IMPLEMENTATION OF OPTIMIZED PIXEL-PROJECTED REFLECTIONS FOR PLANAR REFLECTORS

Byumjin Kim

Computer Graphics and Game Technology

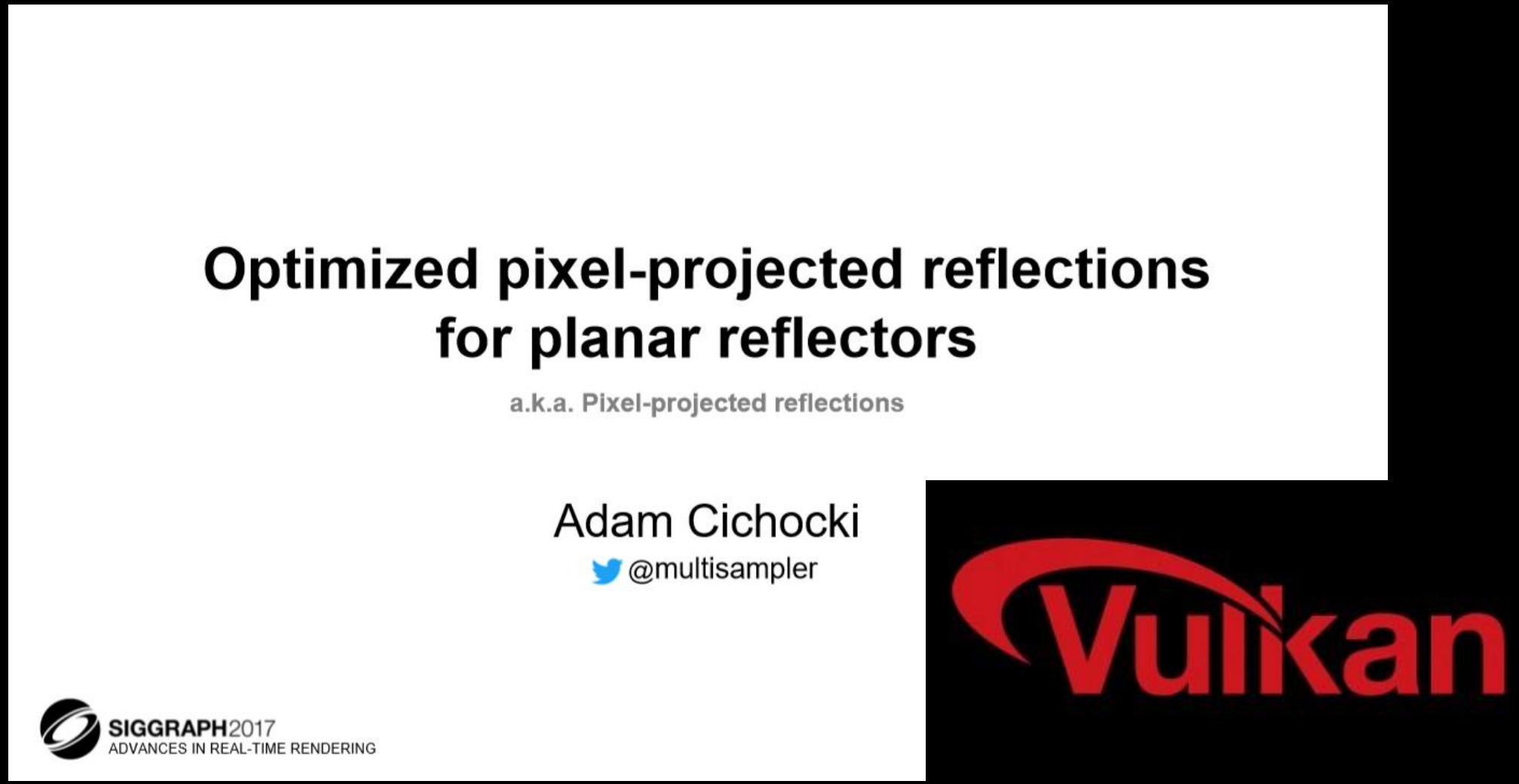
University of Pennsylvania

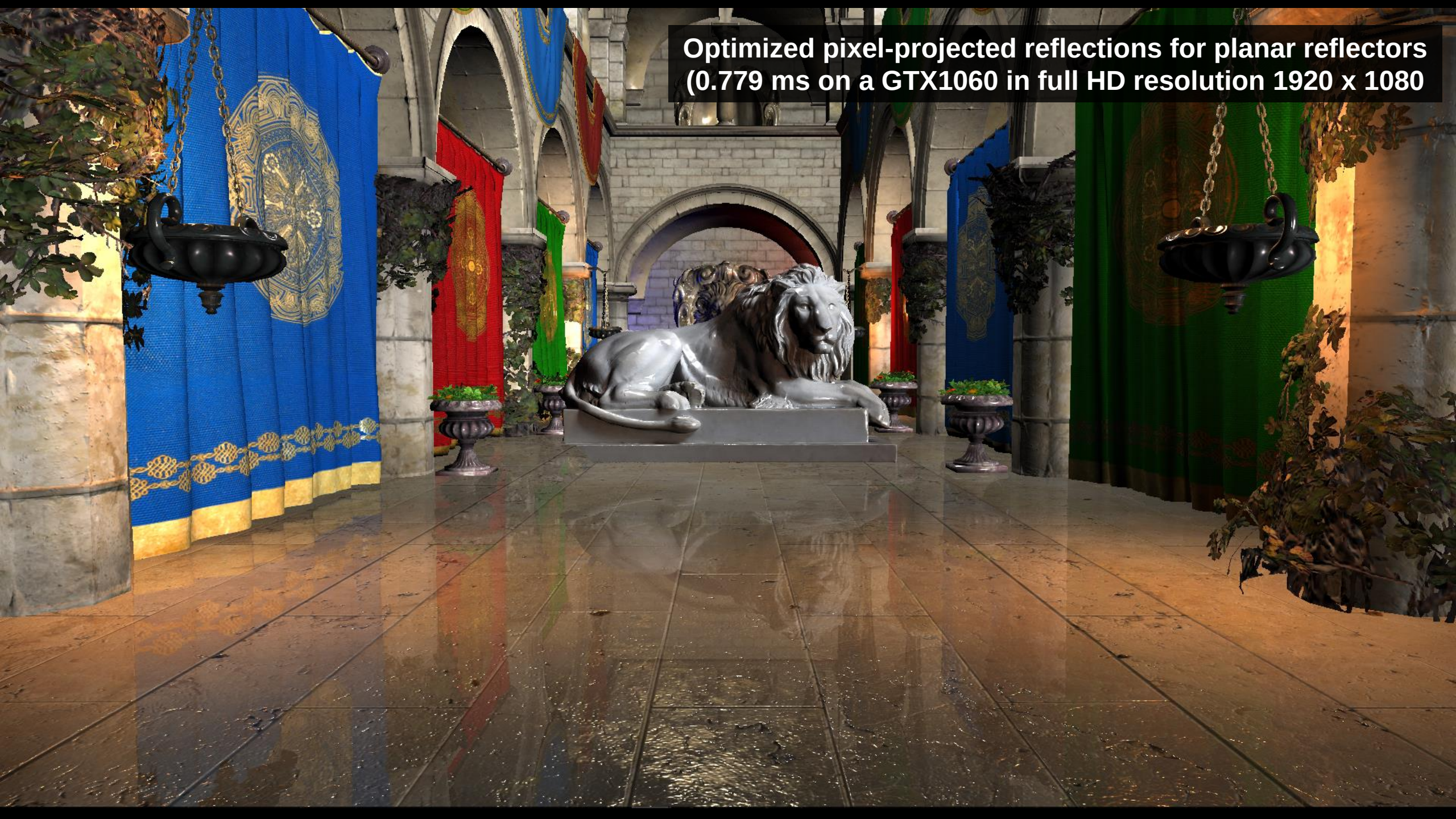
Overview

- Motivation
- Previous work
  - Ray tracing
  - Ray Marching
  - Screen Space Reflections
- Implementation
- Main topic
  - Pixel-projected Screen Space Reflections
  - Improvement
- Results
- Limitations
- Conclusion and Future work

Motivation

Motivation

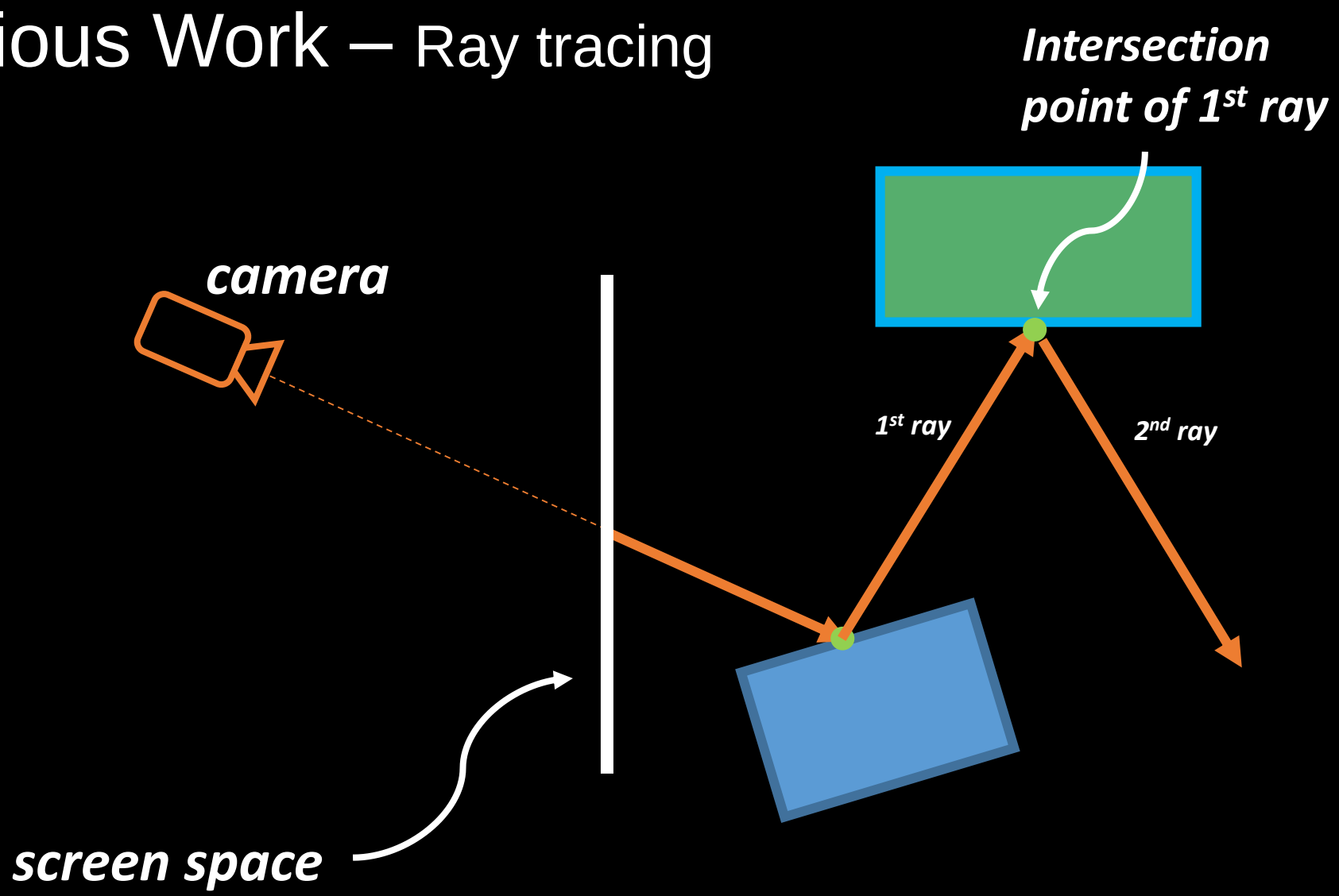

Optimized pixel-projected reflections
for planar reflectors
a.k.a. Pixel-projected reflections
Adam Cichocki
@multisampler
SIGGRAPH2017
ADVANCES IN REAL-TIME RENDERING
Vulkan


Optimized pixel-projected reflections for planar reflectors
(0.779 ms on a GTX1060 in full HD resolution 1920 x 1080

