Universidade Federal de Mato Grosso do Sul
Faculdade de Computação

Computação Gráfica 2022

Trabalho Prático

Paulo A. Pagliosa

O trabalho prático tem dois objetivos principais. O primeiro é uma extensão da API de geração de objetos poligonais por varredura de `Ds`, conforme descrito na Seção 1. O segundo é uma extensão de `cgdemo` com componentes que possibilitem ao usuário, interativamente. manipular os novos métodos de varredura e visualizar os objetos poligonais gerados, como descrito na Seção 2. (O código fonte de `Ds` e `cgdemo` estão disponíveis nos diretórios `cg` e `apps/cgdemo`, respectivamente, em `https://github.com/paulo-pagliosa/Ds`.) Além disso, a Seção 3 descreve uma extensão do traçado de raios básico de `cgdemo`. A implementação dessa extensão é *opcional* e vale como bônus na nota da **P1**.

1 Métodos de varredura

Um ator de uma cena é caracterizado por um modelo geométrico que define a geometria e topologia (formas, dimensões, posição, etc.) e também o material da superfície do ator. Em `Ds`, o modelo geométrico de um ator é representado por um objeto cuja classe deriva de `cg::Primitive`. Contudo. ao invés de ter um primitivo, em `Ds` um ator tem como atributo uma referência para um *mapeador de primitivo*. Este é um objeto de uma classe derivada da classe abstrata `cg::PrimitiveMapper`. A principal funcionalidade de um mapeador de primitivo é transformar um modelo geométrico em elementos gráficos que podem ser renderizados com `OpenGL`. Um objeto da classe `cg:TriangleMeshMapper`, derivada de `cg::PrimitiveMapper`, é um mapeador de primitivo especializado em renderizar modelos geométricos representados por malhas de triângulos. Uma malha de triângulos é um objeto da classe `cg::TriangleMesh`, definido por uma coleção de vértices (mais especificamente, as posições dos vértices em coordenadas locais da malha), normais (se presente, uma normal para cada vértice) e triângulos (um triângulo é caracterizado por três índices que definem seus vértices e normais).

Malhas de triângulos podem ser geradas por processos de *varredura*. Genericamente, um processo de varredura toma uma curva, chamada *geratriz*, e um caminho ao longo e/ou em torno do qual a geratriz efetua algum tipo de deslocamento. Os pontos resultantes do deslocamento da geratriz formam uma superfície no espaço que pode ser representada exata ou aproximadamente por uma malha de triângulos. Por exemplo, a superfície de uma esfera pode ser obtida pela *varredura rotacional* de 360° de um semicírculo em torno de um eixo que passa pelos pontos extremos do semicírculo. A implementação envolve a discretização da geratriz em uma *polilinha*, aberta ou fechada, com um dado número, n_p, de pontos. No caso da esfera, cada ponto, P, da polilinha aberta que aproxima o semicírculo (exceto os extremos) é rotacionado em torno do eixo de rotação. O círculo gerado pela rotação de P também é discretizado em um dado número, n_m, de pontos, isto é, a rotação é efetuada em n_m passos cujo ângulo é $360^{\circ}/n_m$. Com isso, o número total de vértices da "esfera", alem dos polos, é $(n_p - 2) \times n_m$, os quais devem ser ligados apropriadamente para formar os triângulos da malha. O método é implementado na classe `cg::MeshSweeper`.

O primeiro objetivo do trabalho é estender a classe `cg::MeshSweeper` com dois novos processos de varredura. Vamos considerar somente os dois tipos de geratriz a seguir.

- Polígono regular. Os pontos de um polígono são definidos em relação a um sistema local cuja origem é o centro do polígono. O circuncírculo do polígono tem raio unitário e está contido no plano xy, com normal na direção do eixo z do sistema local. O polígono tem um número de pontos, $3 \leq n_p \leq 40$, especificado pelo usuário. O primeiro ponto do polígono é $P_0 = (0, 1, 0)$. O ponto P_i, $i = 1, 2, \ldots, n_p - 1$, pode ser obtido pela rotação de P_0 em torno do eixo z do sistema local de um ângulo $360^{\text{o}}/n_p \times i$. Ao polígono pode ser aplicada uma transformação definida por uma escala $(s_p, s_p, 1)$ em torno da origem, $0.1 \leq s_p \leq 5$ (que altera o raio do circuncírculo do polígono), seguida de uma rotação de $0 \leq \theta_p \leq 360^{\text{o}}$ em torno do eixo z do sistema local, sendo s_p e θ_p parâmetros especificados pelo usuário.

