Exploração da Linguagem Rust para o Desenvolvimento de um *Path Tracer* Paralelo

Yuri Kunde Schlesner
Orientador: Profª Drª Andrea Scwertner Charão

Ciência da Computação
Universidade Federal de Santa Maria

20/10/2014

informáticaufsm
Ciência da Computação

Outline

informáticaufsm
Ciência da Computação

Introdução

- Ferramenta utilizada: Rust
  - Princípios e público-alvo
- Área de aplicação: Path Tracing
  - Requisitos de processamento

Outline

informáticaufsm
Ciência da Computação

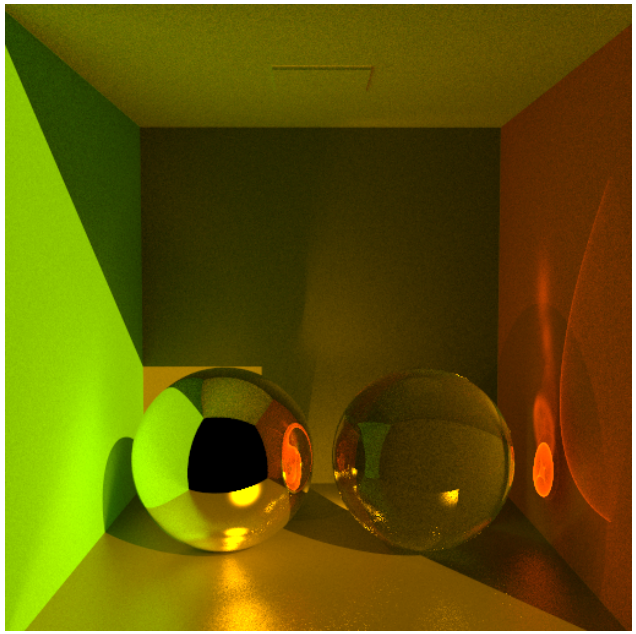
Objetivos

- Portar um renderizador C++ para Rust.

[1] Contagem atualizada, mais precisa que a do texto.

Objetivos

- Portar um renderizador C++ para Rust.
- SmallVCM
  - Pequeno: $\sim$3200[1] linhas de código.
  - Paralelização utilizando OpenMP.
  - Vários algoritmos permitem modularizar o trabalho.



[1] Contagem atualizada, mais precisa que a do texto.

Outline

informáticaufsm
Ciência da Computação

Justificativa

- Rust quer competir com C++
- Poucos estudos e aplicações gráficas no momento
- Validação da linguagem como alternativa viável

Outline

informáticaufsm
Ciência da Computação

Rust

Principais diferenciais da linguagem:

- Compilada, gerando código nativo que não requer *runtime*
- Sistema de Regiões/*Lifetimes*
  - Gerenciamento de memória sem *garbage collection*
  - Prevenção de acessos à memória inválida
- Prevenção de *aliasing*
  - Redução de comportamentos surpreendentes
  - *Thread safety*

Rust

Principais diferenciais da linguagem:

- *Traits*.
- Tipos de dados algébricos e *pattern matching*.
- `unsafe`: controle quando necessário

Lifetimes

- Controle da duração de validade de variáveis pelo compilador.
- Usos da variável depois de sua validade não são permitidas.
- Referências tem validade herdada como parte de seu tipo.
- Permite receber e retornar referências em funções sem comprometer estas garantias.

```
1 fn get_first<T, 'a>(v: &'a Vec<T>) -> &'a T {
2     return v[0];
3 }
```

informáticaufsm
Ciência da Computação

Prevenção de *aliasing*

- Dois tipos de referências: `&T` e `&mut T`.
- Referências compartilhadas não podem ser usadas para modificar objetos.
- Referencias mutáveis não podem ser compartilhadas.
- Previne que escritas em uma referência sejam visíveis através de outras.
- Automaticamente previne condições de corrida em código *multi-threaded*.

Traits

- Similares à *interfaces* ou *type classes*.
  - Trait contém métodos, mas não campos.
  - Tipos que implementam um trait podem ser tratados polimórficamente (durante compilação e execução.)
- Implementação do trait para o tipo é separado da definição do tipo.
  - Novos tipos podem implementar traits existentes.
  - Novos traits podem vir com implementações para tipos existentes.
  - Métodos com mesmo nome em traits diferentes podem coexistir no mesmo tipo.

unsafe

- Permite burlar as regras do sistema de tipos.
- Para implementação de código de baixo-nível ou estruturas de dados.
- Não altera semântica existente, apenas permite mais tipos de operações.
- Blocos de código facilmente identificáveis e auditáveis.

Outline

informáticaufsm
Ciência da Computação

Visão Geral

- O SmallVCM é dividido em uma parte comum e em algoritmos de renderização específicos que utilizam esta infraestrutura.
- A *port* mantém a mesma estrutura de arquivos, tipos e funções, na medida do possível, afim de facilitar comparações.
- Alguns módulos precisaram de uma estratégia diferente para adaptá-los as funcionalidades e limitações de Rust.

