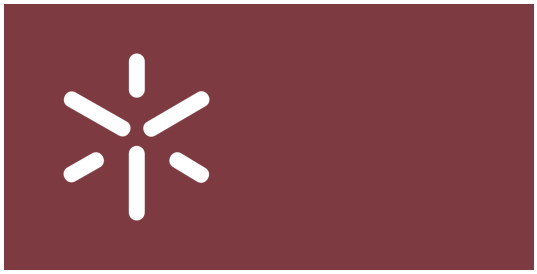

Universidade do Minho

José Malheiro ($PG50509$), **David Duarte** ($PG50315$),
Francisco Peixoto ($PG47194$)

Mestrado em Engenharia Informática

Projeto VI

Julho 2023

Visualização e Iluminação

Conteúdo

1 Parallel Multithreading

1.1 Render

A classe ***StandardRenderer*** é responsável pela renderização da cena, que é um processo intrinsecamente paralelizável. A ideia principal para a introdução do Multithreading foi dividir a imagem a ser renderizada em várias partes e alocar cada parte para uma thread individual. O OpenMP é utilizado para paralelizar a renderização, especificamente nos loops que percorrem

cada pixel da imagem. A complexidade destes 'loops' é **O(W * H * spp)**, onde **W** e **H** são a largura e a altura da imagem, respetivamente, e **spp** é o número de 'samples' por píxel.

A principal vantagem de introduzir o multithreading aqui é a redução do tempo de renderização. No entanto, é importante realçar que a gestão de threads introduz alguma sobrecarga, que pode tornar-se significativa se o número de threads for muito grande, portanto no projeto definiu-se um número de threads fixo igual a 80% do total disponível (todos os testes foram realizados com 10 Threads).

1.2 Shader

A classe ***PathTracerShader*** é usada para introduzir sombras na cena, que é um processo potencialmente paralelizável, principalmente no que diz respeito ao cálculo das contribuições de várias fontes de luz. No entanto, este processo é complicado pela necessidade de recursão para calcular reflexões especulares e difusas, o que tornou a paralelização mais difícil.

A complexidade dos métodos na classe ***PathTracerShader*** varia de ***O(1)*** para operações simples, como cálculo de iluminação direta, a ***O(num_samples * num_area_lights)*** para o cálculo da contribuição de luzes de área.

A introdução do multithreading não acelerou o processo de renderização, pelo que se optou por não introduzir o multithreading nesta parte, também havia a possibilidade de existirem race conditions ao utilizar recursos partilhados, outro factor que influenciou a tomada de decisão.

1.3 Tracer

A classe ***Scene*** é usada para rastrear raios e encontrar interseções com as primitivas da cena. Os métodos desta classe têm uma complexidade que varia de

O(numMaterials * numVertices * numFaces) para o carregamento da cena a partir de um arquivo OBJ;

a ***O(numPrimitives * primitiveIntersectionComplexity)*** para o rastreamento de raios e a verificação de visibilidade.

Embora a introdução do multithreading pudesse acelerar o rastreamento de raios e a verificação de visibilidade, a complexidade desses processos e a sobrecarga na gestão de threads limitou a eficácia da paralelização. Além disso, é importante notar que a paralelização do carregamento

da cena a partir de um arquivo OBJ seria bastante difícil, devido à natureza sequencial das operações de I/O e ao fato de que a leitura de vértices e faces é dependente.

2 Sampling many lights

Uma das abordagens comuns para calcular a iluminação direta é amostrar múltiplas fontes de luz na cena e calcular a contribuição de cada uma delas.

No entanto, à medida que a complexidade da cena aumenta e o número de fontes de luz se torna maior, a amostragem de todas as luzes pode se tornar computacionalmente custosa e resultar num desempenho insatisfatório. Além disso, em muitos casos, nem todas as luzes têm mesma influência no resultado da iluminação.

Para lidar com essas questões, estratégias mais inteligentes podem ser adotadas para selecionar de forma eficiente as fontes de luz a serem amostradas. Uma dessas estratégias é a amostragem estocástica, onde as fontes de luz são selecionadas com base nas suas probabilidades de contribuição.

Nesse contexto, é importante considerar fatores como a potência da luz, a área que ela cobre e a sua distância relativa ao ponto a ser iluminado. Ao levar estes fatores em consideração, podemos calcular uma probabilidade de seleção para cada fonte de luz, dando maior probabilidade às luzes com maior potencial de contribuição.

