

# 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

Introduction

Motivation

Leading Question and Goals

Real Time Path Tracing Explained

Physically Based Approach

Theoretical Basis

Algorithm

History of Path Tracing

Current State of Technology

Research

Implementation Design Overview

Results

Evaluation

Conclusion and Outlook

Conclusion

Outlook

# 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

# 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

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

## 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

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

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

$L_e(x, \omega_o)$  is the **emitted light** from point  $x$ .

$\int_{\Omega} \dots d\omega_i$  is the integral over  $\Omega$  which is the hemisphere at  $x$  (centered around  $n$ ). All possible values for  $\omega_i$  are therefore contained in  $\Omega$ .

$f_r(x, \omega_i, \omega_o)$  is the **BRDF** which determines how much light is reflected from  $\omega_i$  to  $\omega_o$  at  $x$ .

$L_i(x, \omega_i)$  is the **incoming light** at  $x$  from  $\omega_o$ .

$(\omega_i \cdot n)$  is the **normal attenuation** at  $x$ .

## Algorithm

# Algorithm part 1

```
1   max_depth = 5
2   scene = [triangle_1, ..., triangle_n]  # Many triangles defined here
3
4   def trace_ray(ray, depth):
5       if depth >= max_depth:
6           # Return black since we haven't hit anything but we're
7           # at our limit for bounces
8           return RGB(0, 0, 0)
9
10      intersection = None
11      for triangle in scene:
12          intersection = check_intersection(ray, triangle)
13          if intersection:  # Break at first intersection
14              break
15
16      if not intersection:
17          # If we haven't hit anything, we can't bounce again so
18          # we return black
19          return RGB(0, 0, 0)
```

## Algorithm

## Algorithm part 2

```
20     material = intersection.material;
21     emittance = material.emittance
22
23     # Shoot a ray into random direction and recurse
24     next_ray = Ray()
25     next_ray.origin = intersection.position
26     next_ray.direction = random_vector_on_hemisphere(intersection.normal)
27
28     # BRDF for diffuse materials
29     reflectance_theta = dot(next_ray.direction, intersection.normal)
30     brdf = 2 * material.reflectance * reflectance_theta
31     reflected = trace_ray(next_ray, depth + 1)
32
33     return emittance + (brdf * reflected)
34
35 for sample in range(samples):
36     for pixel in pixels:
37         trace_path(ray_from_pixel(pixel), 0)
```

## History of Path Tracing

# 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

# 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

# Current State of Technology

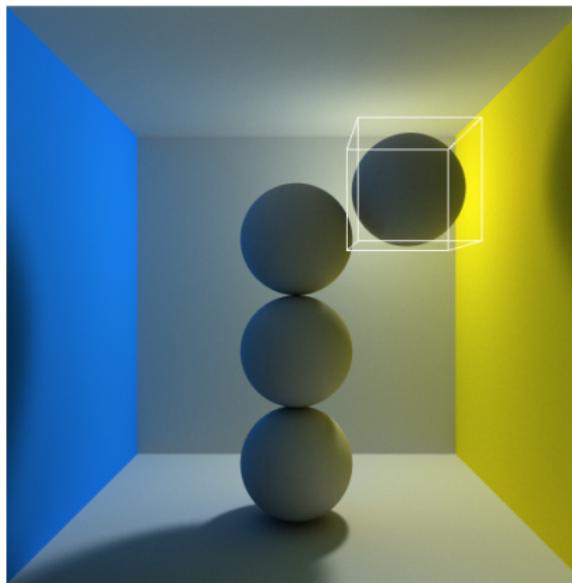


Figure: Sfera real time path tracing video game.

## Current State of Technology

# Current State of Technology



Figure: Sfera real time path tracing video game.

