Poisson Reconstruction for Global Illumination on Surfels

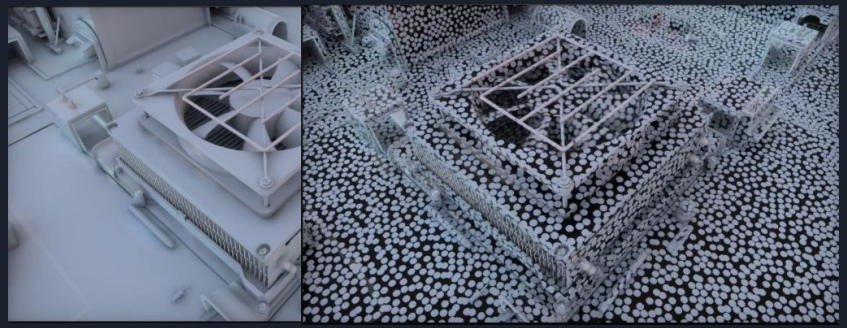
Master Thesis

Antonio Noack, 09.12.2022

Inspiration

- **Interesting paper** from Electronic Arts “Global Illumination Based on Surfels” and SEED “Hybrid Rendering for Real-Time Ray Tracing”
- **Nice looking graphics** for real-time, computer simulated games, e.g., like GTA V
- Personal game engine “Rem’s Engine”,
  because graphics can sell game engines (e.g., Unreal Engine) and games
- Open Source implementation

Image from “Global Illumination Based on Surfels”



Images from “Towards effortless photorealism through real-time raytracing”

Overview

- BSDFs, Path Tracing
- Global Illumination
- Surfels
  - Weights
  - Drawing
  - Distribution
  - Transparency
- Poisson Reconstruction
- Results
- Future Work

Defining Materials: **BSDFs**

Bidirectional Scattering Distribution Functions

Materials (Unity, Unreal Engine, Blender)

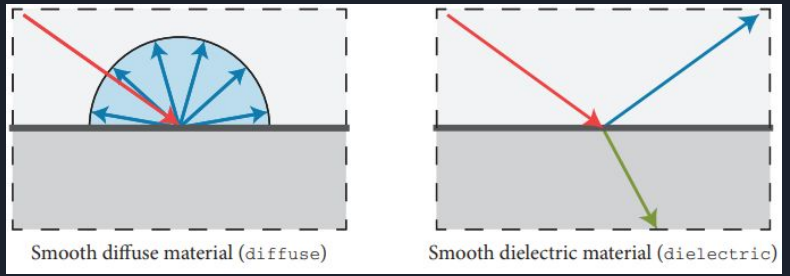


Image from Mitsuba Handbook

How light interacts with a surface:

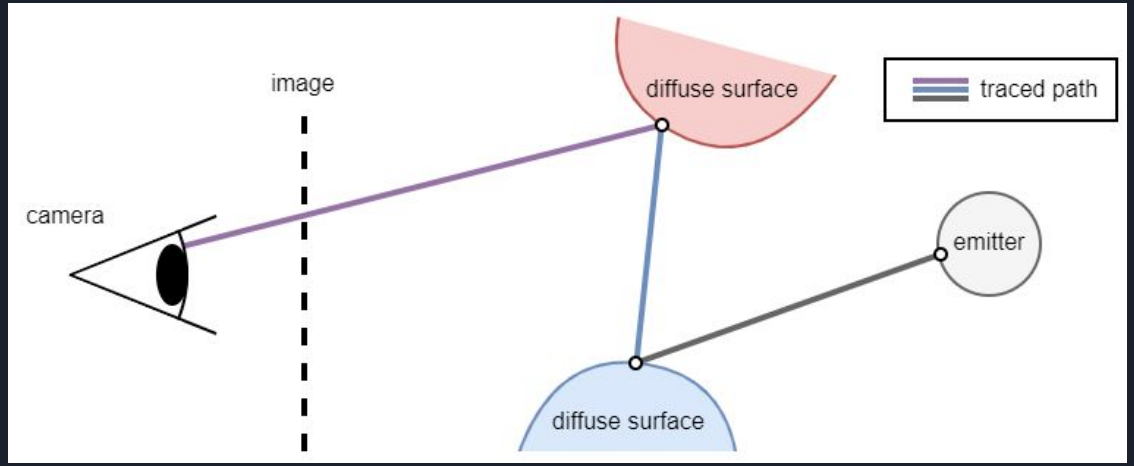
- How much light is absorbed, and what colors (which wavelengths)
- How the remaining light is being distributed over the surface hemisphere

Path Tracing

Simple method to calculate realistic illumination:

- Start with white color at camera
- Trace ray in scene until emissive surface is found
- At every hit, multiply (tint) color by surface color
- To continue ray, choose exit direction by BSDF

Repeat, and average samples until converged

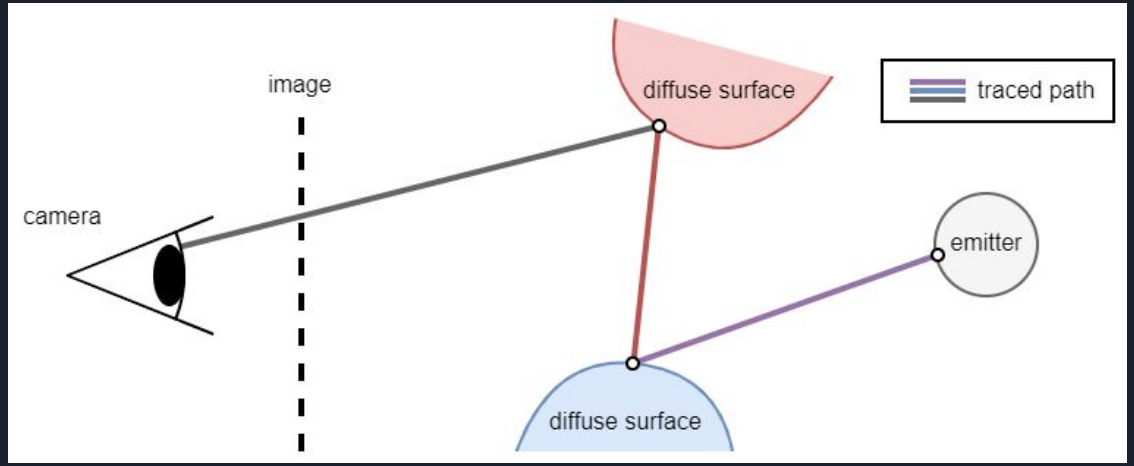


Path Tracing

Simple method to calculate realistic illumination:

- Start with white color at camera
- Trace ray in scene until emissive surface is found
- At every hit, multiply (tint) color by surface color
- To continue ray, choose exit direction by BSDF

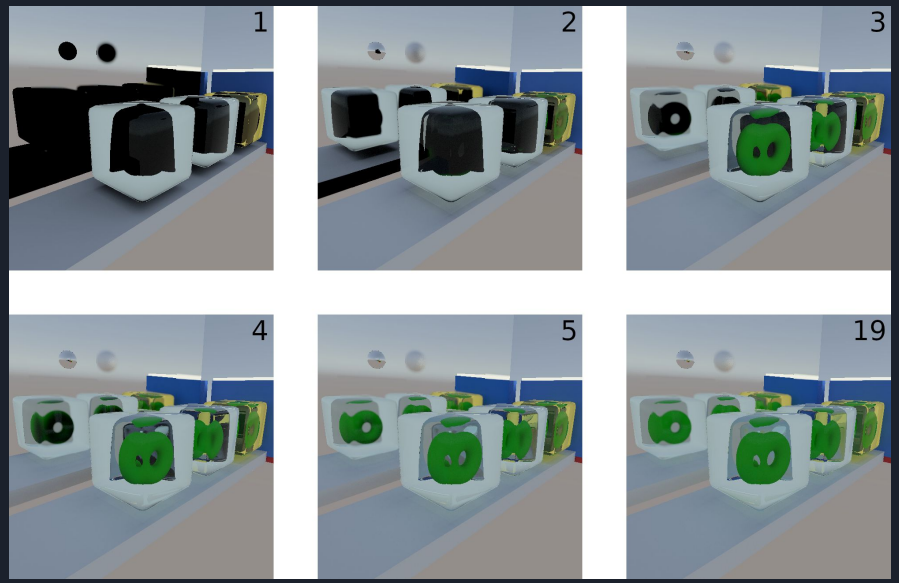
Repeat, and average samples until converged



Path Tracing - Termination

Paths may never find an emissive surface → infinite loop ⚡

→ set an artificial iteration limit + consider path to be completely dark (set color to black)



Path Tracing with limited number of bounces

Global Illumination

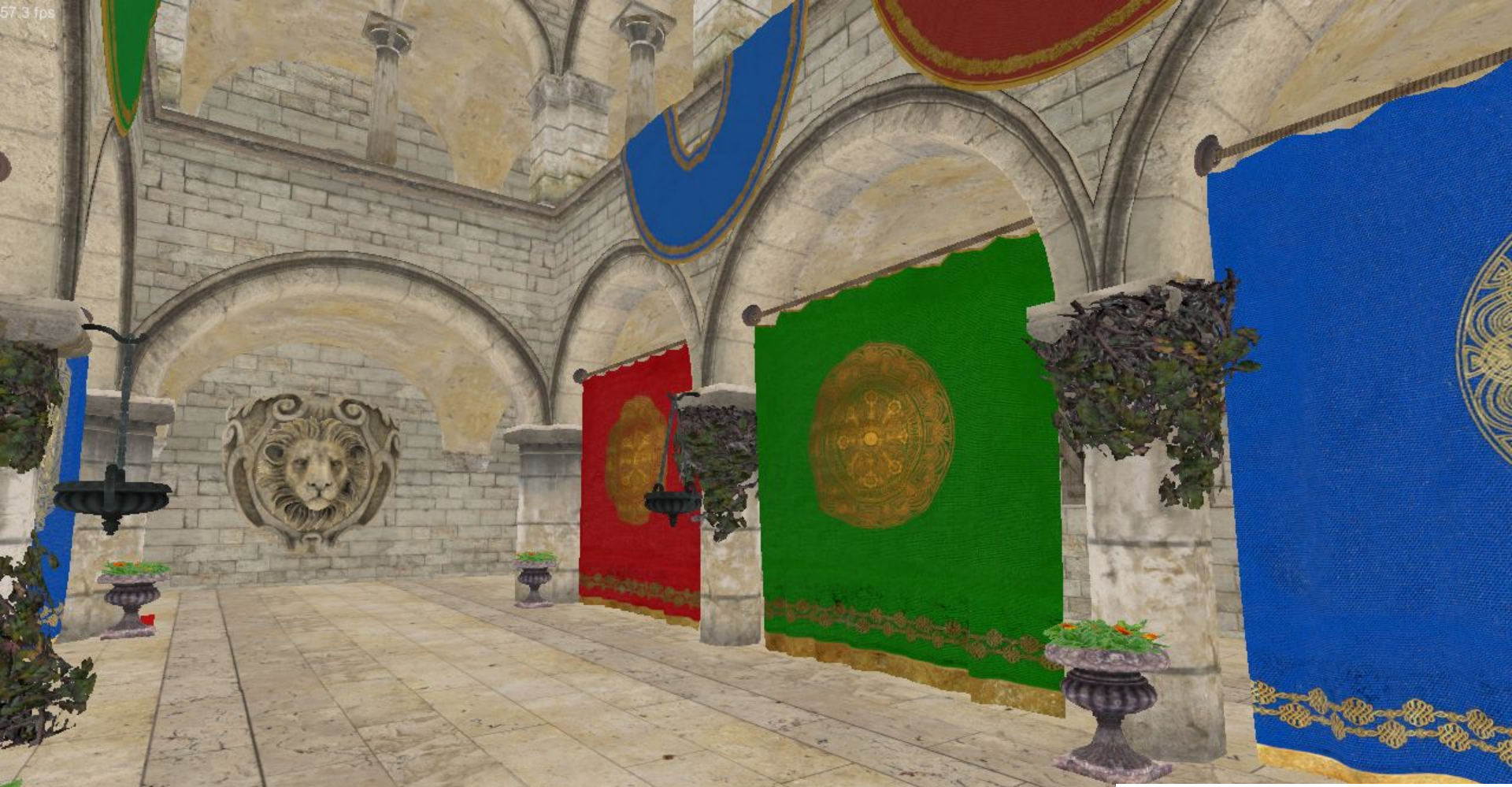
Separate color from illumination information