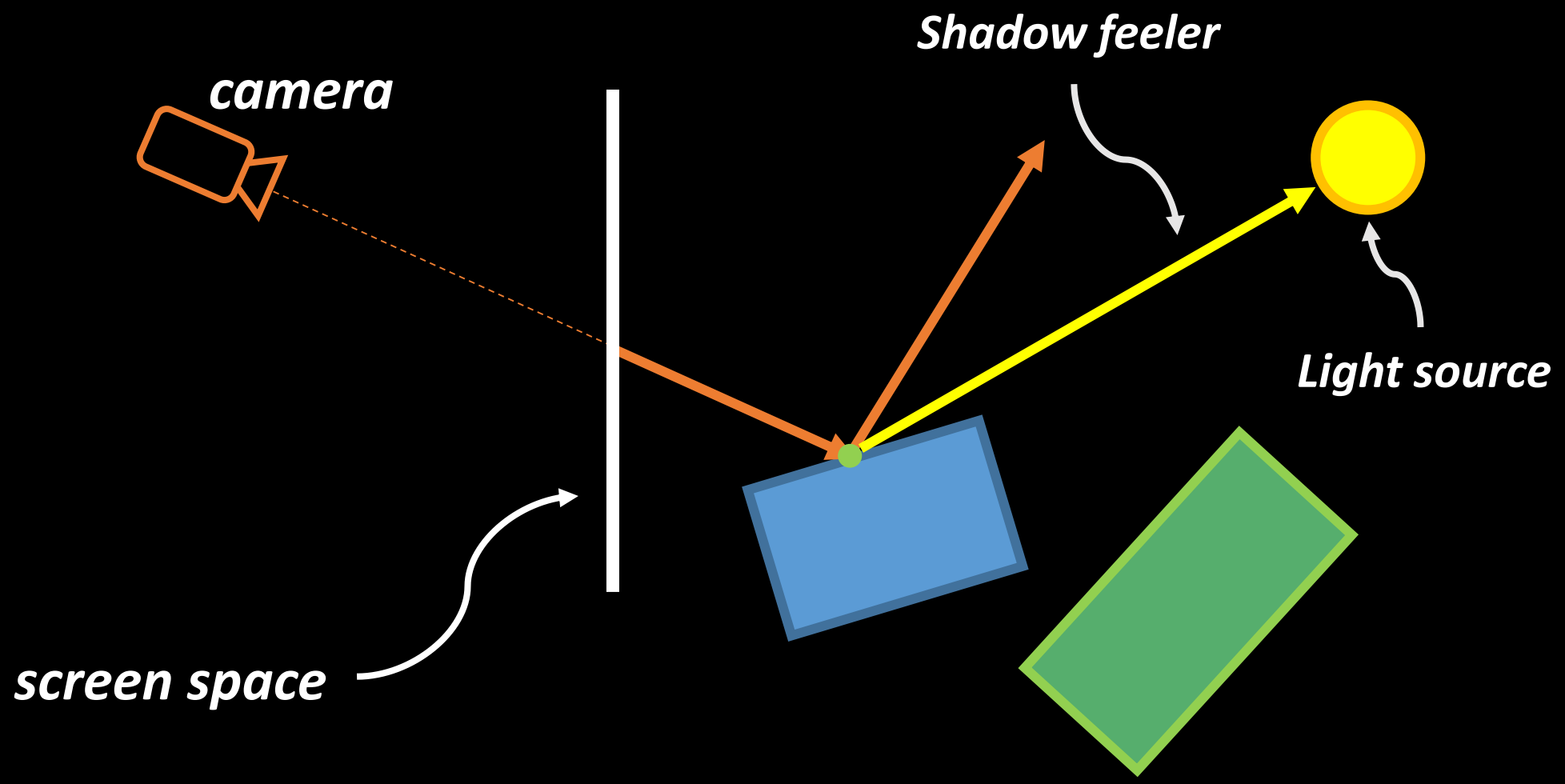
Previous Work

- Ray tracing
- Ray marching
- Screen Space Reflections

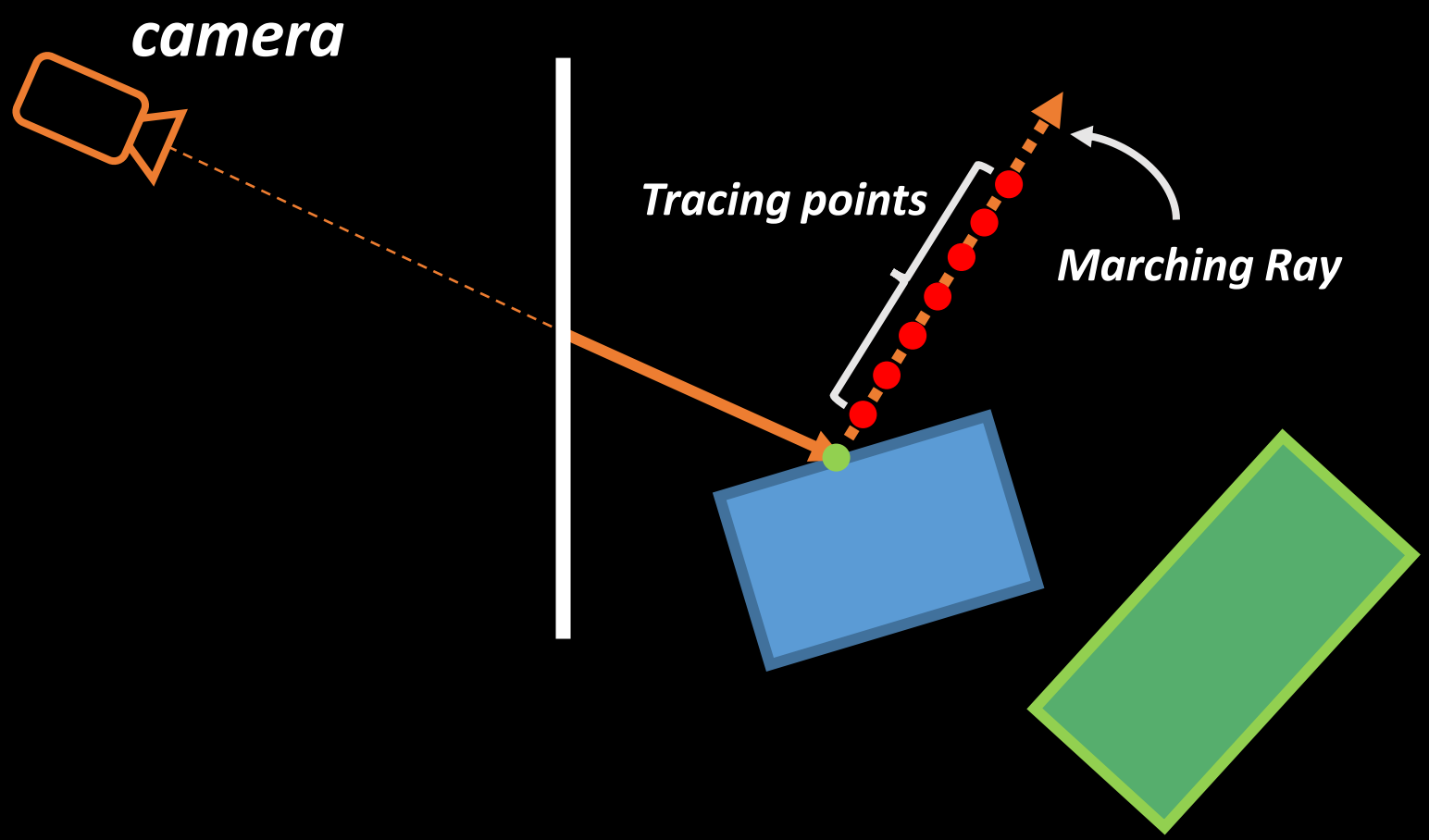
Previous Work – Ray tracing



Previous Work – Ray tracing



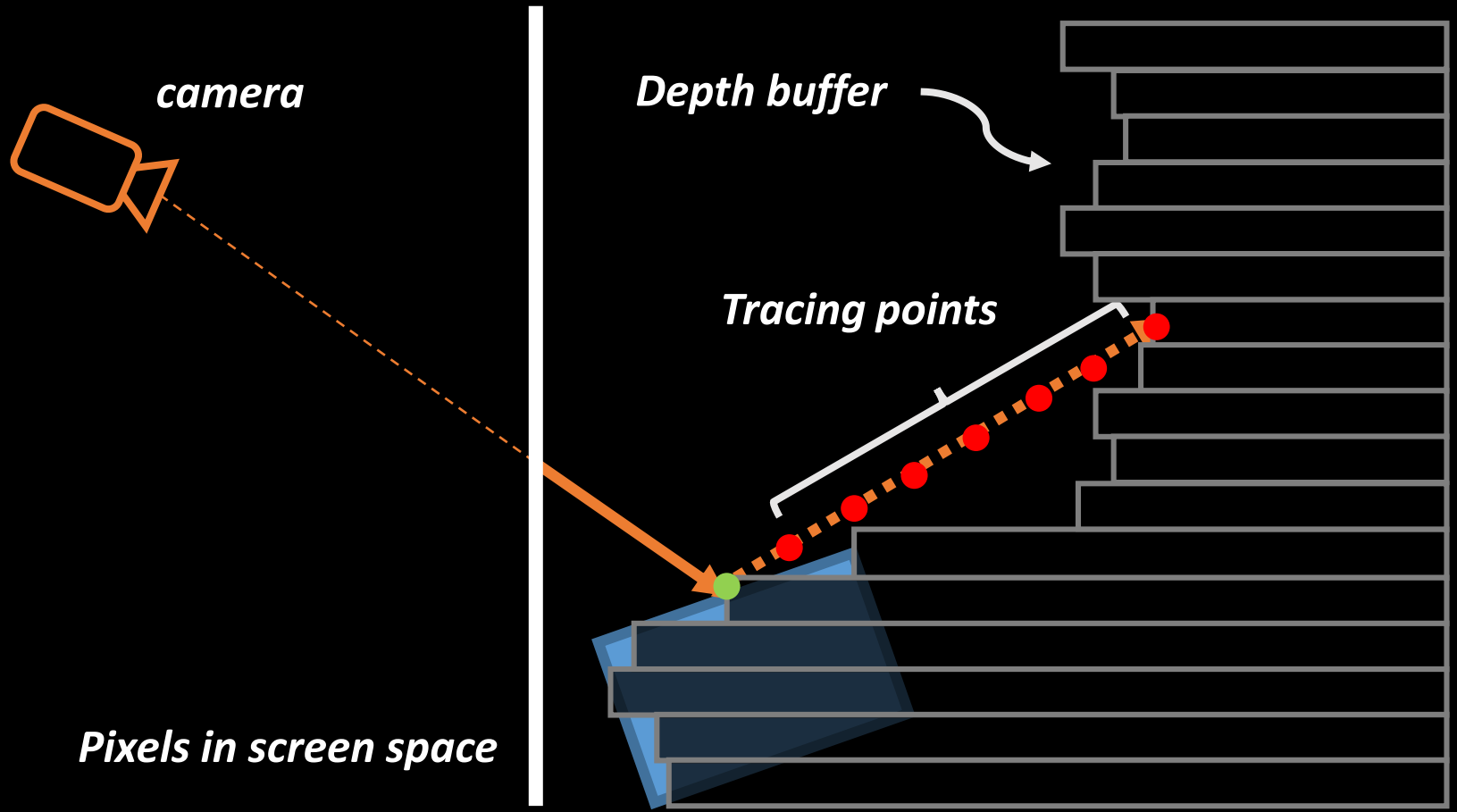
Previous Work – Ray marching



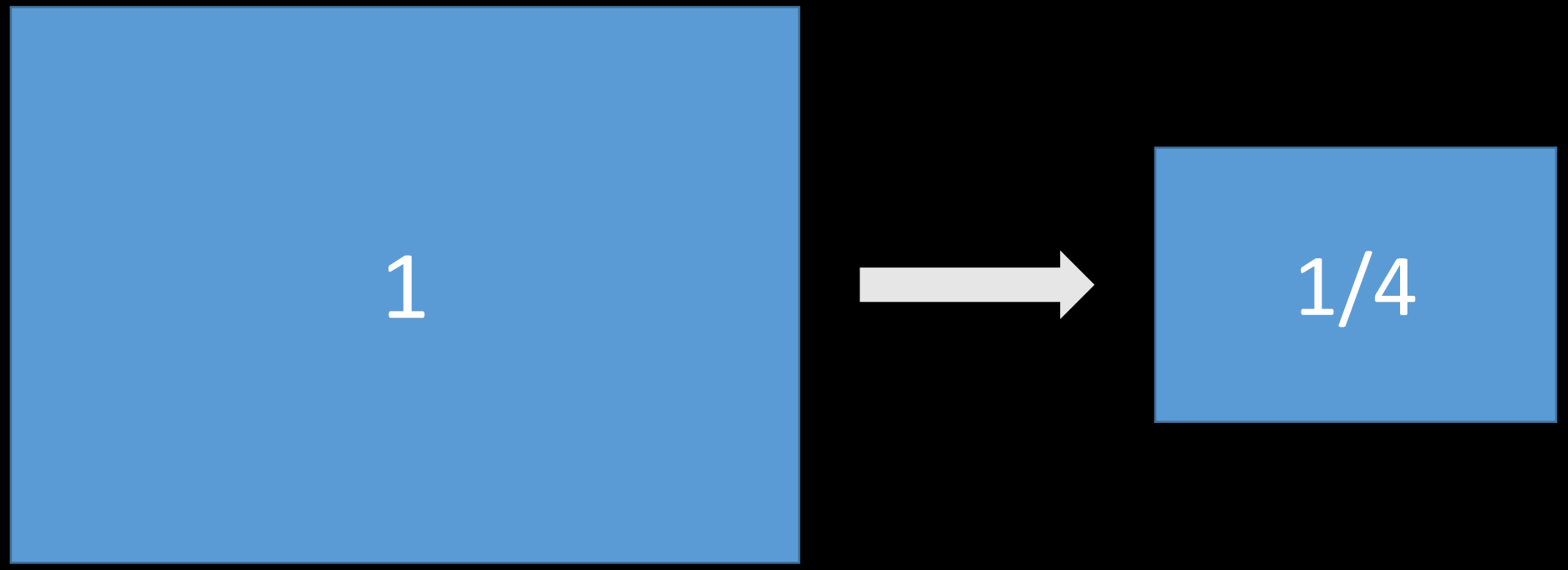
Previous Work – Screen Space Reflections

- Brute Force Screen Space Reflections
- Hi-Z Screen Space Reflections
- Tile Classification

Previous Work – Brute Force Screen Space Reflections

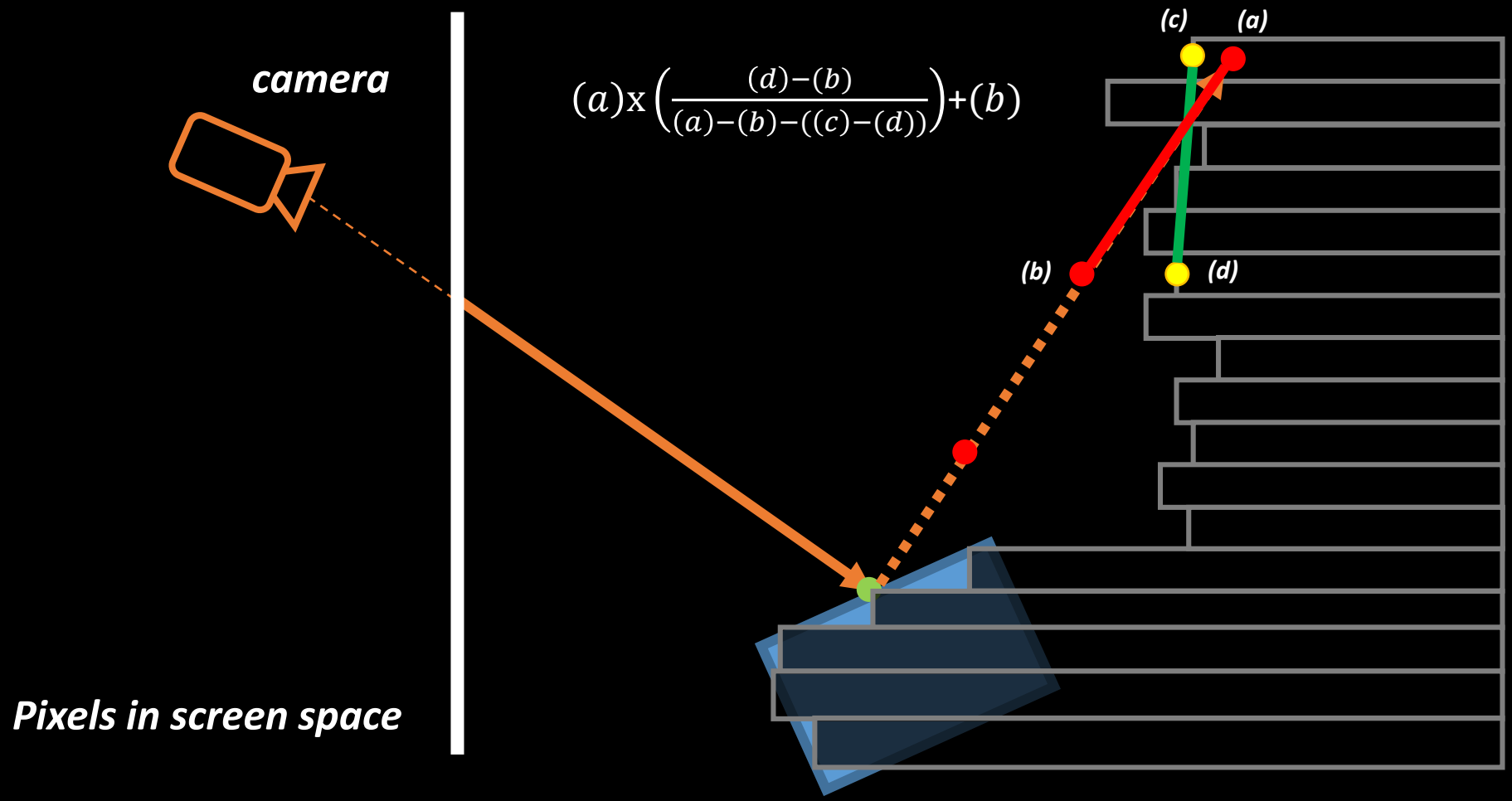


Previous Work – Improved Brute Force SSR [Val14]

