Técnicas de Programação em GPU Aplicadas à Visualização de Modelos com Múltiplas Texturas

Vitor de Andrade
Escola Politécnica - UFRJ
Rio de Janeiro - Brasil
Email: vitor.andrade@poli.ufrj.br

Ricardo Marroquim
COPPE - UFRJ
Rio de Janeiro - Brasil
Email: marroquim@cos.ufrj.br

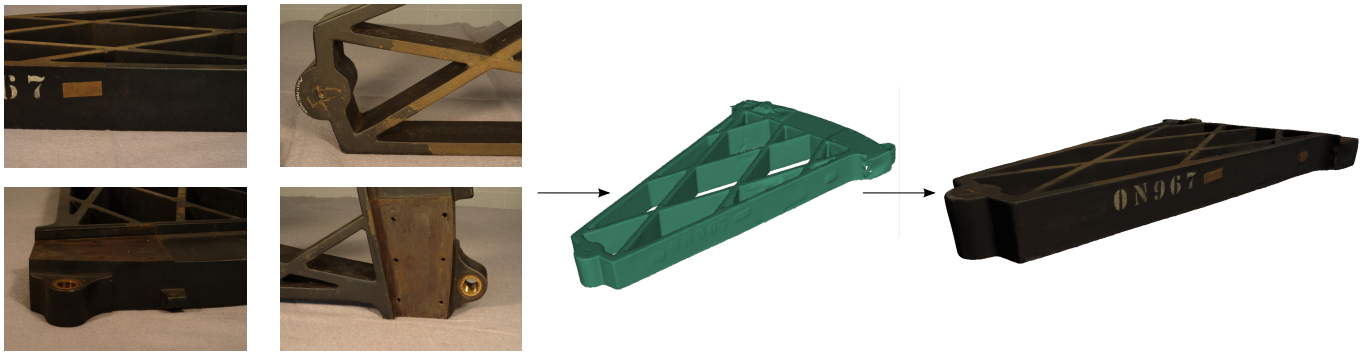

Figura 1. 4 fotos (do total de 41) em alta resolução de um instrumento científico (esquerda), projetadas em um modelo não texturizado (meio), obtendo um modelo final (direita)

***Resumo*—Nas últimas décadas surgiram novas técnicas de digitalização através da utilização de *scanners* 3D. Esta tecnologia permite obter representações virtuais de objetos reais com cada vez mais fidelidade e facilidade. Porém, as técnicas para atribuição de cores a esses modelos (texturas ou BRDFs) ainda apresentam uma série de desafios. Este artigo propõe um sistema de gerenciamento e visualização de objetos contendo várias texturas para auxiliar este processo, focando em aplicações na área de bens culturais. Este trabalho envolve conceitos de renderização em multipassadas, *off-screen rendering*, programação em GPU, *shaders* e OpenGL 4.**

***Keywords*-Renderização em multipassadas; *Off-screen Rendering*; Programação em GPU; Digitalização Tridimensional; Computação Gráfica**

***Abstract*—In the last decades new techniques of digitalization using 3D scanners have emerged. This technology makes it possible to obtain virtual representations of real objects with less effort and more fidelity. However, the techniques for attributing color to these models (textures or BRDFs) still present a series of challenges. In order to aid this process, this paper proposes a visualization and management system for objects containing many textures, focusing in the cultural heritage area. This work involves concepts of multipass rendering, off-screen rendering, GPU programming, shaders and OpenGL 4.**

***Keywords*-Multipass Rendering; Off-screen Rendering; GPU Programming; Three-Dimensional Digitalization; Computer Graphics**

I. Introdução

É de grande importância que existam métodos de se registrar elementos culturais ao longo da história de um país. Livros podem guardar referências aos costumes e o cotidiano da população através dos tempos. Quadros e pinturas armazenam informações sobre as tendências artísticas e muitas vezes sobre o passado. Os museus são locais que podem contar de forma única a história de um país ao armazenar peças que foram importantes para a formação, seja cultural, científica ou econômica da nação. Neste sentido, o avanço da tecnologia proporciona uma nova maneira de se interagir com o patrimônio histórico, de forma que um visitante possa ter contato com a peça em seu contexto original e possa visualizá-la em funcionamento virtualmente.