Why?

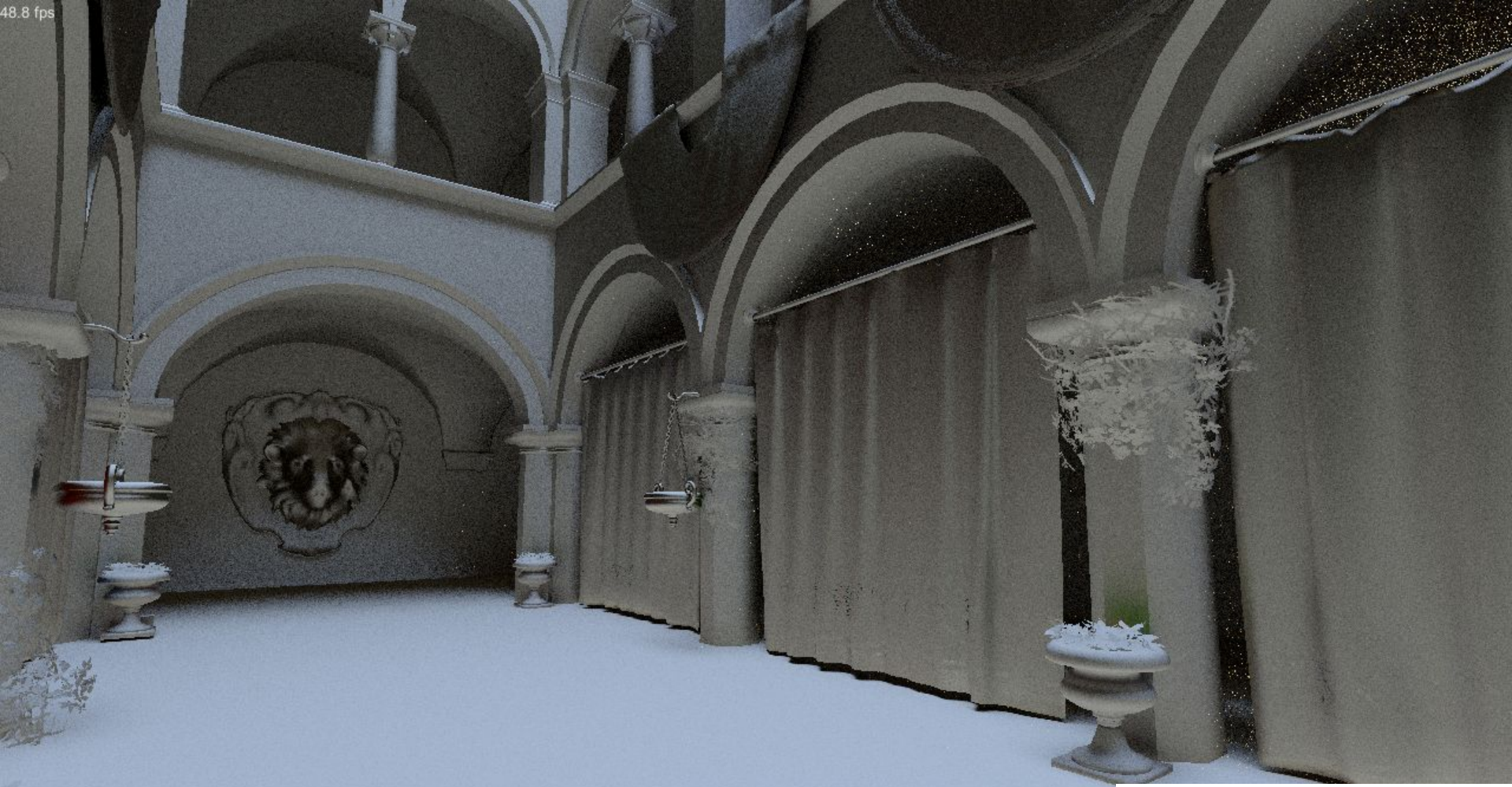
- Illumination may be much less detailed than surface colors (albedo)
- Illumination computationally expensive, surface colors cheap

How?

