Alessandro Alberto de Lima
Daniel Neves Furlaneto

nota final
9,4 (nove e quatro)

SIMULAÇÕES NUMÉRICAS DE DECAIMENTO DE *ROLL* DE UM CASCO DE VLCC

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Graduado em Engenharia.

São Paulo
2004

Alessandro Alberto de Lima

Daniel Neves Furlaneto

SIMULAÇÕES NUMÉRICAS DE DECAIMENTO DE *ROLL* DE UM CASCO DE VLCC

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Graduado em Engenharia.

São Paulo

2004

AGRADECIMENTOS

Gostaríamos de agradecer, primeiramente, aos nossos pais por nos ter dado suporte durante estes anos.

Aos Professores Dr. Julio Romano Meneghini e José A. P. Aranha, sem o qual não seria possível a realização desse projeto.

À toda a equipe do Núcleo de Dinâmica dos Fluidos (NDF) pela estrutura para que fosse feito o trabalho, ajuda em questões ao longo do projeto e amizade conquistada nesses anos. Especialmente aos engenheiros José Alfredo Ferrari Júnior, Jairson Lima e Gustavo Assi.

Aos demais professores que, nos deram condições e conhecimento suficientes para que esse trabalho fosse concluído.

RESUMO

Navios petroleiros, convertidos para realizar o processamento e/ou armazenagem de petróleo, têm sido ancorados em mar aberto para atender ao incremento no nível de produção.

Navios FPSO, como são denominados, ficam estacionários por longo tempo em mar aberto e sujeitos à ação combinada de onda, vento e corrente marítima. Em função disso, principalmente de ondas, podem ocorrer ângulos de *"roll"* com grandes amplitudes o que pode acarretar o mau funcionamento de determinados equipamentos de processamento de operação e até mesmo o desconforto dos tripulantes. Para minimizar esse movimento de *"roll"*, bolinas estabilizadoras são instaladas para amortecer tal movimento.

O projeto visa avaliar utilizando técnicas em Dinâmica dos Fluidos Computacional (CFD) o decaimento no movimento de *"roll"* de um casco de VLCC. Comparações dos resultados das simulações com aqueles obtidos no tanque de provas do IPT são apresentadas neste trabalho. Com a metodologia de simulação testada e com resultados comparados com aqueles obtidos no IPT é possível obter uma metodologia mais econômica e flexível de testes de futuras modificações de projeto.

O movimento de corpo rígido é modelado num programa em C e, paralelamente, a equação de Navier-Stokes é resolvida pelo simulador de CFD, Fluent.

Modelo de malha dinâmica ("Dynamic Mesh") nos permite a manipulação do movimento de corpo rígido do modelo fazendo um reposicionamento dos nós e células da malha, juntamente com o modelo VOF ("Volume of Fluid") que permite a manipulação da interface água - ar.

Com relação ao modelo de turbulência utilizou-se o modelo k – omega SST ("Shear – Strees Transport").

ABSTRACT

Oil-tank ships, converted to be able to process and/or store oil, have been anchored in opened sea to attend the increment in the production level. FPSO Ships, as are they called, are stationary for long time in open sea subjected to the action of wave, wind and maritime currents. Due to those environmental conditions, angles of roll with great amplitude can occur, which can cause equipment failure, operation processing shutoff and the discomfort of crew members. To minimize this roll movement, stabilizing bilge keels are installed.

The project aims to evaluate, using Computational Fluid Dynamics (CFD) techniques, the roll motion damping of a VLCC. It is our intention to compare the results of the simulations with those found in experiments carried out at IPT. With the simulation results and comparisons with those from IPT, it is possible to develop a more economic and flexible form to test future modifications in a FPSO project. The rigid body movement is modeled in a program written in C language, and the Navier-Stokes equations are solved by the CFD simulator, Fluent, in parallel. The dynamic mesh model (Dynamic Mesh), allows the rigid body movement manipulation of the model, remaking the nodes and cells position in the mesh, together with the VOF model ("Volume of Fluid") which allows the manipulation of the interface water - air. The turbulence model k – Omega SST ("Shear - Stress Transport") is employed.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CFD	- *Computacional Fluid Dynamics*
FPSO	- *Floating Production Storage and Offloading*
IPT	- Instituto de Pesquisas Tecnológicas
VLCC	- *Very Large Crude Carrier*
UDF	- *User Defined Function*

LISTA DE SÍMBOLOS

B	- Boca da embarcação
D	- Pontal do Casco
H	- Calado da embarcação
KG	- Altura do centro de gravidade
LCG	- Posição longitudinal do centro de gravidade
Lpp	- Comprimento entre perpendiculares da embarcação
P	- Deslocamento da embarcação
Rxx	- Raio de giração em torno do eixo longitudinal
T	- Período
Tn	- Período natural de *roll*
X	- Eixo de referência longitudinal
Y	- Eixo de referência transversal

1 INTRODUÇÃO

Navios petroleiros, convertidos para realizar o processamento e/ou armazenagem da produção de petróleo, tem sido ancorados em mar aberto, para atender ao incremento no nível de produção.

Navios FPSO, como são denominados, ficam estacionários por longo tempo em mar aberto e sujeitos à ação combinada de onda, vento e corrente marítima. Em função disto, principalmente de ondas, podem ocorrer ângulos de *roll* com grandes amplitudes o que pode acarretar o mau funcionamento de determinados equipamentos de processamento de operação e até mesmo o desconforto dos tripulantes. Para minimizar esse movimento de *roll*, bolinas estabilizadoras são instaladas para amortecer tal movimento.

O uso de ferramentas computacionais para a realização dos *Ensaios de Decaimento de movimento de Roll*, como são chamados, pode ser uma importante ferramenta de projeto nessa linha de pesquisa, pois, uma vez que os modelos computacionais são validados com base em dados experimentais temos à disposição uma ferramenta mais flexível para realização de tal ensaio.

2 OBJETIVO

Este projeto de pesquisa avalia, utilizando técnicas em Dinâmica dos Fluidos Computacional (CFD), o decaimento no movimento de roll de um casco de VLCC. Uma comparação entre os resultados dos ensaios realizados no IPT, "ENSAIO DE ROLLING DE CASCO VLCC COM E SEM BOLINAS ESTABILIZADORAS EM ONDAS DE TRAVÉS", e a simulação numérica será feita para validação do código computacional. Com a metodologia de simulação testada e com resultados comparados com aqueles obtidos no Tanque de Provas, será possível obtermos uma forma mais econômica e flexível de testarmos futuras modificações no projeto.

Tanto as dimensões dos modelos numéricos como as condições de operação foram baseadas nos ensaios realizados no Tanque de Provas do IPT, os quais foram patrocinados pela PETROBRAS.

O movimento de corpo rígido é modelado num programa em C e, paralelamente, a equação de Navier-Stokes é resolvida pelo software de CFD, Fluent.

3 MODELAGEM MATEMÁTICA

Para estudar um dado fenômeno de escoamento, é necessário resolver as equações que regem o comportamento de fluidos reais. Como se tratam de equações não-lineares utiliza-se a simulação numérica aplicando as condições de contorno adequadas para a geometria estudada. Nas próximas páginas descrevemos toda a fundamentação teórica que o método utiliza. Para o volume de controle em estudo, a equação para as quais procuramos as soluções numéricas são as equações de Navier-Stokes. Por se tratar de um escoamento turbulento, modelos foram utilizados para essa modelagem.

3.1 MÉTODO DOS VOLUMES FINITOS

O MVF foi desenvolvido para a análise de problemas complexos de Mecânica dos Fluidos. As equações são obtidas através da realização de balanços da propriedade em questão (seja ela massa, quantidade de movimento, entalpia, etc.) nos volumes elementares, ou volumes finitos, ou então; integrando sobre um volume elementar, no espaço e no tempo.

O fato das equações aproximadas representarem à conservação ao nível de volumes elementares vem do fato que a solução da equação diferencial (por exemplo: Equação de Navier-Stokes) representa a conservação da propriedade em nível de ponto (infinitesimal).

Para obter a solução é utilizada uma técnica de volume de controle que consiste:

- Divisão do domínio contínuo em volumes de controles discretos usando a malha computacional;
- Integração das equações nos volumes de controle individuais para construir equações algébricas para as variáveis discretas dependentes, tais como: velocidades e temperatura;
- Linearização das equações discretas e solução do sistema de equações lineares resultante para produzir valores atualizados das variáveis independentes.

A integração das equações diferenciais será mostrada para um conjunto de equações unidimensionais, sendo que estas podem ser facilmente estendidas para o caso

bidimensional ou tridimensional. Sejam as equações diferenciais unidimensionais para continuidade, quantidade de movimento e quantidade escalar ϕ:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho \cdot u) = 0 \tag{1}$$

$$\frac{\partial}{\partial t}(\rho \cdot u) + \frac{\partial}{\partial x}(\rho \cdot u) = -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\left[\mu\left(\frac{\partial u}{\partial x}\right)\right] + F \tag{2}$$

$$\frac{\partial}{\partial x}(\rho \cdot u \cdot \phi) = \frac{\partial}{\partial x}\Gamma\frac{\partial \phi}{\partial x} + S_\phi \tag{3}$$

Essas equações podem ser integradas em relação ao volume de controle empregando o Teorema da Divergência:

$$\int_{volum,y} \frac{\partial}{\partial x}(\rho \cdot u)dV = \int_A (\rho \cdot u) \cdot dA \tag{4}$$

A integração das equações 1, 2 e 3 nos dá os seguintes resultados:

$$\frac{M_P - M_P^0}{\Delta t} + \dot{M}_e - \dot{M}_w = 0 \tag{5}$$

$$\dot{M}_e \cdot u_e - \dot{M}_w \cdot u_w = -(p_e - p_w)A + \left[\frac{\mu_e}{\Delta x_e}(\mu_E - \mu_P) - \frac{\mu_w}{\Delta x_w}(\mu_P - \mu_W)\right]A + S_\phi \tag{6}$$

$$M_e \cdot \phi_e - M_w \cdot \phi_w = \left(\Gamma_e \frac{\phi_E - \phi_P}{\Delta x_e} - \Gamma_w \frac{\phi_P - \phi_W}{\Delta x_w}\right)A + S_\phi \cdot \Delta V \tag{7}$$

As equações eq.(5), eq.(6) e eq.(7) obtidas são equações algébricas que podem ser resolvidas dado que as variáveis indeterminadas (u, p e ϕ) são interpoladas de uma maneira que relaciona seus valores nas faces do volume de controle aos valores no centro do volume de controle. O Procedimento de discretização se baseia no esquema ilustrado na figura abaixo.

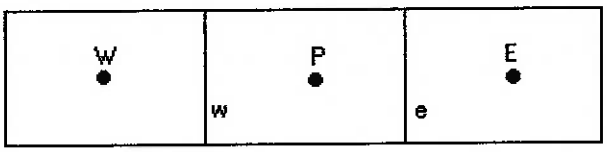

Figura 1: Esquema dos volumes de controle para discretização

A solução das equações expressa acima requerem: o cálculo da pressão nas faces do volume de controle (p_e, p_w), que se determine o fluxo nas faces (M_e, M_w), e a interpolação para relacionar os valores nas faces com os valores das incógnitas (u e ϕ) com os valores nos centros dos volumes de controle.

Os fluxos nas faces são obtidos de tal forma que as velocidades nas faces obedecem a um balanço médio do momento. Já as pressões nas faces são obtidas de tal forma que as velocidades armazenadas no centro da célula obedecem ao balanço de massa.

3.2 Funções de Interpolação para MVF

Ao discretizar uma equação de transporte que possua termos convectivos não nulos aparecerá, na equação discretizada, valores de ϕ nas faces dos volumes de controle. Esses valores precisam ser interpolados entre os valores centrais dos volumes.

Tomando como exemplo um volume de controle unidimensional (direção x) e a seguinte equação de transporte:

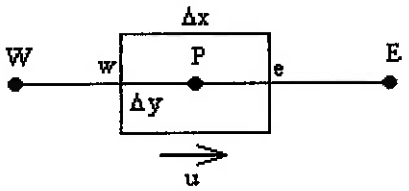

Figura 2: Volume de Controle unidimensional.

$$\frac{d}{dx}(\rho u\phi) = \frac{d}{dx}\left(\Gamma \frac{d\phi}{dx}\right) \tag{8}$$

Integrando a equação acima no volume de controle resulta em:

$$M_e \cdot \phi_e - M_w \cdot \phi_w = D_e(\phi_E - \phi_P) + D_w(\phi_W - \phi_P) \tag{9}$$

onde:

Fluxos convectivos (vazão mássica):

$M_e = \rho_e u_e \Delta y$

$M_w = \rho_w u_w \Delta y$

Termos difusivos (viscosidade):

$D_e = \Gamma_e \Delta y / \Delta x$

$D_w = \Gamma_w \Delta y / \Delta x$

3.3 Upwind de 1ª Ordem

A interpolação das variáveis nas faces é feita da seguinte forma:

$\phi_e = \phi_P$ se $M_e > 0$; $\phi_e = \phi_E$ se $M_e < 0$.

$\phi_w = \phi_W$ se $M_w > 0$; $\phi_w = \phi_P$ se $M_w < 0$.

A ordem de precisão deste método é de Δx (1ª ordem).

3.4 Equação da Conservação de Massa

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(\rho \cdot u_i) = S_m \tag{10}$$

É conhecida como equação da continuidade. Sua validade é muito abrangente, desde escoamentos incompressíveis a compressíveis. O termo *Sm* representa uma fonte que retira ou adiciona massa, ρ é a massa específica do fluido e u_i representa o campo de velocidades do escoamento.

3.5 Equação da Conservação de Quantidade de Movimento

A equação da conservação da quantidade de movimento, na direção x_i, é representada da seguinte forma:

$$\frac{\partial}{\partial t}(\rho \cdot u_i) + \frac{\partial}{\partial x_j}(\rho \cdot u_i \cdot u_j) = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} + \rho \cdot g_i + F_i \tag{11}$$

onde:

p: pressão estática

τ_{ij} : tensor das tensões

gi: aceleração da gravidade na direção i.

Fi: forças externas na direção i.

O tensor das tensões é dado por:

$$\tau_{ij} = \left[\mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right] - \frac{2}{3} \cdot \mu \cdot \frac{\partial u_i}{\partial x_i} \cdot \delta_{ij} \tag{12}$$

onde δ_{ij} representa a dilatação volumétrica.

3.6 O modelo $k-\omega$ standard – modelo de turbulência:

O modelo $k-\omega$ Standard no Fluent é baseado no modelo $k-\omega$ de Wilcox, que incorpora modificações para baixos números de Reynolds, compressibilidade, e shear flow spreading.

Consiste num modelo para modelagem de turbulência. Sua formulação é apresentada à seguir:

$$\frac{\partial}{\partial t}(\rho \cdot k) + \frac{\partial}{\partial x_i}(\rho \cdot k \cdot u_i) = \frac{\partial}{\partial x_j}\left(\Gamma_k \frac{\partial k}{\partial x_j} \right) + G_k - Y_k + S_k \tag{13}$$

e

$$\frac{\partial}{\partial t}(\rho \cdot \omega) + \frac{\partial}{\partial x_i}(\rho \cdot \omega \cdot u_i) = \frac{\partial}{\partial x_j}\left(\Gamma_\omega \frac{\partial \omega}{\partial x_j}\right) + G_\omega - Y_\omega + S_\omega \tag{14}$$

Nessas equações, G_k representa a geração de energia cinética turbulenta devido aos gradientes médios de velocidade. G_ω representa a geração de ω. Γ_k e Γ_ω representam a difusividade efetiva de k e ω, respectivamente. Y_k e Y_ω representam a dissipação de k e ω devido à turbulência. Todos esses termos são calculados como descrito abaixo. S_k e S_ω são os termos fonte que, neste caso, são definidos pelo usuário do solver.

3.6.1 Modelando a difusividade efetiva:

A difusividade efetiva para o modelo $k-\omega$ é dada por:

$$\Gamma_k = \mu + \frac{\mu_t}{\sigma_k} \tag{15}$$

$$\Gamma_\omega = \mu + \frac{\mu_t}{\sigma_\omega} \tag{16}$$

onde, σ_k e σ_ω são os números de Prandtl turbulentos para k e ω, respectivamente. A viscosidade turbulenta, μ_t, é calculada combinando-se k e ω como segue:

$$\mu_t = \alpha^* \frac{\rho k}{\omega} \tag{17}$$

O coeficiente α^* amortece a viscosidade turbulenta causando uma correção para baixos números de Reynolds.

$$\alpha^* = \alpha_\infty^* \left(\frac{\alpha_0^* + \mathrm{Re}_t/R_k}{1 + \mathrm{Re}_t/R_k}\right) \tag{18}$$

onde:

$$\mathrm{Re}_t = \frac{\rho k}{\mu \omega} \tag{19}$$

$$\mathrm{R}_k = 6$$

$$\alpha_0^* = \frac{\beta_i}{3} \tag{20}$$

$$\beta_i = 0.072$$

Podemos notar que, para altos números de Reynolds do modelo $k-\omega$, $\alpha^* = \alpha_0^* = 1$.

3.6.2 Modelando a produção de Energia Cinética Turbulenta:

3.6.2.1 Produção de k:

O termo G_k representa a geração de energia cinética turbulenta. Da equação exata do transporte de k, esse termo poder ser obtido por:

$$G_k = -\rho \cdot \overline{u_i' \cdot u_j'} \frac{\partial u_j}{\partial x_i} \tag{21}$$

onde, u_i' e u_j' se referem às flutuações à que estão submetidos os gradientes médios de velocidades.

Avaliando G_k com relação à hipótese de Boussinesq que relaciona as tensões de Reynolds com os gradientes médios de velocidades:

$$G_k = \mu_t S^2 \tag{22}$$

em que S é o módulo do tensor rate-of-strain, e é dado por:

$$S \equiv \sqrt{2 S_{ij} S_{ij}} \tag{23}$$

onde

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right) \tag{24}$$

3.6.2.2 Produção de ω :

A produção de ω é dada por:

$$G_\omega = \alpha \frac{\omega}{k} G_k \tag{25}$$

onde G_k é dado pela equação

O coeficiente α é dado por:

$$\alpha = \frac{\alpha_\infty}{\alpha^*}\left(\frac{\alpha_0 + \mathrm{Re}_t/\mathrm{R}_\omega}{1+\mathrm{Re}_t/\mathrm{R}_\omega}\right) \tag{26}$$

onde $\mathrm{R}_\omega = 2.95$. α e Re_t são dados por eq.(26) e eq.(19).

Podemos notar que, para altos números de Reynolds do modelo $k-\omega$, $\alpha^* = \alpha_0^* = 1$.

3.6.3 Modelando a Dissipação de Turbulência

3.6.3.1 Dissipação de k:

A dissipação de k é dada por:

$$Y_k = \rho\beta^* f_{\beta^*} k\omega \tag{27}$$

onde

$$f_{\beta^*} = \begin{cases} 1 & \chi_\kappa \le 0 \\ \dfrac{1+680\chi_k^2}{1+400\chi_k^2} & \chi_\kappa > 0 \end{cases} \tag{28}$$

onde

$$\chi_k \equiv \frac{1}{\omega^3}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j} \tag{29}$$

$$\beta^* = \beta_i^*\left[1+\zeta^* F\left(M_t\right)\right]$$

$$\beta_i^* = \beta_\infty^*\left(\frac{4/15+\left(\mathrm{Re}_t/\mathrm{R}_\beta\right)^4}{1+\left(\mathrm{Re}_t/\mathrm{R}_\beta\right)^4}\right)$$

$$\zeta^* = 1.5 \tag{30}$$

$$\mathrm{R}_\beta = 8$$

$$\beta_\infty^* = 0.09$$

3.6.3.2 Dissipação de ω

A dissipação de ω é dada por:

$$Y_\omega = \rho\beta f_\beta \omega^2 \tag{31}$$

onde

$$f_\beta = \frac{1+70\chi_\omega}{1+80\chi_\omega}$$

$$\chi_\omega = \left|\frac{\Omega_{ij}\Omega_{jk}S_{ki}}{\left(\beta_\infty^*\omega\right)^3}\right| \tag{32}$$

$$\Omega_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}-\frac{\partial u_j}{\partial x_i}\right)$$

onde S_{ij} é dado pela equação . Também temos:

$$\beta = \beta_i\left[1-\frac{\beta_i^*}{\beta_i}\zeta^* F(M_t)\right] \tag{33}$$

onde β_i^* é dado pela eq.(21) e equação e $F(M_t)$ é dado a seguir:

3.6.4 Correção de Compressibilidade:

A função de compressibilidade, $F(M_t)$, é dada por:

$$F(M_t) = \begin{cases} 0 & M_t \le M_{t0} \\ M_t^2 - M_{t0}^2 & M_t > M_{t0} \end{cases} \tag{34}$$

onde

$$\begin{aligned} M_t^2 &\equiv \frac{2k}{\alpha^2} \\ M_{t0} &= 0.25 \\ \alpha &= \sqrt{\gamma RT} \end{aligned} \tag{35}$$

Podemos notar que, para altos números de Reynolds do modelo $k-\omega$, $\beta_i^* = \beta_\infty^*$.