- Arco de circunferência. Um arco de circunferência é caracterizado por um ângulo, $0 < \theta_c \leq 360^{\text{o}}$, e um número de segmentos de linha, $2 \leq n_c \leq 40$, em que o arco é subdividido, ambos especificados pelo usuário. Se $\theta_c < 360^{\text{o}}$, a subdivisão do arco resulta em uma polilinha com $n_p = n_c + 1$ pontos, que o usuário pode especificar como sendo aberta ou fechada. Se a polilinha for fechada, o último ponto é ligado ao primeiro por um segmento de linha adicional, ou seja, ao invés de um arco, a geratriz seria um setor circular. Se $\theta_c = 360^{\text{o}}$, a polilinha é um polígono com $n_p = n_c$ pontos. O arco tem centro na origem de seu sistema local e raio unitário, e contido no plano xy, com normal na direção do eixo z do sistema local. O primeiro e último pontos do arco são determinados a fim de que o arco seja simétrico em relação ao plano yz. Como no caso do polígono, ao arco pode ser aplicada uma transformação definida por uma escala $(s_p, s_p, 1)$ em torno da origem, $0.1 \leq s_p \leq 5$, seguida de uma rotação de $0 \leq \theta_p \leq 360^{\text{o}}$ em torno do eixo z do sistema local, sendo s_p e θ_p especificados pelo usuário.

Um dos processos de varredura é para geração de malhas de triângulos que aproximam *espirais*. Uma espiral pode resultar da rotação de uma geratriz (polígono ou arco) em torno do eixo y de seu sistema local. A rotação é dividida em um número de passos; a cada passo, a geratriz pode ser transladada ao longo do eixo de rotação e, em adição, a distância do centro da geratriz até o eixo de rotação pode ser aumentada. Os parâmetros de uma espiral são descritos a seguir.

- Largura inicial, $2b_x \leq w_e \leq 100$, da espiral, sendo b_x o tamanho da caixa envolvente, ou AABB, da geratriz ao longo do eixo x de seu sistema local. Para circuncírculos e arcos com raio unitário sujeitos a uma escala $(s_p, s_p, 1)$, $s_p > 0$, tem-se que $b_x = 2s_p$. A posição inicial da geratriz (no passo 0 de rotação) é definida por uma translação de $(d_e, 0, 0)$, em que $d_e = (w_e - b_x)/2$ é a distância inicial do centro da geratriz até o eixo de rotação (verifique).

- Revoluções, $0.1 \leq r_e \leq 20$, da geratriz. Uma revolução correspondente a um giro completo de 360^{o} da geratriz em torno do eixo y. O número pode ser fracionário. Por exemplo, $r_e = 2.5$ significa 2 revoluções e meia. O ângulo de rotação total da espiral é $\theta_e = 360^{\text{o}} \times r_e$.

- Número de subdivisões, $3 \leq n_e^s \leq 40$, de uma revolução. O número total de rotações da geratriz é $n_e^r = \max(\lfloor r_e + 0.5 \rfloor, 1) \times n_e^s$, e o número de passos é $n_e^r + 1$. Cada passo gera uma seção transversal da espiral com n_p vértices adicionados à malha de

triângulos, em que n_p é o número de pontos da geratriz. No passo 0, os vértices são os próprios pontos da geratriz. Os vértices em um passo k, $k = 1, 2, \ldots n_e^r$, são obtidos pela rotação dos pontos da geratriz em torno do eixo y de seu sistema local de um ângulo $\theta_e/n_e^r \times k$, seguida de translações na altura e largura.

- Incremento de altura, $0 \leq \delta_e^h \leq 10$, por revolução da geratriz. A variação de altura total da espiral é $\Delta_e^h = r_e \times \delta_e^h$. Em um passo $k > 0$, a geratriz sofre uma translação $(0, \Delta_e^h/n_e^r \times k, 0)$. Ao final, a altura da espiral fica $b_y + \Delta_e^h$, em que b_y é o tamanho da caixa envolvente (AABB) da geratriz ao longo do eixo y de seu sistema local. Para circuncírculos e arcos com raio unitário sujeitos a uma escala uniforme, tem-se $b_y = b_x$. Observação: se $\theta_e > 360^\text{o}$, isto é, se $r_e > 1$, deve-se ter $\delta_e^h \geq b_y$ para que a espiral sendo gerada não se auto-intercepte.