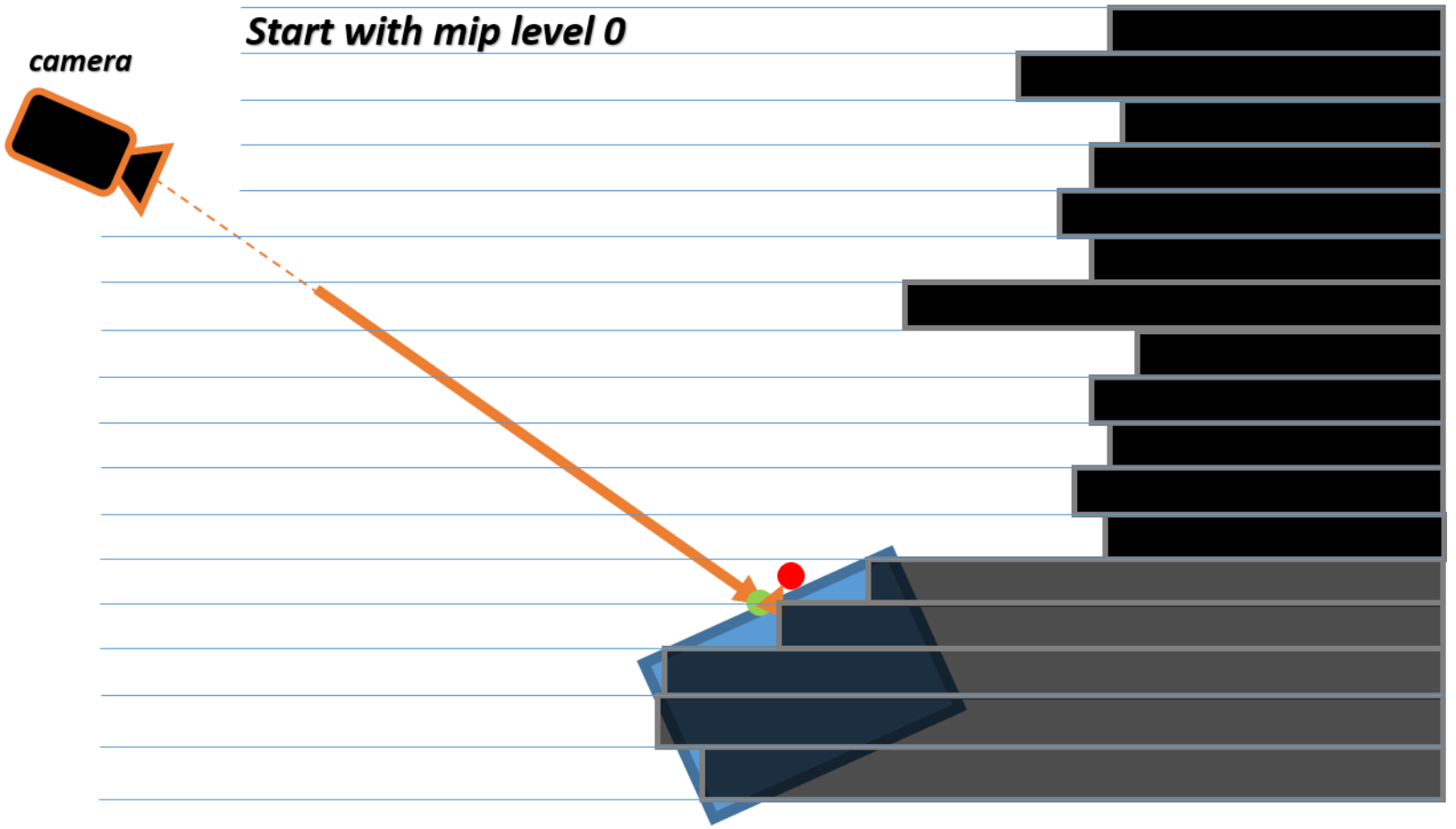


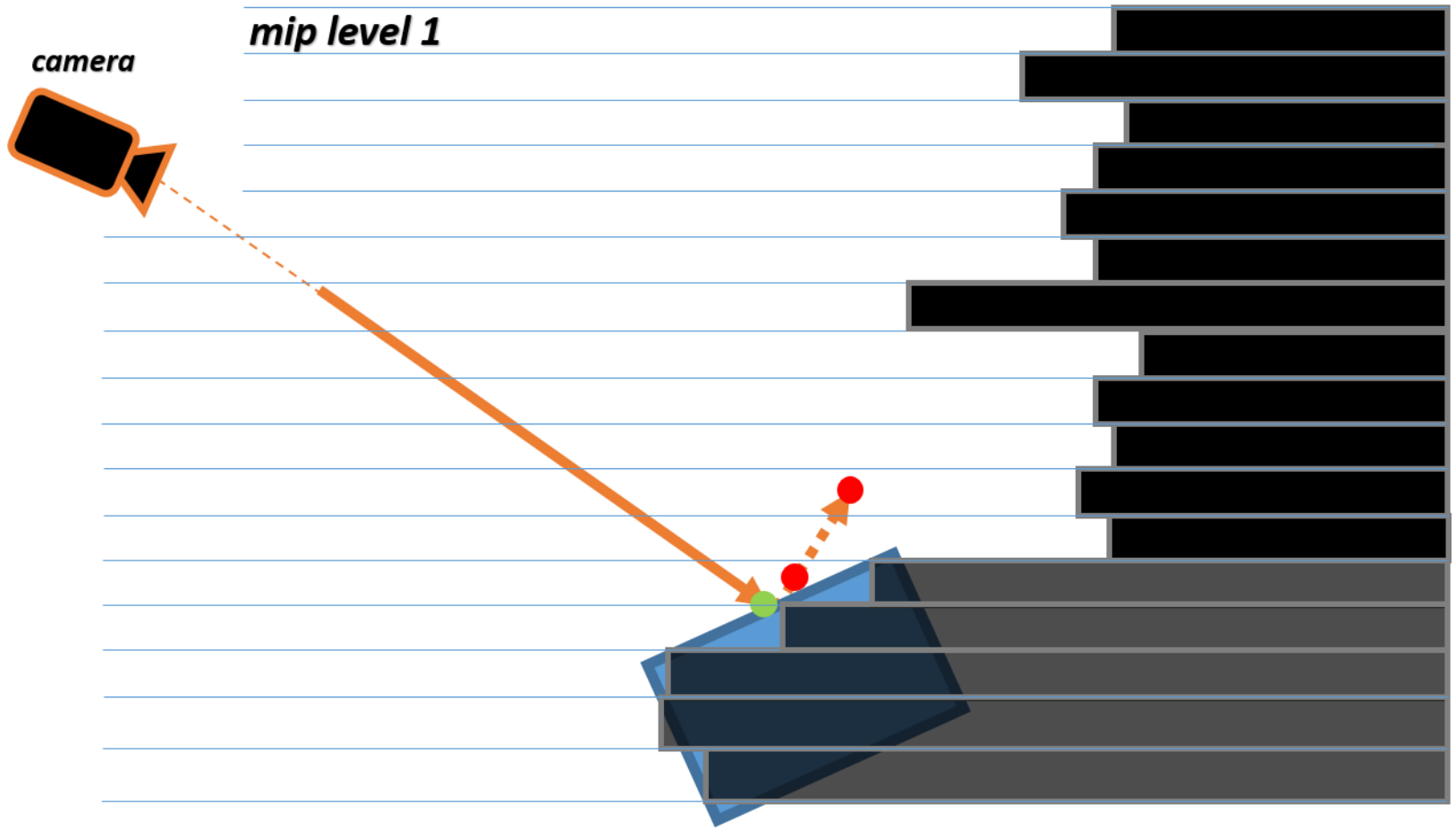
¼ Downsampling

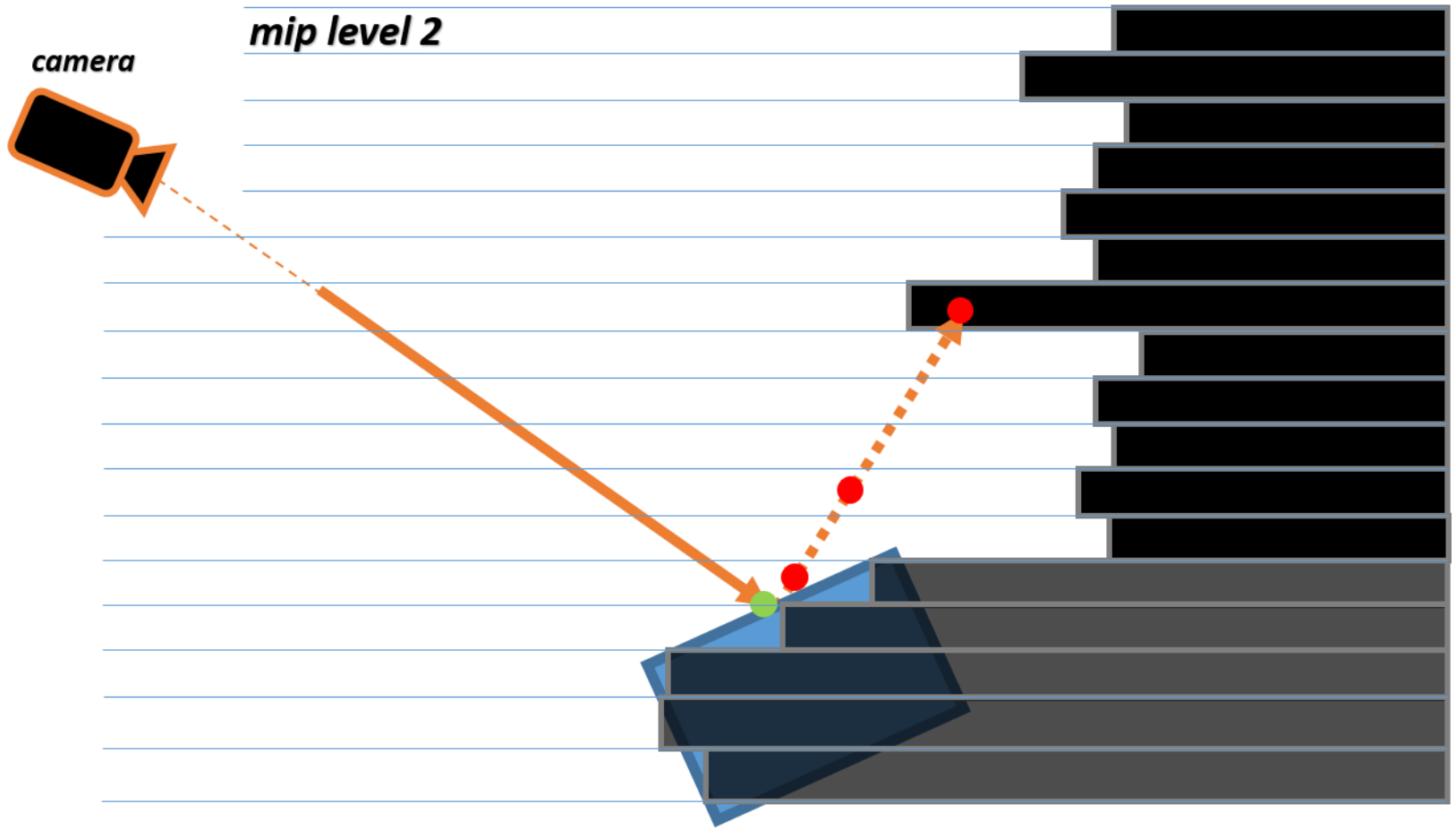
Previous Work – Improved Brute Force SSR [Val14]