Para casos incompressíveis $\beta^* = \beta_i^*$.

Constantes do modelo:

$$\alpha_\infty^* = 1, \quad \alpha_\infty = 0.52, \quad \alpha_0 = \frac{1}{9}, \quad \beta_\infty^* = 0.09, \quad \beta_i = 0.072, \quad R_\beta = 8$$

$$R_k = 6, \quad R_\omega = 2.95, \quad \zeta^* = 1.5, \quad M_{t0} = 0.25, \quad \sigma_k = 2.0, \quad \sigma_\omega = 2.0$$

4 GERAÇÃO DE MALHAS NÃO-ESTRUTURADAS

Existem vários métodos numéricos utilizados para a resolução de problemas em CFD. Entre eles estão o método das diferenças finitas, método de elementos finitos, método espectral e método dos volumes finitos. Este último é utilizado nas simulações deste trabalho e terá uma descrição detalhada adiante. Todos estes métodos têm caráter "euleriano", isto é, a análise é focada num espaço fixo em relação ao sistema de coordenadas adotado, e não na partícula. Desse modo, é necessário que se discretize o domínio do problema a fim de aplicarmos o método de resolução. É nisso que consiste a geração de malhas, da discretização do domínio em vários elementos de forma geral pré-determinada, com a finalidade de estabelecer a posição dos pontos (nós) para os quais serão calculadas as soluções pretendidas. A geração de malhas, a determinação das condições de contorno e condições iniciais e o ajuste dos parâmetros de solução constituem o que se costuma chamar de pré-processamento do problema.

A fase de geração de malhas é muito importante na medida em que a geração de uma malha válida num domínio com uma geometria complexa não é uma operação trivial e pode ter um custo bastante grande em termos de tempo de processamento. Além do mais, a criação de uma malha coerente com as características físicas do problema considerado é crucial, porque a qualidade da solução computada está fortemente relacionada com a qualidade da malha.

4.1 Noções Gerais Relativas a Malhas

Uma malha de um domínio, Ω, é definida por um conjunto, T_h, que consiste de um número finito de segmentos em uma dimensão, segmentos, triângulos e quadriláteros em duas dimensões e os elementos anteriores mais tetraedros, pentaedros e hexaedros em três dimensões. Os elementos, K, de tal malha devem satisfazer a um certo número de propriedades que serão introduzidas a seguir. A primeira diz respeito à conformidade, de acordo com a definição:

Definição: T_h é uma malha conforme de Ω se as seguintes condições são satisfeitas:

1. $\overline{\Omega} = \cup_{K \in T_h} K$

2. Todos os elementos de *Th* têm interior de área (no caso bidimensional) ou volume (no caso tridimensional) não nulos

3. A interseção de dois elementos quaisquer de *Th* se enquadra em um, e apenas um, dos seguintes casos:

- conjunto vazio
- um ponto comum aos dois elementos
- uma aresta comum aos dois elementos
- uma face comum aos dois elementos

Se T_h é uma malha conforme, então dizemos que ela representa Ω de maneira conforme quanto a aspectos geométricos. Na prática, T_h é um particionamento de Ω, tão preciso quanto possível. Quando Ω não é um domínio poligonal (ou poliedral), T_h será apenas uma discretização aproximada do domínio.

Os elementos constituintes de uma malha devem geralmente satisfazer algumas propriedades específicas:

4.1.1 Propriedades geométricas:

- A variação dimensional entre dois elementos adjacentes tem que ser progressiva e descontinuidades de elementos para elementos não podem ser muito abruptas.
- A densidade de elementos em regiões de gradientes elevados de alguma grandeza envolvida no problema deve ser alta.
- Quando os elementos são do tipo triangular, deve-se evitar a presença de ângulos obtusos nos elementos.
- Os elementos devem se adequar às características anisotrópicas do problema.

4.1.2 Propriedades de natureza física:

Essas propriedades estão fortemente ligadas aos aspectos físicos do problema em consideração. A configuração geral e individual dos elementos deve ser definida de acordo com o comportamento do problema.

Existem numerosos algoritmos para a construção de malhas bidimensionais e tridimensionais. A escolha do método está fortemente ligada à geometria do domínio considerado. As malhas geradas podem ser agrupadas em duas classes principais: *malhas estruturadas* e *malhas não-estruturadas*. Uma malha é chamada de estruturada se sua conectividade é do tipo de diferenças finitas. Uma malha é chamada de não-estruturada se sua conectividade é de qualquer outro tipo. Por *conectividade* de uma malha entendemos a definição da conexão entre seus vértices, em outras palavras, a conexão entre os nós globais de uma malha e os nós locais de cada elemento da malha.

Elucidando melhor os conceitos: para uma malha estruturada, a conectividade entre os nós é do tipo *(i, j, k)*, isto é, assumindo que índices de um certo nó sejam (i, j, k), seu vizinho esquerdo terá os índices ((i–1), j, k) e seu vizinho direito terá os índices ((i+1), j, k). Este tipo de malha é mais apropriado para geometrias simples e simétricas, tais como configurações quadrilaterais e hexaedrais. Para geometrias mais complexas, é necessário um tratamento especial para que este tipo de estruturação seja concebido. O presente trabalho lida com simulações que utilizam malhas não estruturadas, que por sua vez apresentam menos restrições geométricas, mas tem um custo computacional maior.

Podemos ainda dividir os diferentes algoritmos de geração de malha em sete classes:

Métodos manuais ou semi-automáticos: adequados para geometrias relativamente simples. Estão nessa classe os métodos enumerativos, nos quais os pontos, arestas, faces e elementos que compõe a malha são dados explicitamente; e métodos apropriados para situações geométricas particulares, como formas cilíndricas e hexaedrais, os quais usam propriedades específicas da geometria explicitamente e a conectividade é conhecida *"a priori"*.

Métodos que utilizam mapeamento: constroem a malha a partir do mapeamento, através de uma transformação conforme de um domínio, de uma malha de geometria simples.

Métodos baseados na solução de um sistema de equações diferenciais a derivadas parciais: essa abordagem se assemelha à segunda, mas aqui a função de mapeamento não é dada a princípio, mas é computada a partir da resolução de equações diferenciais a derivadas parciais, de forma a satisfazer certas propriedades de interesse, tais como densidade de elementos e ortogonalidade.

Métodos baseados na deformação e modificação local de uma malha: este método aplica-se principalmente a malhas de fácil obtenção, do tipo *quadtree*, em casos bidimensionais, ou *octree*, para casos tridimensionais. Nestes casos o domínio está encerrado num quadrilátero ou num paralelepípedo que é divido em subconjuntos na forma de caixas. Esses subconjuntos são construídos pela decomposição baseada em uma árvore quaternária (para dimensão 2) ou árvore octal (dimensão 3). A rede resultante é então utilizada para criar a malha desejada.

Métodos que derivam a malha final, elemento por elemento, dos dados do contorno: basicamente existem duas abordagens: métodos de frente progressiva ("advancing front methods") e algoritmos baseados na construção de Voronoï-Delaunay. Estes métodos criam nós internos e elementos, começando da fronteira do domínio. Esta fronteira pode ser dada de maneira global (por exemplo, definidos de forma analítica) ou de maneira discreta (como uma lista de arestas de faces triangulares). Esta classe de métodos é de particular interesse neste trabalho, pois é a que o software de geração de malhas utilizado (Gambit 2.0.4) emprega.

Métodos que utilizam a composição de malhas de subconjuntos baseados na modificação geométrica ou topológica dessas malhas: neste caso, as malhas dos subconjuntos podem ser obtidas por qualquer um dos métodos anteriores. O problema é então divido em um conjunto de "sub-problemas" de menor complexidade, que são então resolvidos por uma ou mais classes das anteriormente citadas e o resultado final é então obtido por transformações e a adição dos resultados parciais.

Assim sendo, percebe-se que as principais diferenças entre os algoritmos de geração de malhas estão na generalidade do método, principalmente com relação à geometria, e a variedade, quantidade e forma dos dados que tem que ser fornecida ao algoritmo.

O estabelecimento da noção de malha de tal forma que esta seja conveniente em termos da computação futura precede a escolha do método geral de concepção da

malha. Escolhido o método, existem diferentes maneiras pelas quais ele pode ser implementado.

4.1.3 Descrição Geral

Uma malha tem que ser descrita de acordo com a sua aplicação. No caso de simulações de escoamentos externos, que é o que ocorre neste trabalho, são necessárias as definições de objetos sólidos e da zona fluida que os circunda. Nesta definição deverão estar contidas todas as informações necessárias considerando os vários passos na computação. Estas informações incluem geometria, condições de contorno. Elas podem ser agrupadas em três tipos:

4.1.4 Informação geométrica:

Aqui se incluem a descrição da malha, ou seja, como seus elementos cobrem o domínio, e uma espécie de histórico que contenha toda a informação previamente utilizada na construção dos elementos. Também tem que estar descrito o tipo de elemento (segmento, triângulo, quadrilátero, tetraedro, pentaedro, hexaedro ou outro).

A maneira prática da descrição da malha se constitui na listagem dos vértices dos elementos, a conectividade, as coordenadas dos vértices e a topologia, que é a descrição das arestas e faces de um elemento pelos seus vértices.

4.1.5 Informações necessárias ao processamento:

Encontram-se aqui agrupados os dados para computação das matrizes, solução dos sistemas e visualização dos resultados. Estas informações variam de acordo com o algoritmo numérico utilizado para a resolução do problema. Exemplos são o número e a lista dos nós dos elementos.

É preciso frisar que os nós e os vértices de um elemento podem coincidir ou não. Podem existir nós intermediários localizados nas arestas, faces ou interior do elemento. Convenciona-se então uma ordem de numeração, de modo a simplificar a

representação dos elementos. A seguir são dados quatro exemplos de elementos triangulares, com os respectivos nós numerados e indicados:

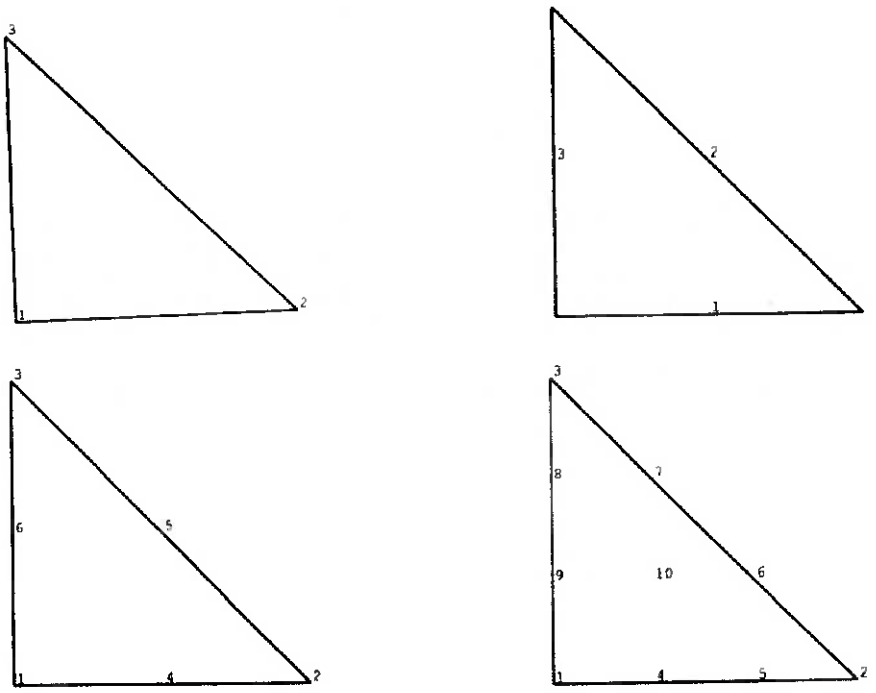

Figura 3: Diferentes numerações dos nós de um elemento triangular

Nesta classificação estão as condições iniciais e de contorno e caracterização física dos elementos (material e propriedades, por exemplo).

4.1.6 Metodologia Geral para Criação de Malhas

A concepção de uma malha pode ser decomposta em três passos:

- Análise do problema;
- Definição formal do processo de geração da malha;
- A construção da malha propriamente dita.

O primeiro passo consiste na análise da geometria do domínio e do problema físico a ser resolvido. Essa análise deve ser feita segundo uma metodologia *top-down*, ou seja, na decomposição de um problema complexo numa série de problemas mais simples.

A construção formal da malha, que constitui o segundo passo, leva em conta os resultados da análise efetuada no primeiro passo e é baseada numa construção *bottom-up*, que é a definição de objetos simples tornando a solução do problema completo possível através da soma das soluções dos objetos.
Por último, a construção da malha propriamente dita é feita através do uso de um algoritmo apropriado de geração de malhas e consiste de duas fases: a definição do conjunto de dados relevantes e a geração real da malha.

4.1.6.1 Métodos de Frente Progressiva (Advancing Front Methods)

Aqui será feita uma introdução geral ao método empregado pelo software utilizado para gerar as malhas das simulações deste trabalho (Gambit 2.0.4). Esta classe de geradores de malhas foi desenvolvida entre as décadas de 70 e 80 e foi a primeira solução automática para a geração de malhas para domínios de geometrias arbitrárias. Basicamente, os algoritmos constroem a malha do domínio a partir da fronteira do mesmo. Os elementos utilizados são triângulos no caso bidimensional e tetraedros no caso tridimensional. Os dados demandados são as fronteiras do domínio ou, mais precisamente, sua discretização poligonal (para dimensão 2) descritos por uma lista de segmentos, ou sua discretização poliedral (para dimensão 3) descritos por uma lista de faces triangulares.
O processo é iterativo: uma *frente*, inicializada por um conjunto de itens da fronteira dada, é analisada a fim de estabelecer uma *zona de partida*, a partir da qual um ou mais elementos internos são criados; a frente é então atualizada e o processo de criação de elementos é repetido se a frente não for um conjunto vazio. O algoritmo pode ser sumariado da seguinte forma:

- Inicialização da frente;
- Análise da frente:
  - Determinação da zona de partida;
  - Análise da região:
    - Criação dos pontos internos e dos elementos internos;

- Atualização da frente.

- Se a frente não for um conjunto vazio, ir para "Análise da frente".

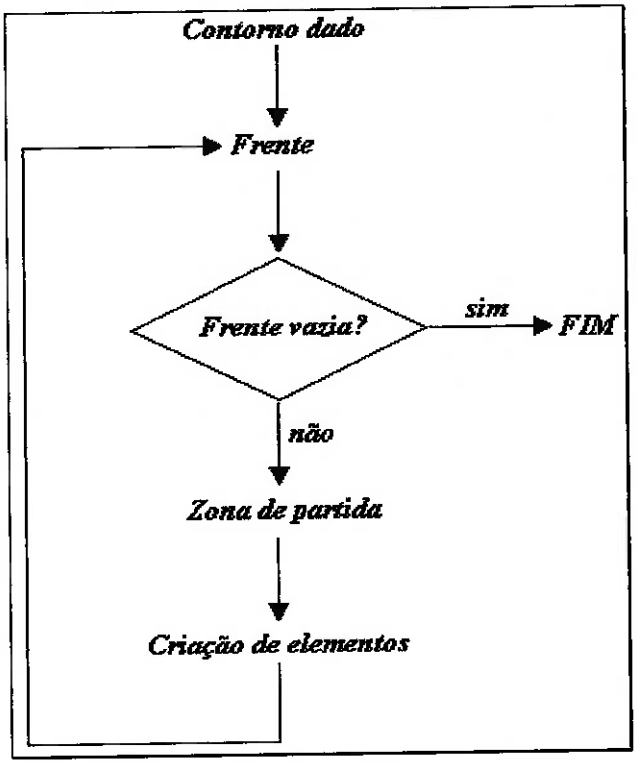

Figura 4: Esquema geral do método de frente progressiva.

A análise da frente e a criação dos elementos podem ser feitas de várias formas. Aqui serão descritas uma forma para o caso bidimensional e uma para o caso tridimensional. Logo após são introduzidas algumas extensões que servem para controlar a criação dos pontos internos e dos elementos, de tal maneira que a malha resultante tenha algumas características particulares, como elementos isotrópicos, elementos anisotrópicos, etc.

4.1.6.2 Métodos de frente progressiva em duas dimensões

Como já foi exposto, este tipo de algoritmo constrói a malha do domínio Ω com triângulos que partem do seu contorno. Na prática, uma aproximação poligonal do contorno é usada em termos de uma lista dos seus elementos constitutivos. O interior do domínio, ou seja, a zona a ser discretizada, está bem definida por causa da

orientação do contorno servindo como dado de entrada. A frente inicial F é definida como o conjunto de segmentos da fronteira C descrevendo o domínio Ω.

Dada F, pode-se detalhar a maneira pela qual os triângulos são criados. Enquanto o processo de criação dos triângulos internos progride, a fronteira C e a frente F são atualizadas. Considerando F o atual estado da frente, então sua análise é baseada no exame das propriedades geométricas dos seus elementos constituintes. Chamando de α o ângulo formado por dois segmentos consecutivos da frente F, então são as três situações:

$\alpha < \frac{\pi}{2}$, os dois segmentos com ângulo α são mantidos e tornam-se dois lados do triângulo criado (*Figura 5: Padrão 1*);

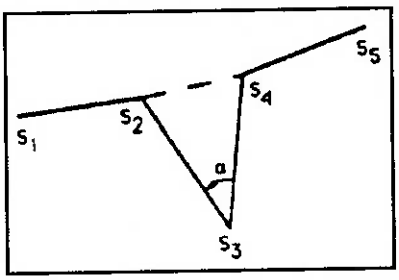

Figura 5: Padrão 1

$\frac{\pi}{2} \leq \alpha \leq \frac{2\pi}{3}$, dos dois segmentos com ângulo α, um ponto interno e dois triângulos são gerados(*Figura 6: Padrão 2*);

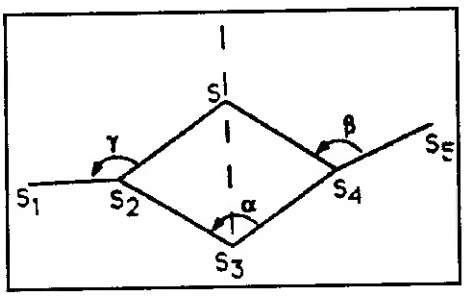

Figura 6: Padrão 2

$\frac{2\pi}{3} < \alpha$, um segmento é mantido, um triângulo é criado com este segmento sendo um dos lados e um ponto interno (*Figura 7: Padrão 3*);

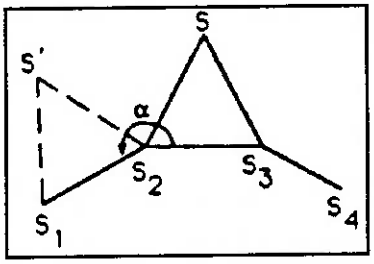

Figura 7: Padrão 3

As posições dos pontos internos criados são definidas de forma que sejam ótimas, significando que os elementos que têm esses pontos como vértices sejam os mais regulares possíveis. No caso do padrão 2, o vértice é gerado na linha bissetriz do ângulo α a uma distância computada a partir dos respectivos comprimentos das arestas da zona de partida: a localização deste ponto interno S é calculada pela fórmula:

$$d_{SS_3} = \frac{1}{6}\left(2d_{S_2S_3} + 2d_{S_3S_4} + d_{S_1S_2} + d_{S_4S_5}\right) \tag{36}$$

No caso dos ângulos β e γ (Figura 9.3) terem seus valores entre $\pi/5$ e $2\pi - \pi/5$ radianos (o valor $\pi/5$ é empírico). Para outros casos, o padrão 1 é utilizado. No caso do padrão 3, um triângulo o mais próximo de um eqüilátero possível é formado usando o segmento mais curto da zona de partida.

Na criação de cada ponto, é necessário verificar se o ponto está dentro do domínio ainda não coberto pelos elementos já construídos. Isto quer dizer cada ponto criado tem que estar dentro do domínio considerado e fora de qualquer elemento existente. Essa verificação, crucial para este tipo de método, baseia-se no conhecimento exato da vizinhança da zona que está sendo criada. No caso bidimensional, um ponto será interno se a intersecção de todos as arestas que dele partem com qualquer aresta da frente é um conjunto vazio. No caso de domínios com um ou mais loops internos

("buracos"), é necessário considerar ainda a condição de que nenhum triângulo formado com o ponto em questão contenha um ponto, em qualquer segmento, do contorno de qualquer loop interno presente.