Uma abordagem comum para realizar esta seleção é calcular a distribuição de probabilidade cumulativa (CDF) das fontes de luz com base nas suas probabilidades individuais.

2.1 DirectLighting()

Esta secção focar-se-á, na maior parte, na estratégia tomada no método do algoritmo do *shader*, *Path Tracer*, o *directLighting()*.

Algoritmo inicial

Na primeira implementação, a função percorre todas as fontes de luz presentes na cena e calcula a contribuição de cada uma delas. Isto inclui o tratamento de luzes ambiente, luzes pontuais e luzes de área. Embora a abordagem seja simples e direta, ela pode se tornar ineficiente quando a cena contém inúmeras fontes de luz. O processamento de todos os *shadow rays* para todas as

luzes podem se tornar o custo mais significativo do *renderer*, especialmente em cenas complexas.

Algoritmo Estocástico

A segunda implementação busca resolver esse problema, adotando uma abordagem estocástica para selecionar uma única fonte de luz a cada chamada da função *directLighting()*. Nessa nova abordagem, as luzes são selecionadas de forma aleatória, levando em consideração as probabilidades de seleção de cada luz com base na sua potência normalizada. É feito usando uma distribuição uniforme para selecionar uma luz com base na função de distribuição cumulativa (CDF). A contribuição da luz selecionada é então calculada individualmente.

Essa estratégia de amostragem estocástica traz algumas vantagens.

1. Reduz significativamente o número de *shadow rays* que precisam ser disparados, já que apenas uma luz é selecionada em cada iteração. Isso resulta em ganhos de desempenho significativos, especialmente em cenas com muitas fontes de luz.

2. Permite controlar a amostragem de luz de forma mais eficiente, direcionando o cálculo de sombras apenas para a luz selecionada, o que pode levar a uma convergência mais rápida e eficaz da imagem.

Em resumo, a segunda implementação da função *directLighting()* adota uma abordagem estocástica de seleção de luz, permitindo uma amostragem mais eficiente e controlada das fontes de luz. Isto resulta em ganhos de desempenho significativos e uma convergência mais rápida da imagem, embora introduza um nível adicional de ruído. Contudo, com o elevado número de **spp's**, este vai ser removido quase na sua totalidade, também com filtros (ex. Filtro de média).

3 Conversão de Imagem

Nesta etapa do projeto, o grupo pretendia converter uma imagem *PPM* para *JPG*, *PFM* e *OpenEXR*. Decidimos utilizar bibliotecas em *Python* e chamar o programa executavel após a imagem ser gerada em *PPM*.

Para a conversão para *JPG*, utilizamos a biblioteca *Pillow*, onde abrimos o ficheiro *PPM*, convertemos para o espaço de cores *RGB* no caso de ser diferente, e por fim, guardamos a imagem convertida para o espaço de cores mencionado anteriormente e guardamos o ficheiro *JPG*.

Relativamente à conversão para *PFM* ou *OpenEXR*, utilizamos a biblioteca *OpenCV*. Para *PFM*, começamos por abrir o ficheiro *PPM* com a flag *IMREAD_UNCHANGED*, extraímos as dimensões da imagem, convertemos os valores de cores para *float32* e reordenamos os canais de cores de *BGR* para *RGB*, normalizamos os valores para valores entre [0,1], por fim criamos o *header PPM*, criamos um ficheiro *PFM*, onde guardamos esse *header* e o resto da informação da imagem *PPM* e guardamos.

Para *OpenEXR*, começamos por abrir a imagem *PPM* com as flags *IMREAD_ANYCOLOR* e *IMREAD_ANYDEPTH*, convertemos a imagem para o formato *OpenEXR* através da normalização dos dados da imagem para *float32* entre valores [0,1], corrigimos o canal de cores uma vez que o vermelho e o azul estavam trocados e por fim guardamos o ficheiro *OpenEXR*

4 Tone Mapping Reinhardt

O *Tone Mapping* é uma técnica utilizada para converter imagens de alto alcance dinâmico (HDR) num formato adequado para ser exibido em dispositivos de baixo alcance dinâmico (LDR), como ecrãs de computador. Imagens HDR contêm uma ampla faixa de níveis de brilho que excedem as capacidades dos dispositivos LDR convencionais, resultando na perda de detalhes e contraste. Os algoritmos de Tone Mapping visam comprimir a faixa dinâmica das imagens HDR, preservando o máximo de informações visuais possível.