- Most materials have multiplicative color term → extract
- Pretend first hit is white, calculate, then multiply illumination by albedo



Color (Sponza)



Global Illumination (Sponza)

Result (Sponza)

Result, GI blurred (Sponza)

GI - Best case

High frequency (detailed) materials, low frequency GI:

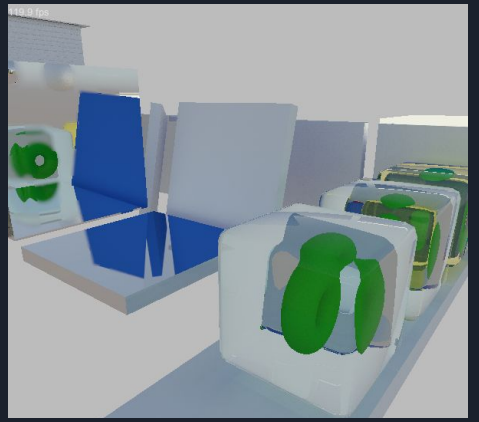
- Diffuse materials (absorbing, random ray exit direction)
- Detailed albedo textures
- Normal maps, material mask textures

Masked materials on surfel global illumination

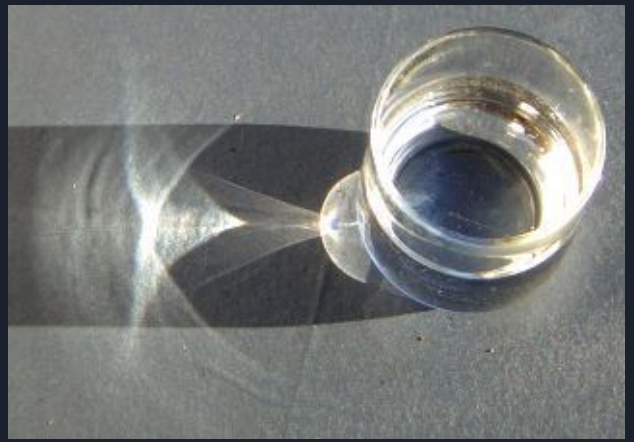
GI - Worst case

High frequency details in GI, few details in color:

- Glass surfaces
- Smooth, reflective surfaces like mirrors, glitter
- Transparent surfaces
- Caustics



GI with sharp reflections and refractions



Heiner Otterstedt,
https://en.wikipedia.org/wiki/File:Kaustik.jpg

Surfels

Surface Elements

- Store spatial, local data, here illumination
- Belong to a surface
  - Specific to normal, and surface properties like roughness, smoothness, BSDF in general

→ weighting function for interpolation needed

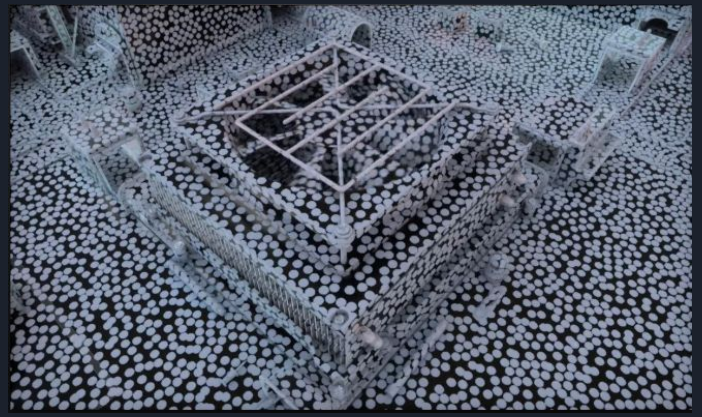


Image from “Towards effortless photorealism through real-time raytracing”

Surfel Weights

$$c_p = \frac{\sum_s c_s \cdot w(s,p)}{\sum_s w(s,p)}$$

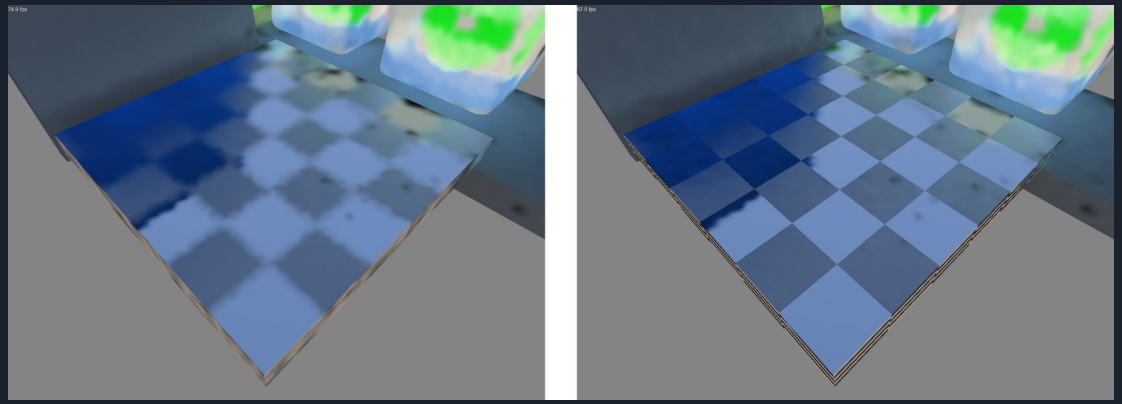
Interpolation formula

Goal is lighting data interpolation without knowledge about other surfels:

Assign weight *w(s,p)* for each surfel *s* to each pixel *p*:

Each surfel is handled with a BSDF matching the original pixel, where it was spawned

Terms: distance, normal, roughness, specularity



Surfel GI without/with weight terms for roughness and specularity

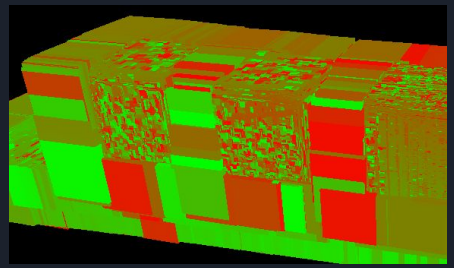
Drawing Surfels in Unity

1) *Instanced rendering* in Unity requires matrix array
   → large overhead + CPU->GPU data transfer needed

2) *Procedural rendering*, all surfel data can remain on GPU

Surfel Count	Instanced FPS	Procedural FPS
262144	53.0	120
524288	27.0	120
1048576	13.5	120
2097152	6.3	76
4194304	3.1	39

Perf. in Unity 2023.1.0a15; Ryzen 5 2600, RTX 3070



Goal: Interpolation of colors (and gradients)