- Incremento de largura, $0 \leq \delta_e^w \leq 10$, por revolução da geratriz. A variação de largura total da espiral é $\Delta_e^w = r_e \times \delta_e^w$. Em um passo $k > 0$, a geratriz sofre uma translação $(d_e + \Delta_e^w/n_e^r \times k, 0, 0)$. Ao final, a largura da espiral fica $w_e + \Delta_e^w$. Observação: se $r_e > 1$, deve-se ter $\delta_e^w \geq b_x$ para que a espiral sendo gerada não se auto-intercepte.

Além disso, para geratrizes fechadas pode-se ter duas "tampas", uma em cada extremidade da espiral, conforme especificado pelo usuário. Cada tampa adiciona n_p vértices à malha. (Com isso, a malha terá vértices com coordenadas repetidas, mas normais distintas. Veja, por exemplo, o método de geração de cilindros em `cg::MeshSwepper`.) Por fim, cuidado especial deve ser tomado quando $\delta_e^h = \delta_e^w = 0$. Neste caso, deve-se ter $r_e < 1$ para que a espiral sendo gerada não se auto-intercepte. Se $r_e \geq 1$, a espiral vira um toro.

O outro processo diz respeito à varredura translacional com torção. Neste, a geratriz translada uma certa distância na direção do eixo z de seu sistema local, em um dado número de passos. Inicialmente, a geratriz pode sofrer um deslocamento no plano xy, o que tira seu centro da origem do sistema local. A cada passo, a geratriz pode sofrer uma transformação de escala seguida de rotação em torno do eixo z (daí a torção). Os parâmetros do método são descritos a seguir.

- Comprimento, $1 \leq l_v \leq 50$, do caminho da varredura, isto é, a distância de translação da geratriz ao longo do eixo z de seu sistema local.

- Deslocamento horizontal, $-b_x \leq o_v^w \leq b_x$, e deslocamento vertical, $-b_y \leq o_v^h \leq b_y$, da geratriz.

- Número de subdivisões, $1 \leq n_v^s \leq 50$, do caminho da varredura. O número de passos da varredura é $n_v^s + 1$. Cada passo gera uma seção transversal da superfície com n_p vértices adicionados à malha de triângulos, em que n_p é o número de pontos da geratriz. Os vértices em um passo k, $k = 0, 1, \ldots n_v^s$, são obtidos por uma transformação dos pontos da geratriz composta de uma escala em torno da origem, uma translação de $(o_v^w, o_v^h, l_v/n_v^s \times k)$ e uma rotação em torno do eixo z, nesta ordem.

- Escala inicial, $0.1 \leq s_v^b \leq 5$, e escala final, $0.1 \leq s_v^e \leq 5$, da geratriz. Em um passo $k \geq 0$, a geratriz sofre uma escala $(s_v, s_v, 1)$, em que $s_v = s_v^b + (s_v^e - s_v^b)/n_v^s \times k$.

- Torção, $-2 \leq r_v \leq 2$, da geratriz. Um valor igual a 1 correspondente a uma rotação completa de 360^o da geratriz em torno do eixo z. O número pode ser fracionário. O ângulo de rotação total da geratriz ao final da varredura é $\theta_v = 360^\text{o} \times r_v$. Em um passo $k \geq 0$, a geratriz sofre uma rotação em z de um ângulo $\theta_v/n_v^s \times k$.

Da mesma forma que na espiral, se a geratriz for fechada e o usuário assim especificar, a superfície pode ser fechada por "tampas" em cada uma de suas extremidades.

2 Componentes da interface gráfica

O segundo objetivo do trabalho é criar componentes na interface gráfica (GUI) de `cgdemo` a fim de que o usuário possa, interativamente, especificar os parâmetros dos processos de varredura descritos na Seção 1 e, em tempo real, visualizar a malha sendo gerada.

Em linhas gerais, os dois novos componentes, um para cada processo de varredura, funcionam como um tipo de *proxy de primitivo*. Um proxy de primitivo é um componente que "envelopa" o mapeador de primitivo de um ator da cena. O proxy de primitivo de um processo de varredura é um objeto de classe derivada de `cg::graph::PrimitiveProxy` cujo mapeador de primitivo é do tipo `cg::TriangleMeshMapper`. É este mapeador que mantém uma referência para a malha de triângulos gerada pela varredura.