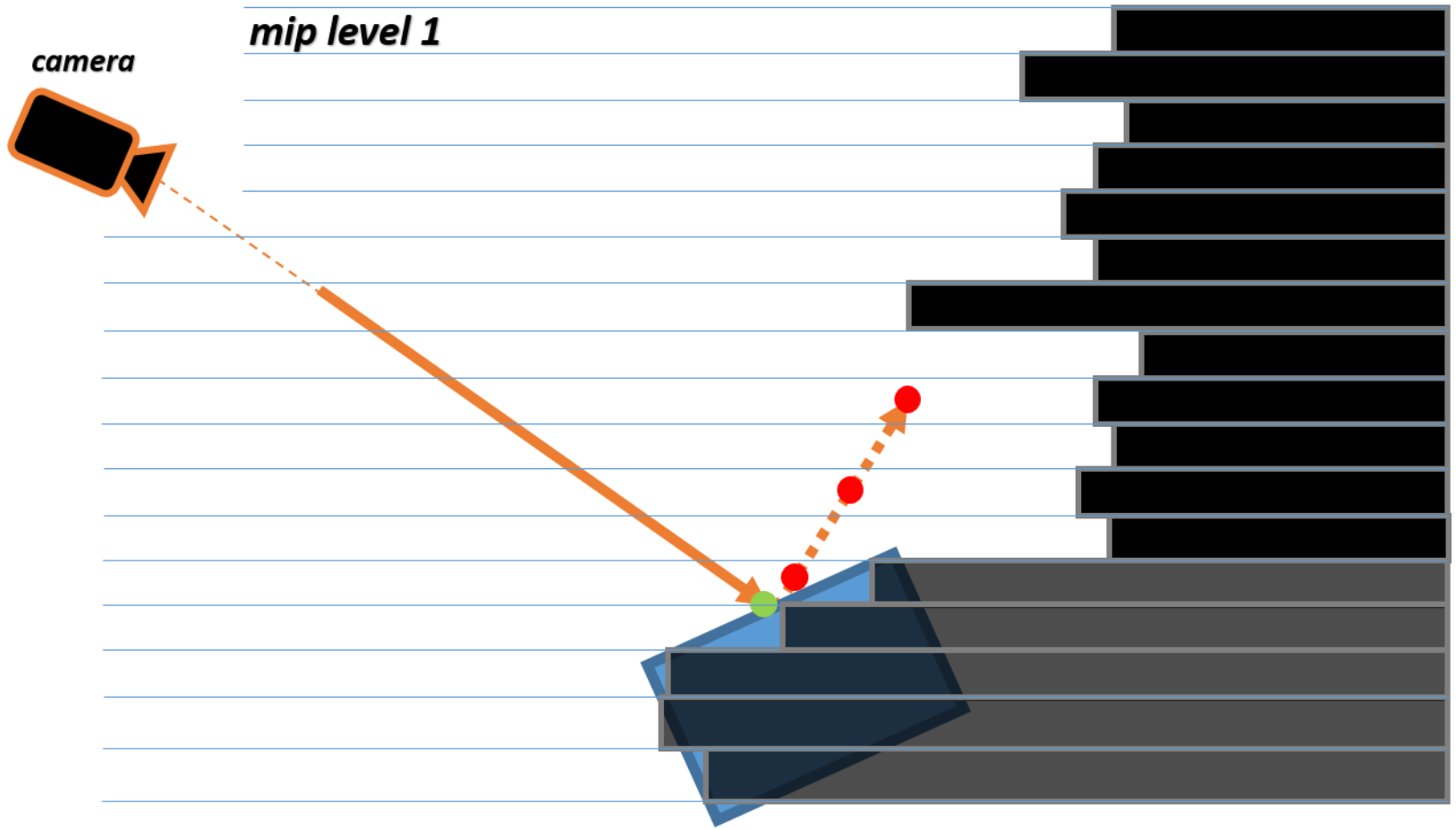


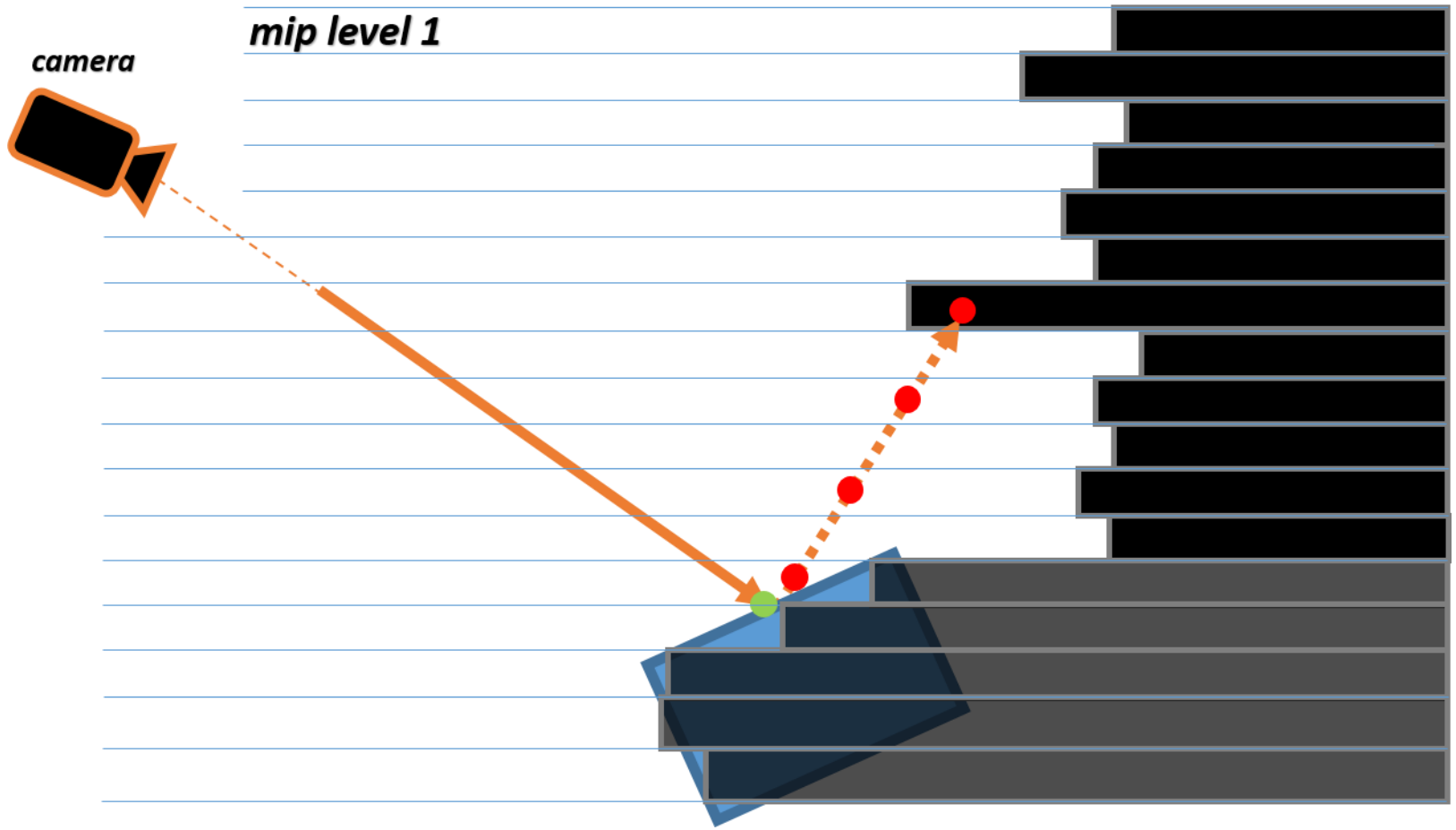
Previous Work – Hi-Z Screen Space Reflections [Uludag14]

