UMA PLATAFORMA COMPUTACIONAL PARA VISUALIZAR SISTEMAS

Celso K Morooka[1] (Unicamp), Dustin M. Brandt[2] (Unicamp),
Vitor de Lima[3] (Unicamp), Vinícius M. Rodrigues[4] (Unicamp)

Departamento de Engenharia do Petróleo, Faculdade de Engenharia Mecânica
Universidade de Campinas, Cx. P. 6122 Campinas, SP 13083-970

[1]morooka@dep.fem.unicamp.br , [2]dustin@dep.fem.unicamp.br
[3]vitor@dep.fem.unicamp.br, [4]vinicius@dep.fem.unicamp.br

Este trabalho apresenta uma plataforma computacional para facilitar a visualização de sistemas marítimos de explotação. O sistema *offshore* de produção de petróleo é formado por diversos componentes tais como plataformas, *risers*, poços, linhas de ancoragens e outros. Existe, portanto, a necessidade de visualizar este complexo sistema reproduzindo-se sua tridimensionalidade. Assim, o sistema de visualização computacional em 3-D, que será apresentado, tem propósito de ajudar o engenheiro a entender melhor todo o sistema. A aplicação foi feita na linguagem C++ com a biblioteca OpenGL e é baseado em programação modular. Programação modular é aplicada quando há a necessidade de se dividir a aplicação em seções separadas para que possam ser adicionadas ou retiradas sem nenhum tipo de problema em relação à aplicação central. Ela facilita atualizações e simplifica a programação. Uma *engine* gráfica foi desenvolvida, a qual possui diversas ferramentas, tais como texturas, sombra e cubo de cores. A implementação da textura e sombra foi feita utilizando *OpenGL Shading Language* (GLSL) . Os modelos possíveis de serem aplicados podem ser importados de diferentes aplicações profissionais de modelagem 3-D disponíveis na literatura. Depois desta importação, as texturas e os efeitos visuais podem ser aplicados. A visualização da superfície do mar irregular também será apresentada neste trabalho. A superfície do mar foi modelada usando o espectro Phillips para mar irregular. Este espectro define ondas irregulares para uma superfície do mar sendo excitada pelos ventos em grandes extensões da superfície. Além da visualização, existem módulos utilizados para aplicações que resolvem numericamente a dinâmica dos componentes dos sistemas marítimos. Para facilitar, o programa interage com um *cluster* de computadores e com outros computadores remotos para reduzir o tempo de processamento das aplicações.

1. Sistemas Marítimos de Produção 2. Plataforma Computacional 3. Visualização

1. INTRODUÇÃO

O projeto de modernos sistemas marítimos de produção de petróleo possui diversos desafios tecnológicos e computacionais. Esses desafios se tornam ainda maiores em águas profundas (profundidade acima de 1000 m). Por se tratar de investimentos de alto risco, na ordem de milhões de dólares, é necessária a realização de uma série de testes e análises adequadas para o seu projeto, sob distintas condições ambientais de ondas, correntezas e até situações mais extremas, como na presença de furacões e de tsunamis, além do estudo de sua interação com o homem e as mudanças provocadas ao meio-ambiente.

Um sistema marítimo para explotação de petróleo em águas profundas é composto por diversos componentes. Essencialmente, abrange plataformas flutuantes, *manifolds*, tubos de distribuição e *risers*, dentre outros componentes. O presente trabalho foi desenvolvido com o foco em um *software* especializado para análise de *risers.* Coelho et. al (2005) e Martins et. al (2003) abordam uma análise do comportamento de *risers* em águas ultra profundas sob forças ambientais.

Um *riser* é um tubo ou um conjunto do mesmo que transportam fluidos e ferramentas do leito marinho, onde está localizado o poço de petróleo, para a plataforma e vice-versa. Um sistema de *riser* pode estar sujeito a diversas intempéries como; correntezas do oceano, ondas, forças induzidas pelo escoamento do fluido interno, deslocamentos da plataforma e vibrações induzidas pelos vórtices ou VIV (Morooka et al, 2006). O VIV é causada pela alteração dos Vórtices quando se chocam contra o corpo no momento em que o fluido passa por ele. O vórtice induz uma força de oscilação perpendicular à direção do escoamento que pode causar danos devido à fadiga no *riser* (Morooka et al, 2005). Portanto, é necessário que os componentes sejam projetados para operarem nos casos mais críticos possíveis, muitas vezes esses casos não são conhecidos realmente e precisam ser determinados através de previsões de cenários com estas condições críticas.