→ sum of *(light * weight)*s + sum of *weights* → additive blending on RGB + W (alpha channel)

→ in following shader division by sum of *weights* ; if weight sum is zero, search neighbor pixels

$$c_p = \frac{\sum_s c_s \cdot w(s,p)}{\sum_s w(s,p)}$$

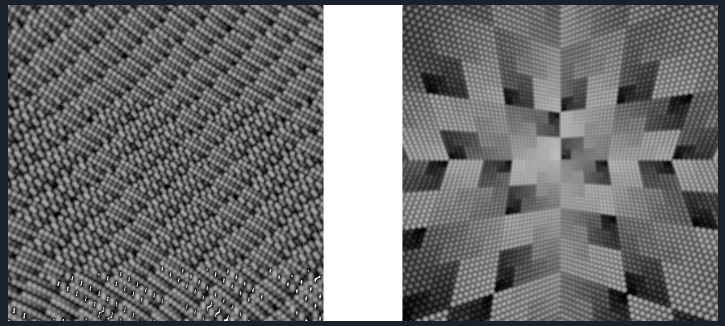
Interpolation formula

Surfel Distribution

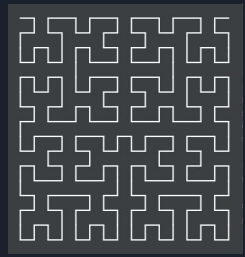
Initial via Fibonacci sphere / projected Hilbert curve

Update: in 16x16 sections, find pixel with lowest weight,
and respawn random surfel there (temporal sample collection)

Non-ideal: high densities mostly remain when moving backwards

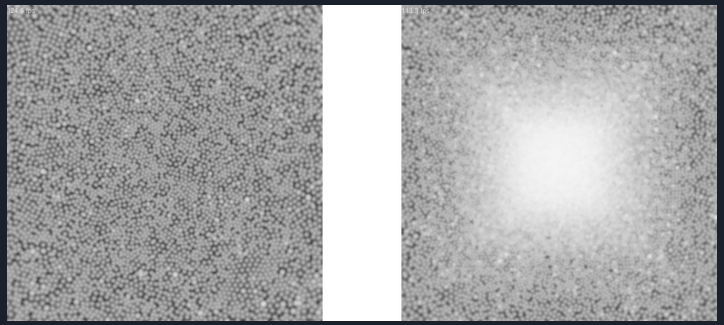


Fibonacci sphere / Hilbert curve initialization



Hilbert curve

Surfel dist. after some time with camera movement (Sponza)

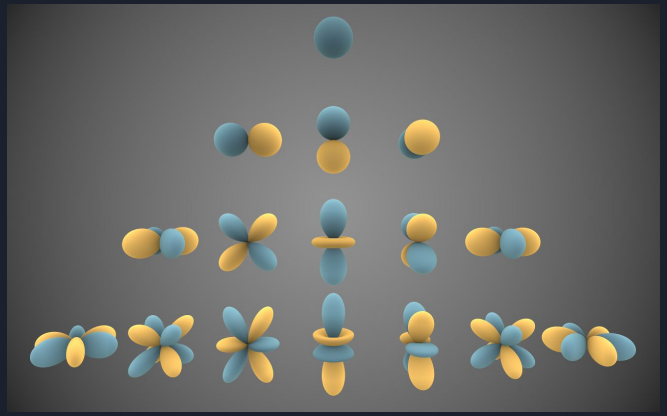


Dense spot after moving away from wall

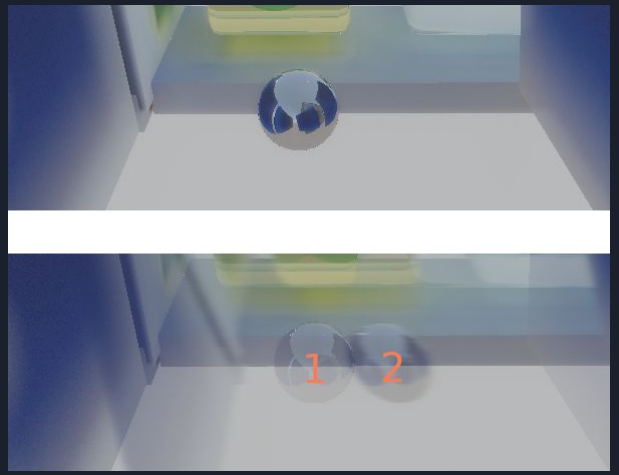
Transparency on Surfels

- Spherical Harmonics
- Dithering
- Handling transparent surfaces as highly angle dependent, like mirrors

Dithered foliage in image from "GTA V - Graphics Study"



Spherical Harmonics from Inigo Quilez

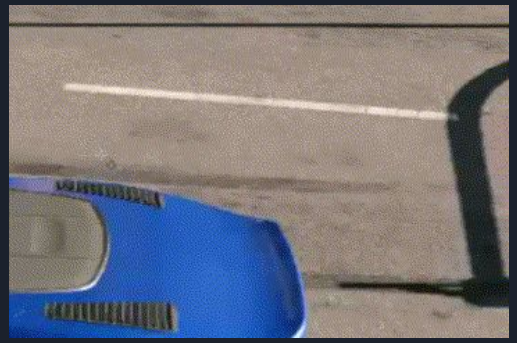


Ghosting on transparent surface

Poisson Reconstruction

Gimp/Photoshop Healing operator:
    Merge foreign features into images

Given base color and gradients, reconstruct the original image
    Often solved iteratively, e.g., with Gauß-Seidel-iteration



Transferring a stripe pattern in GIMP
Image: screenshot from Watch Dogs 2

Idea from **“Gradient Domain Path Tracing”**: gradient information has less fireflies, so it is more stable ~ has less “energy”: only non-zero, if illumination changes