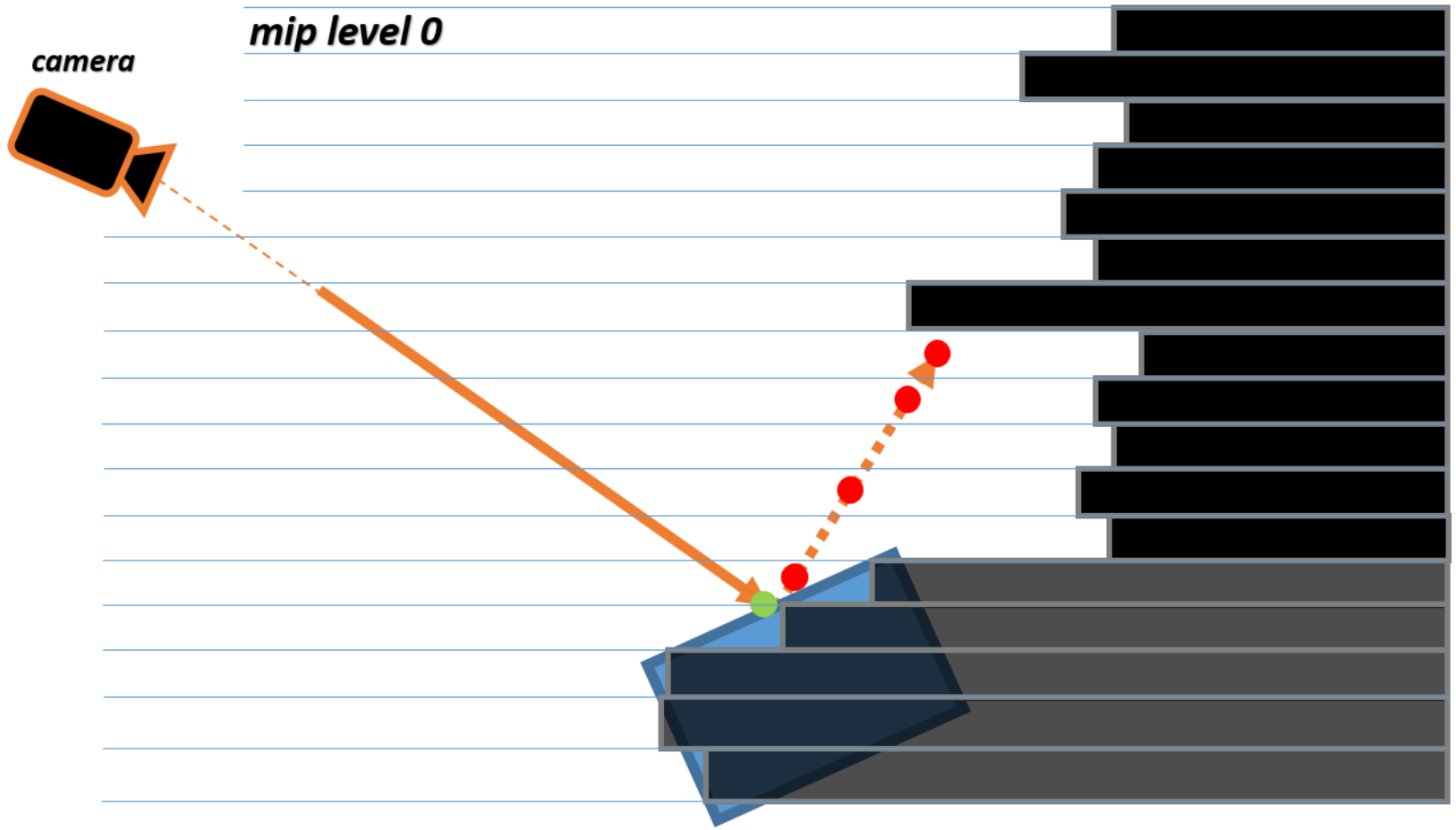


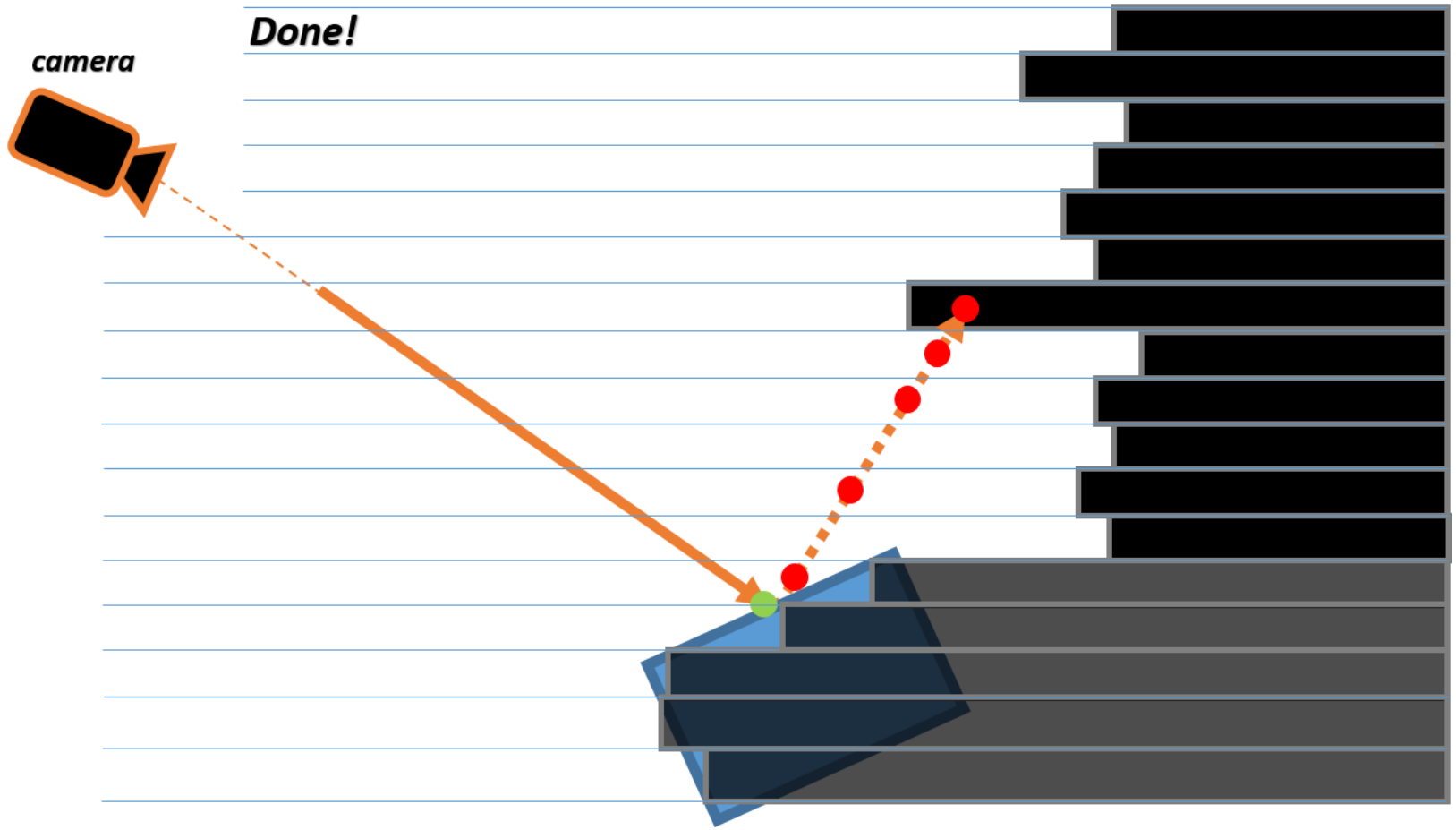

mip level 1
camera


mip level 2
camera

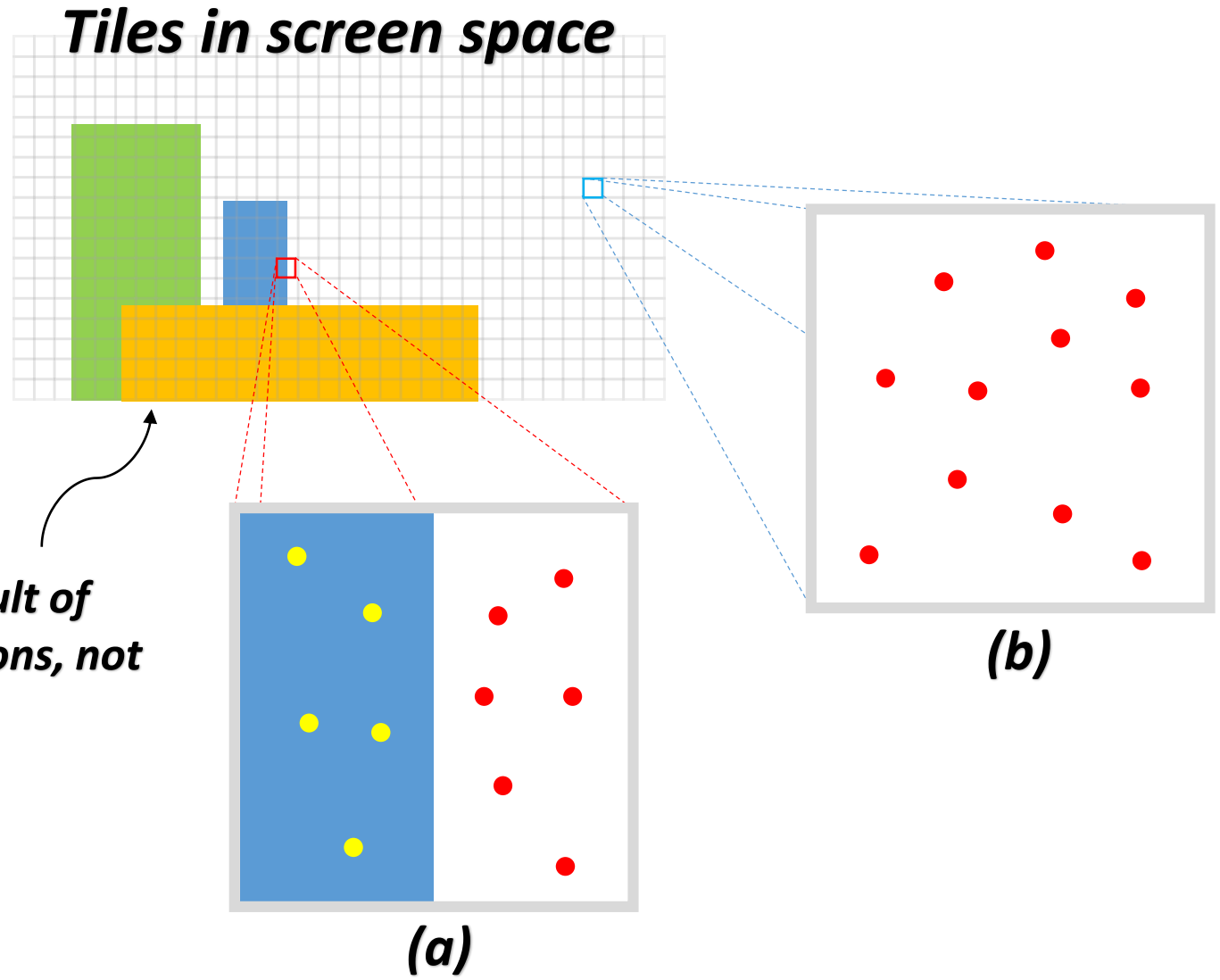

mip level 1
camera


mip level 1
camera


mip level 0
camera


Done!
camera

Previous Work – Tile Classification

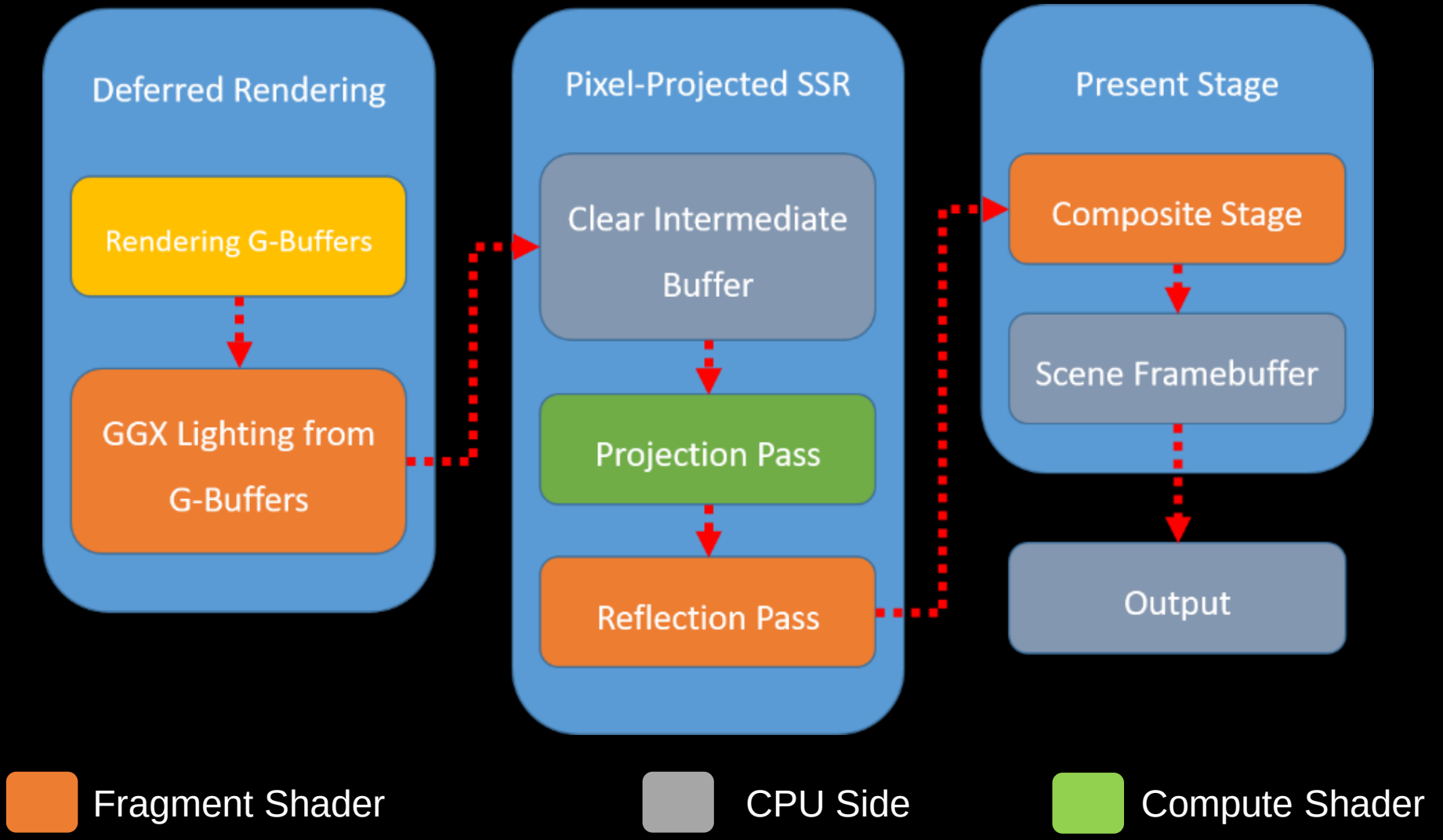


Implementation

Multi-Threading

Memory management

Implementation – Pipeline



Main topic

- Pixel-projected Screen Space Reflections
- Improvement

Pixel-projected SSR – algorithm overview

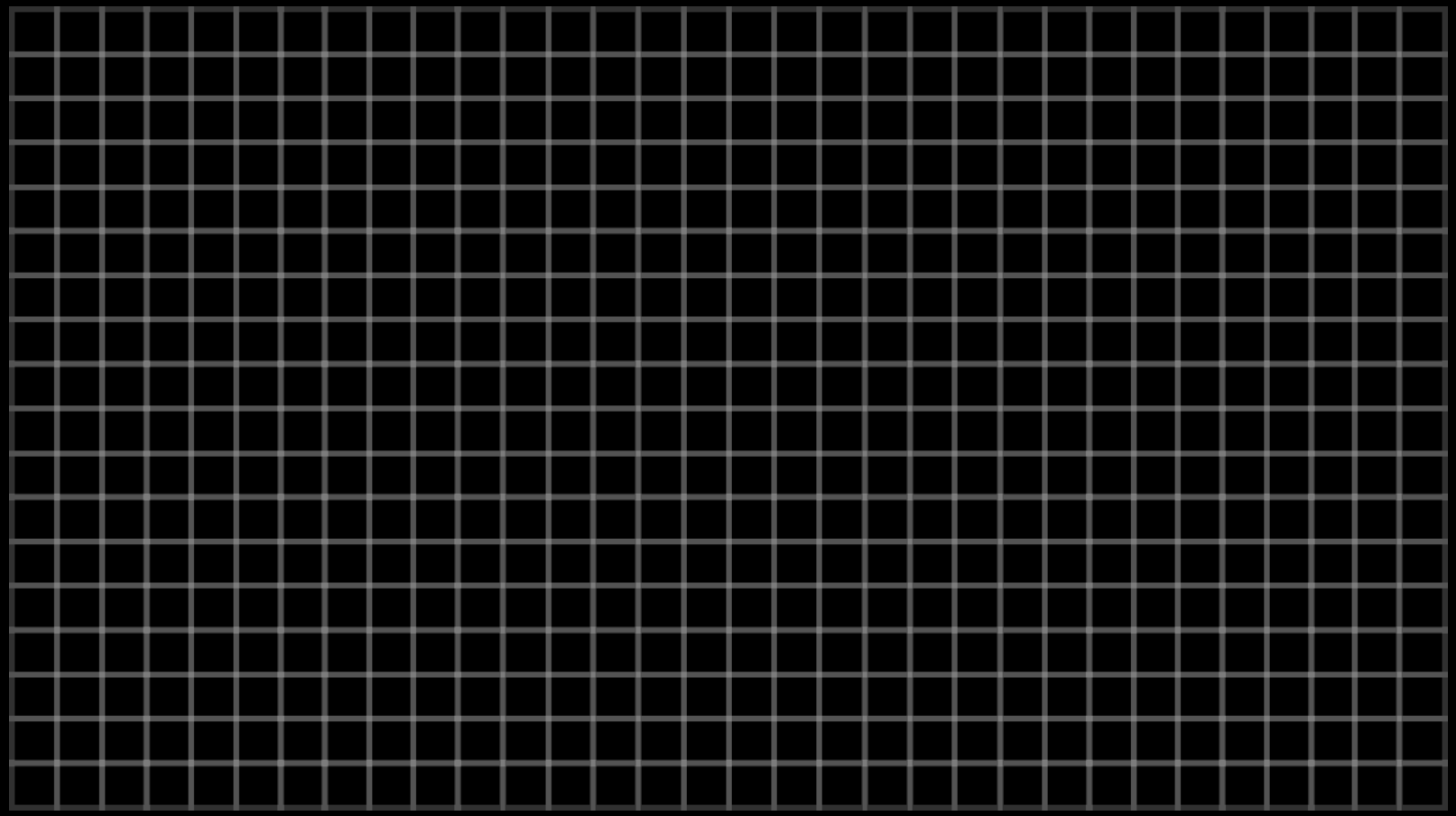
Projection Pass

- Set rectangle shape reflectors.
- For every pixel, find the reflectors that reflect it.
- Project pixels on those reflectors.
- Encode pixel-data and write to **Intermediate buffer**.

Reflection Pass

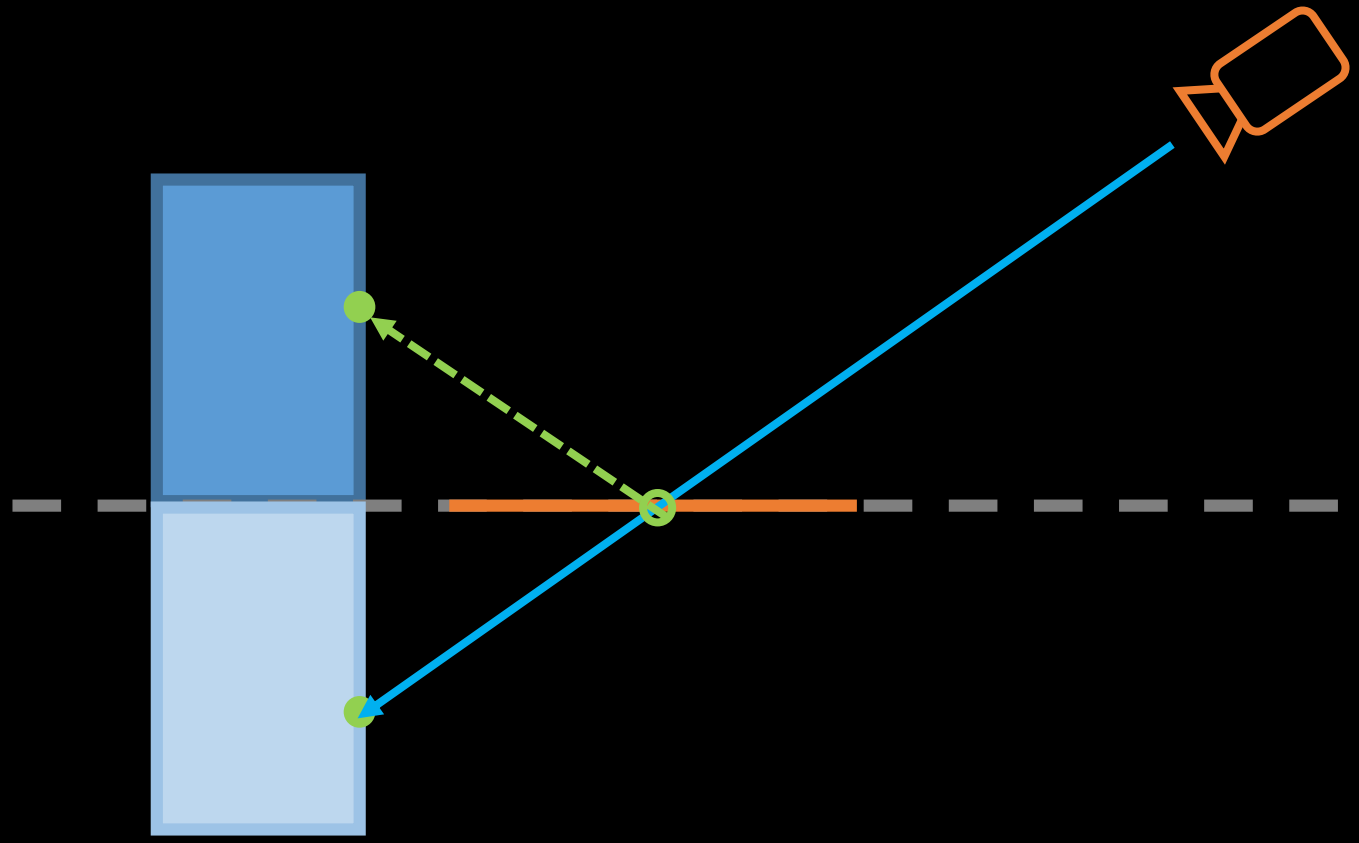
- Read pixel-data from **Intermediate buffer.**
- Decode pixel-data and obtain reflected color
- Write color to reflection buffer

