Feasibility Study of Real Time Path Tracing

Or: How Much Noise Is Too Much?

Sven-Hendrik Haase
Matriculation number: 6341873

Department of Computer Science
University of Hamburg

February 4, 2016

Table of Contents

Motivation

- Current generation graphics made up of many complex tricks

Motivation

- Current generation graphics made up of many complex tricks
- Path tracing is simple in comparison

Motivation

- Current generation graphics made up of many complex tricks
- Path tracing is simple in comparison
- Superior graphics quality

Motivation

- Current generation graphics made up of many complex tricks
- Path tracing is simple in comparison
- Superior graphics quality
- Allows for simulation of caustics, global illumination, light dispersion, etc.

Motivation

- Current generation graphics made up of many complex tricks
- Path tracing is simple in comparison
- Superior graphics quality
- Allows for simulation of caustics, global illumination, light dispersion, etc.
- Real time path tracing appears to be in reach

Leading Question and Goals

- Find out when real time path tracing will be viable
- Theoretical indicators (GPU peak FLOPS)
- Practical indicators (benchmarks)
- Prefer performance to quality

Physically Based Approach

Forward path tracing

Light source ($\Rightarrow$ scene interactions) $\Rightarrow$ observer

Backward path tracing

Observer ($\Rightarrow$ scene interactions) $\Rightarrow$ light source

Theoretical Basis

James Kajiya's rendering equation

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t)\, L_i(\mathbf{x}, \omega_i, \lambda, t)\, (\omega_i \cdot \mathbf{n})\, \mathrm{d}\,\omega_i$$

Simplified rendering equation

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o)\, L_i(\mathbf{x}, \omega_i)\, (\omega_i \cdot \mathbf{n})\, \mathrm{d}\,\omega_i$$

- Path tracing is a numerical approximate solution to the rendering equation

Rendering equation breakdown

$$\boxed{L_o(\mathbf{x},\,\omega_o)} = \boxed{L_e(\mathbf{x},\,\omega_o)} + \boxed{\int_\Omega \boxed{f_r(\mathbf{x},\,\omega_i,\,\omega_o)}\boxed{L_i(\mathbf{x},\,\omega_i)}\boxed{(\omega_i\cdot\mathbf{n})}\,\mathrm{d}\,\omega_i}$$

- $\boxed{L_o(\mathbf{x},\,\omega_o)}$ is the **outgoing light** with $\mathbf{x}$ being a point on a surface from which the light is reflected from into direction ω_o.
- $\boxed{L_e(\mathbf{x},\,\omega_o)}$ is the **emitted light** from point $\mathbf{x}$.
- $\boxed{\int_\Omega \ldots \mathrm{d}\,\omega_i}$ is the integral over Ω which is the hemisphere at $\mathbf{x}$ (centered around $\mathbf{n}$). All possible values for ω_i are therefore contained in Ω.
- $\boxed{f_r(\mathbf{x},\,\omega_i,\,\omega_o)}$ is the **BRDF** which determines how much light is reflected from ω_i to ω_o at $\mathbf{x}$.
- $\boxed{L_i(\mathbf{x},\,\omega_i)}$ is the **incoming light** at $\mathbf{x}$ from ω_o.
- $\boxed{(\omega_i\cdot\mathbf{n})}$ is the **normal attenuation** at $\mathbf{x}$.

Algorithm part 1

```
max_depth = 5
scene = [triangle_1, ..., triangle_n]  # Many triangles defined here

def trace_ray(ray, depth):
    if depth >= max_depth:
        # Return black since we haven't hit anything but we're
        # at our limit for bounces
        return RGB(0, 0, 0)

    intersection = None
    for triangle in scene:
        intersection = check_intersection(ray, triangle)
        if intersection:  # Break at first intersection
            break

    if not intersection:
        # If we haven't hit anything, we can't bounce again so
        # we return black
        return RGB(0, 0, 0)
```

Algorithm part 2

```
        material = intersection.material;
        emittance = material.emittance

        # Shoot a ray into random direction and recurse
        next_ray = Ray()
        next_ray.origin = intersection.position
        next_ray.direction = random_vector_on_hemisphere(intersection.normal)

        # BRDF for diffuse materials
        reflectance_theta = dot(next_ray.direction, intersection.normal)
        brdf = 2 * material.reflectance * reflectance_theta
        reflected = trace_ray(next_ray, depth + 1)

        return emittance + (brdf * reflected)

    for sample in range(samples):
        for pixel in pixels:
            trace_path(ray_from_pixel(pixel), 0)
```

Motivation

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Motivation

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Design Overview

1. herp
2. derp

Results

1. herp
2. derp

Evaluation

1. herp
2. derp

Conclusion

- Recall the goal: When will real time path tracing be viable on commodity hardware?
- Real time means 60 FPS

Conclusion

- Recall the goal: When will real time path tracing be viable on commodity hardware?
- Real time means 60 FPS
-

Outlook

Many venues for improvement:

Outlook

Many venues for improvement:

- Overall: Faster hardware

Outlook

Many venues for improvement:

- Overall: Faster hardware
- Software: Memory access optimizations

Outlook

Many venues for improvement:

- Overall: Faster hardware
- Software: Memory access optimizations
- Intersection: Acceleration Data Structure (BVH, kd-tree)

Outlook

Many venues for improvement:

- Overall: Faster hardware
- Software: Memory access optimizations
- Intersection: Acceleration Data Structure (BVH, kd-tree)
- Convergence: Bidirectional path tracing + Multiple Importance Sampling

Outlook

Many venues for improvement:

- Overall: Faster hardware
- Software: Memory access optimizations
- Intersection: Acceleration Data Structure (BVH, kd-tree)
- Convergence: Bidirectional path tracing + Multiple Importance Sampling
- Image filtering: Bilateral filter

Outlook

Many venues for improvement:

- Overall: Faster hardware
- Software: Memory access optimizations
- Intersection: Acceleration Data Structure (BVH, kd-tree)
- Convergence: Bidirectional path tracing + Multiple Importance Sampling
- Image filtering: Bilateral filter
- Research: Who knows?

Outlook

Many venues for improvement:

- Overall: Faster hardware
- Software: Memory access optimizations
- Intersection: Acceleration Data Structure (BVH, kd-tree)
- Convergence: Bidirectional path tracing + Multiple Importance Sampling
- Image filtering: Bilateral filter
- Research: Who knows?

$\Longrightarrow$ Real time path tracing possible within 4 years