é chamado, evitando o custo de chamar o método como uma função separada, bem

como os custos de criar variáveis temporárias de retorno, internas, etc. No código proposto, o uso de *inlines* acarreta tempos computacionais quase 50% menores devido ao reduzido tamanho das funções propostas.

3.2 Testes de superposição e distância

O critério utilizado para verificar superposição entre *boxes* é bastante simples: se o x_{max} de um dos *boxes* testado é menor que o x_{min} do outro, ou se, ao contrário, o x_{max} do segundo box for menor que x_{min} do primeiro, não há superposição entre os *boxes*. O mesmo vale para as coordenadas y e z. Tal fato toma partido do alinhamento dos *boxes* utilizados com os eixos coordenados, o que simplifica imensamente a definição da geometria dos mesmos (especificada apenas por valores mínimos e máximos nos 3 eixos) e os testes de superposição efeituados, meramente unidimensionais. A superposição ocorre quando nenhum dos 6 testes (o par apesentado para cada uma das 3 coordenadas) resultar em falso. Nesta etapa, adiciona-se às dimensões dos *boxes* a tolerância definida no construtor para cada linha, de modo que colisões possíveis não sejam descartadas acidentalmente.

Já o teste de distância entre elementos, por sua vez, é computacionalmente muito mais custoso. O mesmo se baseia em escrever a equação de ambos os segmentos de reta de forma paramétrica, onde os parâmetros, nomeados t e s, são normalizados de forma a estarem contidos no domínio [0,1] para a imagem dos segmentos. O ponto de menor distância entre as retas infinitas que contêm os segmentos é então descoberto definindo-se um par de pontos dentro das mesmas onde o vetor que os conecte seja simultaneamente ortogonal à ambos os vetores diretores das equações paramétricas. Se o par de pontos definido no espaço s-t estiver dentro do domínio de ambos os segmentos, o mesmo já é o ponto e menor distância. Caso contrário, como pontos equidistantes formam elipses no plano s-t, basta testar as bordas do domínio (Fig. 3).

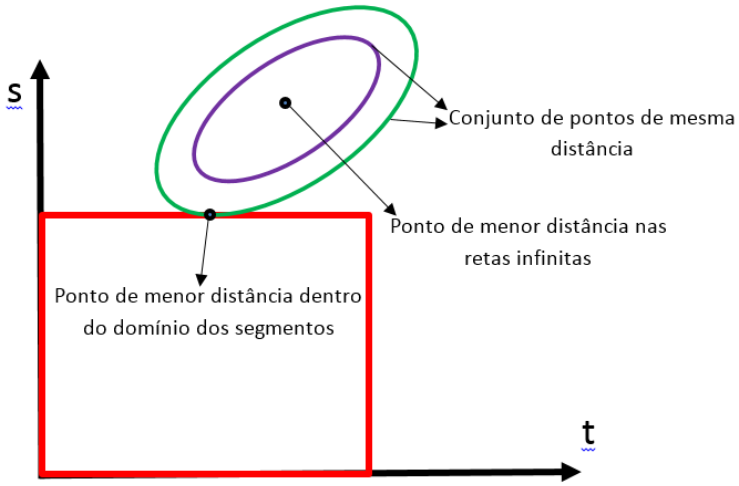

Figura 3. Ilustração das distâncias do plano s-t

Pode-se perceber que não é necessário testar todas as bordas. A depender de onde o ponto de mínimo global se localiza, o mesmo consegue 'enxergar' no máximo um par destas, sendo necessário testar apenas as bordas onde é possível encontrar o mínimo local. Maiores detalhes sobre tal estratégia, tais como o tratamento de segmentos paralelos e as equações para os parâmetros de menor distância, podem ser encontrados na referência (Ericsson, 2005).

4 VALIDAÇÃO E TESTES COMPARATIVOS

De modo a validar a implementação idealizada, dois exemplos numéricos são definidos. Tais exemplos correspondem a arranjos reais de linhas utilizadas pela Petrobras para exploração de petróleo e gás no Brasil. Estes arranjos foram simulados, utilizando o DYNASIM, dentro do Laboratório de Computação Científica e Visualização do Centro de Tecnologia da Universidade Federal de Alagoas (LCCV-CTEC-UFAL).

Estes exemplos comparam, em termos de tempos computacionais e de fidelidade das detecções, os resultados obtidos com a ferramenta implementada e com o método da força-bruta (testes de distância elemento a elemento).

4.1 Exemplo 1

O primeiro exemplo trata de um FPSO típico. Neste, os *risers* (em verde) são flexíveis e assumem aproximadamente a forma de catenárias, enquanto que as linhas de ancoragem (em azul) são tensionadas para reduzir os movimentos do sistema flutuante (Fig. 4).