Em comum, a GUI dos proxies de processos de varredura deve prover controles da Dear ImGui[1]para ajuste dos parâmetros da geratriz: tipo de "curva" (polígono ou arco), número de pontos ou segmentos, ângulo e se aberta ou fechada (no caso de arco), escala e ângulo de rotação. Como referência do uso da Dear ImGui, veja os métodos de inspeção de componentes na classe `cg::graph::SceneWindow`. Qualquer alteração feita pelo usuário em algum parâmetro implica na reconstrução da (polilinha da) geratriz. Em adição, a GUI de cada proxy deve ter controles para ajuste dos parâmetros específicos do processo de varredura correspondente. Para a varredura transacional com torção, por exemplo, comprimento, deslocamentos horizontal e vertical da geratriz, escalas inicial e final, número de subdivisões e torção. Qualquer alteração feita pelo usuário em algum parâmetro ou qualquer alteração na geratriz implica na reconstrução da malha de triângulos. Cada parâmetro deve ter um valor default e restrito aos limites estabelecidos na Seção 1.

A GUI dos proxies deve ter, ainda, um controle para especificação de um nome para a malha de triângulos e um botão para "salvar" a malha de triângulos em arquivo no formato Wavefront OBJ.[2]Como referência, veja a classe `cg::MeshReader`. O arquivo deve ser salvo no diretório `assets/meshes`, com o nome dado para a malha e extensão `.obj`. Com isso, a malha resultante da varredura poderá ser usada em outros objetos de cena.

3 Traçado de raios adaptativo

Para cada pixel de uma imagem, o traçador de raios básico implementado em `cgdemo` (veja o arquivo de cabeçalho `RayTracer.h` e o arquivo de código fonte `RayTracer.cpp`) traça um *único* raio da câmera na direção do *centro* do pixel, sendo a cor do pixel igual a cor do raio. Se o raio de pixel não interceptar ator algum, então sua cor é igual a cor de fundo da cena. Caso contrário, a cor do raio é soma das contribuições da iluminação direta e indireta no ponto de interseção do raio com o ator mais próximo da origem do raio. A iluminação direta é computada, para cada fonte de luz cujos raios chegam no ponto de interseção, pelo modelo de iluminação local de Phong, o qual depende da intensidade da luz e do material e da normal à superfície no ponto de interseção. A iluminação indireta é computada unicamente em termos da reflexão difusa (aproximada pela iluminação ambiente da cena) e

[1] `https://github.com/ocornut/imgui`

[2] `https://en.wikipedia.org/wiki/Wavefront_.obj_file`

reflexão especular (determinada pelo traçado recursivo de raios secundários de reflexão). A profundidade máxima da árvore de recursão e o peso mínimo dos raios secundários podem ser especificados na GUI da aplicação.

Um dos problemas com o traçado de raios básico é o chamado *aliasing*, resultante do processo de amostragem de uma função contínua em poucos pontos discretos, nesse caso, os centros dos pixeis. Como cada pixel na janela de projeção é uma área através da qual a câmera poderia "ver" muitos objetos, a cor devida a somente um raio (passando pelo centro) pode não ser representativa de toda porção da cena visível no pixel. O efeito do *aliasing* é percebido, por exemplo, por "serrilhados" na imagem renderizada. Há técnicas de *antialiasing* que, embora não eliminem totalmente os artefatos causados pelo *aliasing*, amenizam o problema, como descrito a seguir.

Para cada pixel, p, ao invés de um único raio passando pelo seu centro, a técnica consiste em traçar, inicialmente, quatro raios, $\mathrm{R}_{p_i}, i = 1, 2, 3, 4$, passando em cada um dos cantos do pixel, Figura 1 (esquerda). A seguir, computa-se a média, $M_p = \sum_{i=1}^{4} C_{p_i}/4$, das cores desses quatro raios e compara-se cada cor, C_{p_i}, com a média. O resultado dessa comparação é um número que mede o quão uma cor se afasta da cor média:

$$\delta_{p_i} = \mathrm{maxabs}(C_{p_i} - M_p),$$

em que $\mathrm{maxabs}(C)$ denota o máximo componente da cor C em valor absoluto. Se todo δ_{p_i} for menor que determinado limiar, que pode ser chamado de *distância de adaptatividade*, então considera-se que não há variação suficiente dentro do pixel, e a cor de p é igual a média M_p. Caso contrário, o pixel é subdivido em quatro "subpixels", Figura 1 (centro). Para cada subpixel, repete-se o processo recursivamente, isto é, traçam-se quatro raios passando pelos cantos do subpixel, computam-se suas cores e verifica-se o quão estas se afastam da média; se houver variação suficiente, o subpixel é subdividido novamente. Com dois níveis de subdivisão, o número máximo de raios por pixel é igual a vinte e cinco, Figura 1 (direita). O valor da distância de adaptatividade, $0 < \delta_a < 1$, por exemplo, pode ser definido pelo usuário na GUI, bem como o nível máximo de subdivisão, $0 \leq n_a \leq 4$ (se $n_a = 0$, então não é feita a superamostragem).

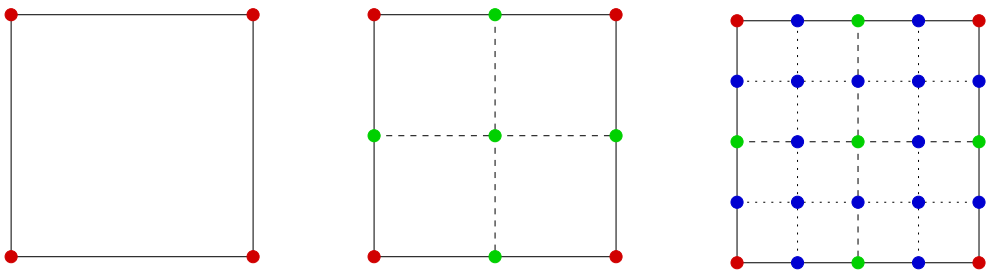

Figura 1: Superamostragem adaptativa. Esquerda: quatro raios são traçados nos cantos do pixel (pontos em vermelho). Centro: subdivisão do pixel em quatro subpixels (no máximo cinco novos raios são traçados nos pontos em verde). Direita: segundo nível de subdivisão, cada subpixel subdividido em outros quatro subpixels (no máximo dezesseis novos raios são traçados nos pontos em azul).

Com este esquema de *superamostragem adaptativa*, poder-se-ia supor que o número de raios traçados é, no mínimo, quatro vezes o número de pixels da janela de projeção. Contudo, exceto para os pixels dos cantos e das bordas da janela, o número de raios *novos* para um pixel p é, como na versão básica, igual a um. Esse novo raio é correspondente ao canto inferior direito do pixel, uma vez que os raios dos cantos superiores e inferior esquerdo já foram traçados quando do processamento dos pixels acima e à esquerda de p,

respectivamente. Portanto, é imprescindível, para maior eficiência da técnica, o emprego de *bufferes* para armazenamento das cores dos raios anteriormente traçados. O mesmo cuidado vale para os raios traçados durante a subdivisão do pixel e que podem ser compartilhados por pixels e subpixels adjacentes (veja a Figura 1).

4 Tarefas

O primeiro passo é baixar o código em `https://github.com/paulo-pagliosa/Ds`. Seja `$/Ds` o diretório contendo `Ds`. Em seguida, crie um diretório `$/Ds/apps/tp` e clone, neste diretório, o conteúdo do diretório `$/Ds/apps/cgdemo`. Este conterá, unicamente, código a ser estendido.

As atividades do trabalho consistem em projetar e implementar:

A1 Uma classe para representação de polilinhas. Dica: para armazenamento dos pontos de uma polilinha, use um vetor com capacidade igual ao número máximo de pontos de uma geratriz.

A2 O processo de varredura para geração de espirais. Dica: o processo pode ser implementado por uma função cujos parâmetros são os parâmetros da varredura e cujo tipo de retorno é um ponteiro para a malha de triângulos gerada. Ou, o processo pode ser um objeto cujos atributos são a geratriz e os parâmetros do processo, com métodos para obtenção e ajuste dos valores dos parâmetros bem como reconstrução e obtenção da malha. Um aspecto a ser considerado é que a malha é reconstruída *interativamente*, cada vez que o usuário altera algum parâmetro da geratriz ou do método. Assim, o número de vértices e/ou triângulos pode variar a todo instante e, em consequência, a memória usada para armazenamento dos elementos da malha. A fim de evitar liberação e (re)alocação de memória a cada reconstrução da malha, pode-se empregar (ou não) um *buffer* temporário para vértices e triângulos. Então, ao final da "edição", os vértices e triângulos são copiados para a malha.