Uma estratégia popular de mapeamento de tom é o algoritmo **Reinhardt**. O algoritmo Reinhardt oferece uma abordagem simples e direta para o mapeamento de tom. Ele funciona dividindo os valores de radiância originais pela soma de 1 e o valor máximo encontrado entre os canais de cor (vermelho, verde e azul). Essa divisão garante que os valores resultantes do mapeamento de tom estejam na faixa de 0,0 a 1,0.

A ideia por trás do mapeamento de tom de Reinhardt é comprimir os valores de radiância alta de forma que eles possam ser exibidos corretamente em dispositivos LDR. No entanto, é importante observar que o algoritmo Reinhardt não leva em consideração as características do sistema visual humano. Como resultado, pode haver perda de detalhes e contraste em cenas com faixas dinâmicas extremamente altas ou condições de iluminação intensa.

4.1 ImagePPM

Para implementar o algoritmo de mapeamento de tom de Reinhardt, foram feitas algumas modificações no código fornecido. No método *ToneMap()*, a implementação original calcula o valor máximo de radiância (max_channel) entre os canais de cor para cada píxel. Em seguida, aplica o algoritmo de Reinhardt a cada canal de cor individualmente, dividindo o valor de radiância original por *(1,0f + max_channel)*.

Vale ressaltar que, embora o algoritmo Reinhardt seja uma abordagem simples e eficaz para o mapeamento de tom, existem técnicas mais avançadas disponíveis que levam em consideração a perceção visual humana e proporcionam uma melhor preservação de detalhes e contraste em imagens HDR.

5 Adicional

5.1 Correção de Gamma

A Correção de Gamma é um processo usado para otimizar a maneira como a luminosidade é interpretada por dispositivos digitais. Este processo é essencial para compensar a perceção não linear da luz pelos seres humanos, melhorando assim a representação geral das cores. A aplicação da Correção de Gamma garante que as cores sejam percebidas conforme pretendido.

5.2 Tone Mapping vs Gamma Correction

O Tone Mapping é uma técnica utilizada quando a gama de cores numa imagem HDR excede a gama que um dispositivo pode exibir. Esta técnica mapeia cores de uma gama de cores mais alta para uma gama de cores menor. O objetivo do Tone Mapping é representar com precisão as cores e os detalhes em condições de iluminação diversificadas, permitindo a visualização adequada em dispositivos com capacidades de exibição de alcance dinâmico mais baixas. Na nossa implementação, aplicamos a Correção de Gamma e o Tone Mapping para garantir a precisão da cor e a representação detalhada das imagens HDR produzidas pelo nosso sistema. Encontramos algumas dificuldades quando aplicamos ambas as técnicas simultaneamente, o que resultou em imagens que pareciam "planas"(*plain*). Por meio de um estudo, percebemos que o uso de ambas as técnicas pode originar uma dupla mapeação da gama de cores, levando à perda de contraste.

Para resolver este problema, optamos por aplicar ou a Correção de Gamma, ou o Tone Mapping, mas não ambos simultaneamente. Esta abordagem resultou em imagens com cores precisas que também se adaptam adequadamente às capacidades de exibição de dispositivos de alcance dinâmico mais baixos.

6 Conclusão

Nesta última fase do projeto de Visualização e Iluminação, foram implementados três temas principais: "**Parallel Multithreading**", "**Sampling many lights**", "**Output JPG / PFM / OpenEXR images**"e "**Tone Mapping Reinhardt**".

No tema "**Parallel Multithreading**", o objetivo era melhorar o desempenho do renderer através da introdução de multithreading. Isso foi feito dividindo a imagem em partes e alocando cada parte para uma thread individual. Embora a introdução do multithreading tenha acelerado o processo de renderização, foi preciso a gestão de threads cuidadosamente analisada para evitar race conditions e sobrecarga excessiva.

No tema "**Sampling many lights**", a estratégia adotada foi a amostragem estocástica das fontes de luz. Em vez de amostrar todas as luzes na cena, apenas algumas são selecionadas com base nas suas probabilidades de contribuição. Isso reduz o número de cálculos necessários e melhora o desempenho do renderer. No entanto, a amostragem estocástica introduz um nível de ruído na imagem, que pode ser gerido ajustando o número de amostras de luz ou aplicando técnicas de redução de ruído durante o pós-processamento.

