Visualização e Iluminação - Parte III

Rafael Correia (pg54162)
Robert Szabo (pg54194)

14 de junho de 2024

Resumo

Este relatório documenta a implementação de uma de janela interativa no projeto realizado ao longo do semestre, com suporte para exibição de imagens estáticas e progresso da renderização em tempo real. Além disso, foi desenvolvida e implementada uma técnica de *Progressive Path Tracing*, que permite a renderização incremental de imagens com controlo de amostragem por pixel. A integração foi realizada utilizando a biblioteca *OpenGL* para renderização em janelas e a estrutura existente do projeto para gerir cenas, câmeras e shaders. Este documento descreve as decisões tomadas, os desafios encontrados e as soluções adotadas.

1 Introdução

Este relatório tem como objetivo documentar a implementação de novas funcionalidades ao sistema de renderização de imagens desenvolvido ao longo do semestre. O foco principal é a adição de uma classe de janela interativa para a visualização em tempo real das imagens geradas e a introdução de um renderizador progressivo (*Progressive Path Tracing*).

A necessidade dessas melhorias surgiu com o propósito de oferecer uma experiência mais dinâmica e interativa durante o desenvolvimento e testes. A implementação de uma janela interativa permite a visualização contínua do progresso da renderização, facilitando a identificação de problemas em tempo real.

A técnica de *Progressive Path Tracing* foi escolhida pela capacidade de renderizar imagens de forma incremental, melhorando progressivamente a qualidade da imagem à medida que mais amostras são calculadas. Isso não apenas contribui com um *feedback* visual para o utilizador, mas também permite pausas durante o processo.

Ao longo deste relatório, serão detalhadas as decisões tomadas durante a implementação, os desafios enfrentados e as soluções adotadas para integrar essas novas funcionalidades à estrutura existente do projeto.

2 Contextualização

O projeto de Visualização e Iluminação desenvolvido ao longo do semestre é composto por várias classes e módulos, cada um responsável por um aspeto específico do processo de renderização. A estrutura do projeto é modular e foi concebida para facilitar a adição de novas funcionalidades e a manutenção do código.

2.1 Estrutura do Projeto

A estrutura principal do projeto é dividida nas seguintes partes:

- **Scene**: Responsável por carregar e armazenar a geometria da cena, incluindo objetos, luzes e materiais. A classe *Scene* gere a interação entre esses elementos durante o processo de renderização.
- **Camera**: Define a posição e a orientação da câmara na cena. A classe *Camera* é responsável por gerar os raios primários que são lançados na cena para calcular a cor dos píxeis.
- **Image**: Armazena os dados da imagem renderizada. A classe *Image* gere o armazenamento e a manipulação dos valores de cor para cada píxel da imagem.

- **Shader**: Implementa os diferentes métodos de *shading*. As classes derivadas de *Shader*, como *WhittedShader*, *DistributedShader* e *PathTracerShader*, definem como a luz interage com os materiais na cena para calcular a cor final dos píxeis.
- **Renderer**: Gere o processo de renderização. A classe *Renderer* é a base para os renderizadores específicos, como o *StandardRenderer* e o *ProgressiveRenderer*, que implementam diferentes estratégias de renderização.
- **Light**: Define as fontes de luz na cena. As classes *AmbientLight*, *PointLight* e *AreaLight* especificam diferentes tipos de luz e como elas contribuem para a iluminação da cena.
- **Utils**: Contém funções utilitárias e estruturas de dados auxiliares utilizadas em todo o projeto, como vetores e cores.

2.2 Implementações Anteriores

No que diz respeito à iluminação, diferentes *shaders* foram desenvolvidos para implementar variadas técnicas de *shading*. Entre estes *shaders*, destacam-se:

- **AmbientShader**: Implementa *shading* ambiente básico, considerando apenas a componente ambiental das fontes de luz.
- **WhittedShader**: Implementa o algoritmo de *shading* de Whitted, que inclui reflexões especulares e iluminação direta de fontes de luz pontuais.
- **DistributedShader**: Extende o algoritmo de Whitted para incluir reflexões especulares distribuídas e *shading* de luzes de área.
- **PathTracerShader**: Implementa o *path tracing*, uma técnica de renderização que simula a propagação da luz através da cena, incluindo interações complexas como difusão e reflexões especulares.

