Visualização de Malhas Tubulares em Movimento

Luiz Fernando Silva
Computing Science Department (DCC)
Federal University of Rio de Janeiro (UFRJ)
Rio de Janeiro (RJ), Brazil
Email: lfestevao@gmail.com

Rodrigo de Toledo
Computing Science Department (DCC)
Federal University of Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
Email: rtoledo@dcc.ufrj.br

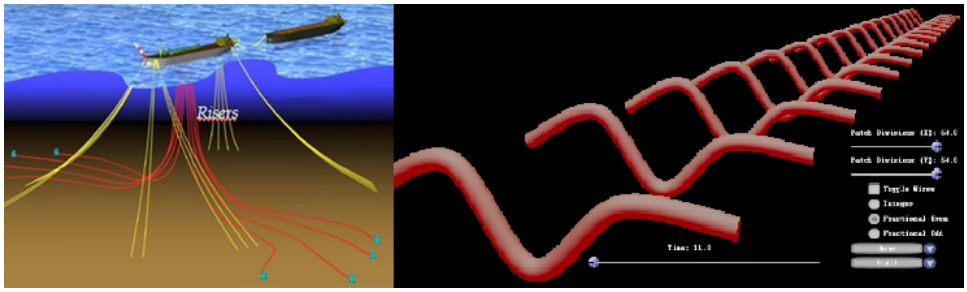

Fig. 1. Amostra do resultado do nosso método: A partir de dados de risers reais (a esquerda), movimento reconstruído para visualização em alta performance (a direita), baseado na entrada.

***Keywords*-GPU;Visualization;**

I. Abstract

In this paper, we propose a technique for visualization, GPU based (Graphics Process Unit), of moving tubular lines, which can represent plumbing, oil pipes, organic veins and others. Our proposal consists in real time frame render for the tubes, loading only once the minimal data to GPU memory, thus reducing the transfer from CPU to GPU. The main base is the ability of dinamic mesh generation allowed by the new video card models.

II. Introdução

A proposta deste trabalho é renderizar vários quadros de uma malha tubular a partir de dados de um encanamento maleável (um *riser*[1] por exemplo) em movimento. Uma restrição é a necessidade de geração de imagens em tempo real, de modo que o observador (câmera) possa se mover pelo ambiente virtual. Este trabalho extende o de Nunes et al. [1].

a) Contribuição: A contribuição dete trabalho adiciona a capacidade de ter uma animação sendo executada pelo sistema sendo observado.

Esta proposta usa os recursos oferecidos pela *GPU* e *Shader Model 5.0* [2] para visualizar uma malha de alta complexidade. Isto é, muitos vértices e arestas, renderização em tempo real, com interatividade, mesmo com uma animação rodando, sem perder em qualidade ou desempenho para aplicações em *VBO*.

[1]Tubo semiflexível de kilômetros de comprimento que conduz o petróleo entre a cabeça de um poço de petróleo no solo marinho até uma plataforma submersível

Aplicações usando o paralelismo da placa de vídeo ganham eficiência. Mas existe um gargalo na transferência de dados de *CPU* para *GPU*, limitando a quantidade de pontos a ser precarregados sem queda de performance [3]. Isso gera um impasse, menos pontos renderizam malhas de baixa qualidade em alta velocidade, entretanto mais pontos resultam em malhas de melhor qualidade com perda de velocidade.

A principal motivação deste trabalho é encontrar um modo de reduzir o impacto deste gargalo. Para isso, é necessário aumentar a vazão de transferência de dados ao decorrer de uma visualização dinâmica.

III. Trabalho Anterior

Existem algumas propostas para renderização de malhas tubulares [4]. No trabalho de Nunes et al. [1] é possível renderizar uma malha tubular vista de qualquer direção. Ele considerava uma tubulação simples, isto é, sem bifurcações e com seção reta constante, tendo alguma curvatura. Este objeto foi matematicamente modelado usando uma linha que corre no centro do tubo, contendo o centro de todos os círculos dos cortes, considerada a espinha dorsal da malha. Os dados usados envolvem o conceito de pontos de controle, nos quais a posição real da curva central do tubo é conhecida, por aferição ou simulação. Por fim esta linha é reconstruída com um ajuste Hermite, dado pelas posições dos pontos e suas tangentes. Esta proposta consegue gerar resultados em tempo real para a ordem de milhões de vértices gerados.

IV. Visualização de Linhas Tubulares em Movimento

O trabalho desta publicação permite renderizar malhas tubulares refletindo o movimento original do sistema. Foi

mantida a consideração de tubulação simples e adicionando as movimentações que o corpo faz, bem como a mudança de sua curvatura ao longo do domínio. O modelo foi acrescido de um parâmetro definindo o tempo decorrido, que reflete o movimento da espinha do sistema. A sequência de dados é decorrente de vários instantes consecutivos em que as medições são feitas. A linha é reconstruída a partir de um ajuste dos pontos de controle, usando o temporizador como fator para dar continuidade a animação.

Há vários tipos de ajustes possíveis e alguns foram escohidos para avaliação. Foram usados o algoritmo de De Casteljau, que originam curvas de Bézier, de primeiro a terceiro níveis e Catmull-Rom.

