DOCUMENTAÇÃO: PROTÓTIPO DE RENDERIZADOR VIA RAY TRACING

INSTRUÇÕES DE COMPILAÇÃO

O projeto é compilado através da ferramenta CMake. O procedimento é padrão: configurar o ambiente através do comando “cmake . ./build” na pasta raiz do projeto e então rodar o compilador manualmente ou através do comando “cmake --build ./build”. Há também a possibilidade de usar a interface gráfica do CMake. De toda forma, existem diversos materiais disponíveis online sobre o uso dessa ferramenta. A etapa de configuração apenas precisa ser realizada uma vez, o projeto pode ser modificado e recompilado sem repetí-la.

Antes de compilar o projeto é necessário compilar a biblioteca yaml-cpp, uma dependência. As instruções estão na fonte da biblioteca. Após compilá-la, será necessário colocar o arquivo .a resultante na pasta lib e o arquivo .dll na pasta bin.

INSTRUÇÕES DE EXECUÇÃO

O executável do projeto deve ser aberto a partir da pasta raiz, isto é, deve-se usar o comando ./bin/raytracer.exe.

Os objetos a serem renderizados são criados a partir das especificações no arquivo de cena carregado. Por padrão o programa carrega o arquivo scenes/scene1.yaml. O programa aceita um número arbitrário de objetos e cada um deles pode ser configurado com os seguintes parâmetros:

- Primitive: configurações geométricas.
    - type: tipo de primitiva, pode ser “plane” ou “sphere”.
    - position: posição global do centro da primitiva.
    - normal: apenas para planos, o vetor normal que o define
    - radius: apenas para esferas, o raio da esfera.
- Material:
    - color: cor para o cálculo de difusão.
    - emissive: cor emitida pelo material.
- Light: “true” ou “false”, sinaliza para o renderizador se este objeto deve ser selecionado como prioridade para amostragem de luz.

UM POUCO DE TEORIA

Visto que é impossível entender o funcionamento do código sem alguns conceitos, e que é impossível explicar o funcionamento do código sem alguns termos, farei um breve resumo da teoria por trás do projeto, entretanto, explicações muito melhores e mais detalhadas podem ser encontradas nos materiais de referência, cuja consulta é de suma importancia para qualquer um que deseje trabalhar com este tipo de software.

O ray tracing é uma técnica de produção de imagens por computador que visa alta fidelidade dos efeitos visuais, e que para isso deriva seus algoritmos do entendimento físico do comportamento da luz no mundo real, o que envolve sempre

a óptica geométrica e por vezes eletromagnetismo, ondulatória, dentre outras ciências. Alguns efeitos simulados pelo ray tracing com excelência são: reflexão, refração, cáusticas, sombras e iluminação global. Entretanto, a complexidade computacional desta técnica é imensamente maior que a de outros algoritmos, o que inviabiliza sua utilização em muitos casos, principalmente quando envolvem aplicações em tempo real.

Em essência, o trabalho de um ray tracer (um software que utiliza a técnica de ray tracing) é traçar raios (reta no espaço que representa o caminho de um raio de luz) partindo de um ponto de detecção em direção a uma cena e determinar qual cor de luz eles trazem até esse ponto, o que geralmente envolve traçar novos raios partindo dos pontos de interseção com a cena. A cor e a intensidade da luz deixando uma superfície em uma direção é determinada pela cor e intensidade da luz que chega nesse ponto vinda de todas as direções e por uma função do material dessa superfície que determina quanto de luz é transmitida de uma direção a outra no hemisfério. Essa última função é chamada de BRDF (bidirectional reflectance distribution function), já esse processo é modelado por uma integral conhecida como Equação de Renderização.

A geração de uma imagem por ray tracing consiste então em computar o valor dessa equação para todos os pixels da tela. Contudo, como essa equação é naturalmente recursiva, e como o cálculo numérico de integrais envolve realizar muitas amostras, a complexidade computacional pode facilmente superar a capacidade do hardware antes que tenhamos obtido resultados satisfatórios.

Felizmente, há diversas maneiras de contornar esse problema, isto é, de conseguir melhores resultados com menos esforço computacional. As técnicas que permitem fazer isso vem, na maior parte, da estatística.

Começando pelo método de integração, a abordagem que melhor se adequa ao caso é a Integração de Monte Carlo com amostragem por importância. Essa técnica, fundamentada na Lei dos Grandes Números estima o valor de uma integral a partir de uma média de amostras do integrando obtidas segundo alguma distribuição. O desvio padrão do resultado da estimativa será tão menor quanto mais a PDF (função de distribuição de probabilidade) com que se escolhem amostras for parecida com integrando. Aqui entra a amostragem por importância: queremos obter uma distribuição que se pareça com o integrando para resultados melhores, contudo, os integrandos quase sempre são compostos de várias funções bastante complexas e descontínuas, de modo que podemos modelar a PDF apenas de acordo com um aspecto da equação, como o material ou a direção das luzes. A escolha de uma dessas técnicas faz com que certas situações tenham excelentes resultados e outras tenham resultados ruins, justamente por selecionar um aspecto da equação e focar apenas nele. A solução para esse problema é a Amostragem por Importância Múltipla.

Se uma certa técnica de amostragem favorece certos aspectos e negligencia outros enquanto outra técnica favorece estes últimos em detrimento dos primeiros, não é estranho supor que a melhor solução seria uma combinação dessas técnicas. A maneira como isso é feito chama-se Amostragem por Importância Múltipla e

consiste em escolher aleatoriamente uma dentre n técnicas de amostragem e pesar os seus resultados de acordo com a distribuição usada na escolha.