Adicionalmente, foram desenvolvidas classes como o **StandardRenderer** que realiza a renderização de forma tradicional, computando todas as amostras por pixel e gerando a imagem final num único passo.

Esta estrutura modular fornece a base para a adição das novas funcionalidades discutidas neste relatório, como a janela interativa e o renderizador progressivo.

3 Desenvolvimento da Janela Interativa

3.1 Motivação e Objetivos

Com o objetivo de melhorar a interatividade e a visualização do processo de renderização, foi desenvolvida uma janela interativa para o projeto. A motivação principal para esta adição é proporcionar aos utilizadores uma forma de visualizar o progresso da renderização em tempo real e interagir com a cena enquanto a renderização está a decorrer. Além disso, a janela interativa permite pausar e retomar a renderização, o que pode ser útil para ajustar parâmetros e observar mudanças sem ter de esperar pela conclusão de uma renderização completa.

Os objetivos específicos do desenvolvimento da janela interativa são:

- **Visualização em Tempo Real**: Permitir aos utilizadores ver o progresso da renderização em tempo real.
- **Interação com a Cena**: Fornecer controlos para pausar, retomar e terminar a renderização, bem como futuras extensões para mover a câmara e alterar parâmetros da cena.
- **Integração com o Projeto Existente**: Garantir que a janela interativa se integra perfeitamente com a estrutura existente do projeto, utilizando as classes e métodos já implementados.

A implementação da janela interativa foi realizada utilizando a biblioteca *OpenGL*, que oferece suporte robusto para renderização gráfica e interatividade. Esta escolha permite tirar partido das capacidades de hardware gráfico moderno para apresentar imagens de alta qualidade de forma eficiente.

3.2 Implementação da Classe *Window*

A implementação da classe *Window* foi fundamental para proporcionar a visualização interativa do processo de renderização. Utilizando a biblioteca *OpenGL*, a classe foi projetada para inicializar uma janela gráfica, atualizar a imagem renderizada em tempo real e responder a eventos de interação do utilizador.

3.2.1 Inicialização da Janela

A inicialização da janela é realizada através do construtor `Window::Window(int width, int height, const char* title)`, que configura o contexto *OpenGL*, define o *viewport* e os parâmetros de projeção, e cria a textura que será utilizada para armazenar e exibir a imagem renderizada. Esta função também configura a função de *callback* para eventos do teclado, permitindo a interação do utilizador com a renderização.

3.2.2 Renderização e Atualização

O método `Window::render(const unsigned char* imageData)` é responsável por iniciar o loop de renderização, que mantém a janela aberta e exibe continuamente a textura atualizada. Este método liga a textura contendo os dados da imagem renderizada e desenha um quadrado que cobre toda a janela, utilizando coordenadas de textura para mapear a imagem corretamente.

Para atualizar a imagem durante a renderização progressiva, o método `Window::update(const unsigned char* imageData)` é utilizado. Este método substitui os dados da textura com os novos dados de imagem fornecidos, e redesenha o conteúdo da janela. A utilização deste método permite que a janela exiba a imagem mais recente da renderização, proporcionando assim um *feedback* visual contínuo ao utilizador.

3.2.3 Interação do Utilizador

A interação do utilizador é gerida através da função de *callback* para eventos de teclado, configurada na inicialização da janela. Este *callback* permite pausar e retomar a renderização, bem como finalizar o processo e fechar a janela. Estas funcionalidades são essenciais para permitir ao utilizador controlar o processo de renderização de forma interativa e intuitiva.

A classe de janela foi integrada ao projeto de forma a garantir que ela funcione harmoniosamente com as classes existentes para gestão de cenas, câmaras e *shaders*, permitindo uma visualização eficiente e interativa do processo de renderização.