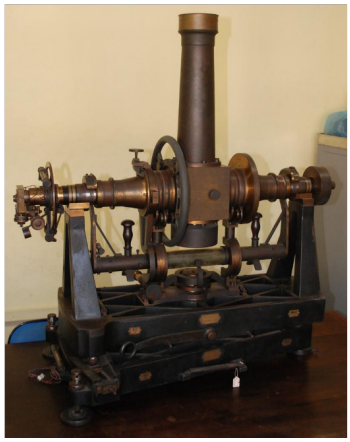

Figura 2. A luneta meridiana Bamberg

Visando contribuir para a preservação de patrimônios históricos, desenvolveu-se uma parceria entre a Universidade Federal do Rio de Janeiro (UFRJ) e o Museu de Astronomia e Ciências Afins (MAST) no Rio de Janeiro, para realizar a digitalização utilizando um *scanner* 3D de um instrumento científico histórico em processo de restauração, a luneta meridiana Bamberg (Figura 2), de forma que se obtenham réplicas virtuais. Estas réplicas serão utilizadas para fazer o treinamento de novos especialistas em manutenção e restauro de instrumentos científicos, além de disponibilizar no museu uma réplica fotorrealista da luneta em seu contexto e utilização originais.

Figura 3. Exemplo de foto sendo projetada em um modelo de uma caneca.

Após o início do projeto, percebeu-se que existem vários desafios no que se refere à texturização do modelo escaneado. Diversas fotos em alta resolução devem ser obtidas e projetadas no modelo geométrico adquirido pelo *scanner* para se obter uma reprodução fiel da peça (Figura 1 e Figura 3). Entretanto, a montagem destas projeções não é uma tarefa simples, uma vez que cada parte da peça é representada em diversas fotos adjacentes e o mapeamento deve ser extremamente preciso (Figura 4). Além disso, há ainda uma complicação relacionada à limitação do número de texturas visualizadas simultaneamente. Considerando estes desafios, foi proposto o desenvolvimento de um *software* para realizar o alinhamento semiautomático das fotos, chamado de Photojibe [1].

Apesar de existirem diversas técnicas para alinhar fotos com o modelo geométrico [2], [3], geralmente é abordado o problema do ponto de vista da projeção de cada foto individualmente. Por outro lado, após realizar o alinhamento de todas fotos existem diversas abordagens para realizar a fusão gerando uma textura única [4], [5], [6]. No entanto, existem muitas dificuldades na elaboração de um sistema de alinhamento semiautomático que geralmente não são mencionadas na literatura. Por exemplo, é desejável acompanhar o progresso do alinhamento em tempo real, sendo necessário visualizar e manipular um único modelo contendo diversas texturas, i.e. diversas fotos projetadas. Este é um caso pouco comum, pois até que a texturização completa seja alcançada, cada vértice pode possuir um número arbitrário e possivelmente grande de coordenadas de texturas, devido à sobreposição das fotos.

Neste trabalho, focamos neste desafio específico de gerenciamento e visualização de várias texturas simultaneamente. Os métodos utilizados para resolver estes desafios envolvem os conceitos de renderização em multipassadas (*Multipass Rendering*), *Off-screen Rendering*, programação em GPU com a utilização de *shaders* e OpenGL 4 e utilização de *buffers* para renderização em *non-immediate-mode*. Nas próximas seções descreveremos como utilizamos estas técnicas em nosso trabalho.

Figura 4. Exemplo de texturas adjacentes com mapeamento impreciso, com perda na qualidade da reprodução.

II. Buffers

Um fator de grande importância para o desenvolvimento de qualquer sistema em computação gráfica é o desempenho. Quando é necessário trabalhar com uma grande quantidade de texturas projetadas em um modelo de alta resolução, o desempenho se torna uma questão ainda mais sensível. Neste sentido, fazemos uso de técnicas recentes de renderização, a fim de melhorar o desempenho e possibilitar a visualização interativa (i.e. ao menos 30 quadros por segundo) dos modelos. Por exemplo, foi utilizada a renderização em *non-immediate-mode* através dos *buffers* VBO e TBO, descritos a seguir.