Uma nova frente F é formada pela supressão dos segmentos que pertençam aos triângulos criados e à antiga frente; e pela adição dos novos segmentos dos triângulos criados, que não sejam comuns a dois elementos. O estado atualizado de F é então processado da mesma forma. A *Figura 8: Estados da frente progressiva* mostra vários estados da frente em evolução correspondendo ao domínio mostrado na *Figura 9: Malha final*. Uma vez que F seja um conjunto vazio, a malha final está constituída.

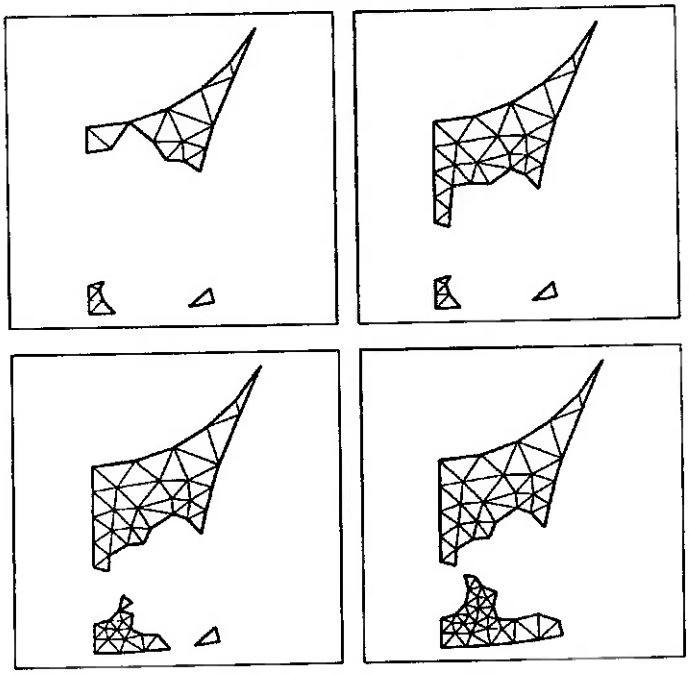

Figura 8: Estados da frente progressiva

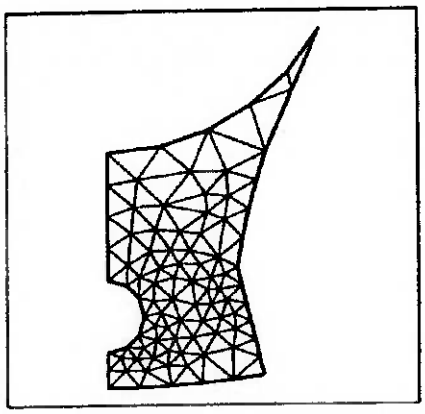

Figura 9: Malha final

No caso de domínios fortemente não convexos, o método pode não convergir. Além disso, uma variação muito aguda na distribuição dos pontos na fronteira pode produzir um resultado negativo similar. Para sanar este problema, consideram-se apenas subconjuntos primários adequados, ou um método diferente tem que ser usado. De fato, este resultado negativo é uma conseqüência da dificuldade em provar a validade do método teoricamente, mas uma implementação mais astuta pode superar este problema.

A triangulação obtida está claramente relacionada ao número e localização relativa dos pontos que discretizam a fronteira. Assim, especificando os pontos da fronteira adequadamente, é possível obter uma densidade variável de elementos em certas regiões da malha.

A malha final pode ser polida a fim de obter triângulos de melhor qualidade. Este processo corrige a posição dos pontos criados usando informações locais globalmente. O resultado é mostrado na *Figura 10: Malha antes e depois de ser polida.*

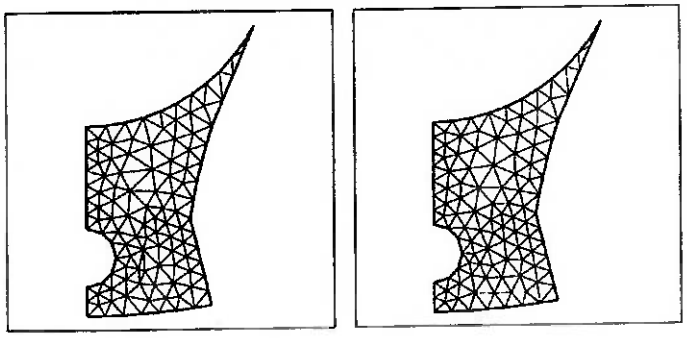

Figura 10: Malha antes e depois de ser polida

Existem numerosas variações do método de frente progressiva. Em particular, a zona de partida pode ser escolhida como:

Uma parte do contorno tal que seus elementos constitutivos satisfaçam certas condições;

A fronteira inteira constitui a frente, e seus elementos constitutivos participam da criação de elementos numa ordem pré-definida.

A primeira abordagem se aplica especialmente a zonas particulares, por exemplo, aquelas que contém ângulos pequenos. A segunda abordagem produz uma inflação da frente inicial (*Figura 11: Frente progredindo por inflação*) ou a propagação de uma linha inicial (*Figura 12: Frente progredindo pelo avanço de uma linha*).

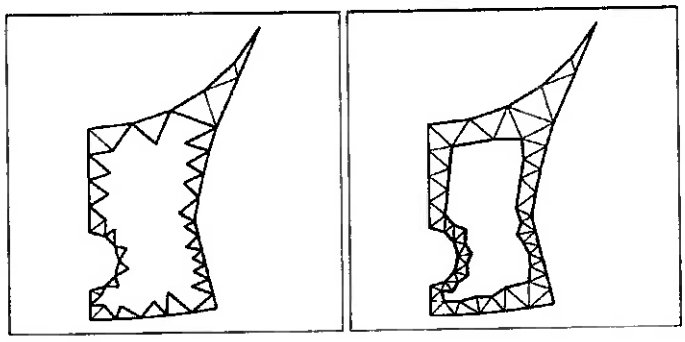

Figura 11: Frente progredindo por inflação

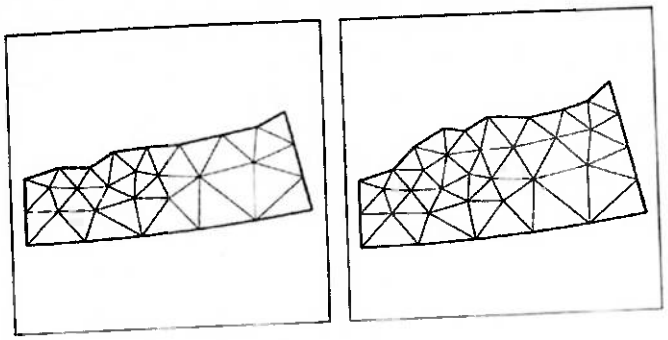

Figura 12: Frente progredindo pelo avanço de uma linha

Este método pode também ser aplicado para a criação de quadriláteros. Baseado no mesmo princípio, o algoritmo intenta em criar quadriláteros com a forma a mais regular possível. Este processo utiliza triângulos em locais impossíveis de serem cobertos por um quadrilátero ou uma combinação deles.

5 MALHA DINÂMICA

5.1 Introdução

O modelo de malha dinâmica é utilizado onde a forma do domínio está mudando com o tempo devido ao movimento nos limites do domínio (por exemplo, você pode especificar as velocidades lineares e angulares sobre o centro de gravidade de um corpo contínuo com tempo) ou um movimento não descrito, onde o movimento subseqüente é determinado com base na solução no tempo atual (as velocidades por exemplo, lineares e angulares são calculadas do contrapeso da força em um corpo contínuo). A atualização da malha do volume é feita automaticamente em cada passo, baseada nas novas posições do objeto que está variando sua posição em função do tempo. Para utilizar o modelo de malha dinâmica você precisa fornecer uma malha inicial do volume e a descrição do movimento de todas as zonas que possuem movimentação no modelo. Essa descrição pode ser feita tanto utilizando perfis do contorno ou utilizando funções descritas pelo usuário (*UDF*).

O Fluent precisa da descrição especifica do movimento para a zona de cada face ou célula. Se o modelo contém regiões móveis e não móveis, é necessário identificar estas regiões agrupando elas em suas respectivas zonas na malha inicial que foi que foi gerada. Além disso, as regiões que se deformam devido ao movimento de suas regiões adjacentes também devem ser agrupadas em zonas separadas na malha inicial. Os contornos entre as várias regiões não precisam ser conformes. Podemos utilizar a opção de "non-conformal" ou "sliding" na interface para conectar as várias zonas no modelo final.

5.2 Equações de conservação da malha dinâmica.

A forma integral da equação de conservação para um escalar genérico, ϕ, em um volume de controle genérico, V, cujo contorno que se move pode ser escrito como:

$$\frac{d}{dt}\int_V \rho\phi dV + \int_{\partial V} \rho\phi(\vec{u}-\vec{u}_g)d\vec{A} = \int_{\partial V} \Gamma\nabla\phi d\vec{A} + \int_V S_\phi dV \qquad (36)$$

onde:

ρ é a densidade do fluido

$\vec{u}$ é o vetor velocidade do escoamento

$\vec{u}_g$ é a velocidade com que os nós da malha dinâmica se movem

Γ é o coeficiente de difusão

S_ϕ é a fonte do termo ϕ

Aqui ∂V é usado para representar os limites do volume de controle V.

O termo derivativo do termo na equação (8.1) pode ser escrito, usando a fórmula inversa da diferença de primeira ordem, assim:

$$\frac{d}{dt}\int_V \rho\phi dV = \frac{(\rho\phi V)^{n+1} - (\rho\phi V)^n}{\Delta t} \tag{37}$$

onde n e $n+1$ são dados respectivamente pelo quantidade no nível atual e no próximo. O $(n+1)$*esimo* nível do tempo V^{n+1} é computado de :

$$V^{n+1} = V^n + \frac{dV}{dt}\Delta t \tag{38}$$

onde dV/dt é o termo derivativo em função do tempo do volume de controle.

Para satisfazer a conservação da lei da malha, o derivativo do tempo do volume de controle é computado de:

$$\frac{dV}{dt} = \int_{\partial V} \vec{u}_g . d\vec{A} = \sum_j^{n_f} \vec{u}_{g.j} . \vec{A}_j \tag{39}$$

onde n_f é o número de faces no volume de controle e $\vec{A}_j$ e o j é o vetor da área. O produto no ponto $\vec{u}_{g.j}.\vec{A}_j$ em cada face do volume de controle é calculada de:

$$\vec{u}_{g.j}.\vec{A}_j = \frac{\delta V_j}{\Delta t} \tag{40}$$

onde δV_j é o volume levado para fora pela face j do volume de controle no passo de tempo Δt.

5.3 Métodos atualizados de malha dinâmica

Estão disponíveis no FLUENT três métodos para refazer a malha do volume nas regiões de deformação referentes ao movimento definido no contorno.

- Mola deslizante
- Camada dinâmica
- Reconstrução da malha local

Detalharemos apenas o funcionamento do método utilizado no experimento.

5.3.1 Método da mola deslizante

Neste método as linhas da malha funcionam como uma rede das molas interconectadas. Os afastamentos iniciais das bordas antes de todo o movimento do contorno constituem o estado do equilíbrio da malha. Um deslocamento em um dado nó do contorno gerará uma força proporcional ao deslocamento ao longo de todas as molas conectadas ao nó. Usando a lei de Hook's, a força em um nó da malha pode ser escrita como:

$$\vec{F}_i = \sum_j^{n_i} k_{ij}(\Delta\vec{x}_j - \Delta\vec{x}_i) \tag{41}$$

onde $\Delta\vec{x}_i$ e $\Delta\vec{x}_j$ são o espaçamento do nó i e seu adjacente j, n_i é o número de nós adjacentes conectados ao nó i, e k_{ij} é a constante de mola (ou rigidez) entre os nós i e j. A constante da mola para a linha conectando os nós i e j são definidos como:

$$k_{ij} = \frac{1}{\sqrt{|\vec{x}_i - \vec{x}_j|}} \tag{42}$$

No equilíbrio a força total em um nó devido a todas as molas conectadas a ele deve ser zero. Esta condição resulta em uma equação iterativa onde:

$$\Delta\vec{x}_i^{m+1} = \frac{\sum_j^{n_i} k_{ij}\Delta\vec{x}_j^m}{\sum_j^{n_i} k_{ij}} \tag{43}$$

Conhecidos os deslocamentos no contorno (depois que as posições do nó do limite foram atualizadas), a equação (8.8) é resolvida usando a varredura de Jacobi no interior de todos os nós. Na convergência, as posições são atualizadas tais que:

$$\vec{x}_i^{n+1} = \vec{x}_i^n + \Delta\vec{x}_i^{m,\,convergido} \tag{44}$$

onde $n+1$ e n são usadas para denotar as posições dos nós para o próximo passo de tempo e o passo de tempo atual, respectivamente. O modelo de mola deslizante é mostrado nas figuras (8.1) e (8.2) para uma malha cilíndrica onde a base do cilindro está movendo.

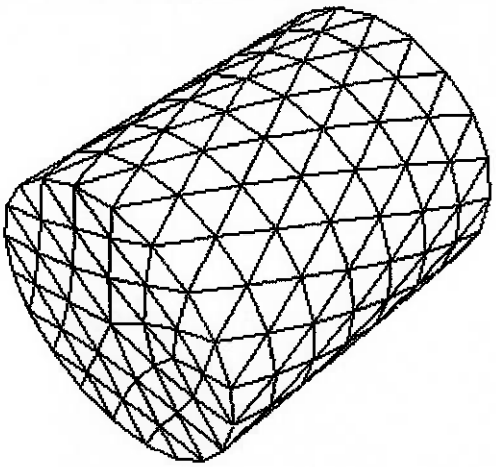

Figura 13: Configuração inicial da malha

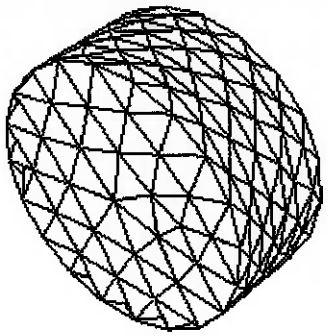

Figura 14: Configuração final da malha

Este método é indicado para casos onde o contorno move-se predominantemente em apenas uma direção, sem compressão ou distensão anisotrópica excessiva da zona da célula.

5.4 Cinemática do corpo sólido

O Fluent utiliza cinemáticas do corpo rígido se o movimento é descrito com base na posição e orientação do centro de gravidade do objeto móvel. Isto é aplicado tanto nas zonas das células como nas zonas das faces. O movimento de um corpo sólido pode ser especificado por uma velocidade linear e angular da velocidade do centro de gravidade. As velocidades podem ser especificadas tanto utilizando perfis do

contorno ou utilizando UDFs. Supõe que o movimento está especificado no sistema de coordenadas inerciais.

Se o movimento for especificado utilizando-se um perfil, as componentes das velocidades devem ser descritas utilizando-se os seguintes parâmetros.

- Velocidade linear (v_x, v_y, v_z) em função do tempo
- Velocidade angular (w_x, w_y, w_z) em função do tempo

Precisam ser especificadas também as posições iniciais do centro de gravidade e orientação do corpo rígido. As atualizações da posição e orientação do centro de gravidade a cada passo de tempo da seguinte forma:

$$\vec{x}_{cg}^{n+1} = \vec{x}_{cg}^{n} + \vec{v}_{cg}\Delta t \tag{45}$$

$$\vec{\theta}_{cg}^{n+1} = \vec{\theta}_{cg}^{n} + G\vec{\Omega}_{cg}\Delta t \tag{46}$$

onde $\vec{x}_{cg}$ e $\vec{\theta}_{cg}$ são a posição e orientação do centro de gravidade, $\vec{v}_{cg}$ e $\vec{\Omega}_{cg}$ são as velocidades linear e angular do centro de gravidade, e G é a matriz transformada que define a escolha do $\underline{\vec{\theta}}$. Por definição, G é a matriz identidade.

Normalmente, $\underline{\vec{\theta}}$ é escolhido para apropriadamente aos ângulos de Euler. Neste caso, o movimento de corpo rígido deve ser especificado usando uma UDF (DEFINE_CG_MOTION) onde a apropriada forma de G pode ser especificado.

A posição dos vetores no corpo rígido são atualizadas baseadas na rotação do vetor da velocidade angular instantânea $\vec{\Omega}_{cg}$. Para um ângulo de rotação finito $\Delta\theta = |\vec{\Omega}_{cg}|\Delta t$, a posição final do vetor $\vec{x}_r$ no corpo rígido para o respectivo $\vec{x}_{cg}$ pode ser expressa pela eq.(47):

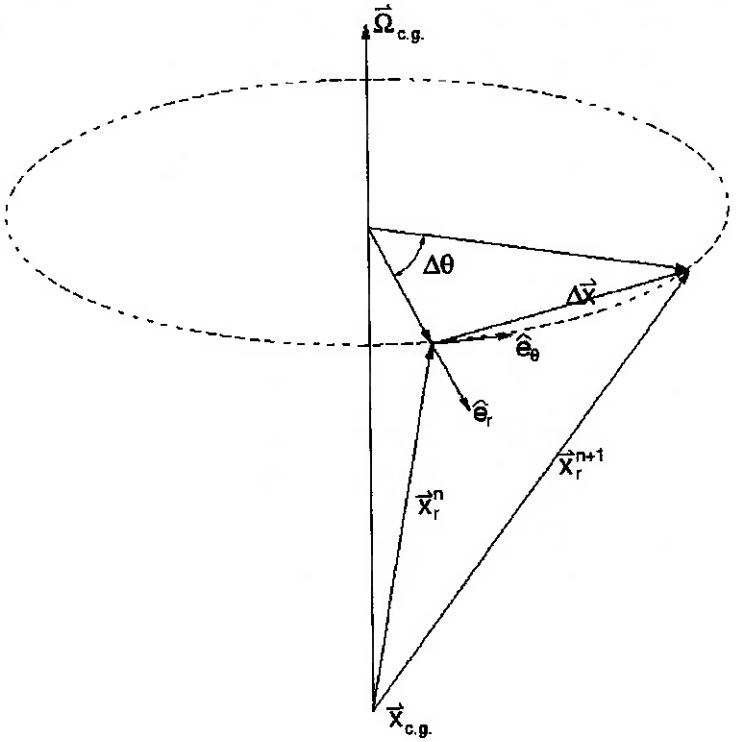

Figura 15: Coordenadas de Rotação de Corpo Rígido.

$$\vec{x}_r^{n+1} = \vec{x}_r^{n} + \Delta\vec{x} \tag{47}$$

onde $\Delta\vec{x}$ é dado por:

$$\Delta\vec{x} = |\vec{x}_r^{n}|[sen(\Delta\theta)\hat{e}_\theta + (\cos(\Delta\theta) - 1)\hat{e}_r] \tag{48}$$

os vetores $\hat{e}_\theta$ e $\hat{e}_r$ são definidos por:

$$\hat{e}_\theta = \frac{\vec{\Omega}_{cg} \times \vec{x}_r}{|\vec{\Omega}_{cg} \times \vec{x}_r|} \tag{49}$$

$$\hat{e}_r = \frac{\hat{e}_\theta \times \vec{\Omega}_{cg}}{|\hat{e}_\theta \times \vec{\Omega}_{cg}|} \tag{50}$$

Se o corpo sólido está se transladando com $\overset{\mathrm{r}}{v}_{cg}$, a posição do vetor $n+1$ do corpo rígido pode ser expresso por:

$$\overset{\mathrm{r}}{x}^{n+1} = \overset{\mathrm{r}}{x}^{n}_{cg} + \overset{\mathrm{r}}{v}_{cg}\Delta t + \overset{\mathrm{r}}{x}^{n+1}_{r} \tag{51}$$

onde $\overset{\mathrm{r}}{x}^{n+1}_{r}$ é dado pela eq.(47).

6 O MODELO VOF

O modelo VOF reside no fato de dois ou mais fluidos (ou fases) não serem miscíveis. Para cada fase adicional que é adicionada ao modelo, uma variável é introduzida: a fração de volume da fase na célula computacional. Para cada célula, a soma das frações de volume de todas as células é unitária. Os campos para todas as variáveis e as propriedades são divididas pelas fases e representam a média nos volumes desses valores, sendo que as frações de volume são conhecidas em cada posição. Sendo assim, as propriedades numa dada célula são representadas por uma das fases ou por uma mistura delas, dependendo dos valores das frações de volume. Em outras palavras, se a q ésima fração de volume na célula é dada por α_q, então três condições são possíveis:

- $\alpha_q = 0$: e a célula está vazia (do q ésimo fluido ou fase).
- $\alpha_q = 1$: e a célula está cheia.
- $0 \leq \alpha_q \leq 1$: e existe uma interface do q ésimo fluido ou fase com os demais.

Com base nos valores de α_q, as propriedades e variáveis serão obtidas para todos os volumes de controle do domínio.

6.1 A equação da fração de volume

O desenho da interface (s) entre as fases é dado em relação à solução da equação da continuidade para as frações de volume de uma (ou mais) fase. Para a q ésima fase, a equação tem a seguinte forma:

$$\frac{\partial \alpha_q}{\partial t} + \vec{v}.\nabla \alpha_q = \frac{S_{\alpha_q}}{\rho_q} \tag{52}$$

O termo fonte, no nosso caso, foi dado como nulo mas é possível a definição de um fluxo de massa para cada fase.