Atualmente existem propostos diferentes configurações para *risers*, e cada um possui sua particularidade, assim como vantagens, desvantagens e complexidades. *Steel Catenary Risers* (SCRs) são tubos feitos de aço em formato de catenária que se estende sobre o leito marinho para absorver os movimentos bruscos causados pelo mar sobre ele, esse modelo tem a vantagem de ser simples e ter um baixo custo. Fadiga induzida pelo VIV e o

peso são um sério problema nesse tipo de sistema. Outra alternativa é o *riser* flexível, o qual é composto por camadas de elastômeros e armaduras de aço que aumentam a flexibilidade da estrutura e conseqüentemente aumentam a resistência a fadiga. Porém uma grande desvantagem para esse tipo de *riser* é a escassez de fornecedores destes componentes no mercado e também limitações severas para aplicações em grandes lâminas d'água. *Riser* com Topo Tensionado (*Top Tensioned Riser* - TTR) consiste em um tubo tencionado pela plataforma ou por uma bóia localizada abaixo do nível do mar, formando um tubo vertical estendido. A extremidade inferior do *riser* é presa sobre uma estrutura aterrada no leito marinho.

O *software* em questão está sendo desenvolvido com a intenção de auxiliar engenheiros na consideração mais fidedigna possível de todos os resultados obtidos através do uso de uma larga variedade de *solvers* numéricos para projeto do sistema marítimo. Além disso, o *engine* de visualização possibilita o usuário do programa a ter uma visão mais detalhada e mais próxima possível da realidade, servindo assim para melhor configurar e prever o comportamento dinâmico do sistema marítimo, seja ele um *riser* ou qualquer outro componente, objeto de projeto. O *engine* de visualização fornece ao usuário uma variedade de detalhes de visualização, como sombras, texturas, deslocamentos nas três dimensões e também dá suporte para importar modelos 3Ds feitos em outros programas comerciais afim de aproximar-se ainda mais do cenário real. Portanto, o *software* tem a finalidade de facilitar a modelagem, processamento e a análise completa de um sistema de *riser*, incluindo a visualização do mesmo por vários ângulos bem como sua configuração (tanto do *riser* bem como todo o ambiente na qual se encontra). Um detalhe fundamental deste projeto é a de usar uma programação modular, que permite o uso de vários *solvers* numéricos para o processamento de dados, permitindo o processamento paralelo em outras máquinas e até a configuração do mesmo para ser executado em *cluster*.

2. COMPONENTES DA APLICAÇÃO

A aplicação foi desenvolvida utilizando programação orientada a objetos, portanto, o código implementado está divido em classes. Existe uma diferença entre o código que descreve os objetos a serem desenhados e o código para desenhar os objetos. O código para desenhar o objeto fica no executável central da aplicação, porém os objetos a serem desenhados estão guardados em bibliotecas dinâmicas. Objetos 3-D podem ser incluídos ou excluídos na aplicação, bastando colocar a bibliotecas dinâmicas na pasta raiz da aplicação. Adicionar ou retirar objetos não causam nenhum tipo de problema em relação ao executável central. Usando-se esta estratégia, os objetos podem ser adicionados sem a necessidade de recompilar o executável central. Um exemplo de código que fica no executável central é o código do *engine* gráfico, sombra, carregadores de malhas e texturas. Um exemplo do código de um objeto a ser desenhado poderia ser uma plataforma.

O executável central comunica-se com as bibliotecas dinâmicas por uma interface comum. A interface define a maneira que o executável central e os objetos interagem. A interface é um conjunto de funções comuns. Os objetos são restringidos a agir pela interface. Desenhar, transladar, girar, abrir a janela de configuração são exemplos de funções comuns que são implementadas no código do objeto. A grande vantagem é que o executável central não precisa saber sobre as propriedades e funções internas de todo os objetos. Objetos usam serviços oferecidos pelo executável central e agem pela interface com a habilidade de abrir uma janela. A Figura 1 apresenta um diagrama dessa estratégia.