→ should accelerate convergence

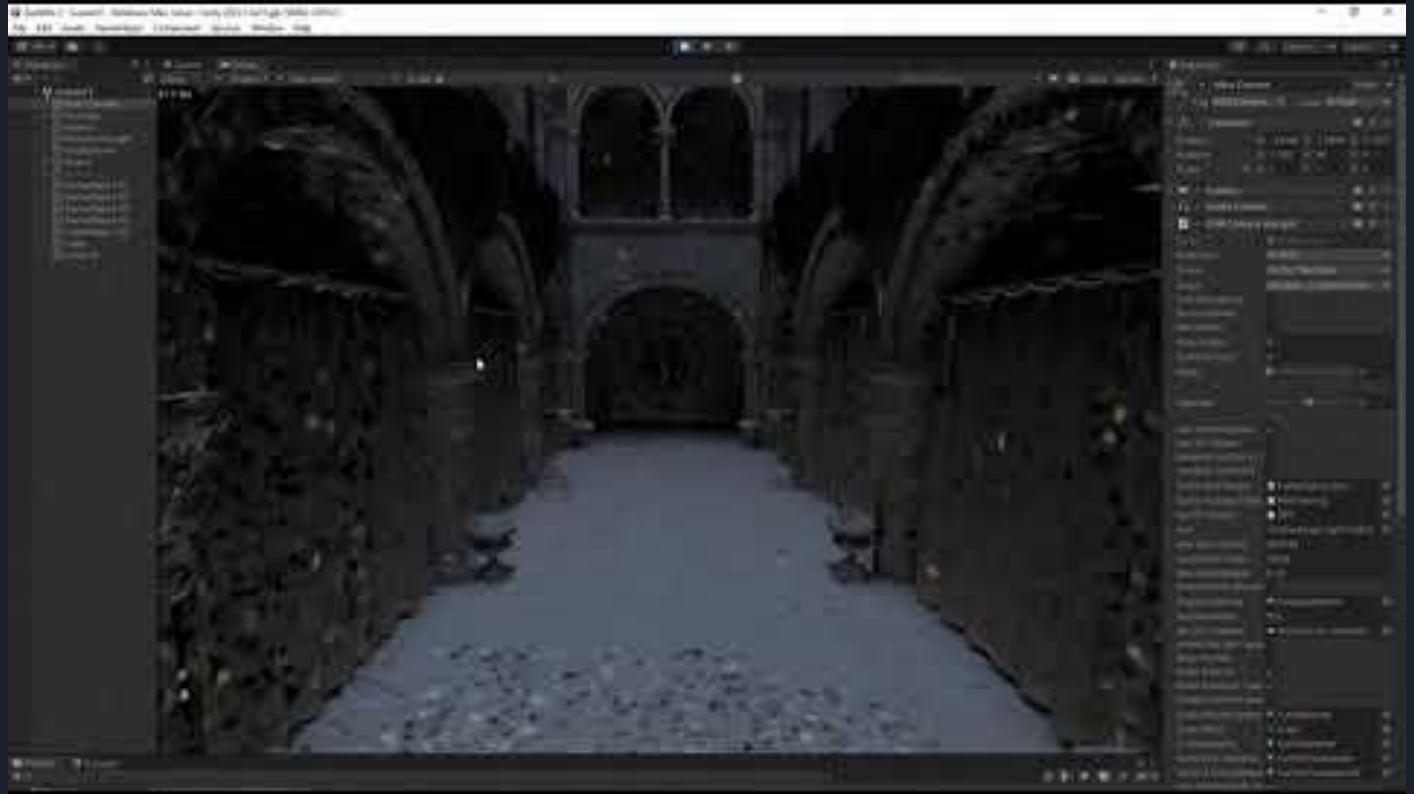
Results

Working surfel distribution

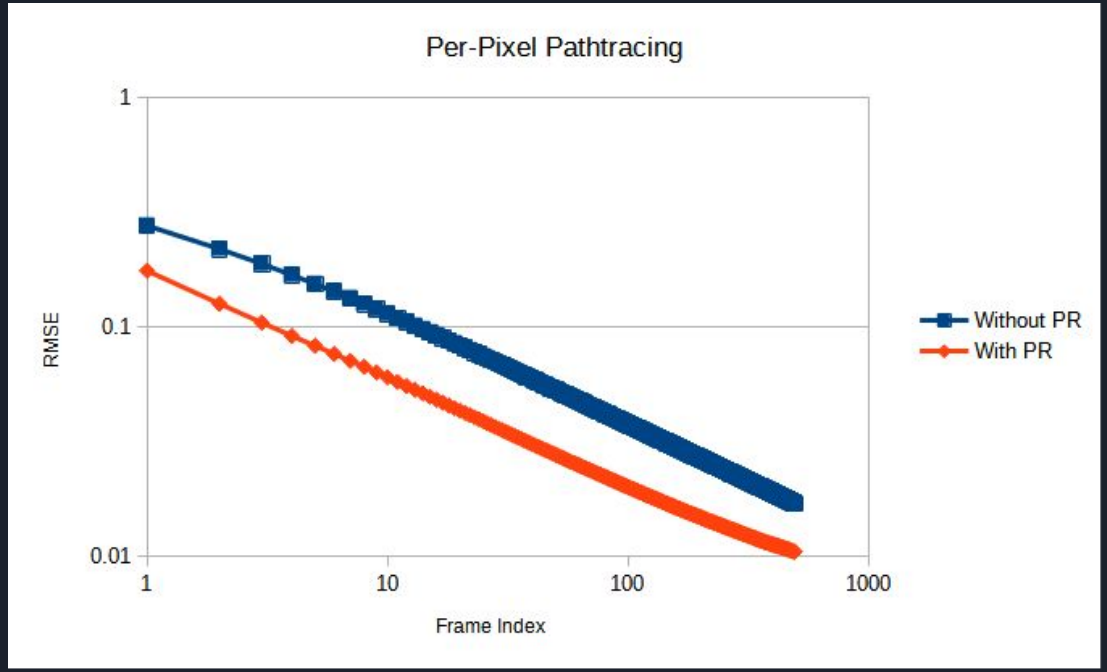
Weighting scheme on BSDF-specific surfels works for colors