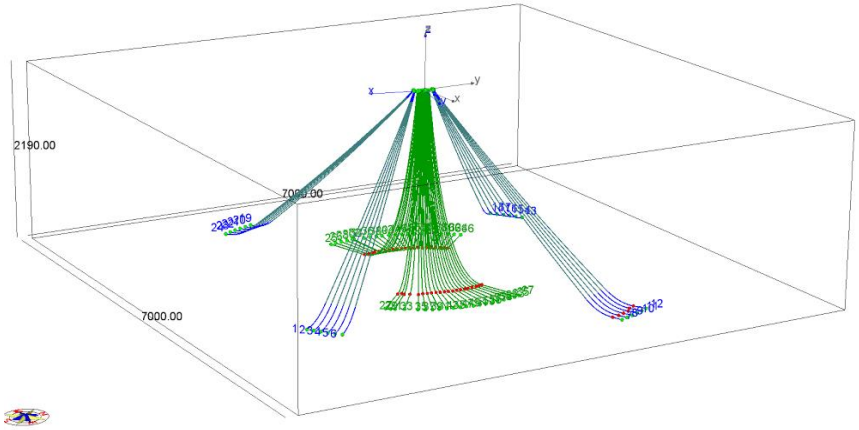

Figura 4. Primeiro arranjo de linhas analisado

A lâmina d'água em questão é de 2.100 m, portanto caracterizada como ultraprofunda (acima de 1.500 m). O arranjo consiste de 67 linhas (entre *risers* e linhas de ancoragem). A primeira das linhas é discretizada por 207 elementos (208 nós); as linhas 2 a 24, por sua vez, são discretizadas por 206 elementos (207 nós); por fim, as demais linhas dispõem de 100 elementos (101 nós) cada.

Uma vez simulado este arranjo no DYNASIM, um histórico de 200 passos de tempo de 0,5 s cada é obtido através de arquivos de saída, que são lidos e armazenado em um bloco de memória. Um ponteiro para esse bloco é então passado ao método **test_collisions** da classe **Box** através de um objeto da mesma.

Os de tempos computacionais para os 200 passos analisados em diferentes distâncias de tolerância totais encontra-se na Tabela 1:

Tabela 1. Tempos computacionais para o exemplo 1

Distância de tolerância entre eixos (m):	***Bounding Boxes* Hierárquicos (s)**	**Método de Força Bruta (s)**
1,00	0,180	108

2,00	0,190	108
3,00	0,220	108
4,00	0,220	108
5,00	0,230	108

Os resultados apresentados ilustram claramente a eficiência do método. Tempos computacionais extremamente reduzidos são observados (dá ordem de 0,2 s para os 200 passos de tempo, ou seja, aproximadamente 1 ms por passo de tempo), em oposição ao método de detecção por força-bruta, em torno de três ordens de grandeza mais lento. Tal diferença se fará ainda mais acentuada para arranjos maiores, uma vez que o algoritmo proposto tem complexidade computacional logarítmica, enquanto o método de força bruta apresenta-se com complexidade quadrática (Ericson, 2005).

É possível reparar que o tempo despendido pela estratégia proposta carrega certa dependência da distância adotada, uma vez que, em sendo maior, a mesma reduz o número de descartes de *boxes*, em oposição ao método da força bruta, onde todos os elementos são testados contra todos os outros independentemente.

Além de verificar o tempo despendido, verifica-se ainda que as colisões detectadas são exatamente as mesmas, ou seja, não há descarte inapropriado de colisões existentes, ou mesmo falsos positivos, atestando para a validade da ferramenta desenvolvida como método preciso de detecção.

4.2 Exemplo 2

Por fim, visando fazer um teste mais completo, de modo a reforçar os resultados anteriores, um histórico maior foi analisado, este contendo 50.000 (cinquenta mil) passos de tempo de 0,2 s cada. O arranjo, apresentado na Fig. 5, corresponde também a uma série de *risers* e linhas de ancoragem, totalizando 130 linhas, todas em catenária, sendo alguns dos *risers* sustentados por boias para alivio de tensões.

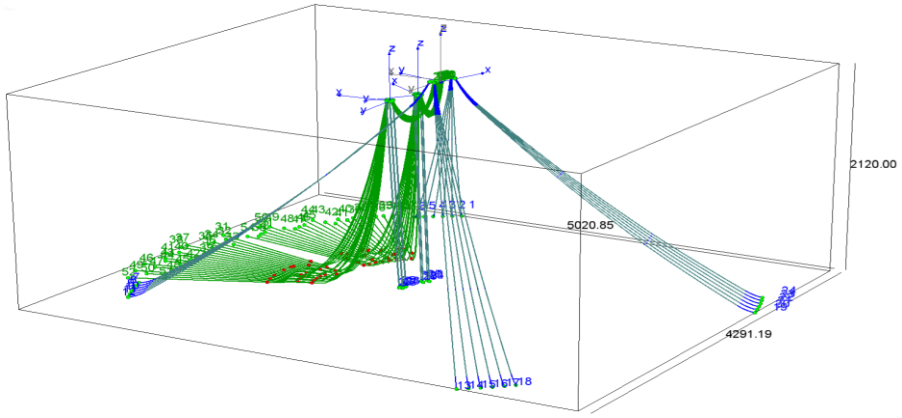

Figura 5. Segundo arranjo de linhas analisado

Neste arranjo o número de elementos discretizando as linhas analisadas varia entre 30, 31, 42, 98 e 5. Tal discretização, em média bem mais pobre que a anterior, representa um caso crítico em termos de performance para o código, visto que os elementos, por terem maior

dimensão mesmo após divididos ao máximo, geram *boxes* de tamanhos elevados, que por sua vez superpõe-se a outros muito mais facilmente, carregando o processo até vários testes elemento a elemento, testes estes bastante caros. Tal fato é ainda agravado pela geometria diferenciada dos *risers*, criada pelas boais.