A equação da fração de volume não será resolvida para a primeira fase, ela será calculada como segue:

$$\sum_{q=1}^{n} \alpha_q = 1 \tag{53}$$

6.2 Propriedades

As propriedades que aparecem nas equações de transporte são determinadas pela presença de componentes de cada fase no volume de controle. Num sistema de duas fases, por exemplo, se as fases são representadas pelos índices 1 e 2, e se a fração de volume da segunda está sendo investigada, a densidade em cada célula é dada por:

$$\rho = \alpha_2 \rho_2 + (1 - \alpha_2) \rho_1 \tag{54}$$

Em geral, para um sistema composto de n fases, a média para a densidade na fração de volume é dada por:

$$\rho = \sum \alpha_q \rho_q \tag{55}$$

Todas as demais propriedades (viscosidade, por exemplo) são calculadas da mesma maneira.

6.3 A equação da quantidade de movimento

Uma única equação é resolvida no domínio em questão, e o campo de velocidades resultante e "compartilhado" pelas fases. A equação de quantidade de movimento, mostrada abaixo, é dependente da fração de volume de todas as fases com relação as propriedades ρ e μ.

$$\frac{\partial}{\partial t}(\rho \vec{v}) + \nabla.(\rho \vec{v}\vec{v}) = -\nabla p + \nabla.\left[\mu\left(\nabla \vec{v} + \nabla \vec{v}^{T}\right)\right] + \rho \vec{g} + \vec{F} \tag{56}$$

7 MODELO FÍSICO

Foi utilizado como base dos ensaios numéricos o modelo que representa o FPSO-VLCC Vidal de Negreiros em escala 1:90. O modelo foi ajustado quanto ao momento de inércia e posição do centro de gravidade, para as condições de calado, conforme os dados apresentados a seguir em escala real e para o modelo 1:90.

Tabela 1: Dados do casco em escala real e para modelo

Característica	Real	Modelo
Lpp (m)	320,00	3,556
B (m)	54,54	0,606
D (m)	28,26	0,314

Tabela 2: *Dados do casco em escala real e para o modelo*

CASO	A		B		C	
	Real	Modelo	Real	Modelo	Real	Modelo
H (m)	21,62	0,240	14,76	0,164	7,00	0,078
Hav (m)	21,62	0,240	14,65	0,163	5,94	0,066
Har (m)	21,62	0,240	14,89	0,165	8,25	0,092
P (t, kgf)	321903	430,790	213340	285,509	97229	130,120
LCG (m)	8,97	0,100	11,70	0,130	7,67	0,085
KG (m)	14,49	0,161	14,70	0,163	15,56	0,173
Rxx (m)	18,26	0,203	21,41	0,238	20,70	0,230
Tn (s)	15,89	1,675	17,70	1,866	11,76	1,240

Os cascos são semelhantes do ponto de vista hidrodinâmico, portanto estas diferenças são aceitáveis, e não prejudicam o presente estudo. Convém lembrar que o modelo utilizado foi ajustado para que o valor do calado fosse respeitado, de modo a garantir tal semelhança. As simulações computacionais a realizadas consideram os dados conforme o modelo ensaiado.

Uma das limitações de se compartilhar as aproximações dos campos é que em casos de altas diferenças de velocidades entre as fases os valores das velocidades calculadas próximas a interface podem ser seriamente afetadas.

8 MODELO COMPUTACIONAL

O movimento de "Rolling" de uma FPSO para ângulos pequenos, pode ser modelado como um pêndulo em movimento plano de oscilação, onde a massa do pêndulo estaria concentrada na posição do CG da seção transversal da FPSO. Temos então como centro de rotação, o ponto onde cruzam a linha d'água e a linha que passa pelo CG e cruza a linha d'água perpendicularmente.

No Fluent utilizamos um modelo de malha dinâmica (Dynamic Mesh), que nos permite a manipulação do movimento de corpo rígido do modelo fazendo um reposicionamento dos nós e células da malha (remeshing), juntamente com o modelo VOF (Volume of fluid) que nos permite a manipulação da interface água - ar. Com relação ao modelo de turbulência utilizamos o modelo k – omega SST (Shear – Strees Transport) para a modelagem da turbulência. Todos os modelos enumerados foram brevemente citados nas páginas acima.

Foram utilizadas malhas não estruturadas para o uso do Método dos Volumes Finitos. As malhas foram geradas num software específico, Gambit. Um detalhe muito importante é o fato de que na linha d'água, especificamente, não devem existir elementos da malha que sejam "cortados" por ela, pois isso pode gerar ondas de interferência que podem alterar os resultados ou até mesmo causar a divergência do método já que o modelo VOF interpola a fração de água no interior desses elementos podendo gerar nestes um nível de água acima da linha real.

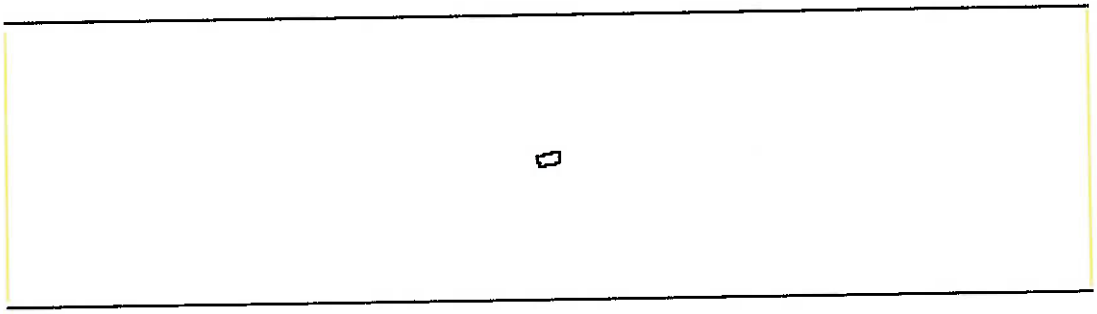

Figura 16: Modelo computacional do tanque de provas numérico

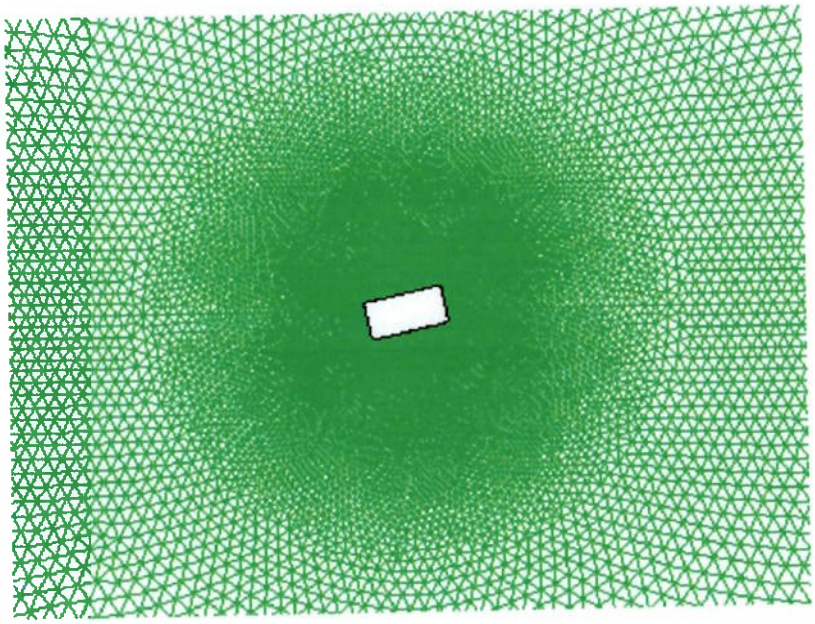

Figura 17: Malha em torno do modelo 2D

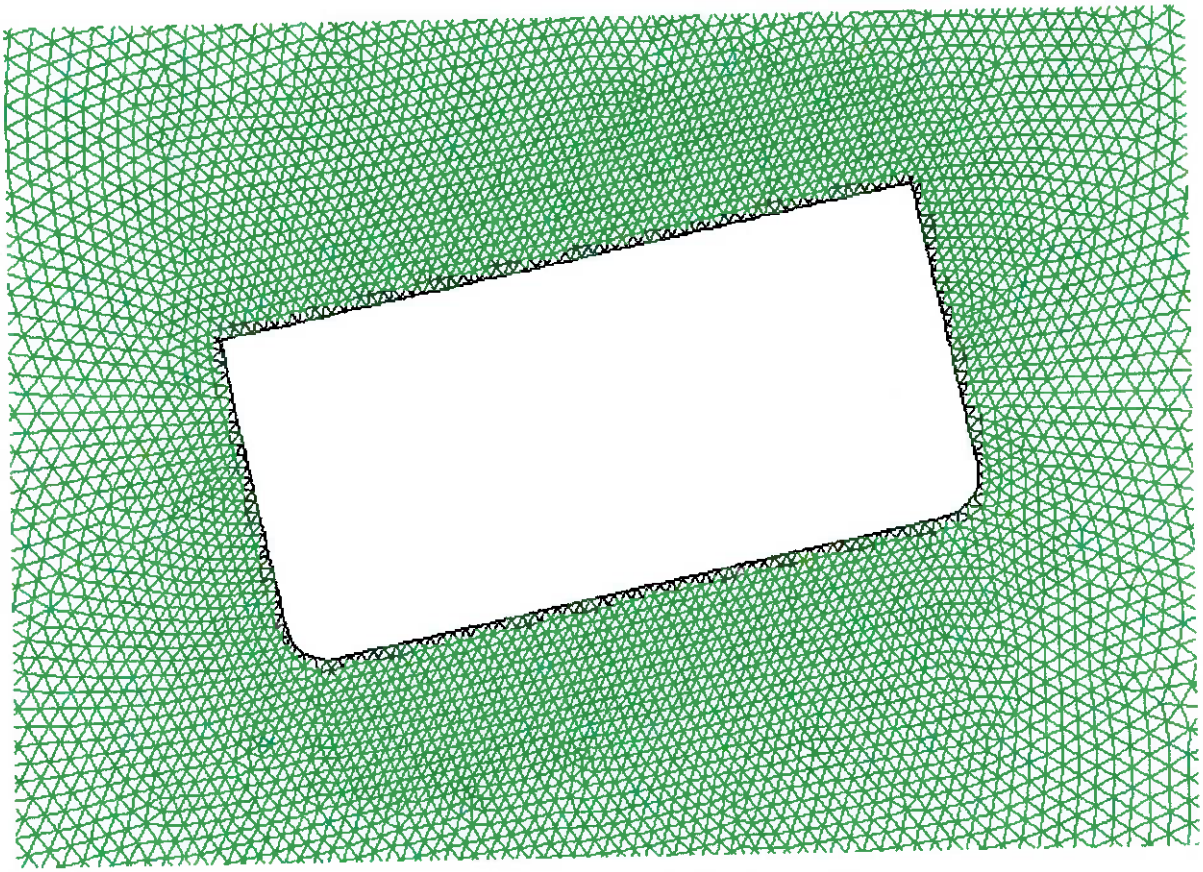

Figura 18: Malha nas proximidades do modelo

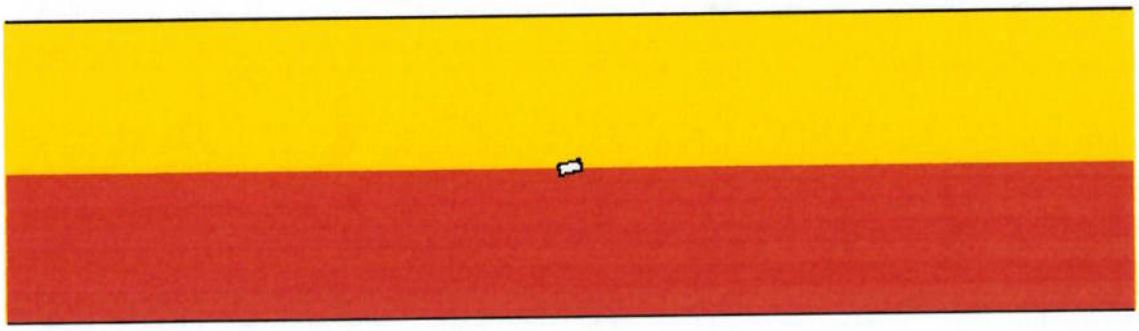

Figura 19: Modelo VOF do tanque numérico

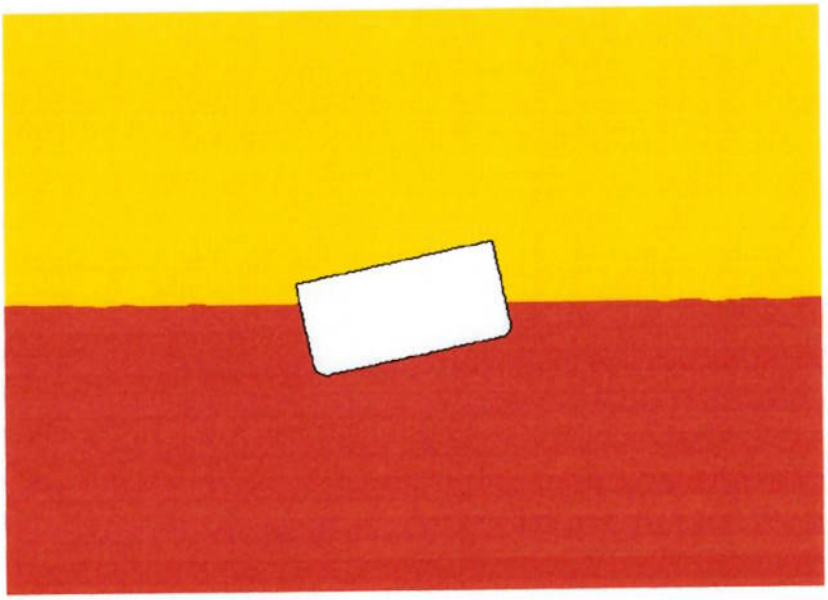

Figura 20: VOF nas proximidades do modelo

9 CÁLCULO ANALÍTICO DO MOMENTO DE RESTAURAÇÃO E FREQÜÊNCIA NATURAL DE OSCILAÇÃO

A freqüência natural de oscilação depende do momento de inércia da embarcação e do momento de restauração do sistema, o primeiro deve ser calculado em relação ao "ponto de articulação" da embarcação enquanto o segundo está relacionado com as características geométricas e dimensões do casco submerso.

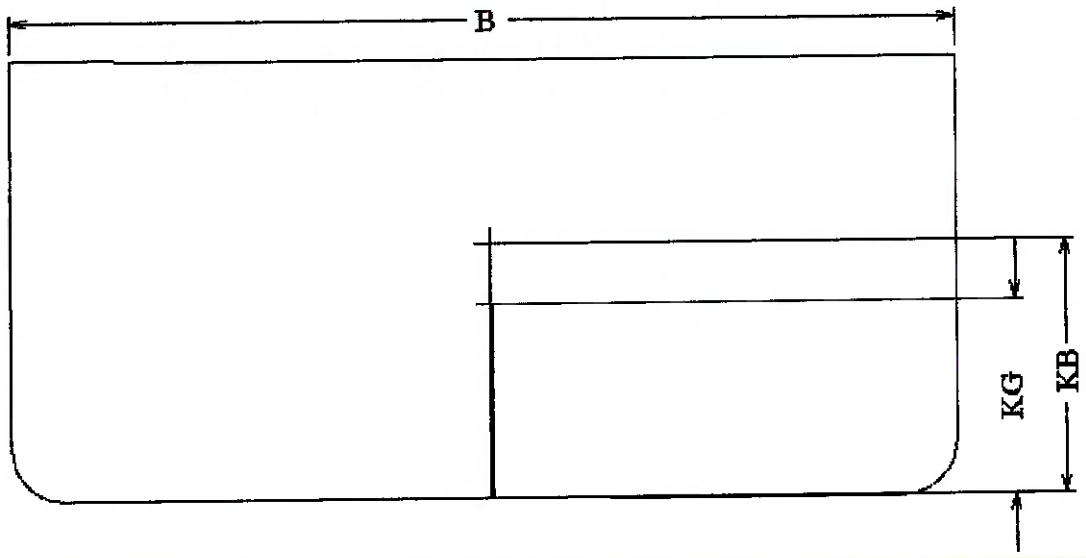

Figura 21: Dimensões do modelo

Dessa forma, é necessário que se conheça o Momento de Inércia J_{total}, o centro de massa (KG) e o centro de flutuação (KB) do modelo, medidos em relação a linha de base do casco. No caso do modelo B sem bolinas temos KG = 0,163m e KB = 0,164 m. O Momento de Inércia pode ser encontrado da seguinte forma:

$$J_{total} = mR_{xx}^2 \tag{57}$$

onde *m* é a massa e R_{xx} é o raio de giração da embarcação. O raio metacêntrico (BM) é calculado em função do momento de inércia da área do casco na linha d'água (J_{WL}) e do volume deslocado através da equação

$$BM = \frac{J_{WL}}{V_{deslocado}} \tag{58}$$

Agora, tendo os parâmetros determinados podemos calcular o *Momento de Restauração* `$R_{rest}$` do modelo, representado na equação

$$R_{rest} = \rho_{agua} V_{deslocado} g\, GM \tag{59}$$

onde

$$GM = KB + BM - KG \tag{60}$$

As propriedades do Modelo B sem bolinas são apresentadas na tabela ***Erro! A origem da referência não foi encontrada.:***

$m = 285{,}509$ Kg
R_{xx}=0,238m
$V_{deslocado} = 0{,}09877\text{m}^3$
$J_{WL} = 0{,}0185046\text{m}^4$
$KG = 0{,}163$m
$KB = 0{,}164$m
$BM = 0{,}18735$m
$GM = 0{,}18835$m
$R_{res} = 182{,}17016\text{N.m}_t$
$J_{total} = 16{,}172\text{Kg.m}^2$

Podemos agora obter a freqüência natural de oscilação do sistema relacionando o Momento de Inércia J_{total} e o Momento de Restauração R_{rest} conforme a equação:

$$\omega_N = \frac{1}{2\pi}\sqrt{\frac{R_{rest}}{J_{total}}} \tag{61}$$

Os resultados analíticos serão então

$\omega = 0,534 rad/s$ e $T = 1,872 s$

Nas simulações realizadas foi utilizado $J_{total} = 15 Kg.m^2$ por se tratar de simulações preliminares.

Para o modelo com bolinas segue um esquema com as principais dimensões da bolina:

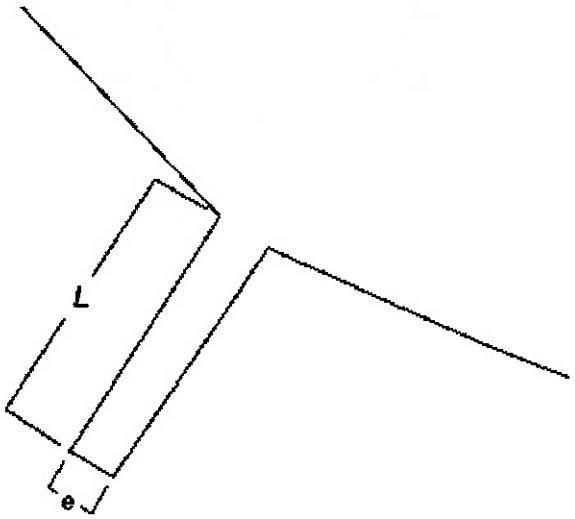

Figura 22: Dimensões da bolina.

Com os seguintes valores para as constantes:

$L = 5,165 mm$

$e = 1 mm$

10 Equacionamento do Movimento de Corpo Rígido usando User Defined Function – UDF

Para a modelagem do movimento de "Rolling" do Navio utilizamos a equação do Momento Angular.

$$I\frac{d^2\theta}{dt^2} = \sum M_o \tag{62}$$

Para a resolução de tal equação em cada instante de tempo foi então utilizado o método de resolução de equações diferencias de Runge-Kutta 4ª ordem (RK4), uma vez que tal método é bastante simples e requer apenas a derivadas de primeira ordem, fornecendo aproximações precisas.

Utilizando o método RK4 teremos então as seguintes equações e condições iniciais para o modelo B e fazendo ma separação de variáveis:

$$\begin{cases} \dfrac{d\theta}{dt} = \omega \\ I_o\dfrac{d\omega}{dt} = \sum M_o \end{cases} \tag{63}$$

$$\{Y\} = \begin{Bmatrix} \theta \\ \omega \end{Bmatrix} \tag{64}$$

Os parâmetros de $\{Y\}$ são repassados para o simulador de CFD que utiliza a velocidade angular para que, em função do intervalo de tempo, o modelo de malha dinâmica (*dynamic mesh*) possa reposicionar a malha a qual sofrerá uma readaptação ("remeshing") a cada iteração.