Pixel-projected SSR – Previous stage

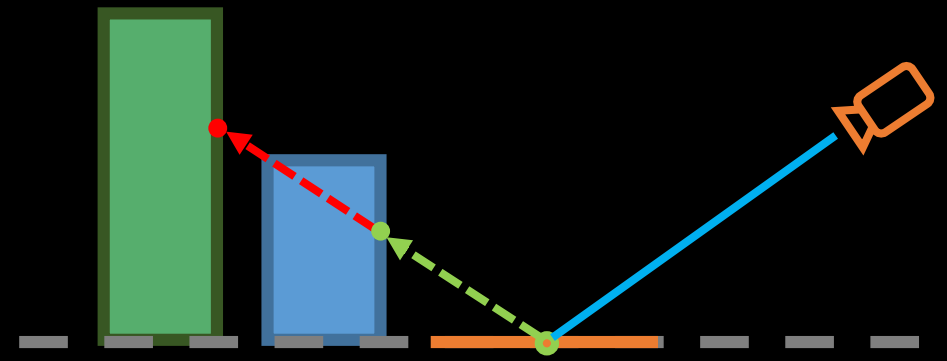


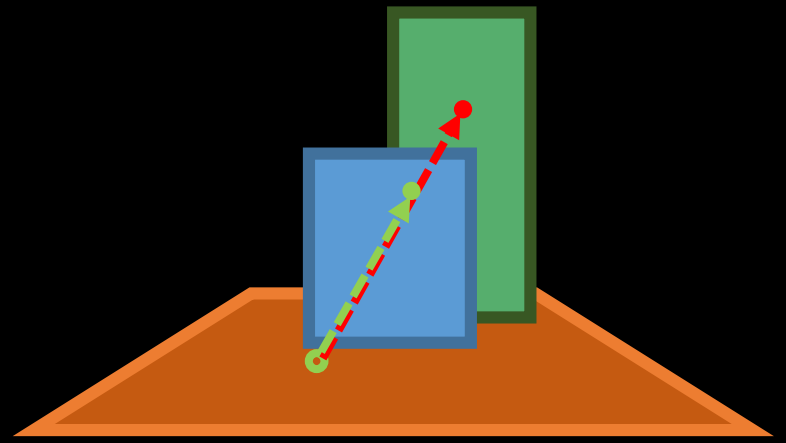
Intermediate Buffer

Pixel-projected SSR – Projection Pass



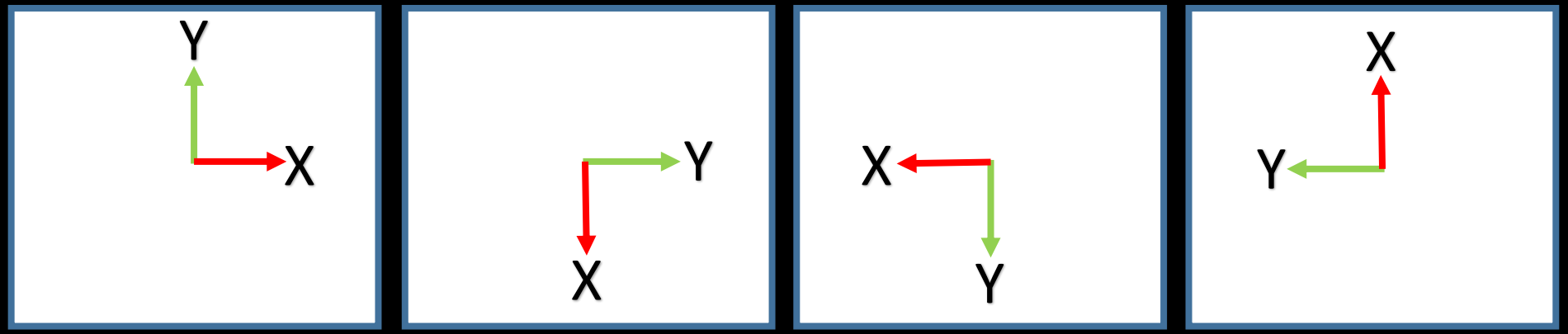
Pixel-projected SSR – Projection Pass



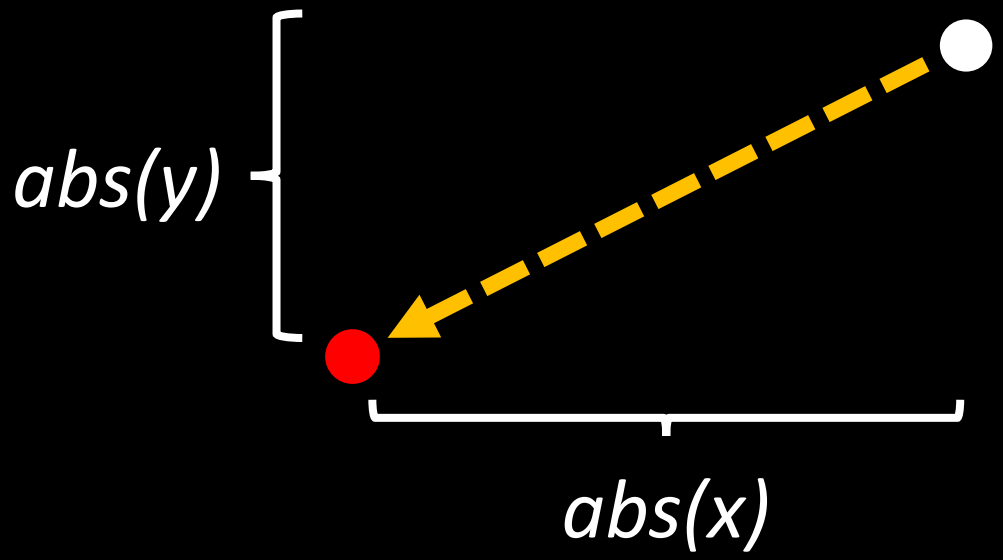


Pixel-projected SSR – Projection Pass

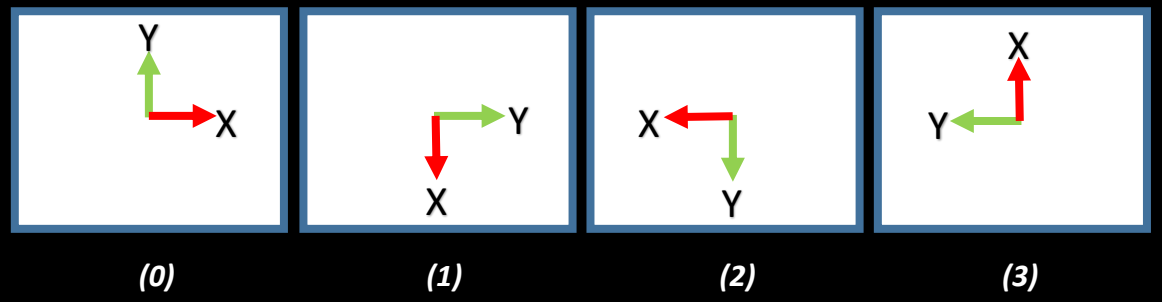
uint atomicMin()

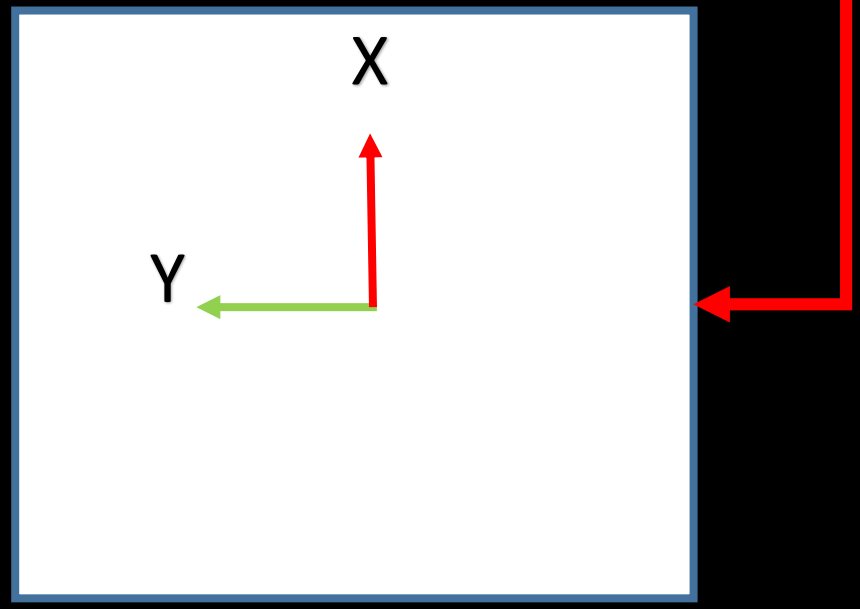


Pixel-projected SSR – Projection Pass

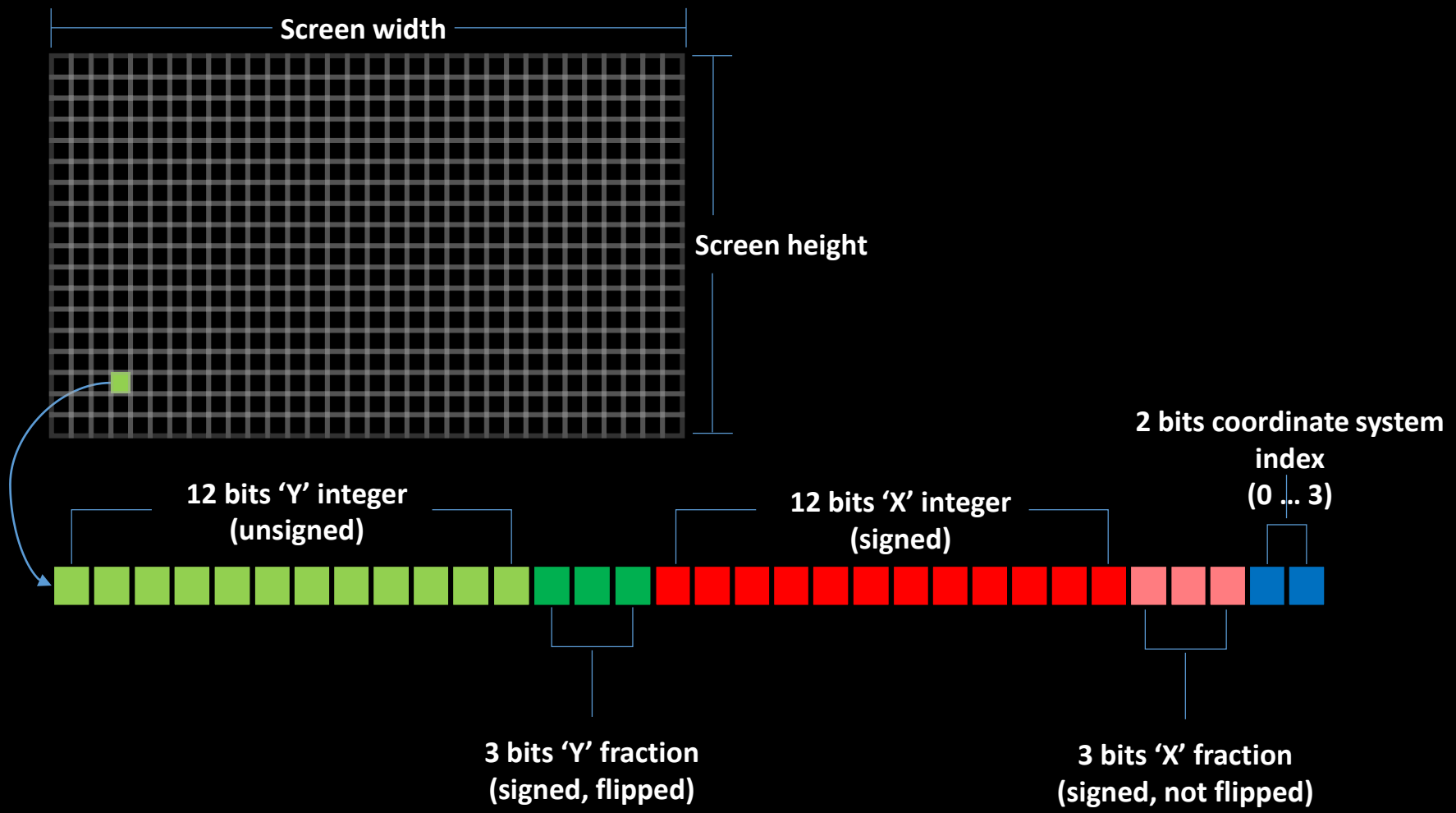


abs(x) > abs(y)





Pixel-projected SSR – Projection Pass



Pixel-projected SSR – Projection Pass

SSBO *Ex) 1920 x 1080 = 2,073,600*

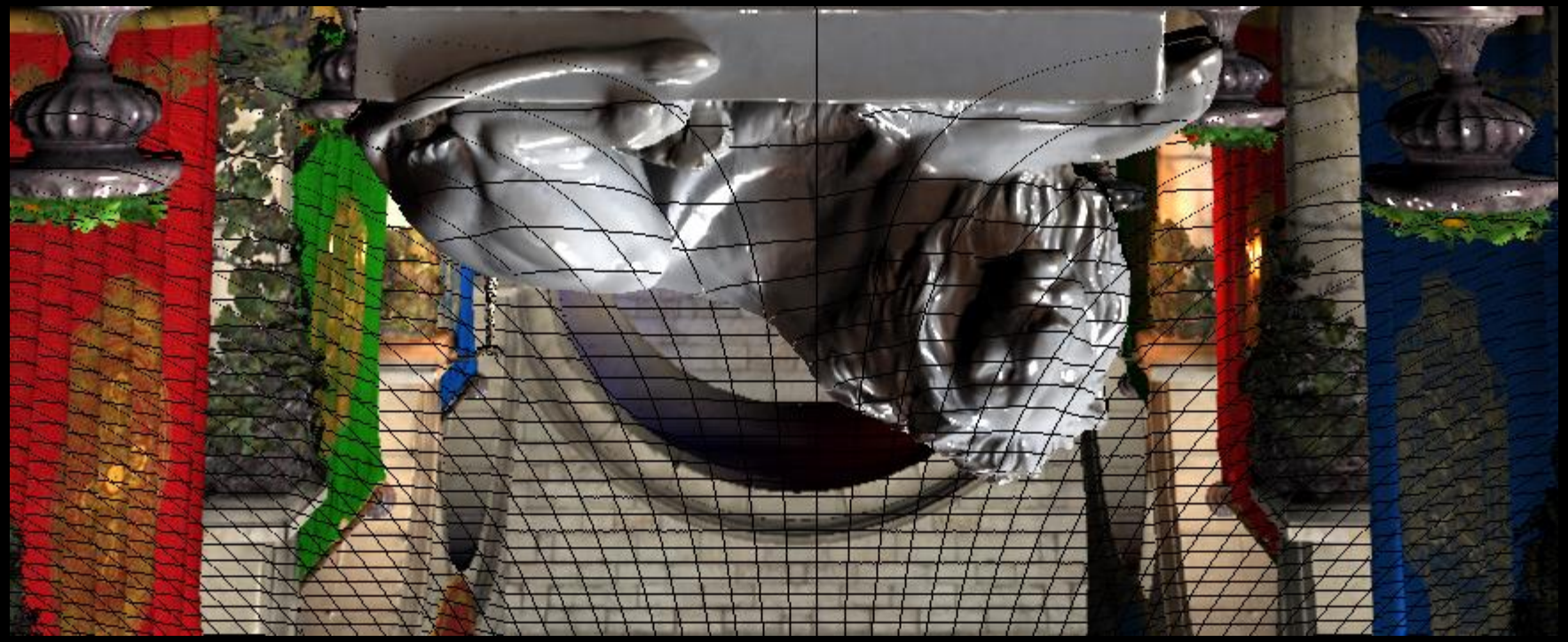
Or

Storage Image ?