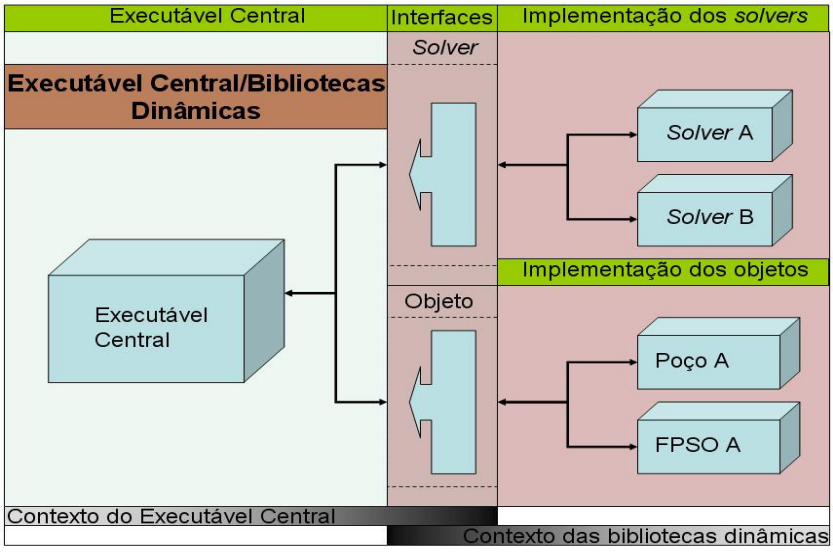


Figura 1. Contextos do executável central e as bibliotecas dinâmicas

Além do aspecto da visualização, o projeto está sendo desenvolvido para modelar *risers*, criar arquivos de entrada, rodar aplicações numéricas e mostrar os resultados obtidos. O código feito para interagir com aquelas aplicações também age através da interface. Uma simulação, para uma aplicação numérica genérica, poderia ser modelada e resolvida utilizando os serviços do executável central pela interface.

3. ENGINE GRÁFICO

O *engine* gráfico é o código responsável pela gestão da posição dos objetos a serem desenhados, reprodução da cena na tela, efeitos visuais (sombra, neblina, luz e outros), transmissão de mensagens de mudanças de estado (clique de mouse, tecla apertada e mais), perspectiva do usuário na cena e outros. Um *engine* gráfico é um componente importante na indústria de visualização e jogos. Existem vários *engine* gráficos comerciais e de código aberto disponíveis. Um *engine* gráfico foi desenvolvido particularmente para cumprir as necessidades deste projeto e, este desenvolvimento foi divido em partes.

A primeira parte é o gerenciamento da cena que controla os objetos na mesma e passa informações importantes para os objetos e para a interface. Pelo gerenciamento da cena, objetos podem ser adicionados e apagados. A janela de visualização pode ter mais de uma cena, embora cada cena possua apenas um gerenciador. Ele também controla as proporções e efeitos globais da cena que inclui sombra, luz e neblina.

A câmera é responsável por definir a posição e volume de visualização do usuário na cena. A câmera pode ser colocada em qualquer orientação na cena. Ela opera usando um sistema local de coordenadas cartesianas que tem um eixo sempre apontando na direção da visão do usuário. Os eixos preservam a noção de esquerda, direita, acima, e abaixo em relação à câmera em vez do sistema global. A câmera pode ser manipulada usando o teclado, mouse ou qualquer controlador. Quando a câmera é adicionada ao gerenciamento, a cena pode ser vista usando esta câmera. Câmeras com orientações predeterminadas facilitam o usuário na visualização da cena.

Os objetos são representações de algo real como *risers*, plataformas, linhas de ancoragem e outros equipamentos. Eles podem ser criados usando aplicações profissionais de modelagem e depois eles podem ser importados na visualização. O *engine* gráfico suporta vários formatos de desenhos 3D. O objeto comunica-se com o gerente da cena pela interface de funções descrita anteriormente. O usuário que faz o desenvolvimento pode modelar o objeto e depois criar um código para o objeto ser interativo. Por exemplo, uma plataforma pode balançar em ondas ou um *riser* movimentar, quando clicado na tela do computador, podendo abrir uma janela de configuração.

O *engine* gráfico encontra-se em desenvolvimento. No entanto, os desenvolvimentos realizados até o presente momento permitem algumas funcionalidades, e espera-se que o *engine* gráfico fique ainda melhor para visualização dos diferentes componentes de sistemas marítimos.