A. Immediate-mode e non-immediate-mode

A renderização em *immediate-mode* consiste em produzir cada quadro a partir de comandos explícitos para desenhar cada primitiva (ex. triângulos). O maior problema desta abordagem é que a cada quadro todos os triângulos da cena são transferidos da memória da CPU para a GPU.

Já a renderização em *non-immediate mode* permite guardar dados como coordenadas de vértices, vetores normais do polígono, coordenadas de textura, entre outros atributos, diretamente na memória da GPU. Desta forma, no momento da renderização da cena o acesso a estes dados é realizado diretamente da memória da placa gráfica, evitando o alto custo computacional da transferência de dados da CPU para a GPU.

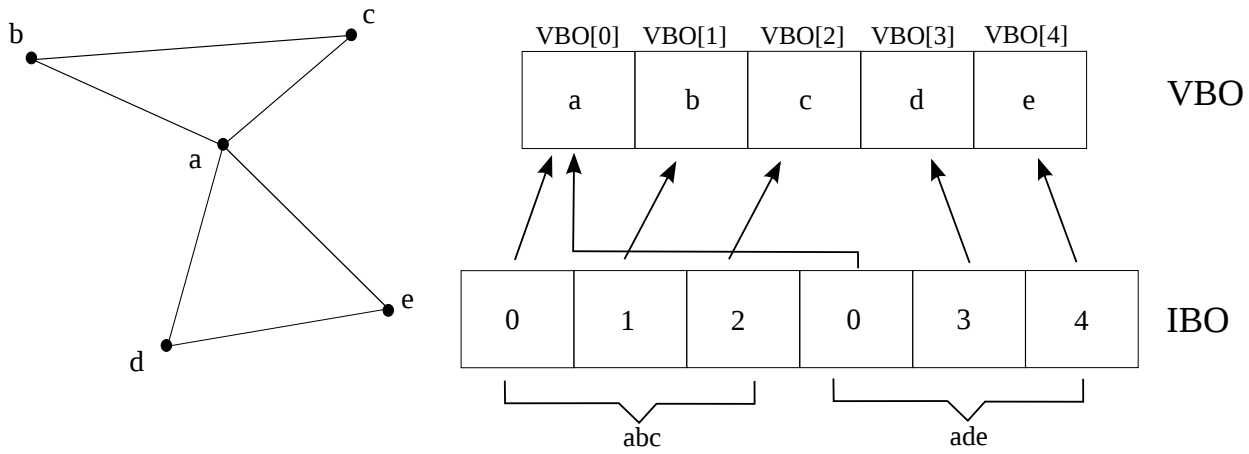

Figura 5. Exemplo de utilização de VBO e IBO.

B. VBO

Um *Vertex Buffer Object* (VBO) permite manter as informações mencionadas acima (coordenadas dos vértices, vetores normais, cores...) armazenadas diretamente na memória da GPU. Ele se relaciona com outro *buffer*, chamado de IBO (*Index Buffer Object*), na criação dos polígonos. Os valores das coordenadas dos vértices são guardados em um VBO e informações de como os polígonos serão construídos, i.e. os índices dos vértices que compõem o polígono, são guardadas no IBO. Este último é utilizado para que não seja necessário definir múltiplas vezes no VBO vértices compartilhados por vários polígonos. Na Figura 5, temos que os triângulos abc e ade, compartilham do vértice a. Então o vértice a possui apenas uma referência no VBO (índice 0), porém o IBO contém duas entradas para este índice, indicando que dois polígonos compartilham este vértice.

C. TBO

Um *Texture Buffer Object* (TBO) é um *array* unidimensional de elementos de textura armazenados em *Buffer Objects*. Ele permite armazenar as coordenadas de textura diretamente na GPU. Como o sistema trabalha com um número muito grande de texturas projetadas simultaneamente, a utilização deste *buffer* é essencial para otimizar o desempenho da renderização.