No cálculo do momento M_o está incluso a parcela correspondente ao amortecimento fluido dinâmico que a cada iteração é calculado pelo Fluent.

11 CÁLCULO DA SOMATÓRIA DOS MOMENTOS EXTERNOS

No cálculo da somatória dos momentos externos são considerados os momentos devido à força peso, empuxo de Arquimedes e forças resultantes da pressão dinâmica.

O momento devido ao peso, juntamente com o momento devido ao empuxo, forma um binário responsável pela restituição da FPSO à sua posição de equilíbrio. Para o cálculo do momento devido ao peso e ao empuxo, basta uma transformação de coordenadas do sistema do corpo para o sistema global utilizado pelo "solver", para então encontrarmos a nova posição do *CG* e do ponto de aplicação do empuxo. No caso do momento devido à pressão dinâmica, são calculadas as componentes da resultante devido à pressão multiplicando-se o vetor área de cada elemento do corpo pela respectiva pressão e então somando as componentes de todas as faces do corpo.

A transformação de coordenadas do sistema fixo (x,y) ao corpo para o sistema global (X,Y) é fornecida pela matriz

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{65}$$

A força devido ao campo de pressões é obtida pela expressão

$$\vec{F} = -\int_A p d\vec{A} \; ; \; -\sum_{i=0}^{n} p_i \vec{A}_i \tag{66}$$

onde n é o número de elementos da superfície do corpo rígido.

12 "STRIP THEORY" (TEORIA DAS TIRAS)

O "strip-theory" é usada desde a década de 50 e pode fornecer estimativas de confiança do desempenho imerso para uma escala muito larga de perfis de casco e de condições de mar. Usando tal teoria, podemos obter uma melhor aproximação do que com apenas uma seção sendo simulada, e não obtendo um tempo computacional proibitivo como em uma simulação tridimensional desse caso.

Método utilizado:

Foram criadas 8 malhas bidimensionais para diferentes seções do casco, selecionadas a partir da carta da nau. O início da simulação se dá com a execução do script "run_simulation" que chama cada um dos demais scripts já nos respectivos nodes onde será executado o solver. A figura demonstra com maior clareza:

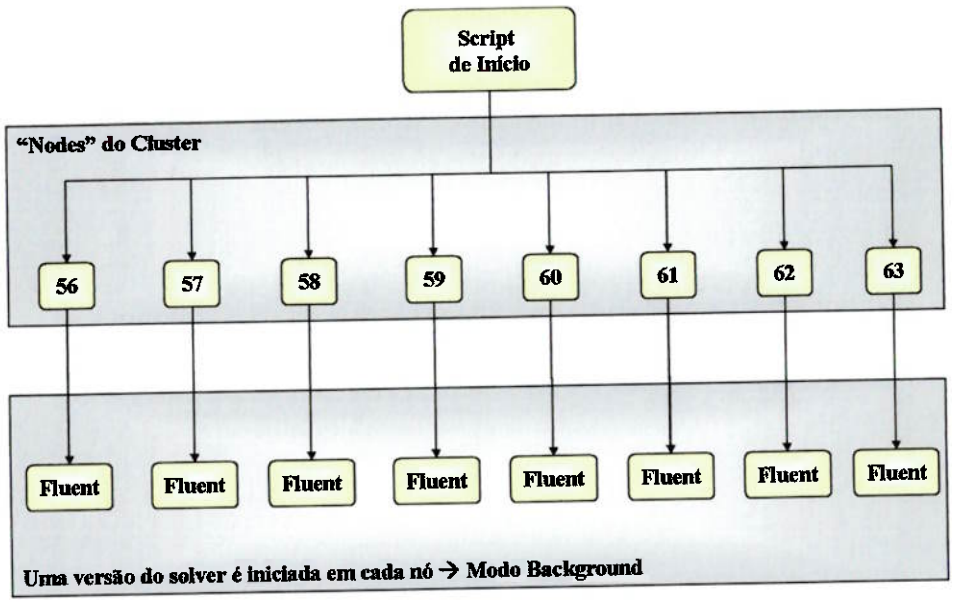

Figura 23: Representação dos scripts

Cada uma das 8 seções fornece o respectivo momento devido à pressão dinâmica calculado pelo solver. Esses momentos serão então considerados ponderando-se o comprimento da nau que cada seção esta representando. Podemos observar o processo da simulação:

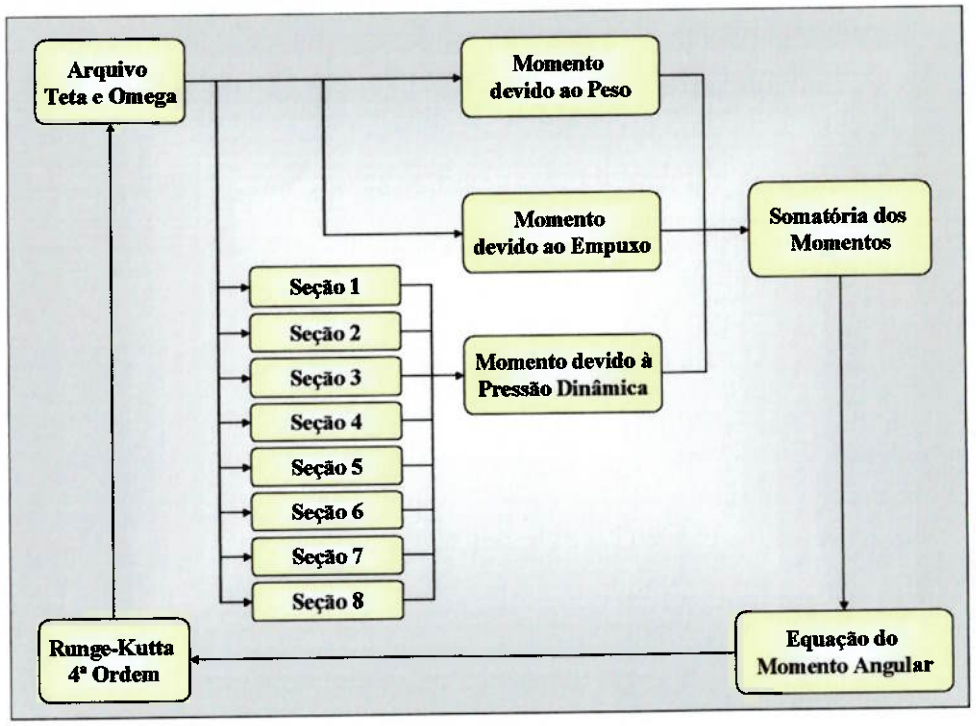

Figura 24: Esquema da simulação com strip - theory

A UDF da seção 1 é a responsável pela sincronia da simulação entre as seções. Isso é feito seguindo as seguintes etapas:

1. A UDF da seção 1 espera que as demais simulações das seções cheguem à convergência ou no limite de iterações determinado para o timestep em questão.
2. Recebe as pressões dinâmicas das demais seções, pondera todas as pressões dinâmicas, inclusive sua própria, e obter assim a pressão dinâmica total no casco.
3. Realiza seu próprio timestep levando em conta a pressão dinâmica total obtida anteriormente.
4. Passa para as demais seções o ângulo de rotação que deve ser aplicado no casco.
5. Caso o tempo de simulação determinado tenha sido alcançado as simulações param, caso contrário volta-se para a etapa 1.

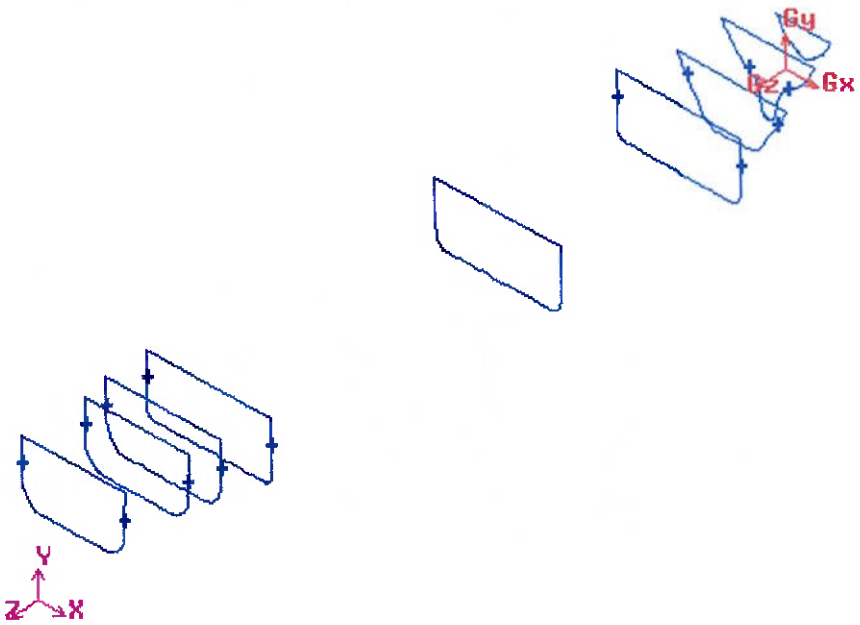

Figura 25: Seções utilizadas e seção de corte

13 RESULTADOS

13.1 Resultados do Ensaio Físico no IPT

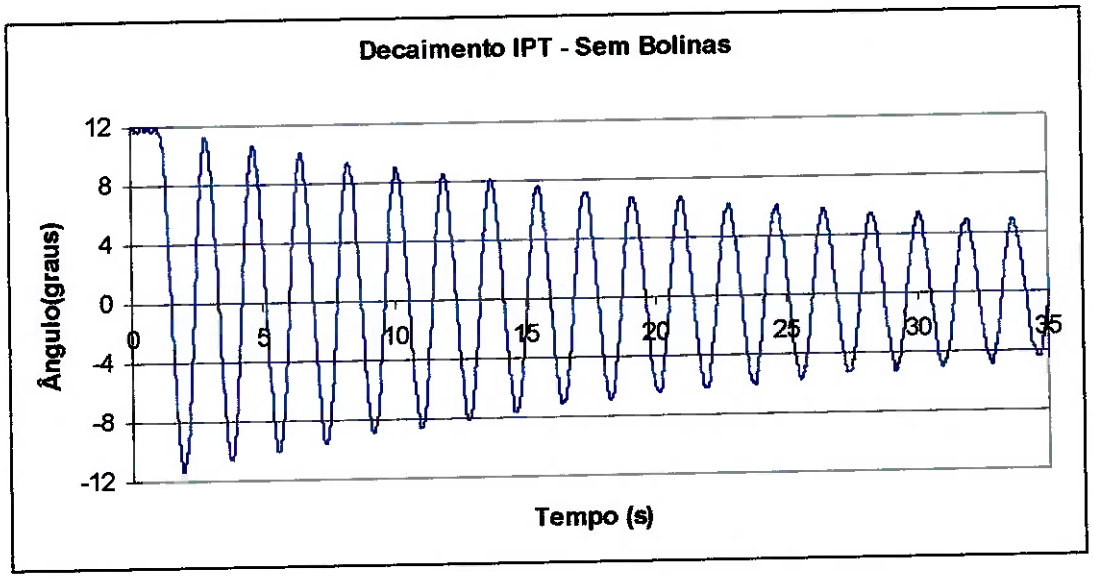

Figura 26: Decaimento do modelo B sem bolinas obtido no IPT

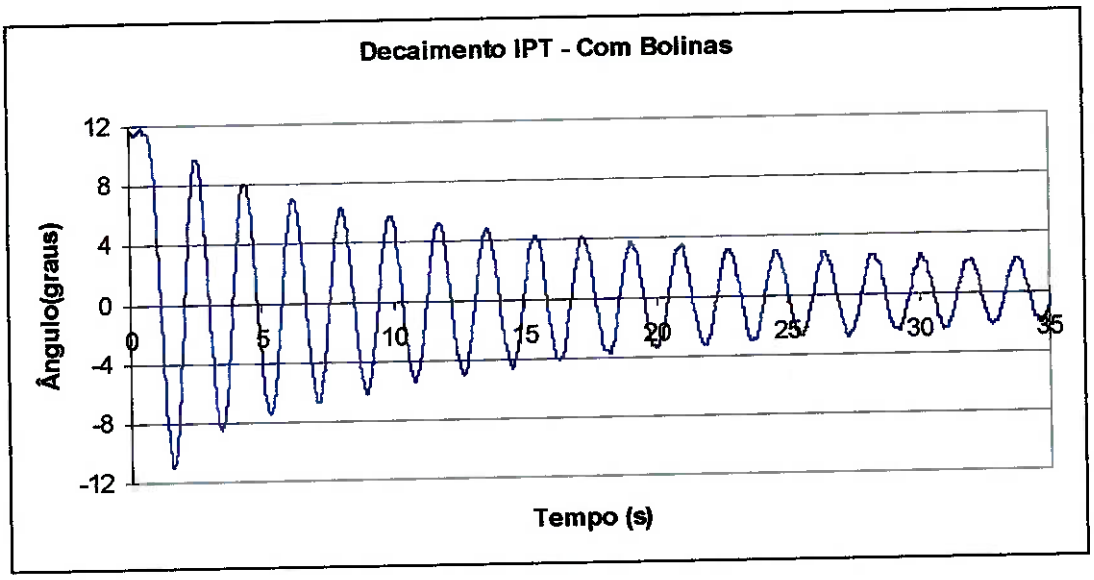

Figura 27: Decaimento do modelo B sem bolinas obtido no IPT

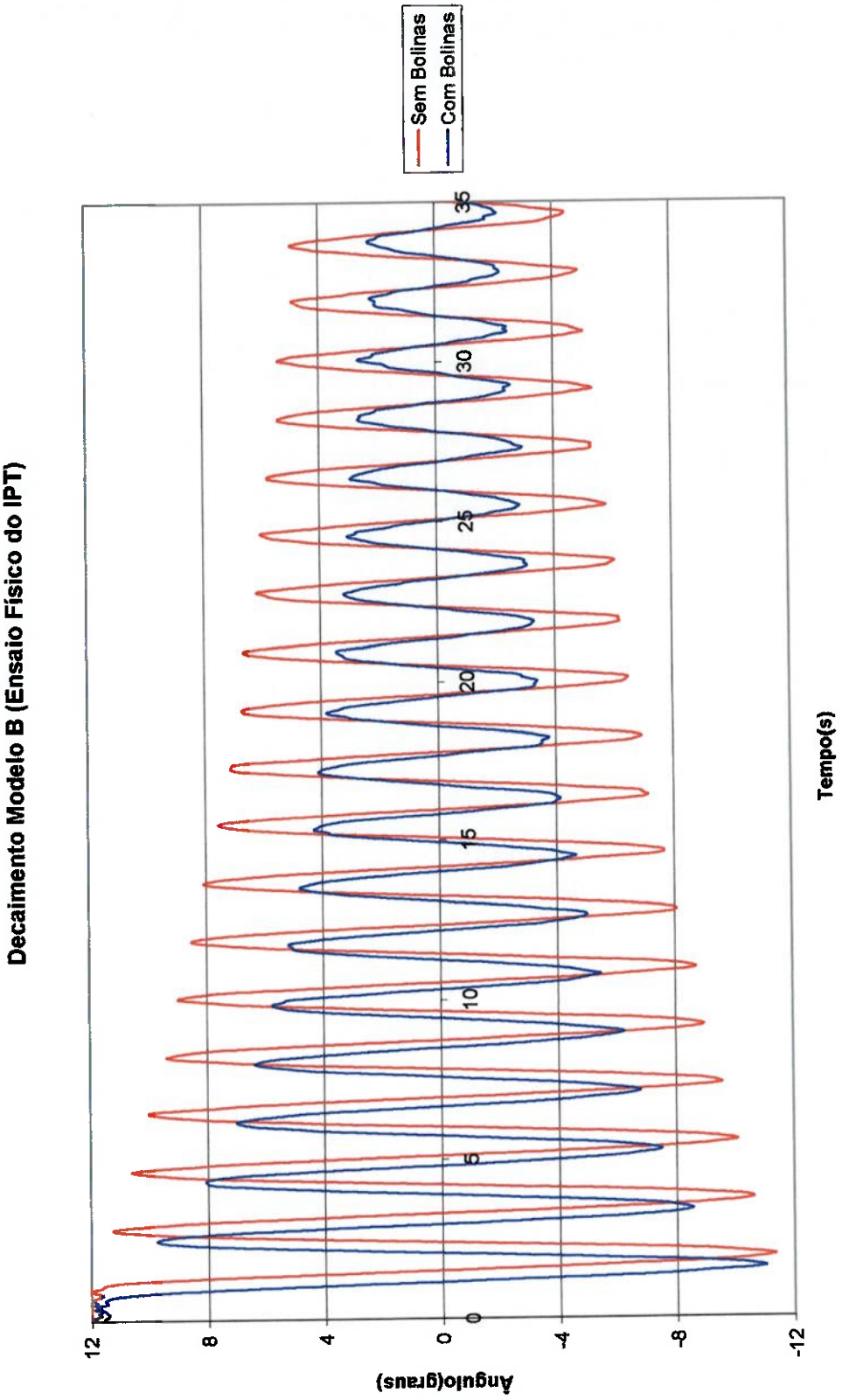

Figura 28: Sobreposição dos decaimentos com e sem bolinas obtidos na simulação Numérica

13.2 Resultados usando Dinâmica dos Fluidos Computacional

Utilizamos para as simulações numéricas os dados referentes ao modelo do caso B de acordo com os moldes apresentados no relatório fornecido pelo IPT. Abaixo podemos verificar os resultados juntamente com algumas ilustrações dos resultados numéricos.

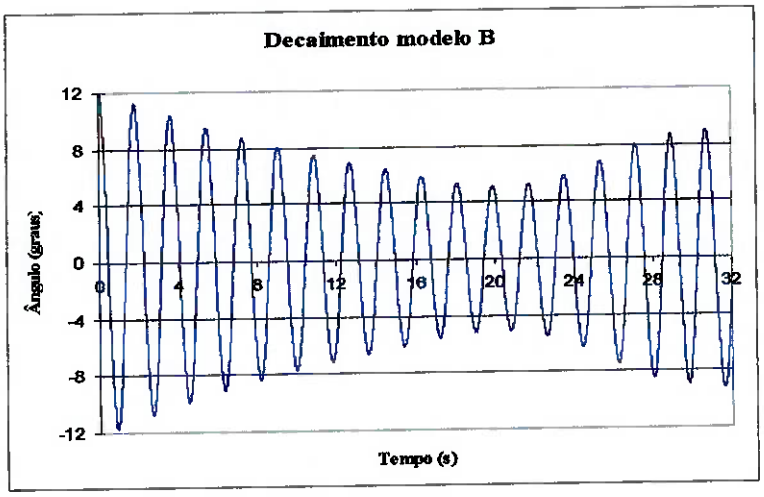

Figura 29: Decaimento do modelo B com 12 graus de inclinação inicial.

Podemos notar que ocorre uma interferência no movimento de "*roll*". Tal interferência pode ser explicada pela largura do "tanque" usado nessas simulações que, diferentemente do tanque de provas utilizado no IPT, não possuía uma dimensão suficiente para evitar a chegada de ondas refletidas nas paredes até o modelo antes do fim do ensaio.

Uma simulação com o modelo totalmente imerso em água foi realizada para o teste da resolução da equação diferencial do movimento e comprovar as possíveis divergências na utilização do modelo VOF – como a reflexão mencionada acima.

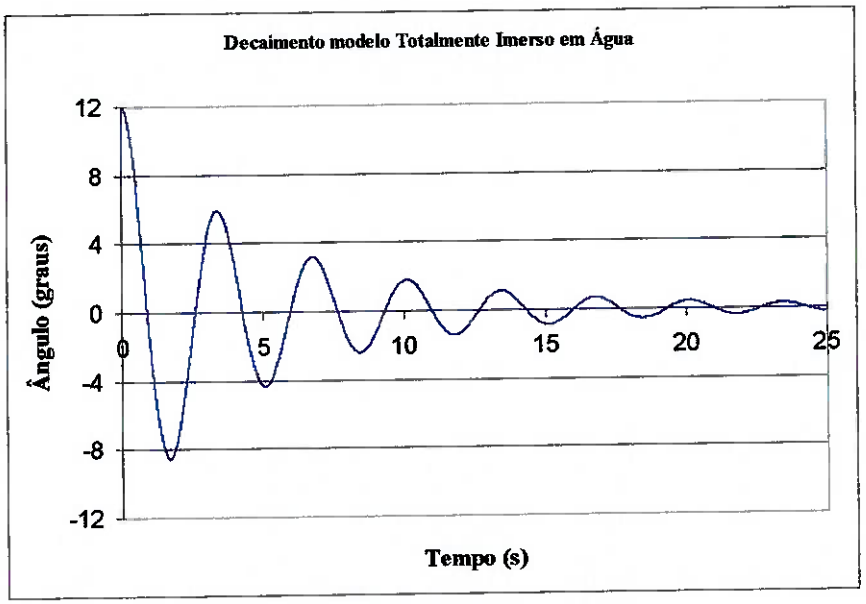

Figura 30: Decaimento do modelo B com 12 graus de inclinação inicial e modelo imerso totalmente em água.

