PHYSICALLY BASED RENDERING

Trabalho final da disciplina tópicos especiais em computação

Aluno: João Hudson de Lacerda Oliveira

Matrícula: 20170153752

Nesse trabalho foram desenvolvidos os requisitos básicos exigidos na disciplina, como: material difuso, material condutor polido, carga de malhas, BVH e multi-threading(OMP). Também foram desenvolvidas 2 features extras: gamma compression e material dielectric.

A cena renderizada conta com 2 modelos importados de um arquivo (.obj), uma arma e um sofá, as demais estruturas são primitivas do software em questão.

O trabalho foi desenvolvido com um processador cellerom dual-core, porém a imagem final foi gerada em outro computador, com processador ryzen hexa-core. Sendo assim o tópico “Acelerações” se refere ao cellerom, já os resultados finais ao ryzen.

Carga de malhas:

O carregador de malhas utilizado foi implementado do zero, por motivo de alguns problemas técnicos que impossibilitaram a instalação de bibliotecas para este fim.

O carregador implementado suporta apenas o formato (.obj) e carrega informações de: malhas, vértices e normais. A carga dos modelos armazena as informações em um map<string, mesh>, dessa forma é possível acessar cada malha individual por seu respectivo nome(essa é a última versão do carregador, versões anteriores eram mais simples). Cada malha carregada possui um vetor de vértices e um material, bem como uma BVH.

O material de cada malha é definido fornecendo ao modelo o um ponteiro para o material e o nome da malha(existe um material padrão definido no modelo para malhas sem materiais).

Como a carga fornece as normais do modelo, o triângulo gera sua normal interpolando as 3 normais que ele recebe fazendo combinação linear com elas.

Obs.: O cálculo de intersecção raio-triângulo foi atualizado para o do Moller, por ser mais rápido e por fornecer as coordenadas barricêntricas.

Acelerações:

Para acelerar o processo de rendering naturalmente lento(em particular para geometrias complexas), foi utilizado paralelização com OMP, cujo a única dificuldade no processo foi a geração de números randômicos de forma thread-safe, que para tal era criado em cada iteração do loop mais externo seu gerador de números randômicos com uma seed baseada no clock, e um único objeto estático para distribuição uniforme. Como resultado obtive quase o dobro da performance(que era de se esperar de um processador dual core).

Além de concorrência para acelerar foi feita uma BVH **por malha,** a ideia era que normalmente os triângulos são bem organizados em malhas, de forma que uma divisão inicial por malha poderia ser efetiva na grande maioria das cenas. O critério de divisão da BVH foi o valor da componente x do centroide de cada triângulo, o critério em si não foi dos melhores, sendo como previsto a construção da BVH ser ineficaz em algumas divisões(normalmente nas últimas), e de tal forma poderia deixar até um pouco mais lento se as divisões fossem aplicadas até termos 1 triângulo por folha. Para contornar foi deixado arbitrário a profundidade da árvore com um intervalo do vetor de triângulos em cada folha(assim cada folha possuía seus triângulos e sua AABB), que nesta cena, foi 5 a melhor profundidade. Resultado? Um ganho bem maior que a paralelização, estimo que tenha minimizado o tempo para 30% do tempo sem a BVH.

Materiais:

Evitei usar OOP na codificação dos matérias, tanto pelo overhead quanto pelas simplificações que foi possível fazer sem a modularização deles. No código é possível ver que os materiais são: LIGHT, DIFFUSE, SPECULAR, DIELECTRIC.

Esse material LIGHT serviu apenas para fazer uma pequena otimização, ao perceber que as luzes da cena eram materiais difusos normalmente com refletância (0,0,0), notei que o cálculo recursivo neste caso era redundante, pois tudo ia resultar em (0,0,0) devido a sua refletância ser esta, sendo assim o material LIGHT não faz chamadas recursivas em seu cálculo de radiância, ele se resume apenas em retornar sua emissão.

Também foram feitas simplificações no material difuso(cortar os valores PI), e o reuso de funções responsáveis por gerar vetores direção dos raios. Alguns problemas apareceram com relação a alguns épsilons do programa que tive que ajustar, outros com relação ao gerenciamento do registro de intersecção, mas as maiores dificuldades foram encontradas na implementação do material dielectric.

Por fim no projeto, foi usado gamma compression para corrigir o brilho final da cor na classe Buffer, elevando cada valor a 1/(2.2).

Informações finais:

Tempo: 22 horas Amostras: 15000

Resolução: 1024x720 Profundidade: 7