Uma vez que uma textura é corretamente alinhada com o modelo geométrico, é gerado um TBO que ficará responsável por guardar as coordenadas desta textura. Além disso, existe ainda um TBO temporário relacionado à textura que está sendo alinhada no momento.

Vários vértices no modelo contêm coordenadas de textura válidas em mais de um TBO (ou seja, há várias texturas projetadas em um mesmo vértice), uma vez que frequentemente há várias fotos sobrepondo uma mesma região do modelo. Neste caso, é feito posteriormente um *blending* entre as cores provenientes destas texturas. Além disso, um vértice não necessariamente contém coordenadas para cada foto. Neste caso, atribui-se a este vértice as coordenadas $st(0, 0)$ para esta textura. Este critério é válido pois na prática nenhum vértice é realmente associado a estas coordenadas, porém para ser estritamente correto seria necessário utilizar um vetor auxiliar para indicar quais texturas são utilizadas para cada vértice.

III. Renderização em Multipassadas

A renderização em multipassadas, ou *multipass rendering*, é uma técnica largamente utilizada na computação gráfica. A ideia por trás desta técnica é simples: dividir a renderização de uma cena complexa em passos, unir o resultado de cada passo com a utilização de um FBO (*Framebuffer Object*) e finalmente utilizá-los na renderização final da cena.

As aplicações para esta técnica são as mais diversas: efeitos de espelhos ou TVs através da renderização de uma cena para uma textura, com a sua posterior projeção em um elemento renderizado na tela; *blurring* alcançado ao renderizar a mesma cena várias vezes com uma pequena variação na posição da câmera, desenhando-as na tela simultaneamente; efeitos de iluminação, como por exemplo, renderizar cada fonte de luz da cena separadamente para controlá-las individualmente, dentre outros.

A técnica mencionada anteriormente de renderizar uma cena para uma textura é chamada de *Off-screen Rendering*. Algumas aplicações para esta técnica envolvem geração de imagens de alta resolução e geração de texturas.

A utilização da renderização em multipassadas em conjunto com a *off-screen rendering* no projeto teve como objetivo resolver o desafio da limitação do número de texturas projetadas no modelo que poderiam ser visualizadas simultaneamente. Isto acontece porque cada placa gráfica possui um limite de *slots* de textura que é, em todo caso, inferior à demanda desta aplicação. A cada passada é renderizado para uma textura, o modelo com um certo número de texturas projetadas. Ao fim de cada uma delas, mistura-se o valor das cores dos pixels da passada atual com o das anteriores, como descrito em detalhes a seguir.

A. Texturização em multipassadas

Inicialmente utilizava-se um *slot* de textura da placa gráfica para cada foto selecionada para visualização. Dessa forma, na nossa aplicação, rapidamente se esgotaria a quantidade de

