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 7, 2016

Table of Contents

Topic

Main topic of the thesis

When will real time path tracing be viable on consumer grade hardware?

Motivation

- Current generation graphics made up of many complex tricks

Motivation

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

Motivation

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

Motivation

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

History of Path Tracing part 1

- 1968: Ray casting by Arthur Appel
- 1980: Ray tracing by Turner Whitted

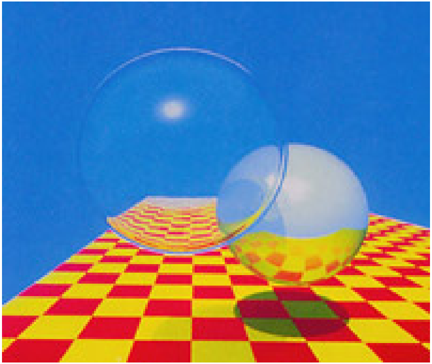


Figure: Turner Whitted's original 1980 image

History of Path Tracing part 2

- 1986: Monte Carlo ray tracing (path tracing) by James T. Kajiya
- 1996: Bidirectional path tracing by Eric Lafortune
- 1997: Metropolis Light Transport by Eric Veach and Leonidas J. Guibas

Current State of Technology

Projects utilizing real time path tracing:

- Jacco Bikker's and Jeroen van Schijndel's *Brigade Renderer*
- David Bucciarelli's *sfera*
- Evan Wallace's *WebGL Path Tracer*

Current State of Technology

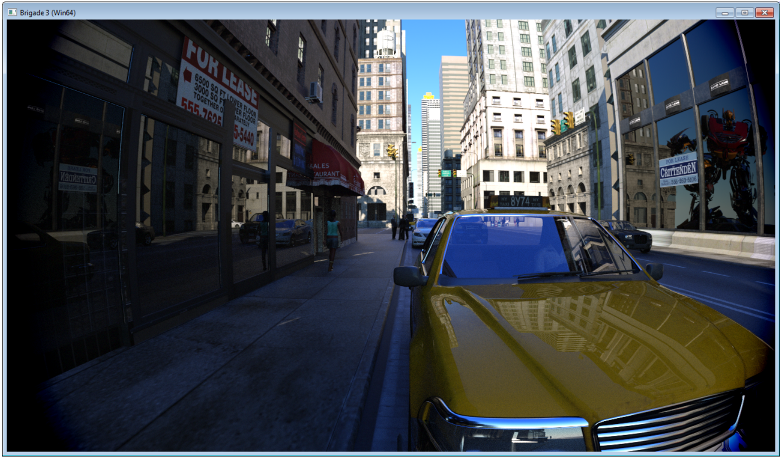


Figure: Brigade Renderer

Current State of Technology

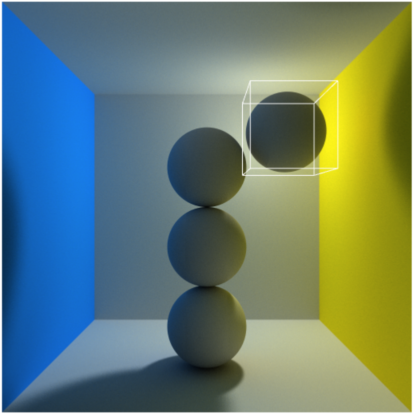


Figure: WebGL real time path tracer

Current State of Technology

Figure: Sfera real time path tracing video game.

Physically Based Approach

Forward path tracing

Light source (⇒ scene interactions) ⇒ observer

Backward path tracing

Observer (⇒ scene interactions) ⇒ light source

Bidirectional path tracing

Observer (⇒ scene interactions) ⇒ camera vertex
Light source (⇒ scene interactions) ⇒ light vertex
Connect all vertices

Theoretical Basis

James Kajiya's rendering equation

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

Simplified rendering equation

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

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

Rendering equation breakdown

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

Rendering equation breakdown

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

$\boxed{L_o(\mathbf{x}, \omega_o)}$ is the **outgoing light** with $\mathbf{x}$ being a point on a surface from which the light is reflected from into direction ω_o.