No tema "**Output JPG / PFM / OpenEXR images**", o objetivo era permitir a conversão das imagens renderizadas para diferentes formatos de arquivo. Para isso, foram utilizadas bibliotecas como Pillow e OpenCV para realizar a conversão para os formatos JPG, PFM e OpenEXR. Embora tenham ocorrido desafios na implementação dessas bibliotecas em C++, foi possível contornar o problema utilizando código Python chamado a partir do código C++.

Por fim, com o "**Tone Mapping Reinhardt**"conseguiu-se implementar o algoritmo Reinhardt, uma abordagem simples e direta para o Tone Mapping, de modo a imagem ser comprimida para se adequar à faixa dinâmica dos dispositivos LDR; se bem que não forneceu os resultados desejados pelo grupo.

Em resumo, este projeto explorou diferentes técnicas para melhorar o desempenho do renderer,

otimizar o processo de amostragem de luz e fornecer opções de saída flexíveis para as imagens renderizadas. Essas melhorias contribuem para a eficiência e qualidade geral do renderer, possibilitando a geração de imagens visualmente impressionantes.

7 Anexos

7.1 Seis Area Lights (com 8 samples cada)

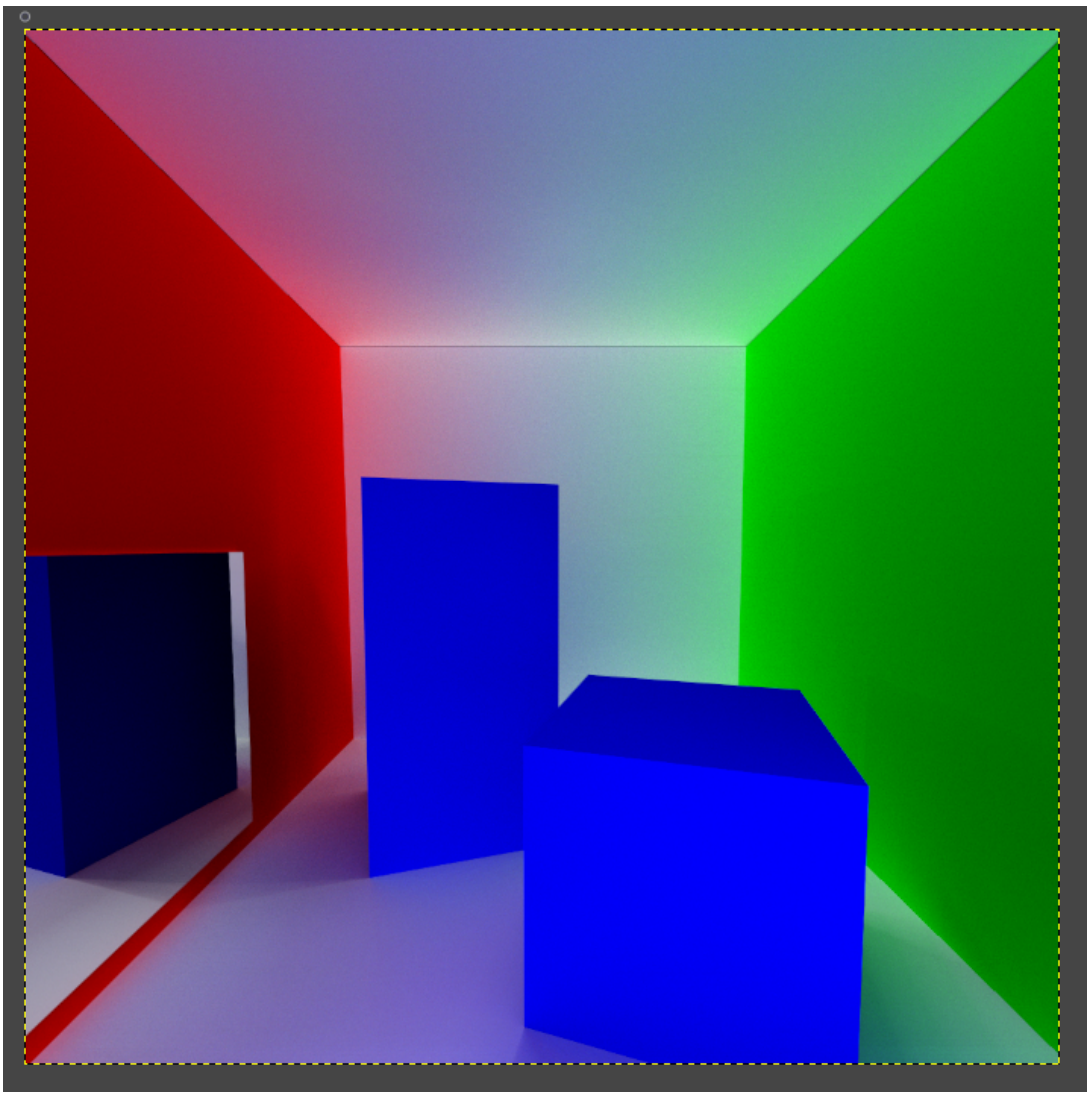

Figura 1: 1024 spp, 6 area lights, 8 samples cada, 10871 segundo.

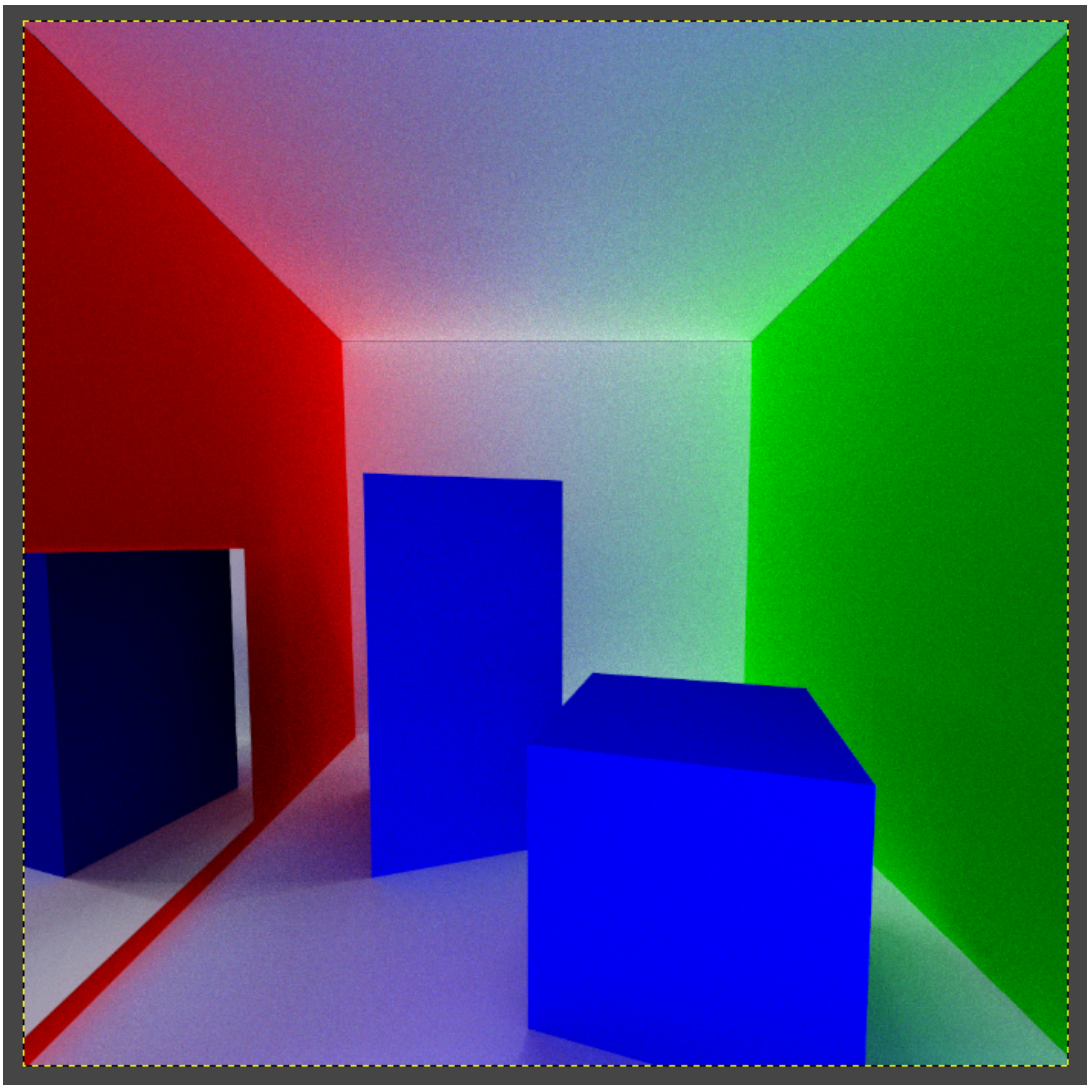

Figura 2: 128 spp, 6 area lights, 8 samples cada, 1359 segundos (antes de melhorar o multi sample).

