Paralelização do compilador da linguagem de programação Rust

Yuri Kunde Schlesner

6 de agosto de 2014

1 Identificação

Resumo: TODO.

Período de execução: Setembro de 2014 a Dezembro de 2014

Unidades participantes:
Curso de Ciência da Computação
Departamento de Eletrônica e Computação

Área de conhecimento: Ciência da Computação

Linha de Pesquisa: Computação Gráfica, Linguagens de Programação

Tipo de projeto: Trabalho de Conclusão de Curso

Participantes:
Profª Andrea Schwertner Charão – Orientadora
Yuri Kunde Schlesner – Orientando

2 Introdução

A *Computação Gráfica* é a area da Ciencia da Computação que estuda tópicos relacionados a criação, análise e manipulação de imagens e conceitos relacionados. Dentre estas, a síntese (ou renderização) de imagens é onde uma imagem é criada de forma computacional, a partir de um modelo matemático e frequentemente buscando o fotorealismo. Tem uma vasta quantidade de aplicações práticas: É usada na engenharia, durante o

projeto de máquinas ou construções; na arquitetura para a visualização de espaços antes que sejam construídos; para entretenimento, em efeitos especiais de filmes ou em jogos 3D e em muitas outras aplicações.

Como a geração de imagens fotorealistas involve essencialmente uma simulação completa da física da luz, um processo proibitivamente lento e complexo, são utilizadas simplificações e modelos. No passado, devido a limitada capacidade computacional disponível, eram utilizadas aproximações grosseiras que, enquanto produziam imagens atrativas, não eram muito realísticas, especialmente no quesito da aparência das superfícies de suas interações com a luz. Com o aumento do poder computacional disponível, vem sendo usados modelos cada vez mais fiéis a realidade e que produzem imagens cada vez mais convincentes, algumas vezes indistinguíveis de uma fotografia real.

Path tracing é um método de renderização que assume que a luz se comporta como uma partícula e calcula uma imagem traçando uma série de raios pelos caminhos através quais a luz viajaria quando refletida através de uma cena. Atualmente é um dos algorítmos mais usados quando são demandadas imagens com um grau de realismo extremamente alto, devido a sua abilidade de simular o comportamento da luz com relativa precisão.

No entanto, este realismo vem ao custo de muito poder de processamento, e mesmo com o avanço tecnológico de CPUs, a renderização de imagens continua sendo uma das tarefas mais árduas para processadores. Sistemas de renderização profissionais são quase exclusivamente escritos em C++, e não em linguagens de mais alto nível, devído as penalidades de performance que impõem, e sistemas mais recentes chegam a fazer o uso de GPUs para acelerar a imensa quantidade de cálculos necessária. Tendo em vista a baixa expressividade de C++comparada a estas outras linguagens, torna-se interessante explorar alternativas que permitam mais fácil desenvolvimento, mas sem sacrificar a performance requerida.

A linguagem de programação *Rust*, um projeto de pesquisa da *Mozilla Research*, tem como seu objetivo ser uma união entre linguagens de programação de sistemas e as tidas como "linguagens de alto-nível", focando simultâneamente em alta-performance, segurança e expressividade. Ela atinge isso usando um modelo tradicional de compilação prévia (*ahead of time*) e um sistema de tipos que permite a verificação automática dos usos de ponteiros durante a compilação, eliminando a possibilidade de acontecerem erros de memória, mas sem introduzir penalidades excessivas de performance ou consumo de memória, ao mesmo tempo que integra conceitos mais recentes de linguagens de programação que aumentam sua expressividade e capacidade de facilmente descrever programas complexos.

3 Objetivos

3.1 Objetivo Geral

O objetivo geral deste trabalho é escrever um *path tracer* simples em Rust que sirva de um exemplo educativo, juntamente com a fundamentação matemática que será descrita no relatório, de como implementar este tipo de algorítmo. Além disto, será uma oportunidade de fazer uma comparação qualitativa e quantitativa entre esta lingaugem e C++, nos aspectos de performance e organização de código.

3.2 Objetivos Específicos

- Re-escrever um renderizador (atualmente escrito em C++) utilizando a linguagem Rust.
- Realizar uma comparação de performance e clareza de código entre as duas versões.
- Implementar melhoras estruturais e algorítmicas na nova versão.
- Documentar a fundamentação teórica dos algorítmos utilizados.
- Analisar como as funcionalidades de Rust ajudam ou não a escrever um programa deste tipo.