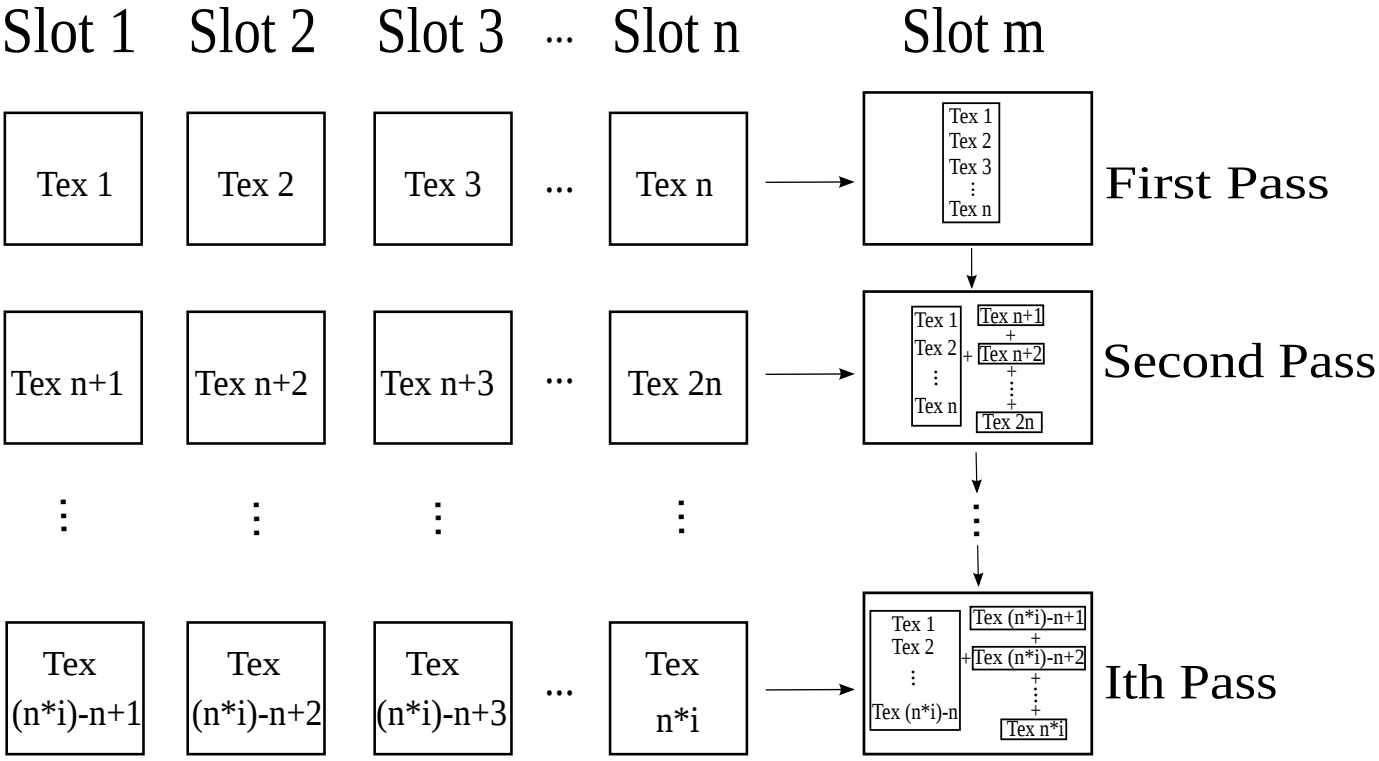

Figura 6. Esquema da renderização em Multipassadas

memória disponível, tornando impossível que se visualizasse o modelo com todas as texturas desejadas. Para solucionar este problema, determinou-se um número n de *slots* de textura a serem utilizados para guardar texturas individuais. Além disso, reservou-se um *slot* de textura extra para a renderização em multipassadas. Caso o número de texturas para visualização seja menor do que o número de *slots* reservados para texturas individuais, a renderização ocorre diretamente para o *back-buffer*. Isto significa que na primeira passada podem ser visualizadas até n texturas simultaneamente sem a utilização da renderização em multipassadas. Caso se deseje visualizar mais de n texturas, uma vez terminada a primeira passada, a cena é renderizada para o FBO (*Off-screen rendering*) e é ativado um *flag* a ser lido pelo *fragment shader*, que indica a utilização do *slot* de textura reservado para a multipassada. Temos então nesta textura, o resultado da renderização com as n texturas iniciais, porém utilizando apenas uma textura efetivamente (aquela relacionada ao *slot* da multipassada). Os n primeiros *slots* podem então ser liberados para que se faça uma nova passada, visualizando outras n novas texturas. Com isso, após a segunda passada teremos no total $2n$ texturas visualizadas. Generalizando, ao final de um número i de passadas, teremos até $n * i$ texturas visualizadas, utilizando-se no máximo $n + 1$ *slots* de textura, independente do número total de texturas (Figura 6).

B. OpenGL 4

Toda a comunicação com os *shaders* foi implementada com OpenGL 4, utilizando o conceito de *Vertex Attribute Pointer* e *Vertex Attribute Array*. No *Vertex Shader*, foram definidas várias variáveis de entrada utilizando o conceito de *layout qualifiers*. Isto permite que cada variável relacione-se a um *attribute index*. Separou-se um índice para as coordenadas dos vértices dos polígonos, um para as coordenadas dos vetores normais, um para as coordenadas da textura atual sendo alinhada e um número de índices determinado pela capacidade do *hardware* para texturas individuais sendo visualizadas (sem a multipassada). Durante a execução, no momento em que se deseja enviar determinados dados para o *Vertex Shader*, relaciona-se o *buffer* em questão a um determinado *attribute index*.