Herança

- Rust não tem herança de *structs*.
- Polimorfismo ainda pode ser realizado utilizando *traits*, mas estes não contém dados, apenas métodos.
- Requer divisão de classes base em partes, se estas também conterem campos de dados compartilhados.

Herança

- Solução: separar dados de métodos.
- Criar estrutura para conter os dados da classe base e disponibilizá-la através do *trait*.
- Mais verboso e menos flexível.
- Na prática, em arquiteturas feitas do zero, geralmente se evita este problema.

Inicialização de Structs

- Diferentemente de C++, não são permitidos valores não inicializados ou ponteiros nulo
- Alguns construtores unidos com métodos de inicialização.
- Maioria dos ponteiros pôde ser transformado em referências não-nulas.
- Em alguns fez-se uso de `Option<T>`, removendo nulos implícitos.

Sobrecarga de operadores

- Sobrecarga de operadores é feita implementando-se *traits* especiais.
- `Add`, `Sub`, `Index`, etc.
- Bastante verboso comparado com a sintaxe de C++:

```
1  friend Vec2x<T> operator+(const Vec2x& a, const Vec2x& b) {
2    // ...
3  }
4  // ... x8 operadores, x2 tipos de vetor
```

```
1  impl<T: Num> Add<Vec2x<T>, Vec2x<T>> for Vec2x<T> {
2    #[inline]
3    fn add(&self, o: &Vec2x<T>) -> Vec2x<T> {
4      // ...
5    }
6  }
7  // ... x8 operadores, x2 tipos de vetor
```

- Pode-se usar macros para diminuir a repetição.

Enums

- Enums não possuem valores inteiros (públicos) associados a elas.
- Teste precisa ser feito utilizando *pattern matching*.

```
1  fn get_name(self) -> &'static str {
2    match self {
3      EyeLight => "eye light",
4      PathTracing => "path tracing",
5      LightTracing => "light tracing",
6      ProgressivePhotonMapping => "progressive photon mapping",
7      BidirectionalPhotonMapping => "bidirectional photon mapping",
8      BidirectionalPathTracing => "bidirectional path tracing",
9      VertexConnectionMerging => "vertex connection and merging",
10   }
11 }
```

informáticaufsm
Ciência da Computação

Instabilidade da Linguagem

- Em ativo desenvolvimento. Ainda não existem versões estáveis.
- Vários *commits* diariamente. Semântica da linguagem era alterada quase que semanalmente.
- Necessitou de constantes re-ajustes ao código para se atualizar.
- Versão estável 1.0 prevista para fim do ano ou início de 2015.

Outline

informáticaufsm
Ciência da Computação

Resultados

- Foi portado infraestrutura e um algoritmo do SmallVCM, representando aproximadamente metade do código original.
- Renderizador portado `EyeLight` produz resultados fiéis ao original.
- Paralelizado utilizando a biblioteca “rayon”, que oferece uma interface simples para computações *multi-threaded fork/join*.
- Infelizmente, outros algoritmos não foram portados.

Desempenho

- Versão em Rust obteve resultados acima do esperado.
- Testes foram feitos com gcc e clang, que assim como o rustc também utiliza LLVM.
- Cerca de 7% mais rápido que o clang, utilizando mesmos algoritmos e arquitetura de código similar.

Iterações	*Threads*	rustc	clang	gcc
100	1	7.665 s (85.45%)	8.253 s (92%)	8.970 s (100%)
400	4	8.802 s (84.59%)	—[2]	10.405 s (100%)

[2] clang não possui suporte a OpenMP.

Outline

informáticaufsm
Ciência da Computação

Conclusão

- Objetivo de portar parte do SmallVCM, mantendo estrutura similar, para a linguagem Rust.
- Simplicidade do código foi afetada negativamente pelo objetivo de manter a mesma estrutura, devido a diferenças nas linguagens.
- *Port* de um algoritmo com sucesso, obtendo resultados corretos.
- Aumento de desempenho quando comparado ao código original, mesmo sem realização de otimizações específicas.

Trabalhos Futuros

Possíveis continuações para o trabalho:

- Portar mais algoritmos, a fim de verificar se a vantagem de desempenho mantém-se neles.
- Realizar uma nova *port* ou re-estruturar a *port* atual utilizando uma estrutura mais idiomática e adequada a Rust e observar ganhos ou perdas em clareza e performance.

Exploração da Linguagem Rust para o Desenvolvimento de um *Path Tracer* Paralelo

Yuri Kunde Schlesner
Orientador: Profª Drª Andrea Scwertner Charão

Ciência da Computação
Universidade Federal de Santa Maria

20/10/2014

informáticaufsm
Ciência da Computação