# 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:

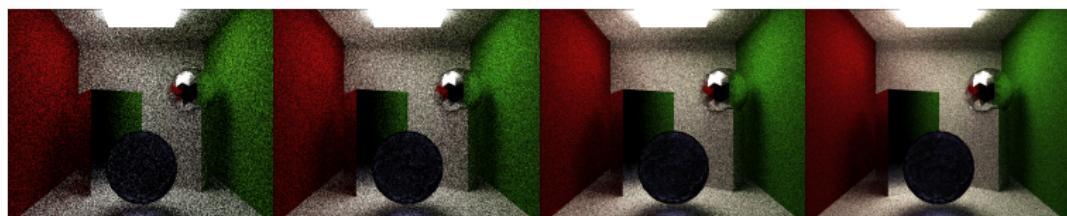
- ▶ Emissive
- ▶ Diffuse
- ▶ Glass
- ▶ Glossy



## Results

# Visual Results

# Visual Results



# Visual Results

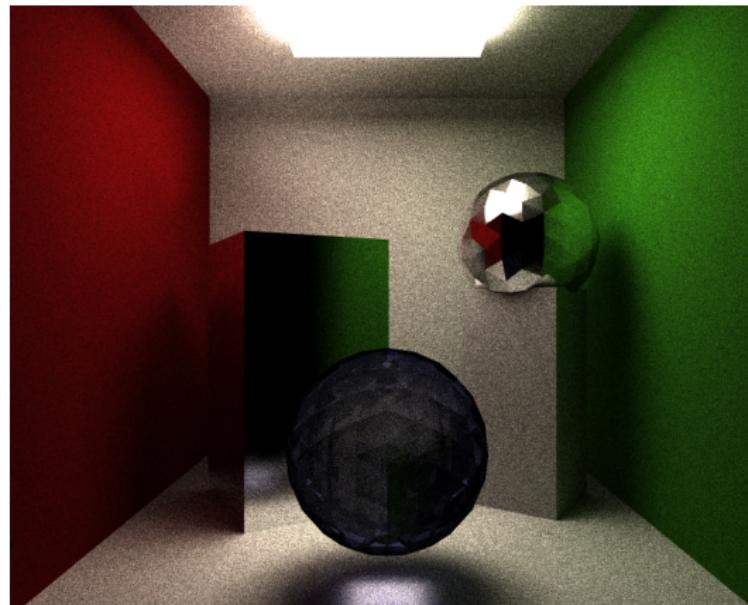


Figure: tracOr\_viewer after integrating for 5000 frames.

# Visual Results

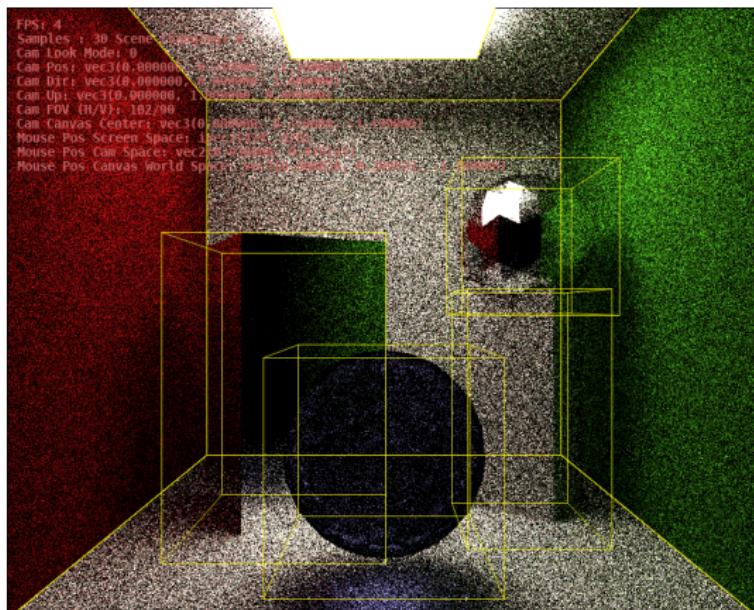


Figure: `tracor_viewer` debug view.