Rendering equation breakdown

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

$\boxed{L_o(\mathbf{x}, \omega_o)}$ 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 $\mathbf{x}$.

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.

Rendering equation breakdown

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

$\boxed{L_o(\mathbf{x}, \omega_o)}$ 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 $\mathbf{x}$.

$\boxed{\int_\Omega \ldots d\,\omega_i}$ 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 $\mathbf{x}$.

Rendering equation breakdown

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

- $\boxed{L_o(\mathbf{x}, \omega_o)}$ 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 $\mathbf{x}$.
- $\boxed{\int_\Omega \ldots d\,\omega_i}$ 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.

Rendering equation breakdown

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

- $\boxed{L_o(\mathbf{x}, \omega_o)}$ 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 $\mathbf{x}$.
- $\boxed{\int_\Omega \ldots d\,\omega_i}$ 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

```python
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)
```

Design Overview

Design Overview

Design goals:

1. **Speed**
2. **Ease of implementation**
3. **Interactivity**
4. **Portability**
5. **Maintainability**

Design Overview

Components:

- Random number generation
- Camera
- Material system
- AABBs
- Triangles
- Shapes and geometry
- Scene
- Intersection/Scene lookup
- Viewer

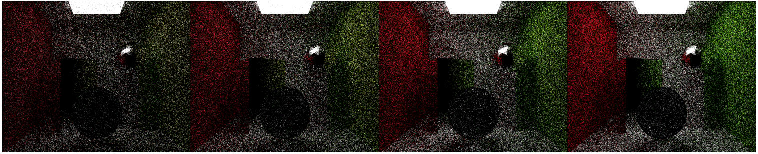
Design Overview

Materials:

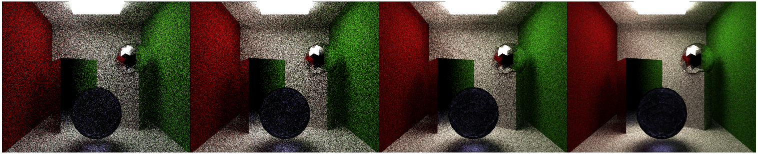
- Emissive
- Diffuse
- Glass
- Glossy

Visual Results

Visual Results



(i) $n = 1$ (j) $n = 2$ (k) $n = 3$ (l) $n = 4$



(m) $n = 10$ (n) $n = 20$ (o) $n = 50$ (p) $n = 100$

Visual Results

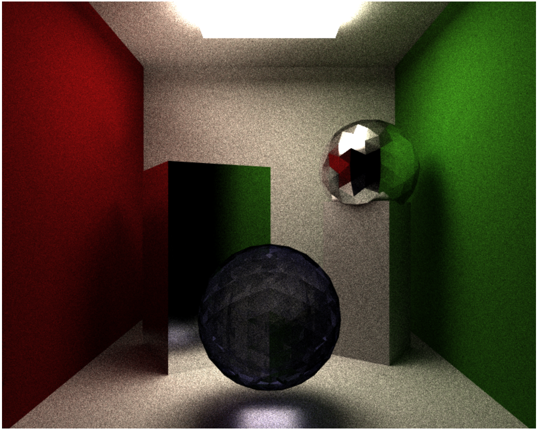


Figure: trac0r_viewer after integrating for 5000 frames.

Visual Results

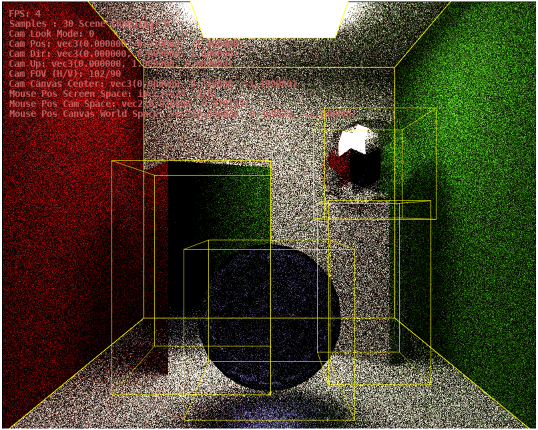


Figure: trac0r_viewer debug view.