4 Implementação do Progressive Path Tracing

4.1 Descrição da Técnica

O *Progressive Path Tracing* é uma técnica de renderização que permite a geração incremental de uma imagem, melhorando gradualmente a sua qualidade à medida que mais amostras são calculadas para cada pixel. Esta técnica é particularmente útil em aplicações interativas, onde uma imagem inicial pode ser gerada rapidamente, e refinada ao longo do tempo, proporcionando uma visualização contínua e em tempo real do progresso da renderização.

No *Progressive Path Tracing*, a cor final de cada pixel é calculada a partir da média de várias amostras obtidas através de traçados de raios. A cada iteração, um novo conjunto de raios é gerado e traçado através da cena, contribuindo para a cor acumulada do pixel. Este processo é repetido até que o número desejado de amostras por pixel (spp) seja alcançado, resultando numa imagem de alta qualidade com redução progressiva de ruído.

4.2 Decisões de Design

A implementação do *Progressive Path Tracing* exigiu uma série de decisões de design para garantir a sua integração eficiente no projeto existente e para maximizar a interatividade e qualidade da renderização. Algumas das principais decisões de design foram:

- **Classe** `ProgressiveRenderer`: Criámos uma nova classe chamada `ProgressiveRenderer` para gerir o processo de renderização progressiva. Esta classe foi derivada da classe base `Renderer`, mantendo a estrutura do projeto.

- **Buffer de Acumulação**: Implementámos um *buffer* de acumulação (`accumulationBuffer`) para armazenar as cores acumuladas de cada pixel. A cada iteração, as novas amostras são adicionadas a este *buffer*, e a cor média é calculada para atualizar a imagem exibida.
- **Normalização das Cores**: Para evitar que as cores acumuladas saturassem, decidimos normalizar os valores das cores dividindo pelo número de amostras acumuladas. Este processo é realizado continuamente durante a renderização para proporcionar uma imagem visualmente precisa em cada etapa.
- **Integração com a Janela Interativa**: A classe `ProgressiveRenderer` foi integrada com a classe de janela (`Window`) para permitir a atualização em tempo real da imagem renderizada. Utilizámos o método `update` da janela para atualizar a textura *OpenGL* com os dados da imagem a cada iteração.
- **Métodos de Renderização e Atualização**: Implementámos os métodos `render` e `renderFrame` na classe `ProgressiveRenderer`. O método `render` coordena o ciclo de renderização progressiva, enquanto o método `renderFrame` realiza o traçado de raios para cada pixel e acumula as cores.
- **Controlo de Interatividade**: Adicionámos controlos para pausar e retomar a renderização, bem como para encerrar a aplicação de forma limpa. Estes controlos foram implementados na classe `Window` e integrados no ciclo de renderização da `ProgressiveRenderer`.
- **Compatibilidade com Shaders Existentes**: Optámos por reutilizar o `PathTracerShader` existente para o *shading*, garantindo consistência na renderização e evitando a necessidade de criar um novo *shader* específico para a renderização progressiva.
- **Salvar Imagens Intermediárias**: Implementámos a funcionalidade para salvar imagens intermediárias a cada certo número de amostras, permitindo a análise do progresso da renderização ao longo do tempo. Esta funcionalidade é útil para depuração e para entender melhor o processo de convergência da imagem.

4.3 Desafios e Soluções

A implementação do *Progressive Path Tracing* apresentou diversos desafios que foram abordados com soluções específicas. Abaixo estão alguns dos principais desafios e as soluções adotadas:

- **Gestão do Buffer de Acumulação**: Um dos desafios foi gerir eficientemente o buffer de acumulação para garantir que as cores dos píxeis fossem corretamente acumuladas e normalizadas. Inicialmente, verificámos que a imagem ficava progressivamente mais escura devido a uma divisão incorreta pelos valores acumulados. **Solução** Normalizámos os valores no momento da atualização da imagem, assegurando que a divisão pelo número de amostras (`frameCount`) fosse feita corretamente.
- **Integração com a Classe de Janela**: A integração da classe `ProgressiveRenderer` com a classe de janela (`Window`) teve desafios relacionados com a sincronização dos dados de imagem e a atualização da textura *OpenGL*. **Solução** Implementámos o método `update` na classe `Window` para garantir que a textura fosse atualizada de forma eficiente e sincronizada com os dados acumulados na `ProgressiveRenderer`.
- **Compatibilidade com Imagens Estáticas e Progressivas**: Manter a compatibilidade entre a renderização estática e a progressiva foi um desafio significativo, especialmente na gestão de diferentes shaders e técnicas de renderização. **Solução** Implementámos uma lógica condicional no `main` para selecionar o tipo de renderização e o shader apropriado, garantindo que ambos os modos pudessem coexistir sem conflitos.
- **Salvar Imagens Intermediárias**: Um desafio adicional foi garantir que as imagens intermediárias pudessem ser salvas corretamente durante o processo de renderização progressiva. **Solução** Utilizámos `dynamic_cast` para garantir que a classe `ImagePPM` fosse corretamente identificada e utilizada para salvar imagens em disco, assegurando que o método `Save` fosse chamado adequadamente.
- **Atualização da Imagem em Tempo Real**: Garantir que a imagem fosse atualizada em tempo real sem causar atrasos significativos foi um desafio técnico. **Solução** Implementámos um ciclo de atualização eficiente, onde a janela chamava o método `update` regularmente

para refletir as novas amostras acumuladas, mantendo a interatividade e a responsividade da aplicação.

- **Pausar e Retomar Renderização**: Implementar funcionalidades para pausar e retomar a renderização exigiu uma lógica adicional para gerir o estado da renderização. **Solução** Adicionámos variáveis de estado (`paused` e `running`) e métodos correspondentes na classe `Window` para controlar o ciclo de renderização de forma eficaz, permitindo que o utilizador pausasse e retomasse a renderização conforme necessário.

5 Integração e Testes

5.1 Reestruturação da *main*

Para acomodar as novas funcionalidades de janela interativa e *Progressive Path Tracing*, a função `main` foi significativamente reestruturada. Abaixo descrevem-se as principais alterações realizadas:

- **Inicialização de Parâmetros pelo Utilizador**: Foram introduzidos métodos que permitem ao utilizador selecionar diferentes configurações, tais como os parâmetros da câmara, o nome do ficheiro `.obj` a ser carregado para a cena, o tipo de renderizador e shader, o número de amostras por píxel, o número de luzes e a ativação do jitter e *Russian Roulette* (no caso do *Path Tracer*). Estes conjunto de métodos foi definido no ficheiro `customValues`, leem a entrada do utilizador e configuram os parâmetros adequadamente.

- **Seleção de Renderizador e Shader**: Dependendo das escolhas do utilizador, o renderizador apropriado (`StandardRenderer` ou `ProgressiveRenderer`) e o shader correspondente (`WhittedShader`, `DistributedShader` ou `PathTracerShader`) são instanciados. Isto garante que a aplicação é flexível e pode se adaptar a diferentes necessidades de renderização.

- **Inicialização da Janela Interativa**: A janela interativa é criada utilizando a classe `Window`. A câmara configurada pelo utilizador é passada para a janela, permitindo a renderização interativa em tempo real.

- **Ciclo de Renderização**: Para o `ProgressiveRenderer`, a função `render` é chamada num loop que permite a renderização progressiva da cena. Durante cada iteração, a imagem é atualizada na janela e as imagens intermediárias são salvas para depuração e análise de progresso.

- **Salvamento da Imagem Final**: Após a conclusão da renderização, a imagem final é salva utilizando a função `Save` da classe `ImagePPM`. Esta operação garante que o resultado da renderização é preservado em disco.