Há ainda muitos outros conceitos importantes que não abordarei aqui por não terem sido usados no projeto mas que valem ser mencionados e poderiam ser matéria de implementação futura como:

- BVH (Bounding Volume Hierarchy), uma estrutura de dados para organizar a cena e permitir cálculo de colisões mais veloz;
- Roleta Russa, uma técnica para traçar caminhos de profundidade variável que permite um considerável ganho de resultados com baixo esforço de computação adicional;
- outros materiais, diferentes materiais têm diferentes interações com a luz, alguns refletem o que vem de uma certa direção, alguns permitem que parte da luz seja transmitida através de si, um bom modelo de materiais suportaria esses efeitos e ainda outros;
- tratamento espectral de cor, diferente do modelo RGB usado no projeto, uma abordagem mais coerente com a física é representar as cores como uma distribuição de energia no espectro eletromagnético visível.

EXPLICAÇÃO DO CÓDIGO

O código está dividido em três partes separadas nas pastas math, objects e renderer; cada uma delas tem uma versão dentro de ./include com os cabeçalhos e outra dentro de ./src com as implementações.

A pasta math contém implementações de entidades matemáticas usadas no projeto: vetores tridimensionais, cores, raios (um vetor mais um ponto de origem, uma reta) e um gerador de números aleatórios.

A pasta objects contém implementações das entidades que serão instanciadas para representar a cena a ser renderizada, bem como os meios para fazer essa instanciação a partir dos arquivos .yaml.

A pasta renderer contém o código responsável por efetivamente produzir uma imagem a partir da cena como representada pelos conteúdos do módulo anteriormente mencionado.

Objects

A cena a ser renderizada é representada por uma classe Scene que contém um conjunto de objetos Object e uma interface para acessá-los. Os objetos são formados por uma primitiva matemática (classe Primitive), que pode ser um plano (classe Plane) ou uma esfera (classe Sphere), e um material (classe Material), que tem parâmetros para cor difusa e emissão apenas. Para facilitar a criação desses objetos compostos existe a classe ObjectBuilder, e para carregar as cenas a partir dos arquivos .yaml existe a classe SceneManager.

Lista de classes

Primitive: classe base para primitivas matemáticas, exige a implementação de um teste de colisão com raios.
Sphere: primitiva, uma esfera.
Plane: primitiva, um plano.
Material: apenas uma cor de difusão e uma cor de emissão.
Object: une uma primitiva e um material.
ObjectBuilder: auxilia na criação de objetos.
Scene: uma lista de objetos.
SceneManager: auxilia na construção de uma cena.

Renderer

Aqui temos três arquivos, a câmera que define os parâmetros geométricos do observador para a criação dos raios; a stb_image_write, uma biblioteca contida inteiramente em um cabeçalho que é usada para escrever imagens em arquivos png; e o renderizador em si. Neste último está contido todo o funcionamento do algoritmo e é o que exige uma explicação mais detalhada.

Quando a função renderToImage é chamada para uma dada cena e uma dada câmera, será feito um número de amostras para cada pixel como definido pela constante SAMPLES. Para cada uma dessas amostras será traçado um raio do foco da câmera em direção ao pixel em questão, esse raio é então processado pela função processRay, que, por sua vez, caso haja intercessão desse raio com algum objeto, pode produzir um novo raio e chamar-se recursivamente até o limite definido pela constante MAX_RAY_DEPTH. É feita então a média dos resultados das amostras para aproximar o que seria a integral da luz chegando naquele ponto, conforme o processo de integração de Monte-Carlo. Esta é a renderização em linhas gerais, mas o processo pelo qual novos raios são criados em caso de colisões precisa de mais atenção.

O programa suporta apenas materiais perfeitamente difusos, isto é, de brdf constante por todo o hemisfério e faz o balanceamento da amostragem desse material (levando em conta o cosseno do angulo da luz com a superfície) e da amostragem da luz. Quando uma intercessão é detectada e um novo raio precisa ser produzido, cada técnica de amostragem tem 50% de chance de ser selecionada. A técnica de amostragem de material produz um raio em uma distribuição cujo pico de probabilidade acontece na normal da superfície e diminui conforme o angulo com ela aumenta. A técnica de amostragem de luz escolhe entre uma das luzes da cena e lança um raio em direção a algum ponto nessa luz. O resultado desse novo raio, determinado recursivamente da mesma forma, é multiplicado pelos outros termos do integrando (o jacobiano da transformação para coordenadas esféricas, o cosseno com a normal, a brdf e o inverso da pdf) e retornado.

Lista de Classes

Camera: define parâmetros que determinam como os raios serão lançados sobre a cena.
Renderer: efetivamente cria uma imagem a partir dos objetos de uma cena.

REFERÊNCIAS

Bibliotecas utilizadas:
https://github.com/nothings/stb

https://github.com/jbeder/yaml-cpp

Teoria:
https://www.realtimerendering.com/raytracing.html#books

https://www.youtube.com/watch?v=5sY_hoh_IDc&list=PLmIqTlJ6KsE2yXzeq02hqCDpOdtj6n6A9&ab_channel=ComputerGraphicsatTUWien

https://graphics.stanford.edu/papers/veach_thesis/thesis.pdf