Visual Results

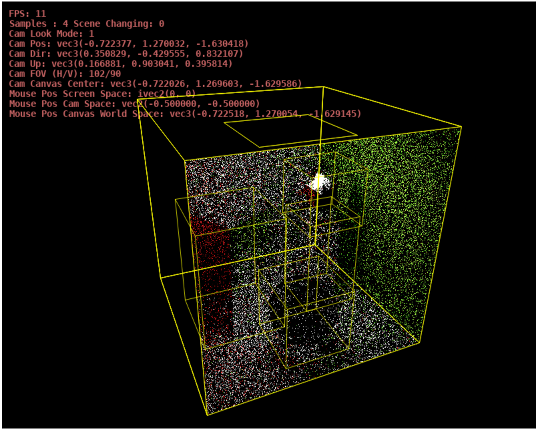


Figure: trac0r_viewer debug view while interactively navigating the scene.

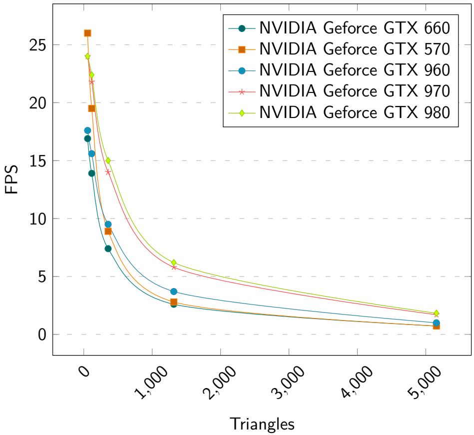
Visual Results

Live presentation!

Evaluation

Evaluation

Performance in correlation to scene complexity

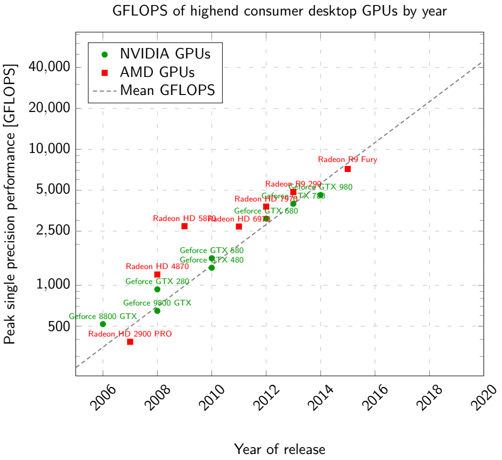


Evaluation

GPU	Average FPS	GFLOPS	$\frac{\text{Average FPS}}{GFLOPS}$
NVIDIA Geforce GTX 660	8.3	1881.6	4.4×10^{-3}
NVIDIA Geforce GTX 570	11.6	1405.4	8.6×10^{-3}
NVIDIA Geforce GTX 960	9.5	2308	4.1×10^{-3}
NVIDIA Geforce GTX 970	13.5	3494	3.9×10^{-3}
NVIDIA Geforce GTX 980	13.9	4612	3.0×10^{-3}

Table: Tested GPU performance and GFLOPS.

Evaluation


GFLOPS of highend consumer desktop GPUs by year
NVIDIA GPUs
AMD GPUs
Mean GFLOPS
Peak single precision performance [GFLOPS]
40,000
20,000
10,000
5,000
2,500
1,000
500
Radeon R9 Fury
Radeon R9 290
Geforce GTX 980
Radeon HD 7970
Geforce GTX 780
Geforce GTX 680
Radeon HD 5870
Radeon HD 6970
Geforce GTX 580
Geforce GTX 480
Radeon HD 4870
Geforce GTX 280
Geforce 9800 GTX
Geforce 8800 GTX
Radeon HD 2900 PRO
2006
2008
2010
2012
2014
2016
2018
2020
Year of release

Conclusion

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

Conclusion

- Recall the goal: When will real time path tracing be viable on commodity hardware?
- Data and research suggests viability in ≈ 4 years

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

Fin

Thank you for your attention.