7.2 Três Area Lights e Ambient Lighting

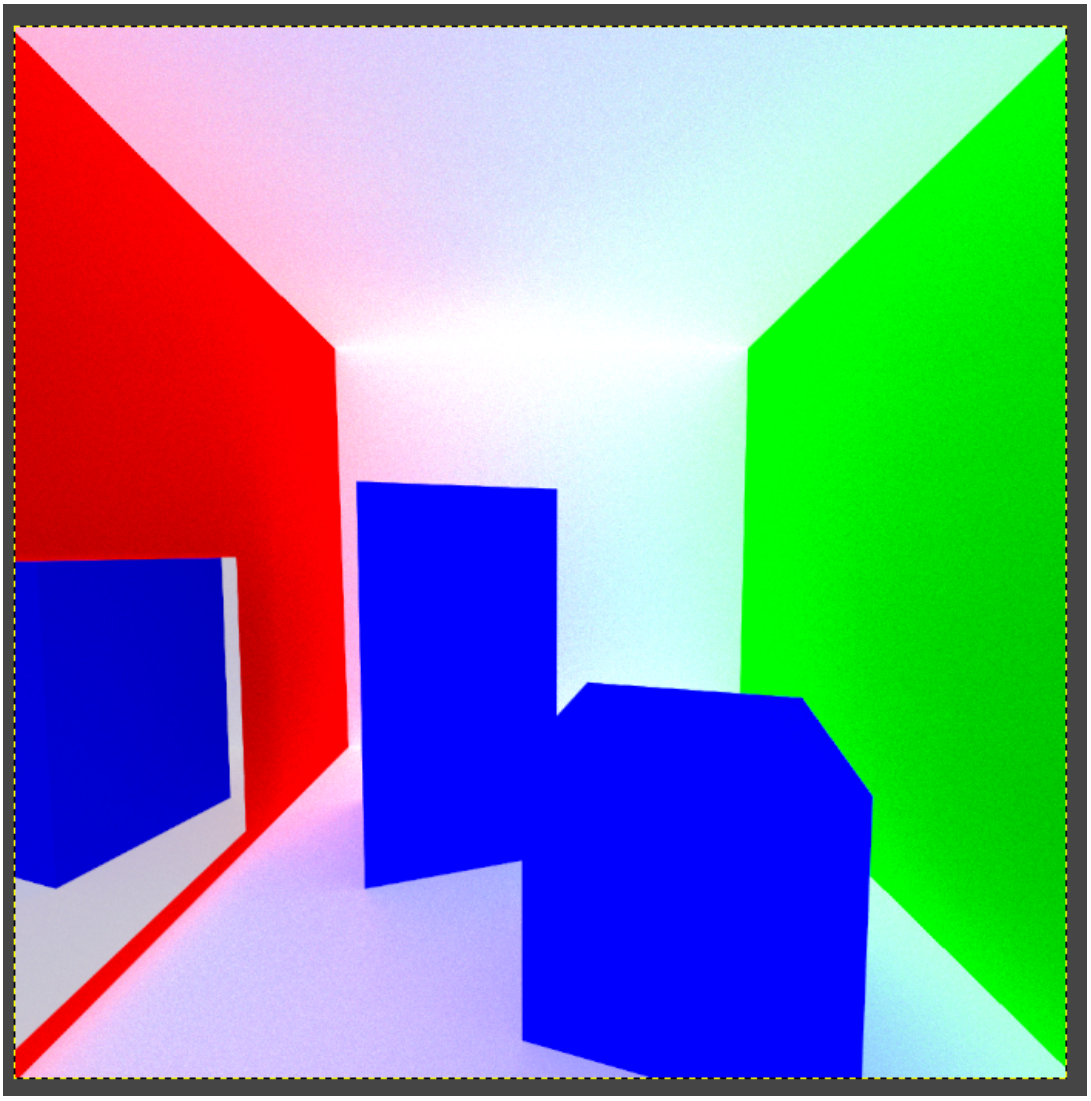

Figura 3: 128 spp, 3 area light, 8 samples cada, ambient light, semi-advanced tone maping + gamma correction, 436 segundos.