Constatado que na simulação como modelo totalmente imerso não houve interferência de onda, aumentamos a largura do tanque para aproximadamente 100 vezes à dimensão da boca do modelo, neste caso podemos chamar este procedimento de teste de malha o mostrou que o tanque virtual de *60mx8m* era suficiente para as simulações.

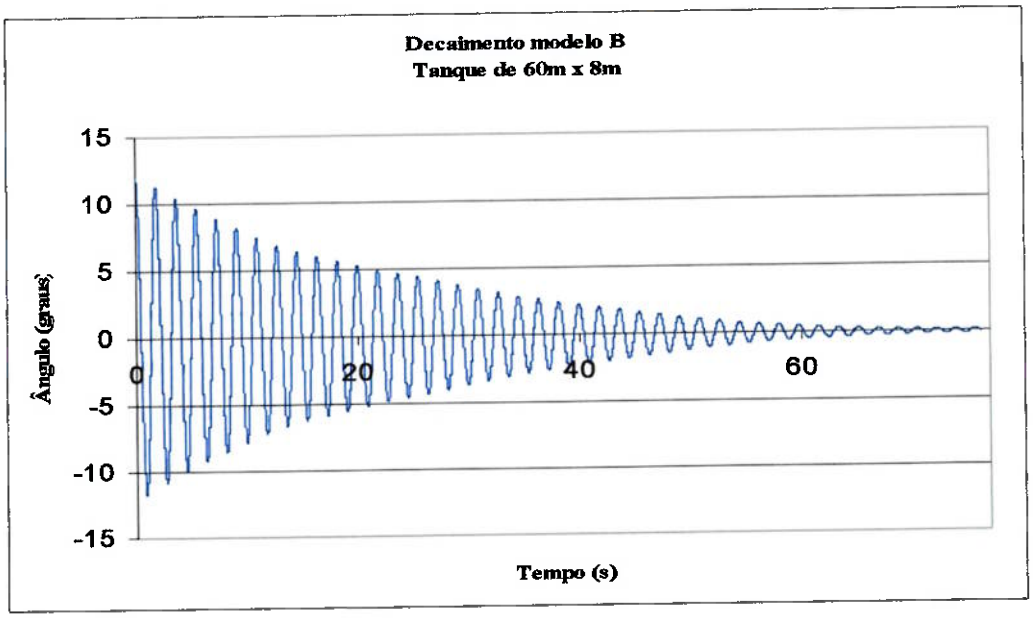

Figura 31: *Decaimento do modelo B com 12 graus de inclinação inicial.*

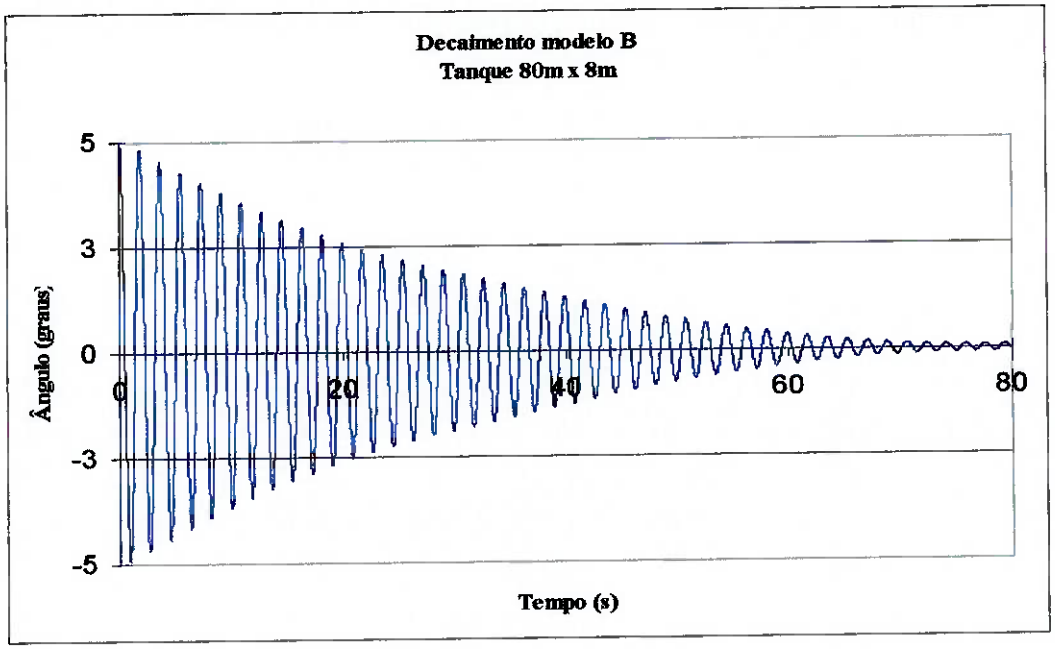

Figura 32: Decaimento do modelo B com 5 graus de inclinação inicial.

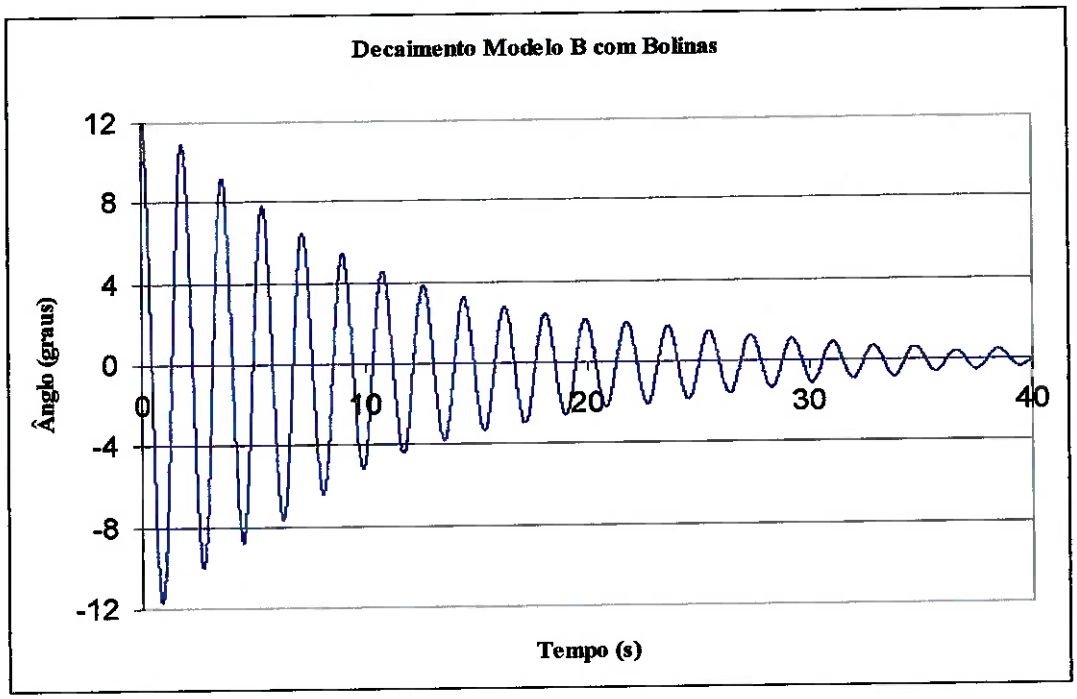

Figura 33: Decaimento do modelo B com bolinas com 12 graus de inclinação inicial.

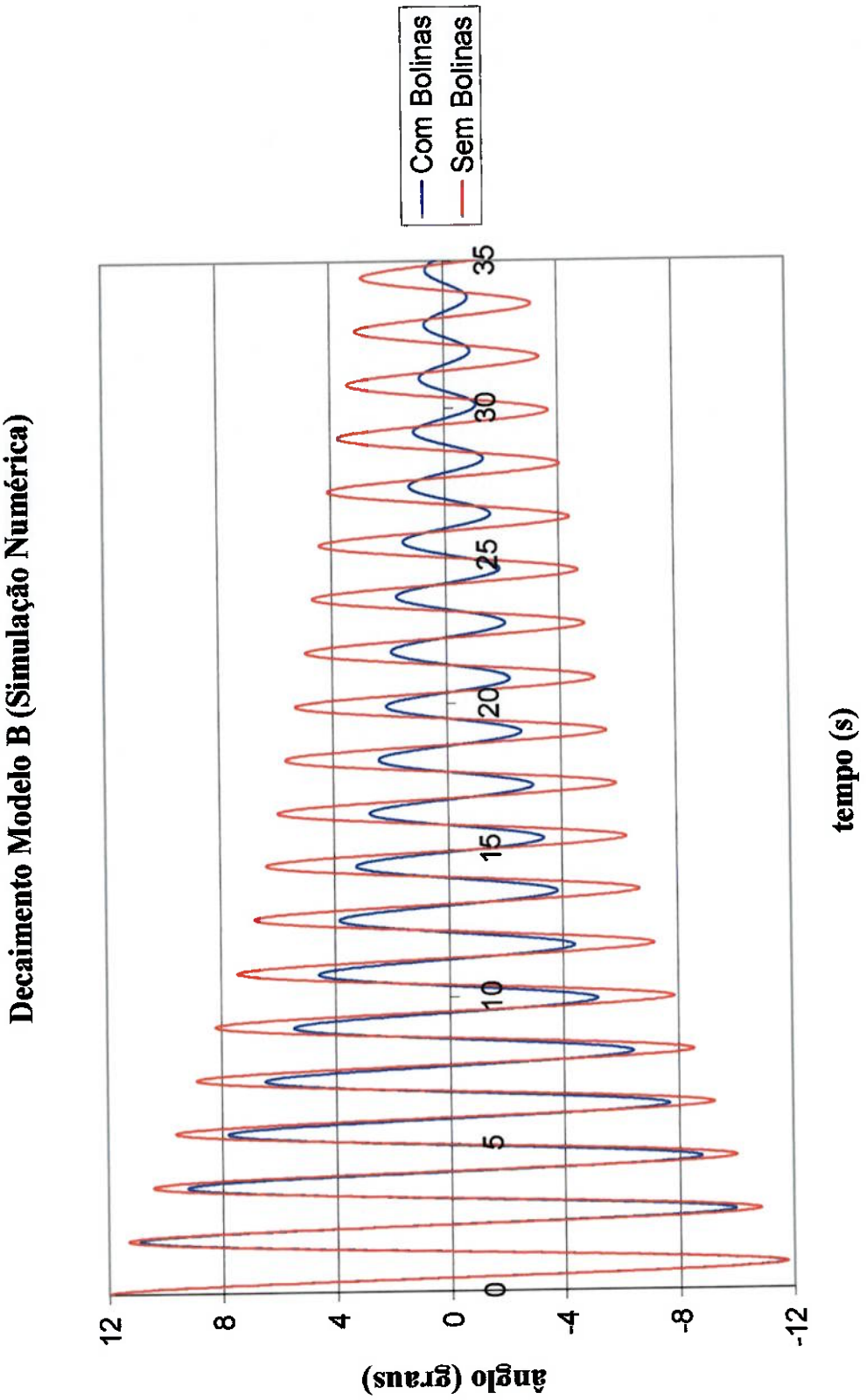

Figura 34: Sobreposição dos decaimentos com e sem bolinas obtidos na simulação Numérica

A análise das séries temporais obtidas resultou nos seguintes valores para a freqüência natural e coeficiente de amortecimento:

Tabela 3: Resultados da simulação numérica para o calado B

Modelo	Freqüência Natural (Hz)	Coef. de Amortecimento
Sem Bolinas IPT(12°)	0.5500	0.007
Sem Bolinas CFD(12°)	0.5371	0.014
Sem Bolinas Strip Theory(12°)	0.7324	0.012
Com Bolinas IPT(12°)	0.5000	0.016
Com Bolinas CFD(12°)	0.5371	0.025
Com Bolinas Strip Theory (12°)	0.7324	0.021

Para uma análise preliminar, este resultado dá uma noção quantitativa da eficiência das bolinas que no caso da seção com 12° de inclinação inicial e com a bolina especificada na modelagem computacional obtivemos um acréscimo no amortecimento de, praticamente, 100% e é devido, principalmente, ao desprendimento de vórtice na estrutura. Abaixo temos uma figura extraída do resultado da simulação que ilustra este fenômeno.

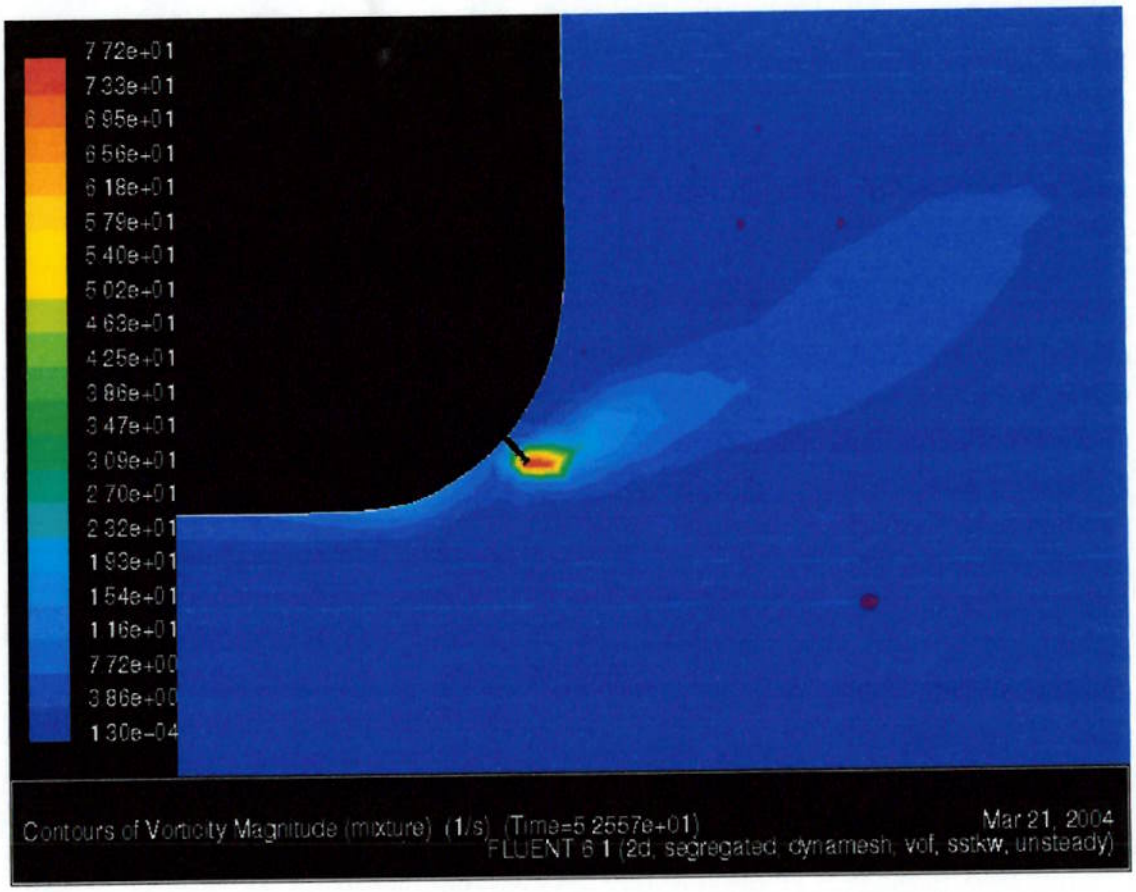

Figura 35: Contornos de vorticidade na bolina para um tempo simulação de aprox. 52.5 segundos.

14 CONCLUSÃO

Os resultados mostram que o amortecimento hidrodinâmico se comporta da mesma maneira nos dois métodos de ensaio se levarmos em consideração a variação na mudança do grau de amortecimento entre os casos estudados neste trabalho. Tanto o ensaio realizado no IPT quanto a simulação numérica apresentaram diferenças não significantes. O fato dos valores de freqüência natural e amortecimento não serem os mesmos dos obtidos experimentalmente se deve, provavemente, à malha utilizada nos cálculos, já que uma malha mais refinada implicaria em um custo computacional inviável para estas simulações.

É importante ressaltar que no caso de uma mudança nas condições de contorno do problema ou até mesmo da geometria o ensaio em CFD apresenta maior flexibilidade em relação ao ensaio tradicional.

Com relação à modelagem com o *strip theory*, notamos que os valores com relação ao aumento do amortecimento hidrodinâmico, para os modelos com e sem bolinas, são qualitativamente os mesmos que foram obtidos no IPT para esse mesmo calado do modelo. Esse fato valida a modelagem numérica. O tempo computacional gasto para cada simulação foi de aproximadamente 3 dias, quando comparado ao tempo necessário para construção dos modelos físicos, faz da CFD uma importante ferramenta para análise da hidrodinâmica do escoamento ao redor de cascos de FPSOs.

ANEXO A – UDF PARA INTEGRAÇÃO DO CAMPO DE PRESSÕES E MOVIMENTO DE CORPO RÍGIDO UMA SEÇÃO TRANVERSAL

```
/*********************************************************
 * Movimento de Corpo rigido
 *
 *    Funcao de movimento do Modelo B
 *
 *********************************************************/

#include "udf.h"
#define  R2D        180./M_PI
#define V_SUB 0.09877       /* m^3 considerando um metro de
profundidade */
#define M_I  15.0                    /* kg*m^2 */
#define M_I_linhadagua 0.0185046  /* m^4 */
#define GAMA 9792.342              /* Kg/(m^2*s^2) */
#define PESO 967.1896                /* N */
#define CALADO 0.164                 /* m  - distancia da linha de
base a linha d'agua*/
#define CG 0.163                     /* m  - distancia da linha de
base ao centro de gravidade */
#define C_Y_empuxo -0.082         /* m  - distancia do centro de
empuxo Y - referencial do corpo */
#define UDF_FILENAME "udf_teta_omega"

/****Vetores Globais******/
real FORCA [ND_ND];     /*Vetor que contem a resultante das forcas
devido a pressao*/
real POLO [ND_ND];      /*Contem as coordenadas do polo de
pressoes*/
real C_EMPUXO [ND_ND];  /*Contem as coordenadas do centro de
empuxo*/
static void calc_c_empuxo(real teta){

      real x;
      real y;
      x = tan(-teta)*M_I_linhadagua/V_SUB;
```

```
      y = C_Y_empuxo;       /* Valor dado em metros com relacao ao
referencial da secao */

      /*matriz de mudanca de base*/
      C_EMPUXO[0]=x*cos(teta)-y*sin(teta);
      C_EMPUXO[1]=x*sin(teta)+y*cos(teta);

}/*calc_c_empuxo*/

static void calcula_forca (Thread *t) {
      face_t f;
      real pressao;
      real A [ND_ND];
      real forca; real forca_x; real forca_y; real forca_x_total=0;
real forca_y_total=0; real forca_total=0;

      /* Calculo da forca devido a pressao */
      begin_c_loop_int(f, t){
            pressao = F_P(f,t);
            F_AREA(A,f,t);
            forca   = pressao*sqrt(((double) A[0]*A[0]+A[1]*A[1]));
            forca_x = pressao*A[0];
            forca_y = pressao*A[1];
            forca_x_total = forca_x_total + forca_x;
            forca_y_total = forca_y_total + forca_y;
            forca_total = forca_total + forca;

      }
      end_c_loop_int(f, t)
      FORCA[0] = forca_x_total;
      FORCA[1] = forca_y_total;
}/*calcula_forca*/

static void polo_pressao (Thread *t){
      face_t f;
      real pressao;
      real A [ND_ND];
      real forca_x; real forca_y;
```

```
        real momento_x; real momento_y; real momento_x_total=0;
    real momento_y_total=0; real momento_total=0;
    real CENTROIDE_FACE [ND_ND];

    /* Calculo do polo de pressao */
    begin_c_loop_int(f, t){
        pressao = F_P(f,t);
        F_CENTROID(CENTROIDE_FACE,f,t);
        F_AREA(A,f,t);
        forca_x = pressao*A[0];
        forca_y = pressao*A[1];
        momento_x = forca_x*CENTROIDE_FACE[1];
        momento_y = forca_y*CENTROIDE_FACE[0];
        momento_x_total = momento_x_total + momento_x;
        momento_y_total = momento_y_total + momento_y;
    }
    end_c_loop_int(f, t)

    calcula_forca(t);
    if (FORCA[0] != 0)
        POLO[1] = momento_x_total/FORCA[0];
    if (FORCA[1] != 0)
        POLO[0] = momento_y_total/FORCA[1];
    printf("\nForca X= %e, Y = %e", FORCA[0], FORCA[1]);
    printf("\nCentro de Pressao X= %e, Y = %e", POLO[0], POLO[1]);

}/*polo_pressao*/

double calcula_momentos (real teta,Thread *t){

    real momento;
    calc_c_empuxo(teta);
    polo_pressao(t);

    momento = FORCA[0]*POLO[1] + FORCA[1]*POLO[0]        /* Momento
devido a pressao */
        + GAMA*V_SUB*C_EMPUXO[0]                  /* Momento devido
ao empuxo */
```

```
            + (CALADO - CG)*sin(teta)*(-PESO);          /* Momento
devido ao peso   */

      printf("\nCentro de Empuxo X= %e, Y = %e", C_EMPUXO[0],
C_EMPUXO[1]);
      printf("\nTeta = %e", teta);
      printf("\nMomento = %e", momento);

      return momento;
}/*calcula_momentos*/

/* read current location and velocity from file */
static void read_teta_omega_file (real *teta, real *omega){

      FILE *fp = fopen(UDF_FILENAME, "r");
      if (fp != NULL){
            float read_teta, read_omega;
            fscanf (fp, "%e %e", &read_teta, &read_omega);
            fclose (fp);
            *teta = (real) read_teta;
            *omega = (real) read_omega;
      }
      else{  /* Para o caso de um arquivo nulo teremos a
      inicializacao da variaveis
                com as condicoes de contorno do problema */
            *teta = M_PI/15;
            *omega = 0.0;
      }
}/*read_teta_omega_file*/

/* write current location and velocity in file */
static void write_teta_omega_file (real teta, real omega){

      FILE *fp = fopen(UDF_FILENAME, "w");
      if (fp != NULL){
            fprintf (fp, "%e %e", teta, omega);
            fclose (fp);
      }
      else
```

```
            Message ("\nWarning: cannot write %s file",
UDF_FILENAME);
}/*write_teta_omega_file*/

/* write current time, location and velocity in file */
static void grava_time_teta_omega (real time, real teta, real
omega){
      FILE *arq = fopen("Teta_Omega.txt", "a");
      if (arq != NULL){
            fprintf (arq, "%e\t%e\t%e\n", time, teta, omega);
            fclose (arq);
      }
      else
            Message ("\nWarning: cannot write %s file",
"Teta_Omega");
}/*grava_time_teta_omega*/

/***********RK4****************/
static void Runge_Kutta(real Yi[], real time, real dtime, Thread
*t){

      real K1[ND_ND]; real K2[ND_ND]; real K3[ND_ND]; real
K4[ND_ND];
      K1[0] = Yi[1];
      K1[1] = calcula_momentos (Yi[0],t)/M_I;
      K2[0] = Yi[1] + dtime/2*K1[1];
      K2[1] = K1[1];
      K3[0] = Yi[1] + dtime/2*K2[1];
      K3[1] = K1[1];
      K4[0] = Yi[1] + dtime*K3[1];
      K4[1] = K1[1];

      Yi[0] = Yi[0] + dtime/6*( K1[0] + 2*K2[0] + 2*K3[0] + K4[0] );
      Yi[1] = Yi[1] + dtime/6*( K1[1] + 2*K2[1] + 2*K3[1] + K4[1] );

}/*Runge_Kutta*/

DEFINE_CG_MOTION(mov_modelo, dt, cg_vel, cg_omega, time, dtime){
```

```
      Thread *t = DT_THREAD (dt);
      face_t f;
      real teta;
      real omega;
      real Yi[ND_ND];

      /* reset velocities */
      NV_S (cg_vel, =, 0.0);
      NV_S (cg_omega, =, 0.0);
      if (!Data_Valid_P ())
            return;
      read_teta_omega_file (&teta, &omega);
      Yi[0] = teta;
      Yi[1] = omega;
      Runge_Kutta(Yi, time, dtime, t);
      /* compute change in velocity */
      teta = Yi[0];
      omega = Yi[1];
      Message ("\nUDF bolinas: time = %f, z_omega = %f, angle(rad)=
%f\n", time, omega, teta);
      write_teta_omega_file (teta, omega);
      grava_time_teta_omega (time, teta, omega);
}/*DEFINE_CG_MOTION*/
```