A3 O proxy do processo da tarefa **A2**. O componente é *requerido* para verificação visual da corretude do processo de varredura. (Afinal, este é um trabalho de computação gráfica, e a implementação de um sem a do outro dificulta a avaliação de ambos.)

A4 O processo de varredura translacional com torção. Valem as dicas da tarefa **A2**.

A5 O proxy do processo da tarefa **A4**. Valem as considerações da tarefa **A3**.

A6 A escrita de arquivos no formato Wavefront OBJ, a fim de "salvar" as malhas geradas pelos processos de varredura como *assets* para uso em outros objetos de cena.

A7 A escrita de arquivos de cenas. Esta tarefa é optativa e vale um bônus de até **1.0** na nota do trabalho. O formato do arquivo é definido pela gramática LL(1) especificada em `$/Ds/apps/tp/reader/grammar.txt`. Cenas são salvas em `assets/scenes`. Como referência, veja o arquivo `$/Ds/apps/tp/scenes/test`. Para os proxies objetos das tarefas **A3** e **A5**, apenas as referências aos arquivos OBJ das malhas geradas são escritas no arquivo de cena, uma vez que um leitor de cena tem métodos para carregar uma malha, mas não um proxy de primitivo (mais detalhes em sala).

A8 A superamostragem adaptativa como descrito na Seção 3. Esta tarefa é optativa e vale um bônus de até **2.5** pontos na nota da **P1**. Se implementada, anexe imagens para comparações da renderização de cenas com e sem a superamostragem.

5 Entrega do trabalho

Além do código fonte do programa, deve ser entregue um arquivo **texto** chamado `README`. Este deve conter o(s) nome(s) do(s) autor(es) e uma descrição de como gerar (caso o Visual Studio 2022 não seja utilizado) e executar o programa. Por exemplo, quais são os argumentos da linha de comando, se o programa deve ser executado de um diretório específico, etc. Em adição, devem estar descritas quais as atividades foram ou não implementadas, parcial ou totalmente, ou se foram implementadas quaisquer outras extensões por iniciativa própria do(s) autor(es), além de um breve manual do usuário (descrevendo os valores default dos parâmetros dos processos de varredura e, por exemplo, se é preciso usar alguma tecla para alguma funcionalidade para a qual não há ajuda textual na GUI).

O trabalho deve ser entregue via AVA em arquivo único compactado (somente um arquivo por grupo), chamado `tp.zip` (nome(s) do(s) autor(es) vão no arquivo `README`). Preste atenção, vamos repetir para não deixar dúvidas: a submissão deve ser feita em um arquivo compactado do tipo `zip` cujo nome deve ser `tp.zip`. Este deve conter **somente** o conteúdo do diretório `$/Ds/apps/tp`, o que inclui:

- o código **fonte** completo do trabalho (mas **sem** o arquivo executável **nem** os arquivos intermediários de backup ou resultantes da compilação);
- makefile para geração do executável com g++, se desenvolvido em Linux;
- o arquivo `README`;
- arquivos de malhas geradas pelos processos de varredura; e, opcionalmente,
- arquivos de cenas e de imagens para testes do programa. As cenas devem ser as mais criativas possíveis.

Não serão considerados e terão nota zero programas plagiados de qualquer que seja a fonte, mesmo que parcialmente, exceção feita ao código fornecido pelo professor. O trabalho deve ser desenvolvido em grupos de **dois** ou **três** estudantes. Excepcionalmente, poderão ser aceitos trabalhos com um único autor.

6 Lista de objetivos

A avaliação do trabalho prático levará em conta se:

☐ O arquivo `tp.zip` foi submetido com os arquivos corretos.

☐ `README` contém as informações solicitadas.

☐ O processo de geração de espirais foi implementado e sua corretude pode ser verificado interativa e visualmente pela inspeção do proxy correspondente.

☐ O processo de varredura translacional com torção foi implementado e sua corretude pode ser verificado interativa e visualmente pela inspeção do proxy correspondente.

☐ A escrita de arquivos OBJ foi implementada corretamente.

☐ Opcionalmente, a escrita de arquivos de cena e/ou a superamostragem adaptativa foi ou foram implementada(s) corretamente.