5.2 Testes e Validação

Os testes realizados focaram-se em garantir a funcionalidade correta e a integração das novas funcionalidades. Abaixo estão algumas áreas de destaque dos testes realizados:

- **Verificação da Reestruturação da `main`**: Testou-se a capacidade de selecionar diferentes configurações de renderizador e shader. Garantiu-se que as escolhas do utilizador fossem corretamente refletidas na configuração do renderizador e do shader.

- **Renderização Interativa**: Testou-se a atualização da imagem em tempo real na janela, garantindo que o `Progressive Path Tracing` produzia uma imagem progressivamente menos ruidosa. Verificou-se a suavidade da atualização e a precisão das cores.

- **Comparação de Desempenho**: Comparou-se o desempenho do `Progressive Renderer` com o `Standard Renderer`. Notou-se que o `Progressive Renderer` é mais lento devido às operações adicionais de atualização da janela e salvar imagens intermediárias. Estas operações são essenciais para fornecer *feedback* visual contínuo e monitorizar o progresso da renderização.

- **Controlo Interativo**: Verificou-se a funcionalidade de pausar/retomar e parar a renderização.

- **Salvar Imagem Final**: Validou-se a integridade das imagens salvas em disco, garantindo que as cores e detalhes das imagens renderizadas eram precisos. Testes foram realizados para assegurar que as imagens intermediárias e finais eram salvas corretamente.
- **Compatibilidade com Diferentes Shaders e Renderizadores**: Realizaram-se testes com diferentes combinações de shaders e renderizadores, verificando a compatibilidade e desempenho de cada configuração.

Os testes confirmaram que as funcionalidades implementadas operavam conforme o esperado, proporcionando uma renderização interativa suave e precisa. A integração da janela com suporte para *Progressive Path Tracing* melhorou significativamente a experiência do utilizador, permitindo um *feedback* visual contínuo e controlo interativo durante o processo de renderização.

6 Conclusão

Este relatório documentou a implementação e integração de uma janela interativa e a técnica de *Progressive Path Tracing* no projeto de renderização. Através da reestruturação da `main`, foi possível permitir ao utilizador selecionar diferentes configurações de renderização e shader, garantindo flexibilidade e personalização. Além disso, a janela interativa proporciona um feedback visual contínuo, permitindo ao utilizador monitorizar o progresso da renderização em tempo real.

Foram realizados testes abrangentes para garantir a funcionalidade correta e a integração das novas funcionalidades. Apesar do sucesso na implementação das principais funcionalidades, foram identificadas algumas áreas para melhorias futuras:

- **Movimentação da Câmara**: Foi realizada uma tentativa de implementar a movimentação da câmara que se revelou-se um desafio significativo. A integração de um sistema de controlo de câmara que permita movimentos pode melhorar substancialmente a experiência do utilizador. Esta funcionalidade permitiria explorar a cena de diferentes ângulos durante a renderização, mas reiniciando o processo de renderização.
- ***Tone Mapper***: Implementar um *tone mapper* mais sofisticado pode melhorar a qualidade visual das imagens renderizadas. A técnica atual pode ser otimizada para lidar melhor com altos contrastes e preservar detalhes em áreas com iluminação extrema, resultando em imagens mais realistas e visualmente agradáveis.
- **Otimização de Desempenho**: Embora o *Progressive Path Tracing* forneça um feedback visual contínuo, ele é inerentemente mais lento devido às operações de atualização da janela e salvamento de imagens intermediárias. Investigar técnicas de otimização, como paralelização e uso de GPUs, pode ajudar a mitigar esse impacto de desempenho.
- **Interface de Utilizador Melhorada**: A adição de uma interface gráfica de utilizador (GUI) para controlar as configurações de renderização em tempo real pode tornar a aplicação mais amigável. A GUI pode permitir ajustes dinâmicos das configurações de iluminação, shaders e parâmetros de renderização sem necessidade de reiniciar a aplicação.

Apesar dos desafios, a integração da janela interativa e do *Progressive Path Tracing* representa um avanço significativo no projeto, proporcionando uma base sólida para futuras melhorias e inovações na renderização interativa.