4. SUPERFÍCIE DO MAR IRREGULAR

A visualização da superfície do mar apresenta um desafio de tentar reproduzir um fenômeno natural que é totalmente aleatório. Na visualização da superfície do mar, utilizou-se o espectro de Phillips para descrever a superfície formada pelos ventos atuando em distâncias de grande extensão (mar desenvolvido), em águas profundas. O espectro de Phillips foi escolhido devido à sua facilidade de implementação, embora existam outros espectros que tentam descrever estados diferentes do mar, tais como JONSWAP, Pierson-Moskowitz, Scott e outros (Chakrabarti, 1987). A Equação 1 define o espectro do Phillips em função do número de ondas dado pelo Tessendorf (2001):

$$S(K) = A\frac{1}{k^4}\exp\left(\frac{-1}{(kL)^2} - (kl)^2\right)\left(\frac{|K \bullet W|}{|K||W|}\right)^2 \tag{1}$$

$$L = \frac{|W|^2}{g} \tag{2}$$

sendo, W é a velocidade do vento, g é a aceleração devida à gravidade, K é o vetor do número de onda, k é o numero de onda, A é uma amplitude arbitrária, e o l é o menor comprimento que será modelado. A Equação 2 define o maior comprimento de onda possível sob o vento. A próxima etapa é a geração dos coeficientes de Fourier através da Equação 3.

$$\hat{h}(K,0) = \frac{1}{\sqrt{2}}\left(X_1 + X_2\sqrt{-1}\right)\sqrt{S(K)} \tag{3}$$

onde, X_1 e X_2 são os números aleatórios gaussianos com uma média zero. A parte real é usada para a decomposição de Fourier pela Equação 4. A Equação 5 define a freqüência fundamental (ω).

$$h(x,t) = \sum_{K=(i,j)} \hat{h}(K,0)e^{\sqrt{-1}(Kx-\omega t)} \tag{4}$$

$$\omega = \sqrt{g|K|} \tag{5}$$

Pelo método de FFT o mapa de alturas pode ser criado na forma de uma grade de pontos espaçados igualmente. Essa grade é depois construída na visualização e repetida para criar um mar imenso. A próxima etapa é emular as propriedades visuais da água.

Durante a renderização da superfície do oceano, calcula-se a reflexão e a refração de luz através da lei de Snell com os índices de refração do meio e da água, e das equações de Fresnel para determinar quanto cada fenômeno contribui na cor final da superfície, onde o coeficiente de reflexão R é dado pelas equações 6, 7 e 8:

$$R = \frac{(R_s + R_p)}{2} \tag{6}$$

$$R_S = \left[\frac{n_i \cos(\phi_i) - n_t \cos(\phi_t)}{n_i \cos(\phi_i) + n_t \cos(\phi_t)}\right]^2 \tag{7}$$

$$R_p = \left[\frac{n_t \cos(\phi_t) - n_i \cos(\phi_i)}{n_t \cos(\phi_t) + n_i \cos(\phi_i)}\right]^2 \tag{8}$$

sendo, n_i e n_t são os índices de refração do meio externo e interno, e ϕ_i e ϕ_t são os ângulos entre o raio incidente e o raio refratado com a normal da superfície.

Devido à conservação de energia o coeficiente de transmissão T é igual a (1 – R) e é usado juntamente com R para calcular a combinação linear da cor do raio refratado e do raio refletido para obter a cor final.

4.1 MAPA DO AMBIENTE

Para simular esses efeitos é gerado um mapa do ambiente (*cube map)*, mostrado na Figura 2, que desenha a cena em seis direções diferentes ortogonais entre si. A partir do *cube map* é obtida a cor do raio refratado e do raio incidente, que aponta na direção do observador.

Figura 2. Um exemplo de *cube map*. (fonte: Nvidia)

As funções padrões do OpenGL sempre utilizam as mesmas equações para calcular as interações da luz com as superfícies dos objetos, não permitindo que existam modelos matemáticos diferentes para diferentes tipos de materiais que possam aparecer na cena. Porém, o OpenGL 2.0 permite que essas funções sejam reescritas pelo programador através de *shaders* e ainda assim sejam executadas pela unidade de processamento gráfico, assim como as funções padrões fazem.