Pixel-projected SSR – Reflection Pass

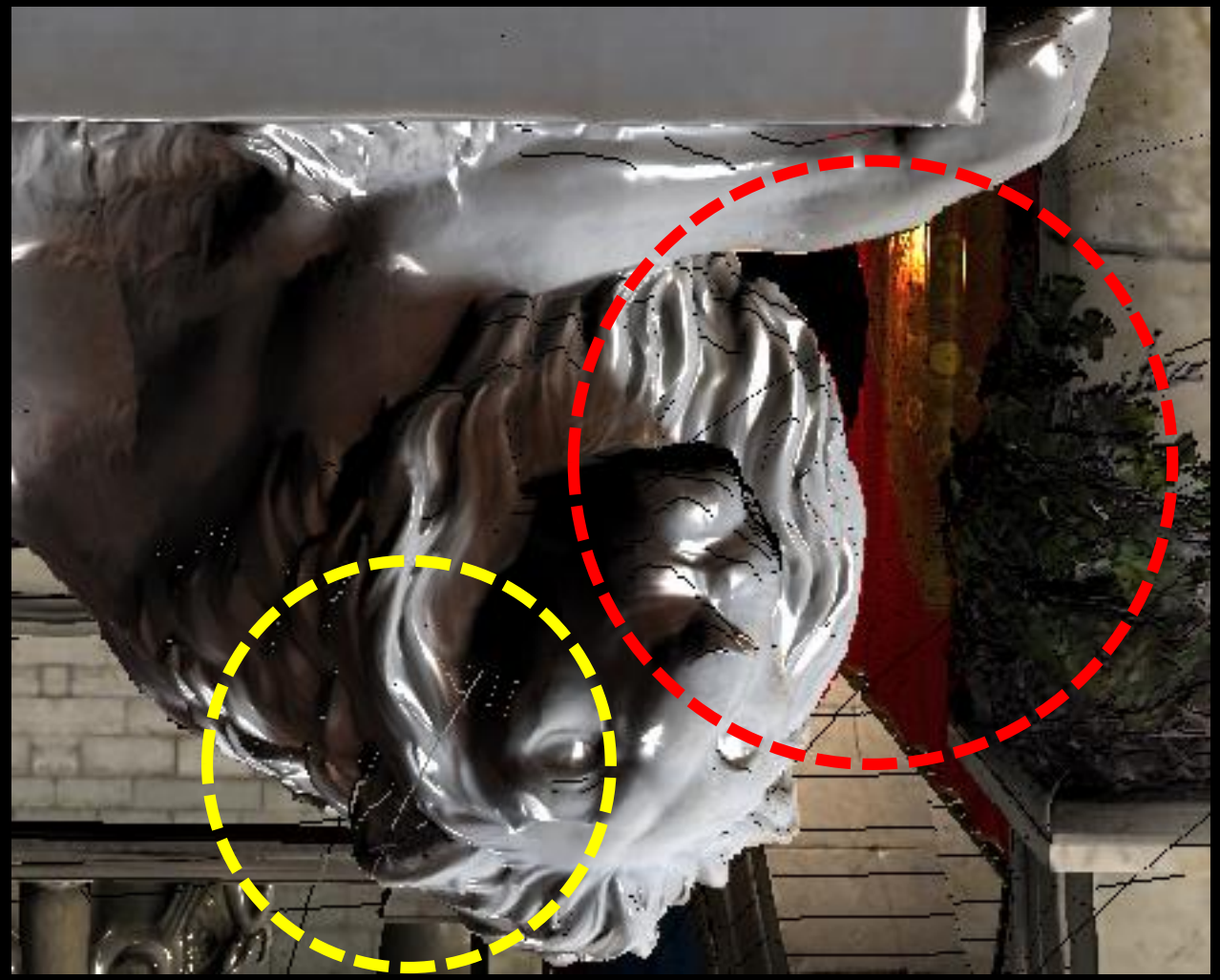
- Read the Intermediate buffer
- Decode the data to retrieve the offsets
- Obtain reflected color from the scene image