Não é necessário que se tenha um índice separado para as coordenadas de textura de multipassada, uma vez que seu mapeamento é feito diretamente no *fragment shader*. Isto é possível porque a textura possui as mesmas dimensões do *viewport*, o que significa que suas coordenadas corresponderão exatamente às coordenadas do fragmento na tela, que podem ser acessadas através da variável *gl_FragCoord.xy*.

C. Fragment Shader

No *fragment shader*, temos a definição dos n *slots* de textura reservados para texturas individuais, assim como aquele reservado para a multipassada. A cada passada, faz-se um *blending* entre as cores provenientes de cada uma das texturas individuais. Em seguida, o *flag* de multipassada emitido no

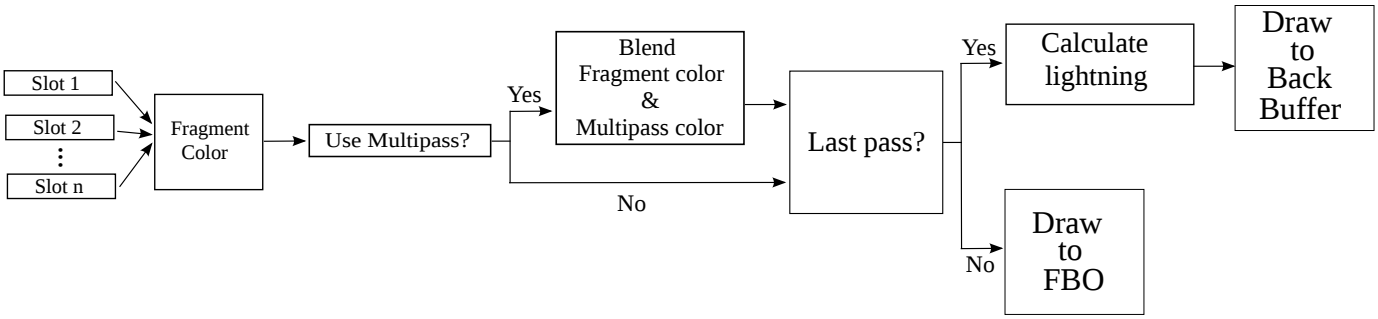

Figura 7. Fragment Shader

momento da renderização é lido. Caso este *flag* indique a utilização da multipassada, faz-se o *blending* destas texturas iniciais também com aquela proveniente do FBO. O valor final do pixel é dado por:

$$C_f = \frac{\sum_{i=1}^{n} C_i}{n} \quad (1)$$

onde n é o número total de texturas e C_i é a cor do pixel associada a i-ésima textura. Note que o número total de texturas é acumulado até a última passada, onde só então é feita a divisão. Para tanto, foi necessário utilizar *GL_TEXTURE_RECTANGLE* para que a soma não fosse truncada ao final de cada passada. Tendo acumulado todas as texturas, podem ser calculados os efeitos de iluminação para que seja emitido o valor final da cor do fragmento (Figura 7).

IV. Resultados

A seguir descreveremos as melhorias obtidas no sistema ao serem implementadas as técnicas de programação em GPU descritas nas seções II e III.

Com o desenvolvimento da renderização utilizando *buffers* e programação em GPU, houve um aumento considerável no desempenho do sistema. Quando a renderização ainda era feita através do *immediate mode*, notava-se que conforme se aumentava o número de texturas registradas, a taxa de quadros por segundo decrescia rapidamente. A partir da quarta textura ela já se tornava proibitivamente lenta. Com a utilização dos VBOs, mesmo que se eleve consideravelmente o número de texturas registradas, são mantidas taxas interativas, como pode ser visto na tabela I.