7 Anexos

7.1 Testes - Imagens Intermédias

Para uma resolução 512x512, usando *Path Tracer Shader* (com Russian Roulette) e *Progressive Renderer*.

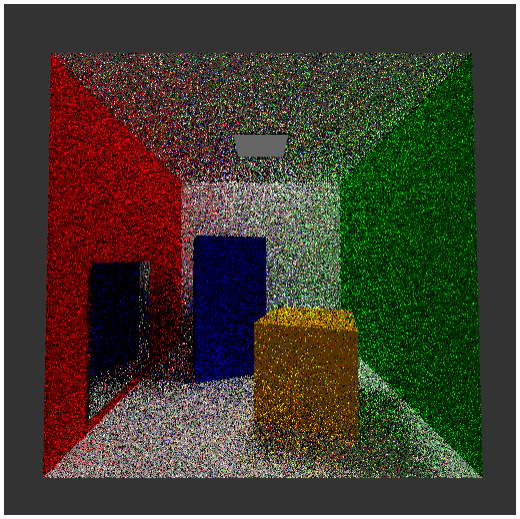

Figura 1: Imagem após 1 amostra por pixel

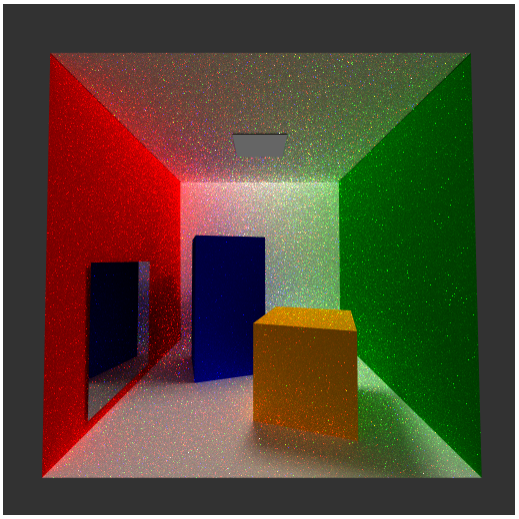

Figura 2: Imagem após 1024 amostras por pixel

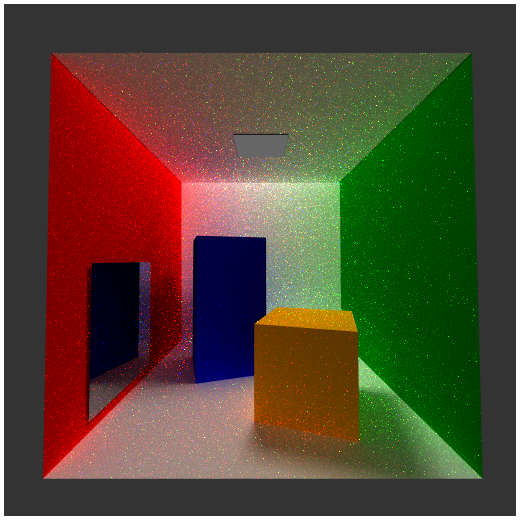

Figura 3: Imagem após 2048 amostras por pixel

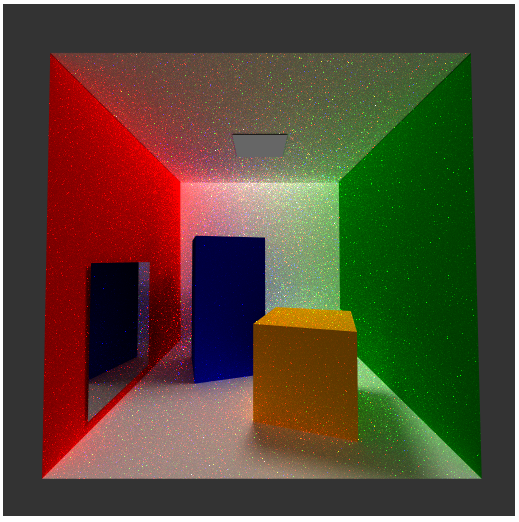

Figura 4: Imagem após 3072 amostras por pixel