Procedural rendering is performant

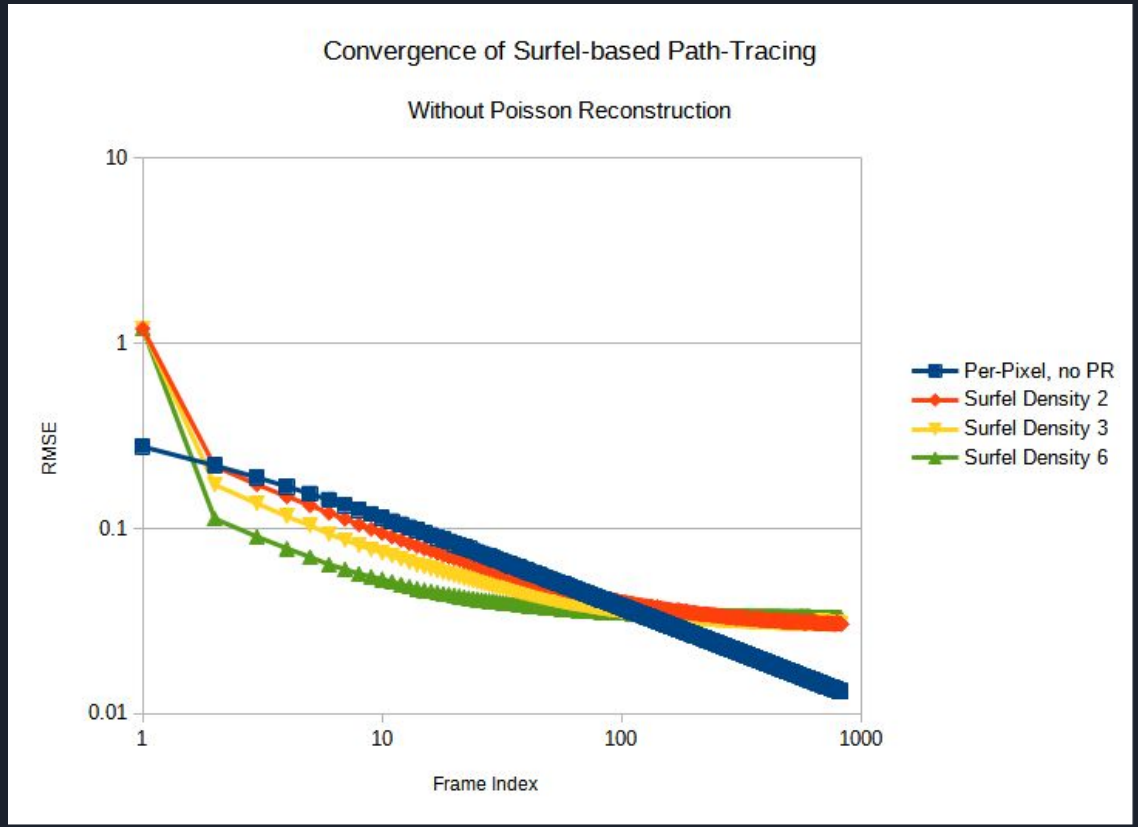
Surfel distribution after some time with camera movement (Sponza)



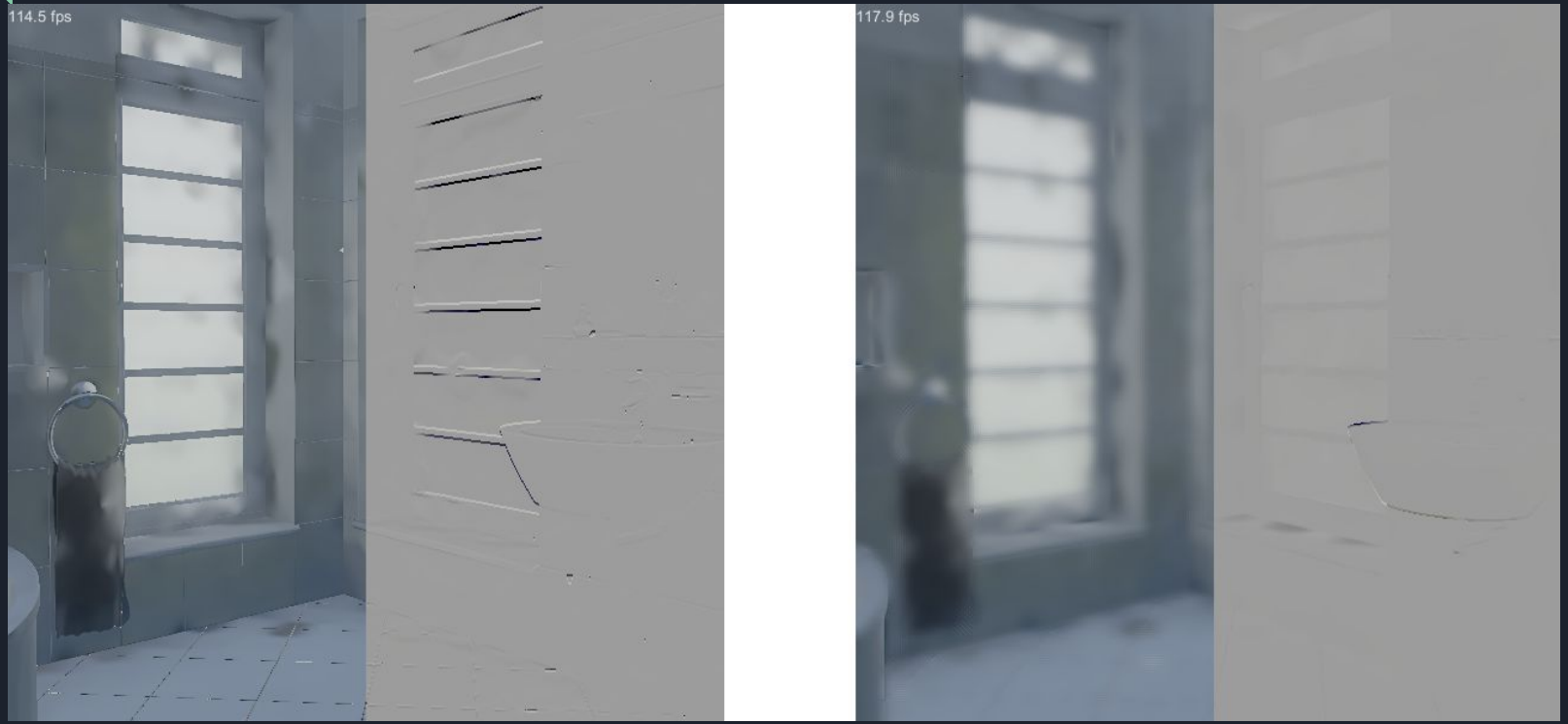
Per-Pixel Path Tracing, comparison with/without Poisson Reconstruction