4 Justificativa

Rust é uma linguagem relativamente nova e, embora aplicações gráficas de alta performance sejam um dos seus públicos alvo, ainda não existe uma quantidade significativa de programas deste tipo que valide a lingaugem para este propósito. A experiência e resultados adquiridos durante a realização deste trabalho podem ajudar a guiar o desenvolvimento da linguagem para atingir este fim.

Além disto, embora existam uma variadade de renderizadores pequenos que tem como objetivo servir de exemplo para outras implementações, eles no geral não possuem uma documentação adequada que explique os princípios teóricos e matemáticos por trás deles. Espera-se que este trabalho preencha este nicho, afim de ajudar futuros pesquisadores ou estudantes em busca deste conhecimento.

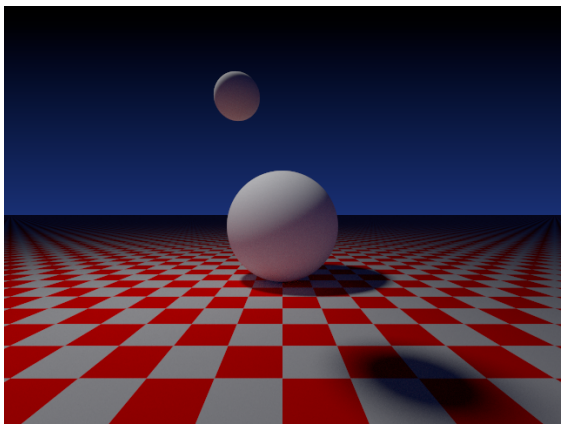

Figura 1: Um exemplo de uma imagem gerada utilizando *path tracing*. Note como a luz que atinge o plano xadrez é refletida devolta para iluminar a esfera, um fenômeno conhecido como *iluminação indireta* e que é corretamente simulado pelo algoritmo. (referenciar)

5 Revisão de Literatura

TODO.

6 Metodologia

Como tem como objetivo principal produzir uma síntese do conhecimento da área esta pesquisa tem um objetivo exploratório(descritivo?), e como irá produzir um trabalho que, além de guiar esta exploração, exemplificará a aplicação deste conhecimento, é também aplicada.

7 Plano de Atividades e Cronograma

1. **Re-escrever o renderizador utilizando a linguagem Rust**: O renderizador será baseado em código pré-existente do autor, escrito em C++, que será utilizado como uma base funcional para a implementação em Rust.

2. **Realizar uma comparação entre as duas versões**: As duas versões do renderizador, a nova feita em Rust e a antiga em C++, serão comparadas nos quesitos de performance e de organização do código, afim de identificar avanços ou regressos, e se esses podem ser atribuídos a características de cada linguagem.

3. **Implementar melhoras estruturais e algorítmicas**: A partir daqui, a nova versão do renderizador será aprimorada com algoritmos mais avançados, que permitam uma melhora de performance ou de qualidade de imagem. A maior parte do trabalho aqui será a revisão da bibliografia existente para identificar algorítmos interessantes e entender seu funcionamento.
4. **Documentar a fundamentação teórica dos algorítmos**: Com a experiência adquirida durante a implementação, as observações e conclusões do autor serão sintetizadas com o objetivo de tornar claro o funcionamento destes algorítmos e como eles interagem.
5. **Analisar como as funcionalidades de Rust ajudam ou não a escrever um programa desse tipo**: Continuando o trabalho da etapa 2, serão ressaltadas quaisquers constatações feitas durante o desenvolvimento sobre as vantagens ou desvantagens de se utilizar a Rust para este tipo de trabalho.

Espera-se que o desenvolvimento das atividades siga o seguinte cronograma:

Etapa	Setembro	Outubro	Novembro	Dezembro
1	✓			
2	✓			
3		✓	✓	
4	✓	✓	✓	✓
5				✓

Tabela 1: Cronograma de Atividades

8 Recursos

Para a realização deste trabalho será utilizado apenas equipamento pessoal do pesquisador, visto que não é necessário o uso de qualquer equipamento especial além de um computador para desenvolvimento.

9 Resultados Esperados

Ao término deste trabalho, espera-se ter uma implementação de um renderizador, escrito em Rust, que seja capaz de produzir imagens fotorealísticas através da implementação de uma variedade de técnicas relacionadas a *path tracing*. Além disso, espera-se que o relatório do trabalho sirva como uma síntese da teoria e funcionamento dos algorítmos implementados, de forma a guiar implementações futuras.

10 Referências

TODO.