# Visual Results

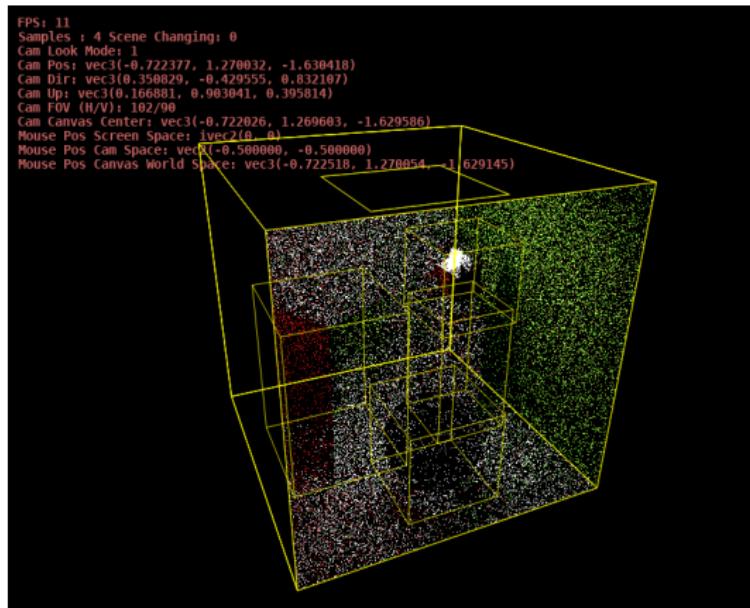


Figure: trac0r\_viewer debug view while interactively navigating the scene.

# Visual Results

Live presentation!

Introduction

- 
- 

Real Time Path Tracing Explained

- 
- 
- 
- 

Evaluation

Research

- 
- 
- 

Conclusion and Outlook

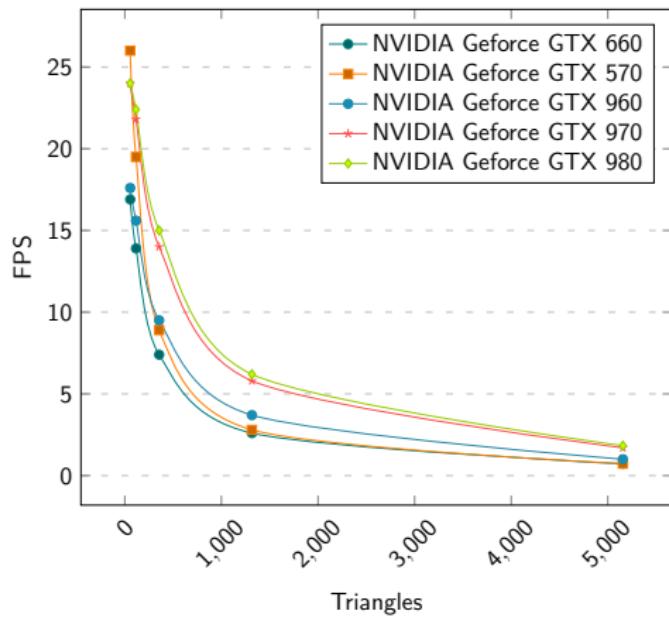
- 
- 

# Evaluation

## Evaluation

# Evaluation

Performance in correlation to scene complexity



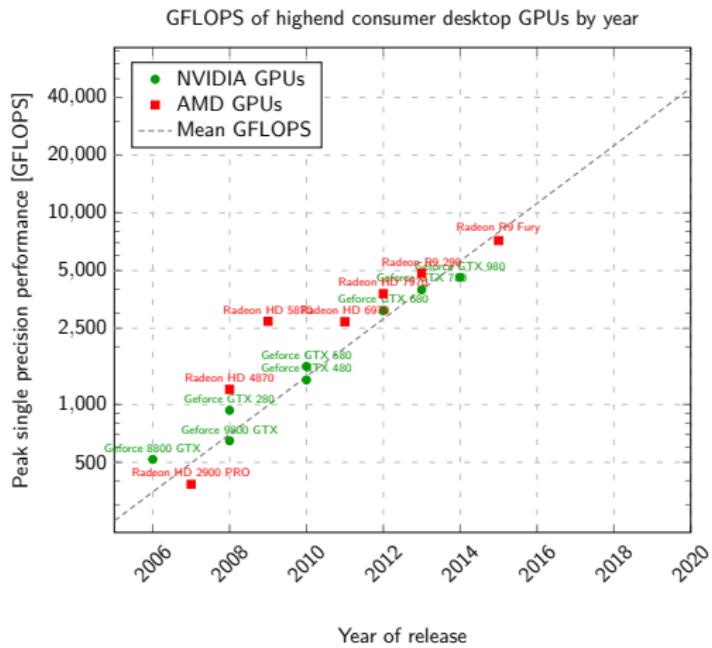
# Evaluation

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

Table: Tested GPU performance and GFLOPS.

## Evaluation

# Evaluation



# 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  $\approx 4$  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?
- ⇒ Real time path tracing possible within 4 years