5. SOMBRA E SHADERS

Os shaders são pequenos programas, geralmente escritos em linguagem de alto nível, que descrevem uma série de operações envolvendo vetores, matrizes e texturas e são divididos em pelo menos duas categorias:

***1-Shaders* de vértices:** controlam as transformações que os vértices do objeto sofrerão durante os primeiros passos da renderização (como rotação e translação).

***2-Shaders* de *pixels*:** fazem parte de um dos últimos passos da renderização e são responsáveis por determinar a cor que cada pixel pertencente à superfície terá quando for escrito na tela.

5.1 SHADOW MAPPING

O *shadow mapping* é um método para a geração de sombras baseado na renderização em dois passos, no primeiro passo, a cena é desenhada a partir do ponto de vista da luz e são guardadas as distâncias entre as superfícies visíveis e o emissor de luz, no segundo passo a cena é renderizada normalmente, porém uma verificação extra é feita: quando a distância entre a luz e a superfície for diferente que a calculada no primeiro passo, o pixel é desenhado sombreado, pois deve existir algum obstáculo à passagem da luz.

A Figura 3 apresenta uma imagem do mapa de distância (esquerda) e a imagem final com a sombra aplicada (direita). Brabec et al. (2003) fornece um tratamento completo para o *shadow mapping*. A Figura 4 apresenta um diagrama sobre o método de *shadow mapping* e de como se determina se o pixel está sob sombra.

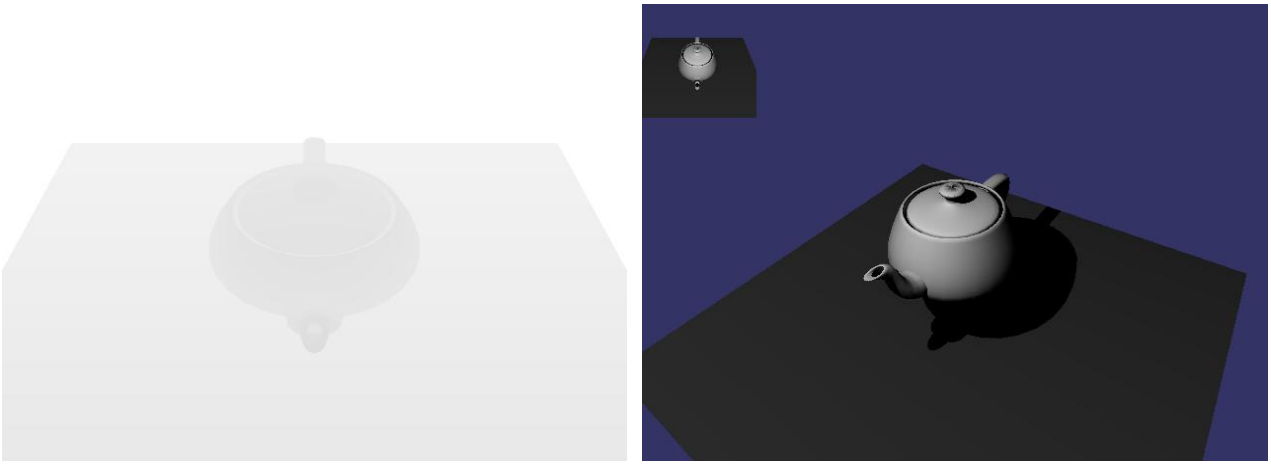


Figura 3. O mapa de distância e imagem final com sombra (fonte: NVIDIA Co. www.nivida.com)