ANEXO B – UDF PARA INTEGRAÇÃO DO CAMPO DE PRESSÕES E MOVIMENTO DE CORPO RÍGIDO PARA SEÇÃO MESTRA – STRIP THEORY

As linhas de código abaixo implementam a UDF integrante do Strip – Theory. Devemos lembrar que desta dada abaixo para a do modelo 2D com uma única seção devem ser desconsideradas as linhas que levam em consideração as outras seções. Basicamente, as UDFs das seções secundárias somente integram o campo de presssões e nos devolvem o momento devido à pressão dinâmica.

```
/*******************************************************
 * Movimento de Corpo rigido
 *
 * Função de movimento do Modelo B seção mestra
 *
 *******************************************************/

#include "udf.h"
#define  V_SUB          0.3091   /* m^3 considerando um metro de
profundidade */
#define M_I  15.0                /* kg*m^2 */
#define M_I_linhadagua 0.05254   /* m^4 */
#define GAMA 9792.342           /* Kg/(m^2*s^2) */
#define PESO 3026.383            /* N */
#define CALADO 0.164             /* m */
#define CG 0.163                 /* m - comprimento do pêndulo */
#define C_Y_empuxo -0.082        /* m - distancia do centro de empuxo
Y - referencial do corpo */
#define UDF_FILENAME "udf_teta_omega"
/****Vetores Globais******/
real FORCA [ND_ND];      /*Vetor que contem a resultante das forcas
                         devido a
                         pressao*/
real POLO [ND_ND];       /*Contem as coordenadas do polo de
pressoes*/
real C_EMPUXO [ND_ND];   /*Contem as coordenadas do centro de
empuxo*/
```

```
/* Função que calcula o centro de Empuxo dado uma ângulo de
inclinação em  relação ao referencial do corpo */
static void calc_c_empuxo(real teta){
      real x;
      real y;
      x = tan(-teta)*M_I_linhadagua/V_SUB;
      y = C_Y_empuxo;      /* Valor dado em metros com relacao ao
      referencial da    secao */
      /*matriz de mudanca de base*/
      C_EMPUXO[0]=x*cos(teta)-y*sin(teta);
      C_EMPUXO[1]=x*sin(teta)+y*cos(teta);
}/*calc_c_empuxo*/
/* Função que integra o campo de pressões ao redor do corpo, no caso
do modelo do casco */
static void calcula_forca (Thread *t) {
      face_t f;
      real pressao;
      real A [ND_ND];
      real forca; real forca_x; real forca_y; real forca_x_total=0;
real forca_y_total=0; real forca_total=0;
      /* Calculo da forca devido a pressao */
      begin_c_loop_int(f, t){
            pressao = F_P(f,t);
            F_AREA(A,f,t);
            forca   = pressao*sqrt(((double) A[0]*A[0]+A[1]*A[1]));
            forca_x = pressao*A[0];
            forca_y = pressao*A[1];
            forca_x_total = forca_x_total + forca_x;
            forca_y_total = forca_y_total + forca_y;
            forca_total = forca_total + forca;
      }
      end_c_loop_int(f, t)
      /*printf("\nForcas X = %e, Y = %e, F= %e", forca_x_total,
forca_y_total, forca_total);*/
      FORCA[0] = forca_x_total;
      FORCA[1] = forca_y_total;
}/*calcula_forca*/
```

```
/* Dadas as forces resultantes e o campode pressões, calcula o polo
de pressão */
static void polo_pressao (Thread *t){
      face_t f;
      real pressao;
      real A [ND_ND];
      real forca_x; real forca_y;
      real momento_x; real momento_y; real momento_x_total=0; real
momento_y_total=0;
      real CENTROIDE_FACE [ND_ND];
      /* Calculo do polo de pressao */
      begin_c_loop_int(f, t){
            pressao = F_P(f,t);
            F_CENTROID(CENTROIDE_FACE,f,t);
            F_AREA(A,f,t);
            forca_x = pressao*A[0];
            forca_y = pressao*A[1];
            momento_x = forca_x*CENTROIDE_FACE[1];
            momento_y = forca_y*CENTROIDE_FACE[0];
            momento_x_total = momento_x_total + momento_x;
            momento_y_total = momento_y_total + momento_y;
      }
      end_c_loop_int(f, t)
      calcula_forca(t);
      if (FORCA[0] != 0)
            POLO[1] = momento_x_total/FORCA[0];
      if (FORCA[1] != 0)
            POLO[0] = momento_y_total/FORCA[1];
}/*polo_pressao*/

static void escreve_flag (){
      /* Abertura dos arquivos de flags para escrever 1 */
      FILE *fl2 = fopen("../flags/flag2", "w");
      fprintf(fl2, "%d", 0);
      fclose(fl2);
      FILE *fl3 = fopen("../flags/flag3", "w");
      fprintf(fl3, "%d", 0);
      fclose(fl3);
      FILE *fl4 = fopen("../flags/flag4", "w");
```

```
	fprintf(fl4, "%d", 0);
	fclose(fl4);
	FILE *fl5 = fopen("../flags/flag5", "w");
	fprintf(fl5, "%d", 0);
	fclose(fl5);
	FILE *fl6 = fopen("../flags/flag6", "w");
	fprintf(fl6, "%d", 0);
	fclose(fl6);
	FILE *fl7 = fopen("../flags/flag7", "w");
	fprintf(fl7, "%d", 0);
	fclose(fl7);
	FILE *fl8 = fopen("../flags/flag8", "w");
	fprintf(fl8, "%d", 0);
	fclose(fl8);
} /* escreve_flag */

int verifica_flag (char *arquivo){
	int flag = 0;
	/* Abertura dos arquivos de flags para verificacao e leitura
*/
	FILE *fl = fopen(arquivo, "r");
	if (fl != NULL)
		fscanf(fl, "%d", &flag);
	fclose(fl);
	return flag;
} /* verifica_flag */

void verifica_todos (){
	int flag;
	/* Aguarda as secoes subordinadas escreverem "1" no flag */
	printf ("\nAguardando...\n");
	flag=0;
	while (flag == 0){
		flag = verifica_flag ("../flags/flag2");
	}
	printf ("OK!\n");
	flag=0;
	while (flag == 0){
		flag = verifica_flag ("../flags/flag3");
```

```
        }
        printf ("OK!\n");
        flag=0;
        while (flag == 0){
                flag = verifica_flag ("../flags/flag4");
        }
        printf ("OK!\n");
        flag=0;
        while (flag == 0){
                flag = verifica_flag ("../flags/flag5");
        }
        printf ("OK!\n");
        flag=0;
        while (flag == 0){
                flag = verifica_flag ("../flags/flag6");
        }
        printf ("OK!\n");
        flag=0;
        while (flag == 0){
                flag = verifica_flag ("../flags/flag7");
        }
        printf ("OK!\n");
        flag=0;
        while (flag == 0){
                flag = verifica_flag ("../flags/flag8");
        }
        printf ("OK!\n");
}

double le_arquivo (char *nome_arquivo){
        real valor;
        /* Abertura dos arquivo para verificacao e leitura */
        FILE *arq = fopen(nome_arquivo, "r");
        if (arq != NULL)
                        fscanf(arq, "%e", &valor);
        fclose(arq);
        return valor;
} /* le_arquivo */
```

```
/* Faz a ponderação dos momentos de todas as seções baseados no
comprimento característico de cada uma delas */
double calcula_momentos (real teta,Thread *t){
            real momento; real momento_pressao; real momento_sec1;
real momento_sec2; real momento_sec3; real momento_sec4; real
momento_sec5; real momento_sec6; real momento_sec7; real
momento_sec8;
      calc_c_empuxo(teta);
      polo_pressao(t);
      momento_sec2 = le_arquivo ("../momentos/momento_sec2.txt");
      momento_sec3 = le_arquivo ("../momentos/momento_sec3.txt");
      momento_sec4 = le_arquivo ("../momentos/momento_sec4.txt");
      momento_sec5 = le_arquivo ("../momentos/momento_sec5.txt");
      momento_sec6 = le_arquivo ("../momentos/momento_sec6.txt");
      momento_sec7 = le_arquivo ("../momentos/momento_sec7.txt");
      momento_sec8 = le_arquivo ("../momentos/momento_sec8.txt");
      momento_sec1 = FORCA[0]*POLO[1] + FORCA[1]*POLO[0];      /*
      Momento devido  a pressao */
      momento_pressao = (momento_sec1*0.311 + momento_sec2*0.133 +
      momento_sec3*0.2225 + momento_sec4*1.4223 +
      momento_sec5*0.8889 + momento_sec6*0.2666 +
      momento_sec7*0.1778 + momento_sec8*0.1778)/3.5999;
      momento = momento_pressao     /* Momento devido a pressao */
      + GAMA*V_SUB*C_EMPUXO[0]      /* Momento devido ao empuxo */
      + (CALADO - CG)*sin(teta)*(-PESO);/* Momento devido ao peso
      */
      printf("\nMomento = %e", momento);
      return momento;
}/*calcula_momentos*/
/* read current location and velocity from file */
static void read_teta_omega_file (real *teta, real *omega){
      FILE *fp = fopen(UDF_FILENAME, "r");
      if (fp != NULL){
            float read_teta, read_omega;
            fscanf (fp, "%e %e", &read_teta, &read_omega);
            fclose (fp);
            *teta = (real) read_teta;
            *omega = (real) read_omega;
      }
```

```
      else{  /* Para o caso de um arquivo nulo teremos a
inicializacao da variaveis
                  com as condicoes de contorno do problema */
            *teta = M_PI/15;
            *omega = 0.0;
      }
}/*read_teta_omega_file*/
/* escreve a posição e velocidade instantâneas no arquivo */
static void write_teta_omega_file (real teta, real omega){
      FILE *fp = fopen(UDF_FILENAME, "w");
      if (fp != NULL){
            fprintf (fp, "%e %e", teta, omega);
            fclose (fp);
      }
      else
            Message ("\nWarning: cannot write %s file",
UDF_FILENAME);
}/*write_teta_omega_file*/
/* write current time, location and velocity in file */
static void grava_time_teta_omega (real time, real teta, real
omega){
      FILE *arq = fopen("Teta_Omega.txt", "a");
      if (arq != NULL){
            fprintf (arq, "%e\t%e\t%e\n", time, teta, omega);
            fclose (arq);
      }
      else
            Message ("\nWarning: cannot write %s file",
"Teta_Omega");
}/*grava_time_teta_omega*/
/***********RK4***************/
static void Runge_Kutta(real Yi[], real time, real dtime, Thread
*t){
      real K1[ND_ND]; real K2[ND_ND]; real K3[ND_ND]; real
      K4[ND_ND];
      K1[0] = Yi[1];
      K1[1] = calcula_momentos (Yi[0],t)/M_I;
      K2[0] = Yi[1] + dtime/2*K1[1];
      K2[1] = K1[1];
```

```
        K3[0] = Yi[1] + dtime/2*K2[1];
        K3[1] = K1[1];
        K4[0] = Yi[1] + dtime*K3[1];
        K4[1] = K1[1];
        Yi[0] = Yi[0] + dtime/6*( K1[0] + 2*K2[0] + 2*K3[0] + K4[0] );
        Yi[1] = Yi[1] + dtime/6*( K1[1] + 2*K2[1] + 2*K3[1] + K4[1] );
}/*Runge_Kutta*/

DEFINE_CG_MOTION(mov_modelo, dt, cg_vel, cg_omega, time, dtime){
        Thread *t = DT_THREAD (dt);
        /* face_t f;*/
        real teta;
        real omega;
        real Yi[ND_ND];
        /* reset velocities */
        NV_S (cg_vel, =, 0.0);
        NV_S (cg_omega, =, 0.0);
        if (!Data_Valid_P ())
              return;
        /************************************************/
        /**** Verifica se as outras secoes terminaram ***/
        /************************************************/
        verifica_todos ();
        read_teta_omega_file (&teta, &omega);
        Yi[0] = teta;
        Yi[1] = omega;
        Runge_Kutta(Yi, time, dtime, t);
        /* compute change in velocity */
        teta = Yi[0];
        omega = Yi[1];

        Message ("\nUDF bolinas: time = %f, z_omega = %f, angle(rad)=
        %f\n", time, omega, teta);
        write_teta_omega_file (teta, omega);
        grava_time_teta_omega (time, teta, omega);
        cg_omega[2] = omega;
        /* Escreve "0" no flag */
        escreve_flag ();
}/*DEFINE_CG_MOTION*/
```

ANEXO C – UDF PARA INTEGRAÇÃO DO CAMPO DE PRESSÕES E MOVIMENTO DE CORPO RÍGIDO PARA SEÇÕES ESCRAVAS – STRIP THEORY

```
/*******************************************************
 * Movimento de Corpo rigido
 *
 *    Funcao de movimento do Modelo B seção escrava
 *
 *******************************************************/

#include "udf.h"
#define ZONE_ID1   4                /* face ID of the moving object
*/
#define R2D        180./M_PI
#define V_SUB 0.09877                /* m^3 considerando um metro de
profundidade */
#define M_I  15.0                    /* kg*m^2 */
#define M_I_linhadagua 0.0185046     /* m^4 */
#define GAMA 9792.342               /* Kg/(m^2*s^2) */
#define PESO 967.1896                /* N */
#define CALADO 0.164                 /* m  - distancia da linha de
base a linha d'agua*/
#define CG 0.163                     /* m  - distancia da linha de
base ao centro de gravidade */
#define C_Y_empuxo -0.082          /* m  - distancia do centro de
empuxo Y - referencial do corpo */
#define UDF_FILENAME "../1/udf_teta_omega"

/****Vetores Globais******/

real FORCA [ND_ND];     /*Vetor que contem a resultante das forcas
devido a pressao*/
real POLO [ND_ND];      /*Contem as coordenadas do polo de
pressoes*/
real C_EMPUXO [ND_ND];  /*Contem as coordenadas do centro de
empuxo*/

static void calc_c_empuxo(real teta){
```

```
      real x;
      real y;
      x = tan(-teta)*M_I_linhadagua/V_SUB;
      y = C_Y_empuxo;      /* Valor dado em metros com relacao ao
referencial da secao */

      /*matriz de mudanca de base*/
      C_EMPUXO[0]=x*cos(teta)-y*sin(teta);
      C_EMPUXO[1]=x*sin(teta)+y*cos(teta);

}/*calc_c_empuxo*/

static void calcula_forca (Thread *t) {
      face_t f;
      real pressao;
      real A [ND_ND];
      real forca; real forca_x; real forca_y; real forca_x_total=0;
real forca_y_total=0; real forca_total=0;

      /* Calculo da forca devido a pressao */
      begin_c_loop_int(f, t){
            pressao = F_P(f,t);
            F_AREA(A,f,t);
            forca   = pressao*sqrt(((double) A[0]*A[0]+A[1]*A[1]));
            forca_x = pressao*A[0];
            forca_y = pressao*A[1];
            forca_x_total = forca_x_total + forca_x;
            forca_y_total = forca_y_total + forca_y;
            forca_total = forca_total + forca;

      }
      end_c_loop_int(f, t)
      /*printf("\nForcas X = %e, Y = %e, F= %e", forca_x_total,
forca_y_total, forca_total);*/
      FORCA[0] = forca_x_total;
      FORCA[1] = forca_y_total;
}/*calcula_forca*/
```

```
static void polo_pressao (Thread *t){
      face_t f;
      real pressao;
      real A [ND_ND];
      real forca_x; real forca_y;
      real momento_x; real momento_y; real momento_x_total=0; real
momento_y_total=0;
      real CENTROIDE_FACE [ND_ND];

      /* Calculo do polo de pressao */
      begin_c_loop_int(f, t){
            pressao = F_P(f,t);
            F_CENTROID(CENTROIDE_FACE,f,t);
            F_AREA(A,f,t);
            forca_x = pressao*A[0];
            forca_y = pressao*A[1];
            momento_x = forca_x*CENTROIDE_FACE[1];
            momento_y = forca_y*CENTROIDE_FACE[0];
            momento_x_total = momento_x_total + momento_x;
            momento_y_total = momento_y_total + momento_y;
      }
      end_c_loop_int(f, t)

      calcula_forca(t);
      if (FORCA[0] != 0)
            POLO[1] = momento_x_total/FORCA[0];
      if (FORCA[1] != 0)
            POLO[0] = momento_y_total/FORCA[1];
      /*printf("\nMomento X= %e, Y = %e", momento_x_total,
momento_y_total);*/
      /*printf("\nForca X= %e, Y = %e", FORCA[0], FORCA[1]);*/
      /*printf("\nCentro de Pressao X= %e, Y = %e", POLO[0],
POLO[1]);*/

}/*polo_pressao*/

static void escreve_flag (){

      /* Abertura dos arquivos de flags para escrever 1 */
```

```
      FILE *fl = fopen("../flags/flag2", "w");
      fprintf(fl, "%d", 1);
      fclose(fl);
} /* escreve_flag */

int verifica_flag (){

      int flag = 1;

      /* Abertura dos arquivos de flags para verificacao e leitura
*/
      FILE *fl = fopen("../flags/flag2", "r");
      if (fl != NULL)
            fscanf(fl, "%d", &flag);
      fclose(fl);
      return flag;
} /* verifica_flag */

double calcula_momentos (real teta,Thread *t){

      real momento;

      calc_c_empuxo(teta);
      polo_pressao(t);

      momento = FORCA[0]*POLO[1] + FORCA[1]*POLO[0];        /*
Momento devido a pressao */
            /*+ GAMA*V_SUB*C_EMPUXO[0]              /* Momento devido
ao empuxo */
            /*+ (CALADO - CG)*sin(teta)*(-PESO);          /* Momento
devido ao peso   */

      /* printf("\nCentro de Empuxo X= %e, Y = %e", C_EMPUXO[0],
C_EMPUXO[1]);     */
      /*printf("\nTeta = %e", teta);*/
      printf("\nMomento = %e", momento);

      return momento;
}/*calcula_momentos*/
```

```
#if !RP_NODE

/* read current location and velocity from file */
static void read_teta_omega_file (real *teta, real *omega){

      FILE *fp = fopen(UDF_FILENAME, "r");

      if (fp != NULL){

            float read_teta, read_omega;
            fscanf (fp, "%e %e", &read_teta, &read_omega);
            fclose (fp);

            *teta = (real) read_teta;
            *omega = (real) read_omega;
      }

      else{  /* Para o caso de um arquivo nulo teremos a
inicializacao da variaveis
                  com as condicoes de contorno do problema */

            *teta = M_PI/15;
            *omega = 0.0;
      }
}/*read_teta_omega_file*/

/* write current location and velocity in file */
static void write_teta_omega_file (real teta, real omega){

      FILE *fp = fopen(UDF_FILENAME, "w");

      if (fp != NULL){
            fprintf (fp, "%e %e", teta, omega);
            fclose (fp);
      }

      else
```

```
            Message ("\nWarning: cannot write %s file",
UDF_FILENAME);

}/*write_teta_omega_file*/

/* write current time, location and velocity in file (cumulative) */
static void grava_time_teta_omega (real time, real teta, real
omega){

      FILE *arq = fopen("Teta_Omega.txt", "a");

      if (arq != NULL){
            fprintf (arq, "%e\t%e\t%e\n", time, teta, omega);
            fclose (arq);
      }

      else
            Message ("\nWarning: cannot write %s file",
"Teta_Omega");

}/*grava_time_teta_omega*/

/* write current time, location and velocity in file */
static void grava_momento (real momento){

      FILE *arq = fopen("../momentos/momento_sec2.txt", "w");

      if (arq != NULL){
            fprintf (arq, "%e", momento);
            fclose (arq);
      }

      else
            Message ("\nWarning: cannot write %s file",
"momento_sec2");

}/*grava_time_teta_omega*/

#endif /* !RP_NODE */
```

```
DEFINE_CG_MOTION(mov_modelo, dt, cg_vel, cg_omega, time, dtime){

#if !RP_NODE
      Thread *t = DT_THREAD (dt);
      face_t f;
      real teta;
      real omega;
      int flag = 1;

      /* reset velocities */
      NV_S (cg_vel, =, 0.0);
      NV_S (cg_omega, =, 0.0);

      if (!Data_Valid_P ())
            return;

      read_teta_omega_file (&teta, &omega);

      grava_momento (calcula_momentos(teta,t));

      Message ("\nUDF bolinas: time = %f, z_omega = %f, angle(rad)=
%f\n", time, omega, teta);
      /*write_teta_omega_file (teta, omega);*/
      grava_time_teta_omega (time, teta, omega);

      cg_omega[2] = omega;

      /* Escreve "1" no flag */
      escreve_flag ();

      /* Aguarda a secao mestra escrever "0" no flag */
      printf ("\nAguardando a secao mestra...\n");
      while (flag == 1){
            flag = verifica_flag ();

      }
      printf ("OK!\n");
```

```
}/*DEFINE_CG_MOTION*/
```