Os resultados para este teste são apresentados na Tabela 2.

Tabela 2. Tempos computacionais para o exemplo 2

Distância de tolerância entre eixos (m):	*Bounding Boxes* Hierárquicos (s)	Método de Força Bruta (h)
1	81,4	1,36
2	81,5	1,36
3	102	1,36
4	102	1,36
5	102	1,36

Mais uma vez observa-se uma imensa diferença entre a performance das duas estratégias de detecção, embora um pouco menos significativa do que a anterior, uma vez que o número de elementos analisados por passo de tempo é bem menor (o que reduz a diferença entre as complexidades dos dois códigos). Tal redução tem intuito de acelerar a geração do arquivo de saída para o elevado número de passos de tempo utilizado. Na prática, uma análise mais fina seria necessária, no que a estratégia proposta certamente teria desempenho muito superior.

Contudo, mesmo para essa análise mais simples, o tempo despendido pelo DYNASIM na simulação é 3 (três) ordens de magnitude superior ao despendido para a detecção de colisões, ou seja: a ferramenta implementada tem impacto quase nulo no tempo de simulação, o que a viabiliza como recurso para análise de colisões em tempo real.

Novamente observa-se uma dependência da distância adotada para a estratégia proposta. Também se observa, novamente, total concordância entre todas as detecções realizadas pelos dois métodos.

Os resultados são gerados utilizando-se um desktop DexPC com processador Core i5 da Intel de 3,0 GHz, 4 GB de Memória RAM e com o compilador GCC, onde se utilizou o nível de otimização -03.

5 CONSIDERAÇÕES FINAIS

A estratégia proposta mostrou-se extremamente eficiente em termos de custos computacionais, realizando análises, para ambos os exemplos analisados, em menos de 2 ms por passo de tempo para todas as distâncias utilizadas, representando impacto insignificante no desempenho do DYNASIM e de outros sistemas aos quais venha a ser incorporada.

A estratégia também se mostrou estável e fiel em suas detecções, não acusando falsos positivos ou descartando colisões existentes de forma indevida.

O código apresentado supre uma demanda vital dos sistemas de análise de estruturas offshore delgadas, sendo de grande importância neste cenário.

AGRADECIMENTOS

Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela concessão das bolsas de Iniciação Científica (IC) e de Produtividade em Pesquisa (PQ), respectivamente, bem como à Petrobras, através do Centro de Pesquisas Leopoldo Américo Miguez de Mello (Cenpes), pelo financiamento do projeto no qual este trabalho se insere.

REFERÊNCIAS

Bai, Y., Bai, Q., 2005. *Subsea pipelines and risers*. Oxford: Elsevier Science & Technology.

Berg, M D., Cheong, O., Kreveld, M. V., Overmars, M., 2008. *Computational Geometry: Algorithms and Applications*. Springer. 3 ed.

Chakrabarti, S., 2005. *Handbook of Offshore Engeneering*. Elsevier Science, 1 ed.

Det Norske Veritas, 2009. *Recommended Practice RP-F203: Riser Interference*.

Ericson, C., 2005. *Real Time Collision Detection*. Morgan Kaufmann Publishers, San Francisco.

He, J.W., Low, Y.M., 2012. An Approach for Estimating the Probability of Collision Between Marine Risers. *Applied Ocean Research*, 35, 68– 76.

KetcheL, J.S., Larochelle, P.M., 2005. Collision Detection of Cylindrical Rigid Bodies Using Line Geometry. *Proceedings of the International Design Engineering Technical Conferences & Computer and Information in Engineering Conference*. Long Beach, USA.

Nishimoto, K., Fucatu, C.H., Masetti, I.Q., 2001. Dynasim - A Time Domain Simulator of Anchored FPSO. *Proceedings of the 20th International Conference on Offshore Mechanics and Arctic Engineering*. Rio de Janeiro.

Orcina, 2015. OrcaFlex. URL: http://www.orcina.com/.

Rustad, A.M., 2007. Modeling and Control of Top Tensioned Risers. Tese de Doutorado – Norwegian University of Science and Technology/Trondheim.

Scott, M., 2014. *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*. O'Reilly Media.

Stroustrup, B., 2014. *Programming: Principles and Practice Using C++*. Addison-Wesley. 2 ed.

Wu, W., 2003. *Interaction Between Two Marine Risers*. PhD thesis, University of Glasgow.