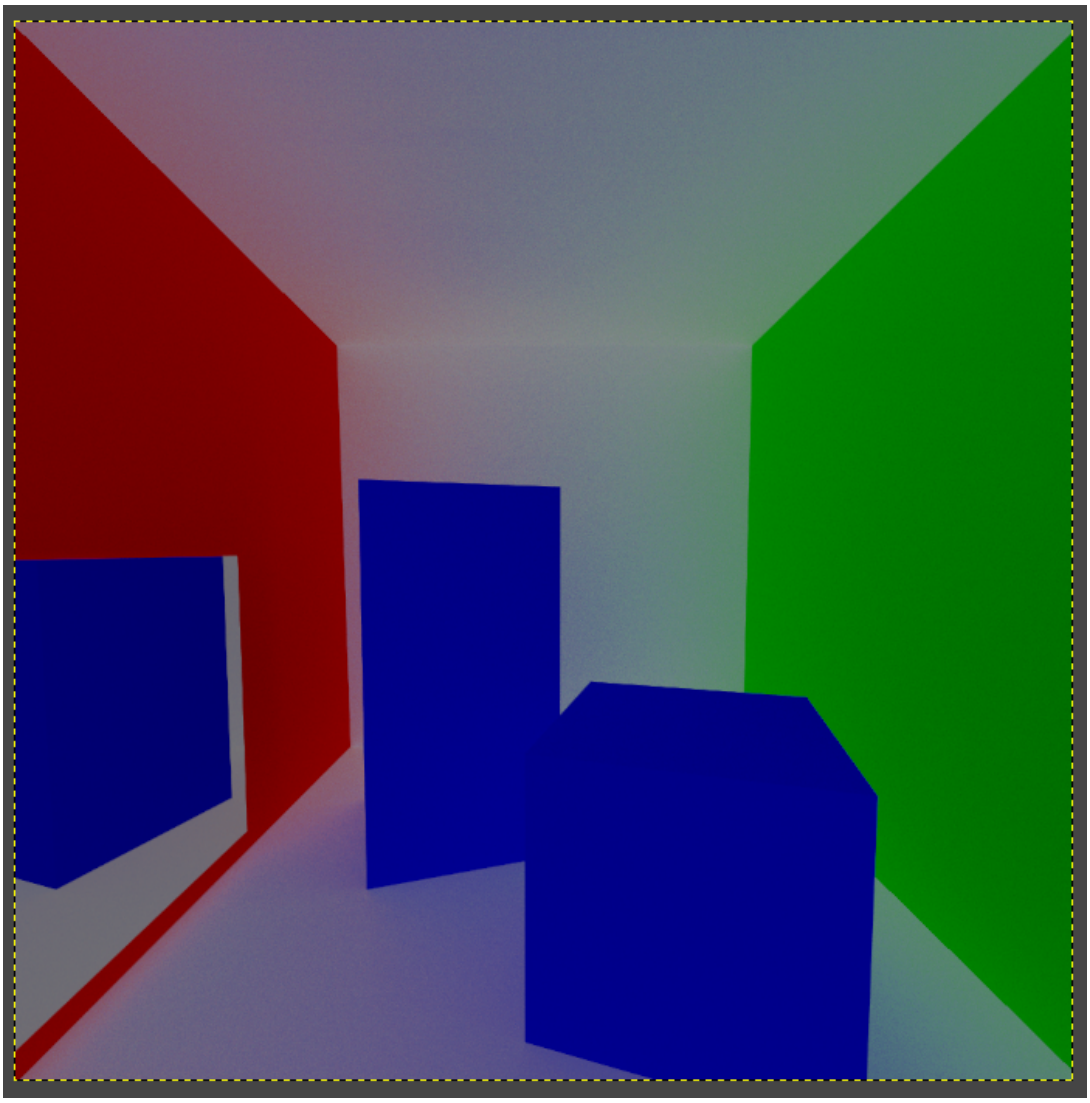

Figura 4: 128 spp, 3 area lights, 8 samples cada, ambient light, advanced tone maping + gamma correction, 436 segundos.