ANEXO D - UNIX SHELL SCRIPTS PARA INÍCIO DA SIMULAÇÃO

run_simulation

```
# Script para simulacao por Strip Theory V0.3
################################
# Secao 1
################################
# Apagando Arquivos Antigos
rm /home/petrobras/B8sec/1/Teta_Omega.txt
rm /home/petrobras/B8sec/1/udf_teta_omega
rm /home/petrobras/B8sec/1/outputfile
rm -r /home/petrobras/B8sec/1/libudf
#Máquina em que o fluent deve ser executado
rsh node56 /home/petrobras/B8sec/run_node56
################################
# Secao 2
################################
# Apagando Arquivos Antigos
rm /home/petrobras/B8sec/2/Teta_Omega.txt
rm /home/petrobras/B8sec/2/udf_teta_omega
rm /home/petrobras/B8sec/2/outputfile
rm -r /home/petrobras/B8sec/2/libudf
#Máquina em que o fluent deve ser executado
rsh node57 /home/petrobras/B8sec/run_node57
################################
# Secao 3
################################
# Apagando Arquivos Antigos
rm /home/petrobras/B8sec/3/Teta_Omega.txt
rm /home/petrobras/B8sec/3/udf_teta_omega
rm /home/petrobras/B8sec/3/outputfile
```

```
rm -r /home/petrobras/B8sec/3/libudf
#Máquina em que o fluent deve ser executado
rsh node58 /home/petrobras/B8sec/run_node58
################################
# Secao 4
################################
# Apagando Arquivos Antigos
rm /home/petrobras/B8sec/4/Teta_Omega.txt
rm /home/petrobras/B8sec/4/udf_teta_omega
rm /home/petrobras/B8sec/4/outputfile
rm -r /home/petrobras/B8sec/4/libudf
#Máquina em que o fluent deve ser executado
rsh node59 /home/petrobras/B8sec/run_node59
################################
# Secao 5
################################
# Apagando Arquivos Antigos
rm /home/petrobras/B8sec/5/Teta_Omega.txt
rm /home/petrobras/B8sec/5/udf_teta_omega
rm /home/petrobras/B8sec/5/outputfile
rm -r /home/petrobras/B8sec/5/libudf
#Máquina em que o fluent deve ser executado
rsh node60 /home/petrobras/B8sec/run_node60
################################
# Secao 6
################################
# Apagando Arquivos Antigos
rm /home/petrobras/B8sec/6/Teta_Omega.txt
rm /home/petrobras/B8sec/6/udf_teta_omega
rm /home/petrobras/B8sec/6/outputfile
rm -r /home/petrobras/B8sec/6/libudf
#Máquina em que o fluent deve ser executado
```

```
rsh node61 /home/petrobras/B8sec/run_node61
################################
# Secao 7
################################
# Apagando Arquivos Antigos
rm /home/petrobras/B8sec/7/Teta_Omega.txt
rm /home/petrobras/B8sec/7/udf_teta_omega
rm /home/petrobras/B8sec/7/outputfile
rm -r /home/petrobras/B8sec/7/libudf
#Máquina em que o fluent deve ser executado
rsh node62 /home/petrobras/B8sec/run_node62
################################
# Secao 8
################################
# Apagando Arquivos Antigos
rm /home/petrobras/B8sec/8/Teta_Omega.txt
rm /home/petrobras/B8sec/8/udf_teta_omega
rm /home/petrobras/B8sec/8/outputfile
rm -r /home/petrobras/B8sec/8/libudf
#Máquina em que o fluent deve ser executado
rsh node63 /home/petrobras/B8sec/run_node63
```

run_node56

```
################################
# Secao 1
################################
cd /home/petrobras/B8sec/1
#Iniciando Simulacao em Background
/usr/Fluent.Inc/bin/fluent 2d -g < /home/petrobras/B8sec/1/script_fluent01 >
outputfile 2>&1 &
```

run_node57

```
################################
# Secao 2
################################
cd /home/petrobras/B8sec/2
#Iniciando Simulacao em Background
/usr/Fluent.Inc/bin/fluent 2d -g < /home/petrobras/B8sec/2/script_fluent01 >
outputfile 2>&1 &
```

run_node58

```
################################
# Secao 3
################################
cd /home/petrobras/B8sec/3
#Iniciando Simulacao em Background
/usr/Fluent.Inc/bin/fluent 2d -g < /home/petrobras/B8sec/3/script_fluent01 >
outputfile 2>&1 &
```

run_node59

```
################################
# Secao 4
################################
cd /home/petrobras/B8sec/4
#Iniciando Simulacao em Background
/usr/Fluent.Inc/bin/fluent 2d -g < /home/petrobras/B8sec/4/script_fluent01 >
outputfile 2>&1 &
```

run_node60

```
################################
# Secao 5
################################
cd /home/petrobras/B8sec/5
#Iniciando Simulacao em Background
/usr/Fluent.Inc/bin/fluent 2d -g < /home/petrobras/B8sec/5/script_fluent01 >
outputfile 2>&1 &
```

run_node61

```
################################
# Secao 6
################################
cd /home/petrobras/B8sec/6
#Iniciando Simulacao em Background
/usr/Fluent.Inc/bin/fluent 2d -g < /home/petrobras/B8sec/6/script_fluent01 >
outputfile 2>&1 &
```

run_node62

```
################################
# Secao 7
################################
cd /home/petrobras/B8sec/7
#Iniciando Simulacao em Background
/usr/Fluent.Inc/bin/fluent 2d -g < /home/petrobras/B8sec/7/script_fluent01 >
outputfile 2>&1 &
```

run_node63

```
################################
# Secao 8
```

```
################################
cd /home/petrobras/B8sec/8
#Iniciando Simulacao em Background
/usr/Fluent.Inc/bin/fluent 2d -g < /home/petrobras/B8sec/8/script_fluent01 >
outputfile 2>&1 &
```

ANEXO E - SCRIPTS DE INICIALIZAÇÃO DO FLUENT EM BACKGROUND

```
;Script de Setup e Simulacao do Fluent para bolinas
;Lendo o case
rc ./sec_modelo_maior.cas
;Compile UDF
define/user-defined/compiled-functions compile libudf , mov_modelo1b.c , ,
;Load UDF
define/user-defined/compiled-functions load libudf
;Definindo Dynamic Zones
define/dynamic-zones/create sec1 solid-body-motion mov_modelo , , , 12 ,
;Marcando a regiao com agua
adapt/mark-inout-rectangle , , -40 40 -10 0
;Initialize
solve/initialize/initialize-flow
;Escolha do Time-Step
solve/set/time-step 0.001
;Patch
solve/patch agua , 0 , mp 1
;Iterando - parametro1: time steps parametro2: max it por time step
solve/dual-time-iterate 40000 100
;Escrevendo Dados
wd ./sec_modelo_maior.dat
yes
;Saindo do Fluent
exit
yes
```

ANEXO F - RELATÓRIO DE MODELOS E CONSTANTES USADAS NO FLUENT CFD

FLUENT
Version: 2d, segregated, dynamesh, vof, sstkw, unsteady (2d, segregated, dynamic mesh, VOF, SST k-omega, unsteady)
Release: 6.1.22
Title:

Models

Model	Settings
Space	2D
Time	Unsteady, 1st-Order Implicit
Viscous	k-omega turbulence model
Heat Transfer	Disabled
Solidification and Melting	Disabled
Species Transport	Disabled
Coupled Dispersed Phase	Disabled
Pollutants	Disabled
Soot	Disabled

Boundary Conditions

Zones

name	id	type
fluid	2	fluid
wall	3	wall
parede_e	4	symmetry
parede_d	5	symmetry
modelo	6	wall
default-interior	8	interior

Boundary Conditions

fluid

Condition	Value
Material Name	air
Specify source terms?	no
Source Terms	()
Specify fixed values?	no
Fixed Values	()
Motion Type	0
X-Velocity Of Zone	0
Y-Velocity Of Zone	0
Rotation speed	0
X-Origin of Rotation-Axis	0
Y-Origin of Rotation-Axis	0
Deactivated Thread	no
Laminar zone?	no
Set Turbulent Viscosity to zero within laminar zone?	yes
Porous zone?	no
Porosity	1

wall

Condition	Value
Wall Motion	0
Shear Boundary Condition	0
Define wall motion relative to adjacent cell zone?	yes
Apply a rotational velocity to this wall?	no
Velocity Magnitude	0
X-Component of Wall Translation	1
Y-Component of Wall Translation	0
Define wall velocity components?	no
X-Component of Wall Translation	0
Y-Component of Wall Translation	0

Wall Roughness Height	0
Wall Roughness Constant	0.5
Rotation Speed	0
X-Position of Rotation-Axis Origin	0
Y-Position of Rotation-Axis Origin	0
X-component of shear stress	0
Y-component of shear stress	0

parede_e

Condition	Value

parede_d

Condition	Value

modelo

Condition	Value
Wall Motion	0
Shear Boundary Condition	0
Define wall motion relative to adjacent cell zone?	yes
Apply a rotational velocity to this wall?	no
Velocity Magnitude	0
X-Component of Wall Translation	1
Y-Component of Wall Translation	0
Define wall velocity components?	no
X-Component of Wall Translation	0
Y-Component of Wall Translation	0
Wall Roughness Height	0
Wall Roughness Constant	0.5
Rotation Speed	0
X-Position of Rotation-Axis Origin	0
Y-Position of Rotation-Axis Origin	0
X-component of shear stress	0
Y-component of shear stress	0

default-interior

Condition	Value

Solver Controls

Equations

Equation	Solved
Flow	yes
Volume Fraction	yes
Turbulence	yes

Numerics

Numeric	Enabled
Absolute Velocity Formulation	yes

Unsteady Calculation Parameters

Time Step (s)	0.01
Max. Iterations Per Time Step	50

Relaxation

Variable	Relaxation Factor
Pressure	1
Density	1
Body Forces	1
Momentum	1
Volume Fraction	1
Turbulence Kinetic Energy	1

Specific Dissipation Rate	1
Turbulent Viscosity	1

Linear Solver

Variable	Solver Type	Termination Criterion	Residual Reduction Tolerance
Pressure	V-Cycle	0.1	
X-Momentum	Flexible	0.1	0.7
Y-Momentum	Flexible	0.1	0.7
Volume Fraction	Flexible	0.1	0.7
Turbulence Kinetic Energy	Flexible	0.1	0.7
Specific Dissipation Rate	Flexible	0.1	0.7

Discretization Scheme

Variable	Scheme
Pressure	Body Force Weighted
Pressure-Velocity Coupling	PISO
Momentum	First Order Upwind
Turbulence Kinetic Energy	First Order Upwind
Specific Dissipation Rate	First Order Upwind

Solution Limits

Quantity	Limit
Minimum Absolute Pressure	1
Maximum Absolute Pressure	5000000
Minimum Temperature	1
Maximum Temperature	5000
Minimum Turb. Kinetic Energy	1e-14
Minimum Spec. Dissipation Rate	1e-20
Maximum Turb. Viscosity Ratio	10000

Material Properties

Material: water-liquid (fluid)

Property	Units	Method	Value(s)
Density	kg/m3	constant	998.2
Cp (Specific Heat)	j/kg-k	constant	4182
Thermal Conductivity	w/m-k	constant	0.6
Viscosity	kg/m-s	constant	0.001003
Molecular Weight	kg/kgmol	constant	18.0152
L-J Characteristic Length	angstrom	constant	0
L-J Energy Parameter	k	constant	0
Thermal Expansion Coefficient	1/k	constant	0
Degrees of Freedom		constant	0

Material: air (fluid)

Property	Units	Method	Value(s)
Density	kg/m3	constant	1.225
Cp (Specific Heat)	j/kg-k	constant	1006.43
Thermal Conductivity	w/m-k	constant	0.0242
Viscosity	kg/m-s	constant	1.7894e-05
Molecular Weight	kg/kgmol	constant	28.966
L-J Characteristic Length	angstrom	constant	3.711
L-J Energy Parameter	k	constant	78.6
Thermal Expansion Coefficient	1/k	constant	0
Degrees of Freedom		constant	0

Material: aluminum (solid)

Property	Units	Method	Value(s)
Density	kg/m3	constant	2719
Cp (Specific Heat)	j/kg-k	constant	871

Thermal Conductivity	w/m-k	constant	202.4

LISTA DE REFERÊNCIAS

[1] Anagnostopoulos, P. and Bearman, P.W., Response characteristics of a vortex-excited cylinder at low Reynolds number, *J.Fluids and Structures*, Vol 6, pp39-50, 1992.

[2] Ássi, G.R.S., Desenvolvimento de um canal de água circulante para experimentos em dinâmica dos fluidos; XIX Congresso de Iniciação Cientifica em Engenharia; São Carlo, 2003

[3] Auada, R.B., and Meneghini, J.R. "Unstructured Mesh Generation: quality enhancement through smooting", "Sixth Latin American Congress of Heat and Mass Transfer/Latcym-96".

[4] Cozens, P. "Numerical modelling of the roll damping of ships due to vortex shedding," PhD Thesis, Imperial College, 1987.

[5] Graham, J.M.R. "Computation of viscous separated flow using a particle method," Numerical Methods in Fluid Mechanics, Vol 3, 1988.

[8] Meneghini, J.R. "Projeto de Pesquisa no tópico geração e desprendimento de vórtices no escoamento ao redor de cilindros" Tese de Prof. Livre Docente, EPUSP, 2002.

[9] Meneghini, J.R. and Bearman, P.W., Numerical simulation of control of bluff body flow using a discrete vortex method incorporating viscous diffusion, *Proc IUTAM Sym on Bluff Body Wakes*, Dynamics and Instabilities, Gottingen, 1992.

[10] Meneghini, J.R. and Bearman, P.W., "Numerical simulation of high amplitude oscillatory flow about a circular cylinder using a discrete vortex method." AIAA paper 93-3288, 1993.

[11] Meneghini, J.R., and Bearman, P.W. "Numerical simulation of high amplitude oscillatory flow about a circular cylinder," *Journal of Fluids and Structures*, Academic Press, vol.9, pp. 435-455, 1995.

[12] Meneghini, J.R., and Bearman, P.W. An investigation of the effect on vortex shedding of a sudden transverse disturbance applied to a circular cylinder," *Journal of Wind Engineering and Industrial Aerodynamics*, vol 69-71, pp. 229-238, 1997.

[13] Meneghini, J.R., Siqueira, C.L.R., and Bearman, P.W. "Numerical Simulation of the Effect of Wave Form and Sudden Transverse Displacement on Vortex Shedding from a circular Cylinder," *American Institute of Aeronautics and Astronautics* AIAA Paper 97-2006, pp. 1-12, 1997.

[14] Saltara, F., Meneghini, J.R., and Bearman, P.W. "Numerical simulation of vortex shedding from an oscillating circular cylinder", Computational Methods and Experimental Measurements VIII, ed. P. Anagnostopoulos et al., Computational Mechanics Publications, UK, pp. 409-418, 1997.

[15] Saltara, F., Meneghini, J.R., and Siqueira, C.L.R., "The Simulation of Vortex Shedding from an Oscillating Circular Cylinder", em co-autoria com C. L. R. Siqueira, e F. Saltara, Publicado nos "Proceedings of The Eight (1998) International Offshore and Polar Engineering Conference-ISOPE-98", Volume III, pp.356-363, Montreal, Canada, May 24-29, 1998a.

[16] Saltara, F., Meneghini, J.R., and Siqueira, C.L.R., "The Simulation of Vortex Shedding from an Oscillating Circular Cylinder with Turbulence Modelling", em co-autoria com F. Saltara e C. Siqueira, Publicado nos Proceedings of FEDSM'98, 1998 ASME Fluids Engineering Division Summer Meeting, June 21-25, 1998, Washington, D.C., 1998b.

[17] Siqueira, C.L.R., Meneghini, J.R., and Saltara, F., "Numerical Simulation of a Fixed and an Oscillating Circular Cylinder in Oscillatory Flow", em co-autoria com C. L. R. Siqueira, e F. Saltara, Publicado nos "Proceedings of The Eight (1998) International Offshore and Polar Engineering Conference-ISOPE-98", Volume III, pp.364-371, Montreal, Canada, May 24-29, 1998a.

[18] Siqueira, C.L.R., Meneghini, J.R., and Saltara, F., "An Investigation of Vortex-Induced Vibration of a Circular Cylinder in Water", em co-autoria com A. Fujarra, C. P. Pesce, e P. Parra, Publicado nos Proceedings of FEDSM'98, 1998 ASME Fluids Engineering Division Summer Meeting, June 21-25, Washington, D.C., 1998b.

[20] Summer, D., Price, S.J. & Païdoussis, M.P., Investigation of Side-by-Side Circular Cylinders in Steady-Flow by Particule Image Velocimetry, *1998 Conference on Bluff Body Wakes and Vortex-Induced Vibration, ASME Summer Meeting*, Washington, DC, 1998.

[21] White, F. M., Mecânica dos Fluidos. McGraw-Hill. Rio de Janeiro, 2002.

[22] Zdravkovich, M.M., The Effects of Interference Between Circular Cylinders in Cross Flow, Journal of Fluids and Structures, v. 1, 239-261, 1987.