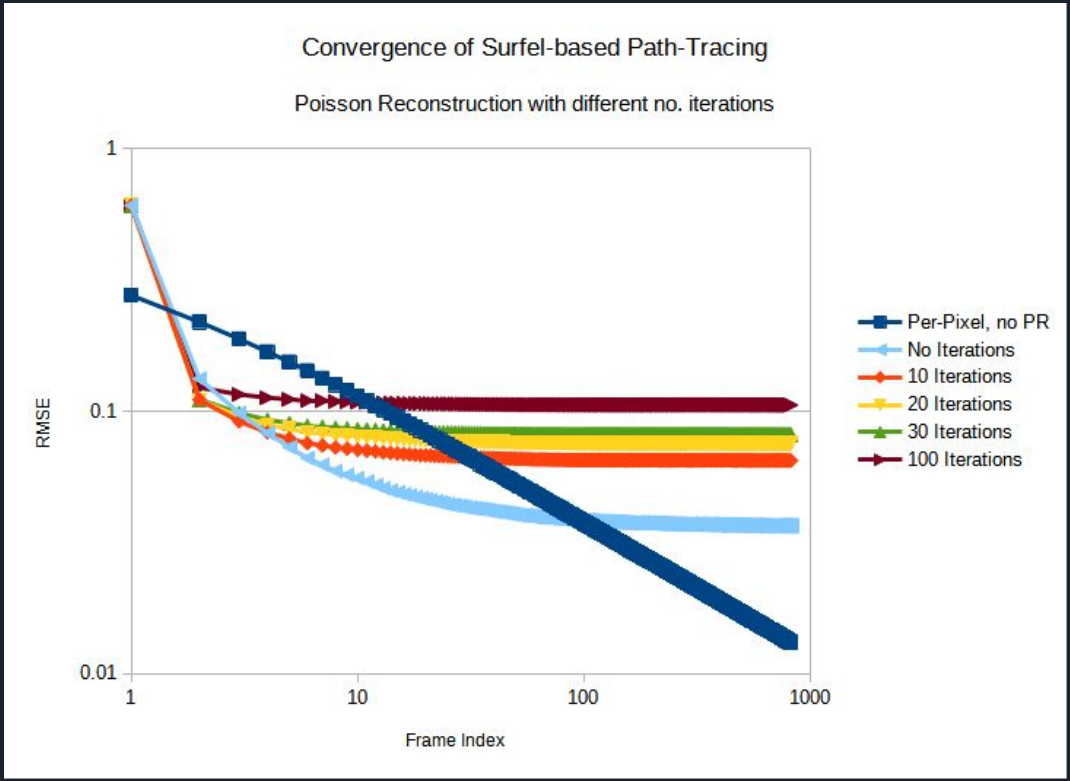
Surfels vs Per-Pixel



Surfel Gradients: too smooth
Weighting gradients doesn’t work that easily



Surfel Gradients: too smooth
Unsuitable for Poisson Reconstruction



Future Work

- **Better gradient calculation** for edges, e.g., by averaging sides of edge separately, then summing
  Neither sum nor average is correct

- Draw surfels using quads instead of boxes (performance)
- Increase path tracing efficiency, e.g., using ReSTIR (spatiotemporal reservoir resampling)
- More efficient gradients, e.g., by bidirectional gradient domain path-tracing (C++ → Unity translation needed, plus replacements for dynamic allocations)
- Transparent/extremely rapidly changing regions could be handled with spherical harmonics, and really smooth reflections using per-pixel PT
- Despawn surfels, where very dense
- Surfel migration by gradient of density
- Handle reflections and direct shadows separately for sharp results

Thank you for your attention

Sources:

Mitsuba Handbook image:
https://www.mitsuba-renderer.org/releases/current/documentation.pdf, sec. 8.2, page 52

”Global Illumination Based on Surfels” image: page 178
Andreas Brinck and Xiangshun Bei and Henrik Halén and Kyle Hayward

“Shiny Pixels and Beyond: Real-Time Raytracing at SEED” image: page 57/58
Tomasz Stachowiak, SEED – Electronic Arts

”GTA V - Graphics Study” image: *Adrian Courrèges*,
https://www.adriancourreges.com/blog/2015/11/02/gta-v-graphics-study/

Spherical Harmonics image: *Inigo Quilez*
https://en.wikipedia.org/wiki/File:Spherical_Harmonics.png