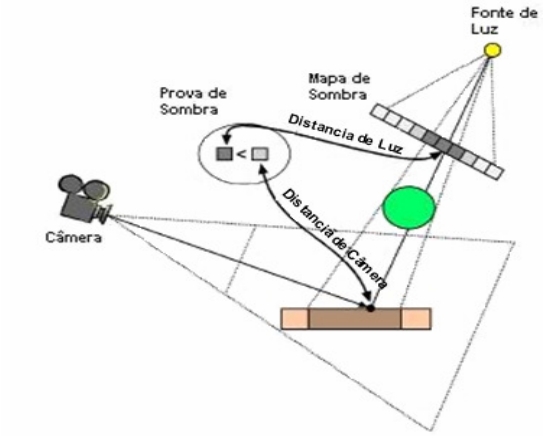
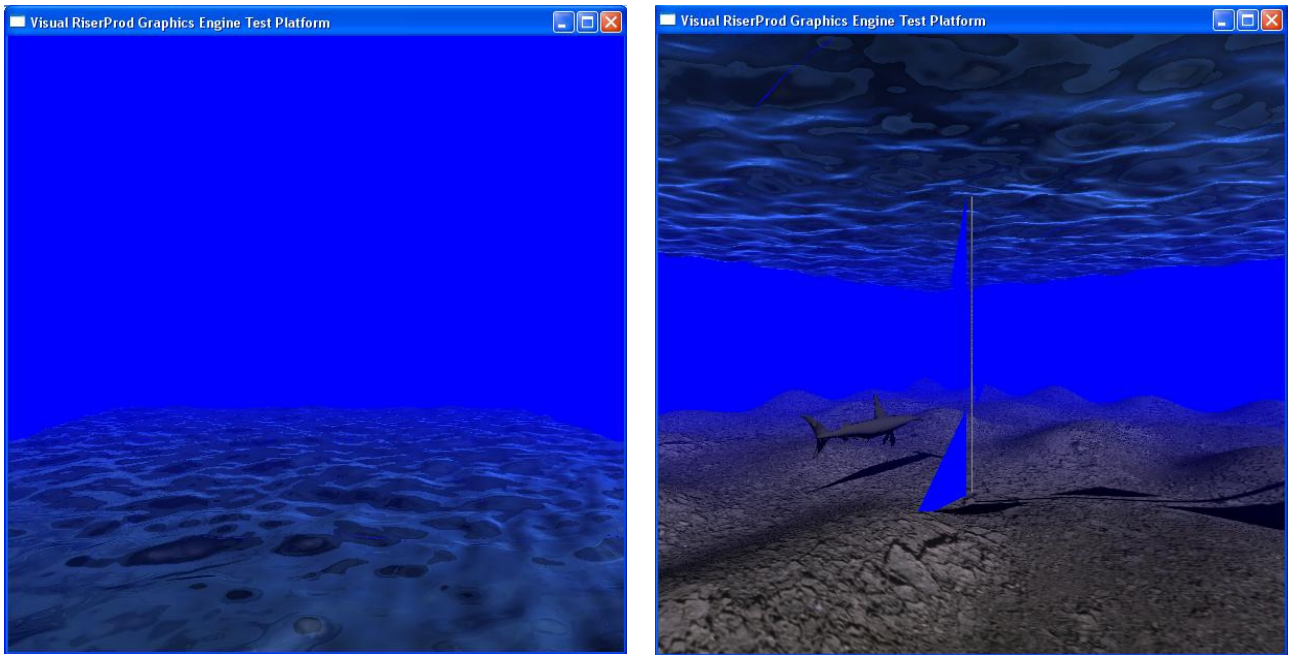


Figura 4. Diagrama de sombra (Fonte: Brabec, 2005 com alterações)

6 RESULTADOS

A superfície do mar gerada a partir do espectro de Phillips está apresentado na Figura 5a. Um *cube map* de um céu claro com a lua é aplicado sobre o mar. A Figura 5b mostra uma cena com o efeito de sombra. O modelo do objeto "tubarão" foi gerado como exemplo numa aplicação de modelagem 3-D e posteriormente importado ao *engine* gráfico. O *riser* rígido e o perfil de correnteza (os triângulos azuis) espalham sombras. O usuário pode mover-se na cena através do teclado e mouse do computador. O tubarão, *riser*/perfil, leito marítimo e superfície são objetos guardados em bibliotecas dinâmicas e são carregados na cena.



(a) **(b)**

Figura 5: (a) Superfície do mar gerado pelo espectro de Phillips. (b)cena com sombra.

7 CONCLUSÃO

O presente trabalho apresenta resultados do início de concepção e da construção de uma ferramenta computacional para auxiliar um engenheiro no projeto de sistemas marítimos de explotação. O alvo central é integrar a visualização, modelagem e análise do sistema global em apenas uma ferramenta. Para alcançar este objetivo, desafios de diferentes áreas do conhecimento deverão ser superados.