Já com o desenvolvimento da renderização em multipassadas, pôde ser resolvido o problema da limitação do número de texturas a serem visualizadas simultaneamente no modelo. Com ele o usuário pode visualizar o modelo com um número muito maior de texturas alinhadas, possibilitando a visualização de modelos complexos totalmente texturizados. Ainda mais, o número máximo de texturas simultâneas não é mais dependente do *hardware*. A tabela II indica os valores de taxas de quadros por segundo medidos na visualização de um modelo para diferentes quantidades de texturas registradas, de forma que possa ser avaliada a performance do sistema conforme se aumenta o número de passadas necessárias para a renderização. Neste teste foram utilizados 7 *slots* de textura para texturas individuais e mais um *slot* para a multipassada. Percebe-se então que mesmo quando foram projetadas mais de 40 texturas simultaneamente no modelo, as taxas de renderização mantiveram-se interativas. É importante lembrar que cada textura corresponde a uma foto em alta resolução (aproximadamente 3.2M pixels).

As configurações do computador utilizado para todos os testes indicados nesta seção são: processador *Intel Core i7* com 8 núcleos, placa de vídeo *NVidia GeForce 470 GTS* e 16GB de memória RAM.

Tabela I
TABELA COMPARATIVA DE TAXAS DE QUADROS POR SEGUNDO EM *immediate mode* E *non-immediate mode* PARA VÁRIOS NÚMEROS DE TEXTURAS REGISTRADAS.

Registered Textures	Immediate Mode FPS	Non-Immediate Mode FPS
0	22.6	1000
1	11.2	556
2	6.9	496
3	4.7	494
4	3.6	493
5	2.9	493
6	2.3	493
7	2.0	490

Tabela II
TABELA DE TAXAS DE QUADROS POR SEGUNDO MEDIDAS COM A UTILIZAÇÃO DA RENDERIZAÇÃO EM MULTIPASSADAS.

Registered Textures	Number of Passes	FPS
6	1	550
7	2	495
14	3	333
21	4	310
28	5	250
35	6	245
42	7	230

V. Conclusões

Neste artigo, foram apresentados os conceitos de renderização em multipassadas, *off-screen rendering* e renderização em *non-immediate mode* aplicados à texturização de modelos 3D. Os resultados encontrados demonstram o sucesso obtido na aplicação destas técnicas. A utilização dos *buffers*

proporcionou uma grande melhora no desempenho da visualização do modelo. Além disso, a utilização da renderização em multipassadas e da *off-screen rendering* permitiram a utilização de um grande número de texturas projetadas em um único modelo, permitindo com que se faça a texturização de modelos complexos de forma eficiente e visualizando o modelo completo em todos os momentos.

REFERÊNCIAS

[1] R. Marroquim, G. Pfeiffer, F. Moura de Carvalho, and A. Oliveira, "Texturing 3d models with low geometric features," in *Graphics, Patterns and Images (Sibgrapi), 2011 24th SIBGRAPI Conference on*, aug. 2011, pp. 1 –8.

[2] R. Pintus, E. Gobbetti, and R. Combet, "Fast and robust semi-automatic registration of photographs to 3d geometry," in *The 12th International Symposium on Virtual Reality, Archaeology and Cultural Heritage*, October 2011.

[3] M. Corsini, M. Dellepiane, F. Ponchio, and R. Scopigno, "Image-to-geometry registration: a mutual information method exploiting illumination-related geometric properties," *Computer Graphics Forum*, vol. 28, no. 7, pp. 1755–1764, 2009.

[4] A. Baumberg, "Blending images for texturing 3d models," in *Proc. Conf. on British Machine Vision Association*, 2002, pp. 404–413.

[5] N. Bannai, R. B. Fisher, and A. Agathos, "Multiple color texture map fusion for 3d models," *Pattern Recogn. Lett.*, vol. 28, no. 6, pp. 748–758, Apr. 2007.

[6] M. Callieri, P. Cignoni, M. Corsini, and R. Scopigno, "Masked photo blending: mapping dense photographic dataset on high-resolution 3d models," *Computer & Graphics*, vol. 32, no. 4, pp. 464–473, Aug 2008.