Para a implementação em GPU, evita-se o gargalo da transferência pré-carregando-os antes de começar a renderização, removendo o custo de carregar novos dados a cada frame. Para tanto, usa-se uma textura em GPU com estrutura organizacional similar a uma lista de dados por frame. Por sua vez, dados relativos aos frames são listas de dados por ponto de controle.

No *Vertex Shader*, o dado de cada ponto de controle é acessado usando o seu índice e o valor do temporizador atualizado em CPU. Assim cada ponto de controle sabe a posição que deve estar no instante T e em $T+1$, se ajustando conforme a animação. No *Hull Shader* se define os parâmetros para o *Tessellator*, que gera os vértices que estarão na cena [5]. No *Domain Shader* os vértices reconstruem a tubulação. Por fim, no *Fragment Shader* a malha é texturizada conforme as normais de cada vértice.

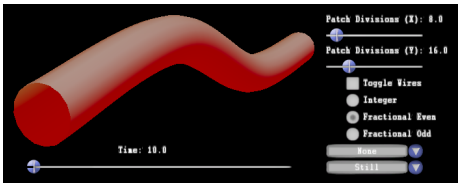

Fig. 2. Exemplo de Linha tubular simples. Imagem produzida pelo programa.

V. Resultados

Para medir o desempenho da aplicação, ela foi submetida a vários testes. O mais expressivo foi o de comparação, para 500 pontos de entrada, gerando pontos conforme os parâmetros de tecelagem. Esta proposta foi comparada às técnicas já existentes, VBO e uma variação proposta pelo trabalho base. Todos os vértices gerados estiveram visíveis 100% do tempo. Caso hajam pontos fora do fustrum, o *culling* deixa a aplicação ainda mais eficiente.

Para comparar os tipos de ajuste que propostos, fez-se testes similares. A tecelagem usada foi a de 16 por 16, replicando 20 vezes cada malha gerada, estressando o programa. Pode-se notar na Tabela V que há pouca diferença em performance: Como não há muita diferença no desempenho, o usuário pode escolher qual a opção melhor atende sua necessidade. É importante ressaltar que dentre as opções na Tabela V, a animação aumenta em qualidade visual da esquerda para direita.

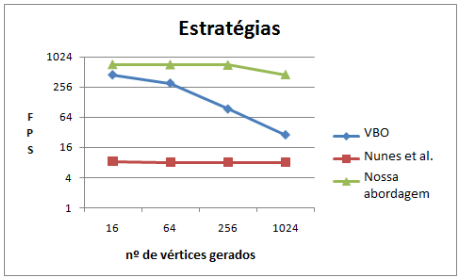

Fig. 3. Comparação entre as estratégias propostas. Em azul, aplicação similar usando VBO, em vermelho a proposta de [1], atualizando os pontos de controle a cada quadro e em verde a proposta deste trabalho. Vantagem a partir de 128 mil pontos, desbancando a principal técnica usada anteriormente.

TABLE I
Desempenho semelhante das estratégias de interpolação

Interpolation strategy	None	Linear	Quadratic	Cubic	Hermite
FPS	120,70	120,64	120,36	120,33	120,35

VI. Conclusão

Neste paper se apresenta a técnica de pré-carregamento como uma forma de evitar o gargalo da tranferência CPU-GPU. Assim a vazão global aumenta conforme a animação prossegue. Também mostra que esta abordagem é particularmente apropriada para este tipo de aplicação.

Trabalhos futuros envolvem estender esta aplicação para considerar LOD progressivo [6], e outras complexidades como bifurcações.

Agradecimentos

Agradecemos aos colegas Alexandre Valdetaro e Gustavo Nunes que desenvolveram a aplicação base para este projeto e nos ajudaram tirando eventuais dúvidas.

References

[1] G. Nunes, A. Valdetaro, A. Raposo, B. Feijó, and R. de Toledo, "Rendering tubes from discrete curves using hardware tessellation," *Journal of Graphics Tools: JGT*, 2012, accepted to be published.

[2] G. B. N. B. F. A. R. Rodrigo B. Pinheiro, Alexandre Valdetaro, "Introduction to multithreaded rendering and the usage of deferred contexts in directx 11," in *SBC - Proceedings of SBGames*, 2011.

[3] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008. [Online]. Available: http://www.idav.ucdavis.edu/publications/print_pub?pub_id=936

[4] J. Sánchez-Reyes, "Single-valued tubular patches," *Comput. Aided Geom. Des.*, vol. 11, no. 5, pp. 565–592, Oct. 1994. [Online]. Available: http://dx.doi.org/10.1016/0167-8396(94)90304-2

[5] A. Tatarinov, "Instanced tessellation in directx 10," in *Game Developers Conference 08*, 2008. [Online]. Available: http://developer.nvidia.com/object/gamefest-2008-subdiv.html

[6] H. Hoppe, "Progressive meshes," *Proceedings of SIGGRAPH 96*, pp. 99–108, August 1996, iSBN 0-201-94800-1. Held in New Orleans, Louisiana.