As próximas etapas do desenvolvimento incluem detecção de intersecção devido à configuração dos *risers*, integração de outros *solvers*, gerador de relatórios, inclusão de espectros de ondas diferentes do utilizado, e outros. A detecção de intersecção seria muito útil no desenvolvimento de campos onde o espaço é extremamente limitado. Além de colisões, quando um *riser* fica um próximo ao outro, a turbulência gerada pelo escoamento em torno de um *riser* pode afetar a outro *riser* adjacente, o que pode causar colisão entre eles ou aumentar a fadiga dos componentes.

Existem outros *solvers* para a análise do comportamento estático e dinâmico de *risers* e sistemas marítimos em geral. Atualmente, o *software* suporta apenas um *solver*. Quanto mais *solvers* uma ferramenta pode suportar, ela se dotará de maior capacidade para solução de problemas. O projeto oferece um *framework* para interagir com qualquer *solver*, entretanto, janelas de configuração de código para leitura, escrita e monitoração do estado interno de um caso para o *solver* é necessário.

Na prática, sob o sistema marítimo proposto age uma grande variedade de carregamentos ambientais em diversas condições extremas. Um gerador de relatórios pode normalizar a análise dos casos e evitar erros humanos no processamento de dados. O usuário pode definir um documento padrão de resultados incluindo gráficos e tabelas e aplicá-los para os resultados de diferentes carregamentos ambientais.

AGRADECIMENTOS

Os autores gostariam de agradecer a Capes, CNPq e FINEP (CTPetro), e Petrobras pelo apoio ao presente trabalho.

REFERÊNCIAS

Brabec, S. Annen, T. Seidel, H. Practical Shadow Mapping, Journal of Graphics Tools, Vol. 7, Number 4, 2003.

Chakrabarti, S.K., Hydrodynamics of Offshore Structures, CML Publications, Great Britain, 1987.

Coelho, F.M., Vieira, S.C., Morooka, C.K., Costa, C.G., Franciss, R., Simulação Numérica do Comportamento de Risers de Produção em Águas Ultra Profundas. 3º Congresso Brasileiro de P&D em Petróleo e Gás, Salvador, BA, 2 a 5 de outubro de 2005.

Martins, F.P., Kubota, H.Y., Morooka, C.K., Ferrari Jr., J.A., Ribeiro, E.J.B., Estudo do Comportamento Dinâmico In-Line e Transversal de "Riser" Rígido de Produção, 2º Congresso Brasileiro de P&D em Petróleo e Gás, Rio de Janeiro, RJ, 15 a 18 de junho de 2003.

Morooka, C.K., Idehara, A.Y., Matt, C.G., Behavior of a Pipeline with Free Span Under Steady Current, 27th Iberian Latin- American Congress on Computational Methods in Engineering,13-504, 2006.

Morooka, C.K., Coelho, F.M., Ribeiro, E.J.B., Ferrari Jr., J.A., Franciss, R., Dynamic Behavior of a Vertical *Riser* and Service Life Reduction, 24th International Conference on Offshore Mechanics and Arctic Engineering (OMAE), Halkidiki, Greece., 2005.

Tessendorf, J., "Simulating Ocean Water", In Simulating Nature: Realistic and Interactive Techniques, SIGGRAPH, 2001.

A COMPUTATIONAL PLATFORM FOR VISUALIZING SYSTEMS

This work presents a computational platform to facilitate the visualization of offshore petroleum production systems. An offshore petroleum production system is comprised of many diverse components like platforms, risers, wells, anchoring chains and many others. In the design phase it is very important to be able to visualize the proposed system in three-dimensions. A visualization system is currently under development to help the engineer better understand a complex offshore petroleum production systems. That platform is being developed in C++ using modular programming concepts and utilizing the widely used 3D library, OpenGL. Modular programming elements are used in the aim to allow the platform to be divided into interoperable blocks which greatly facilitates the development and customization of the project as a whole. 3D elements can be modeled in professional 3D modeling packages and then imported into the platform. This paper will introduce the basic theory behind modular programming oriented to the aims of the platform. The basics of a rudimentary graphic engine will also be introduced which includes cube maps, shadows, fog, textures and the use of the OpenGL Shading Language (GLSL). This paper will also report on the construction of a 3-D irregular sea surface based on the Phillips Spectrum.

1. Offshore Petroleum Production System 2. Computational Platform 3. Visualization

Os autores são os únicos responsáveis pelo conteúdo deste artigo.