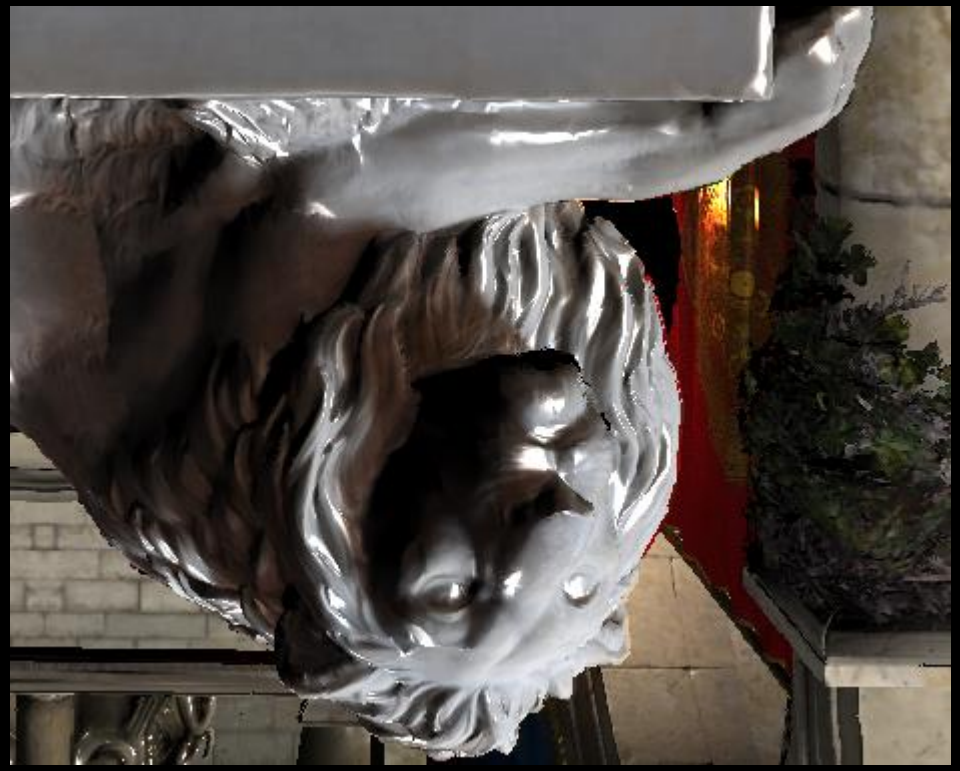
Pixel-projected SSR – Reflection Pass



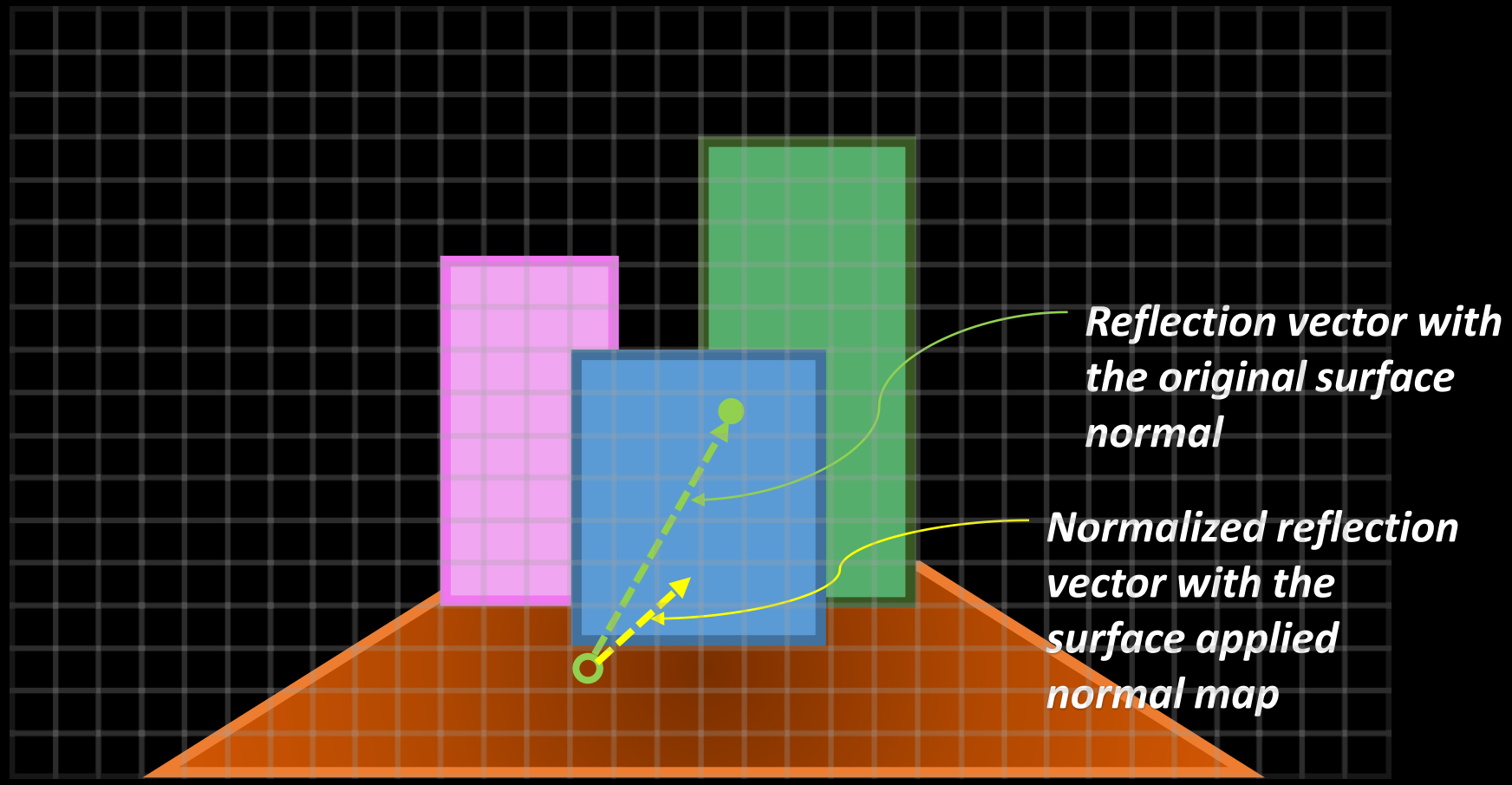
Need holes patching

Pixel-projected SSR – Reflection Pass

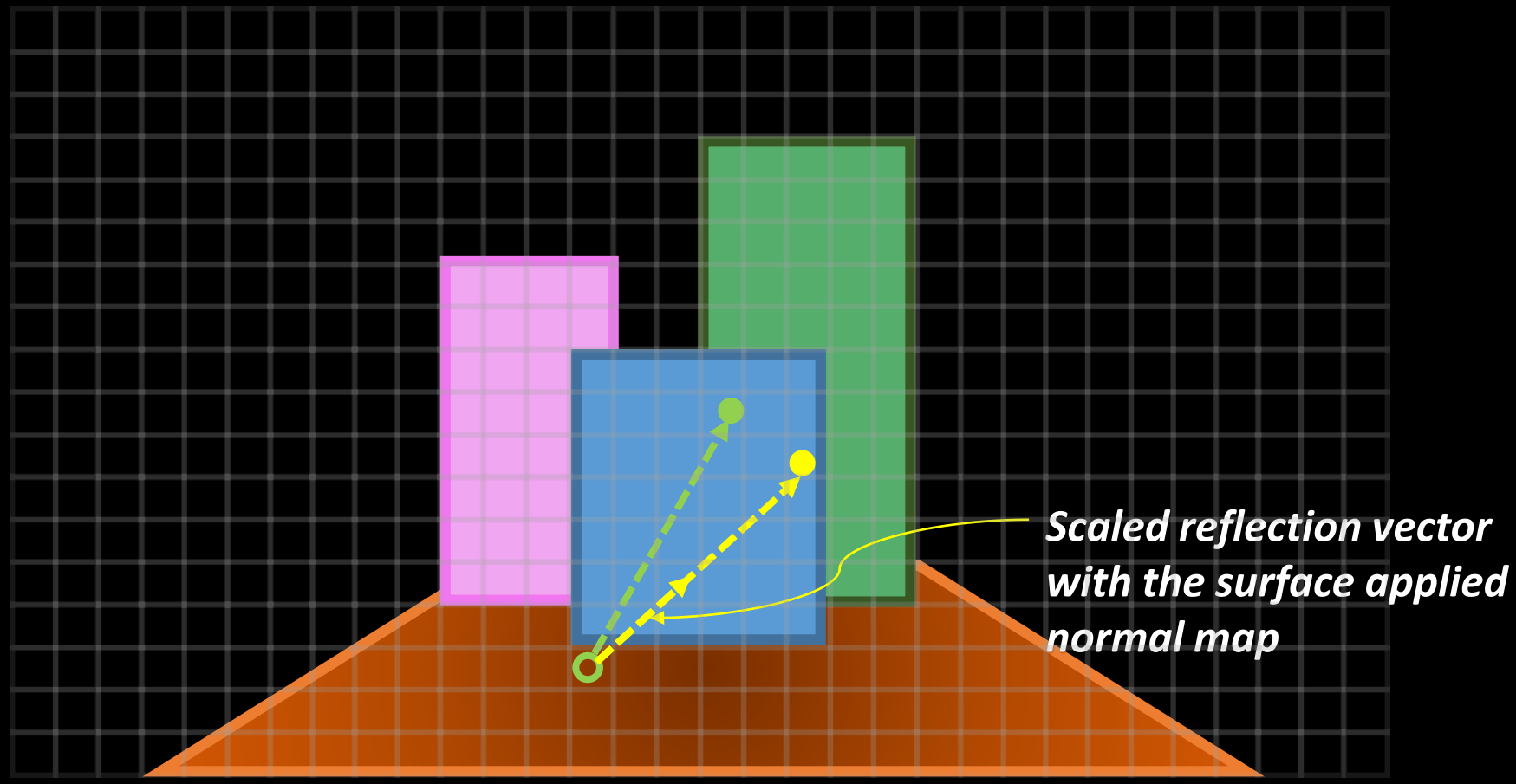




Pixel-projected SSR – Apply normal map

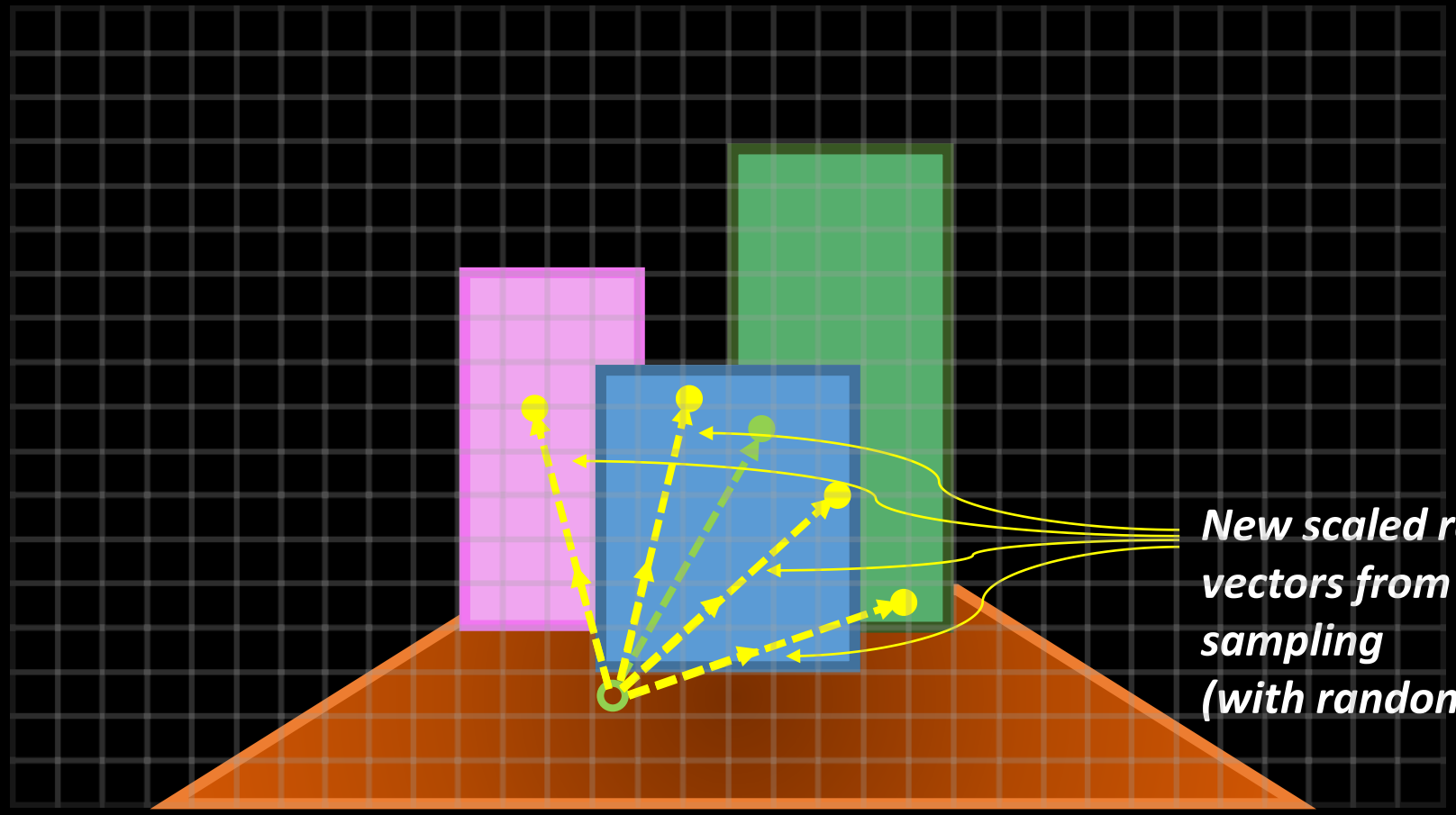


Pixel-projected SSR – Apply normal map




With normal map

Pixel-projected SSR – Apply roughness



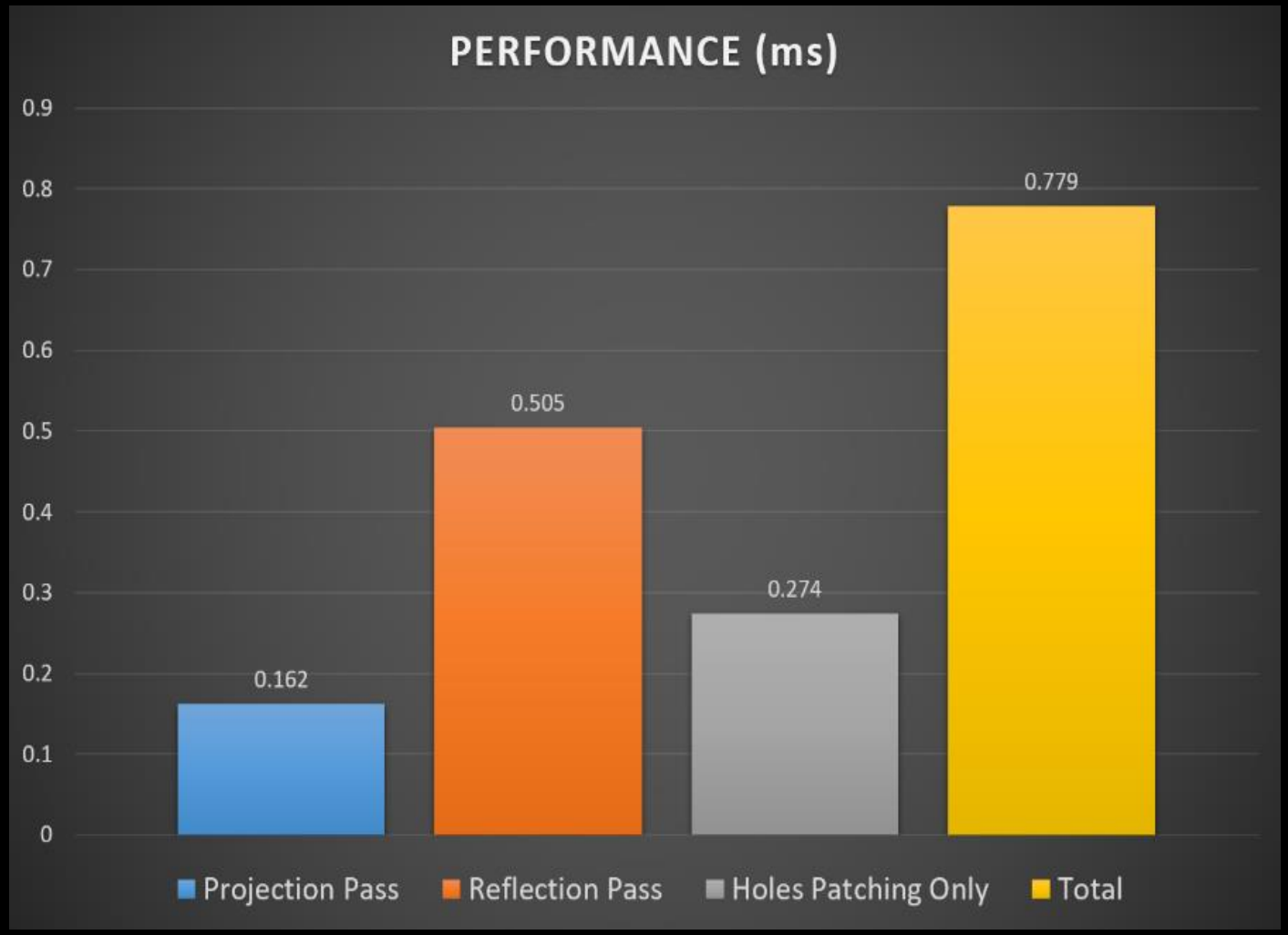
New scaled reflection vectors from importance sampling (with random seed per pixel)


Roughness 0.5

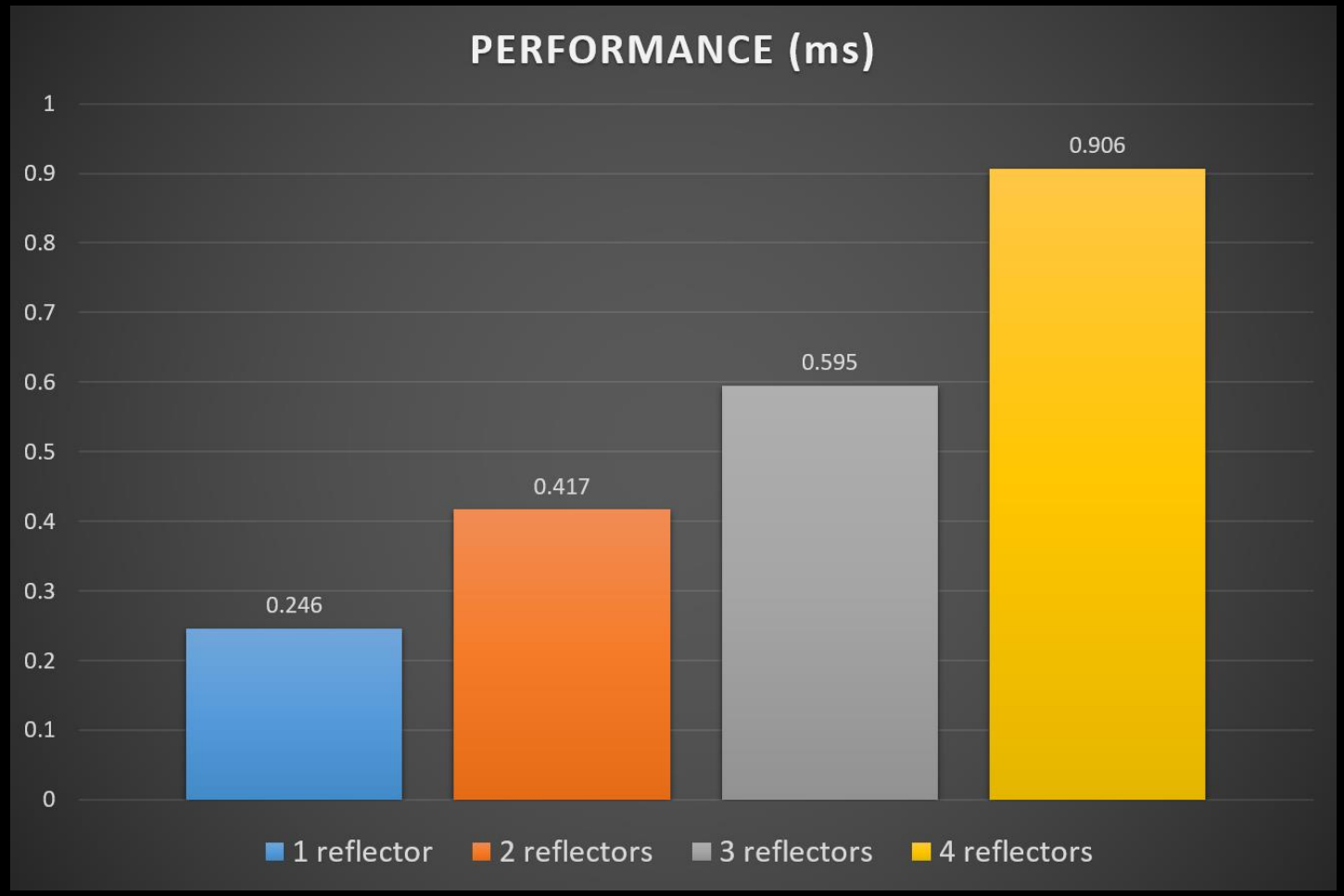
Results

- NVIDIA GTX 1060
  - ✓ Core Speed: 1404 MHz
  - ✓ Memory Speed: 2002 MHz
  - ✓ Memory Bus Width: 192 Bit
- Intel Core i7-6700HQ Quad Core Processor (2.6 GHz)
- Full HD resolution 1920 x 1080
- Tested this approach on a complex scene which uses Crytek’s Sponza and a Lion modeling designed by Geoffrey Marchal

Performance of each stage



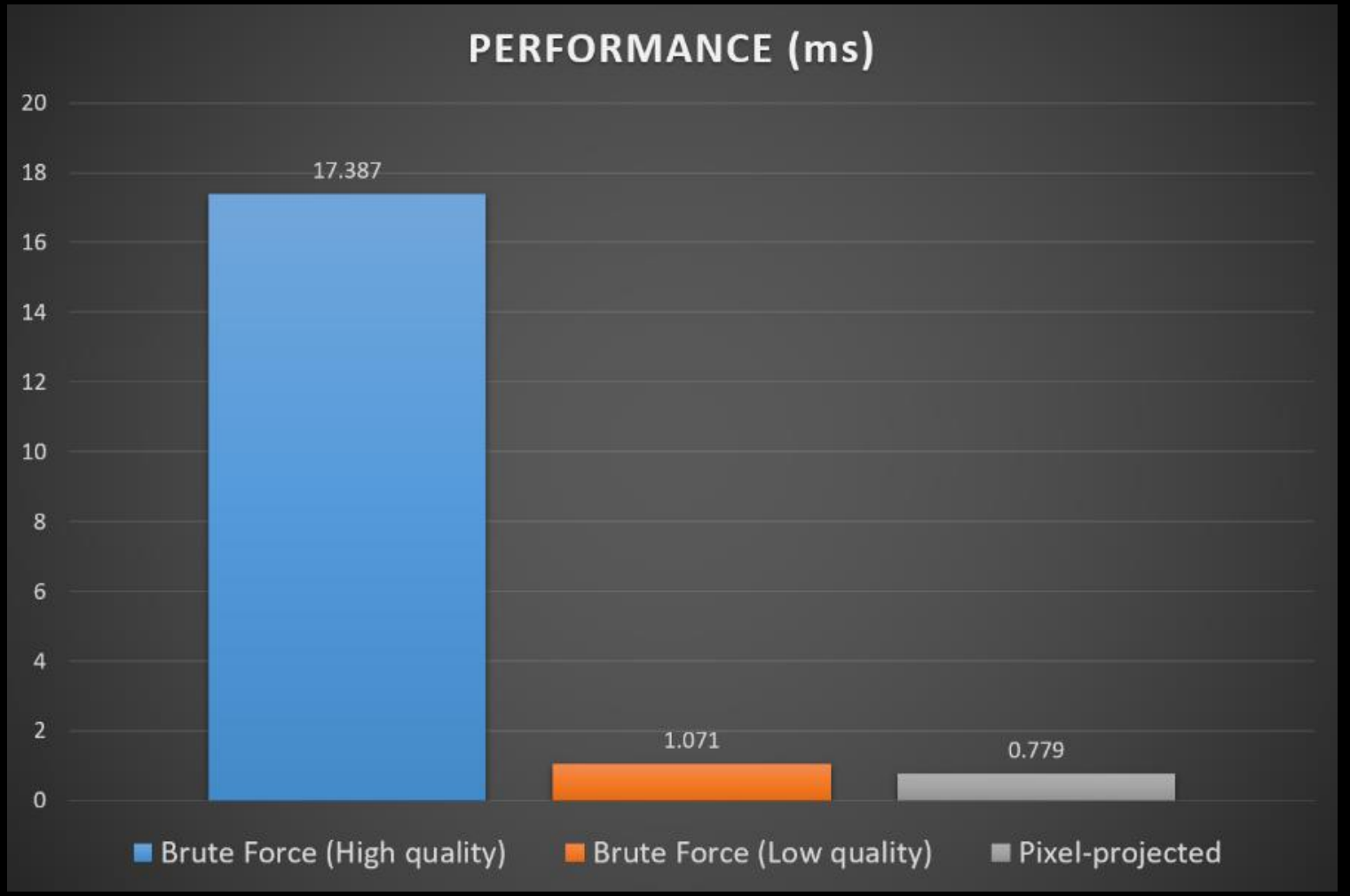
Comparison with different number of reflectors



Using small size of reflectors

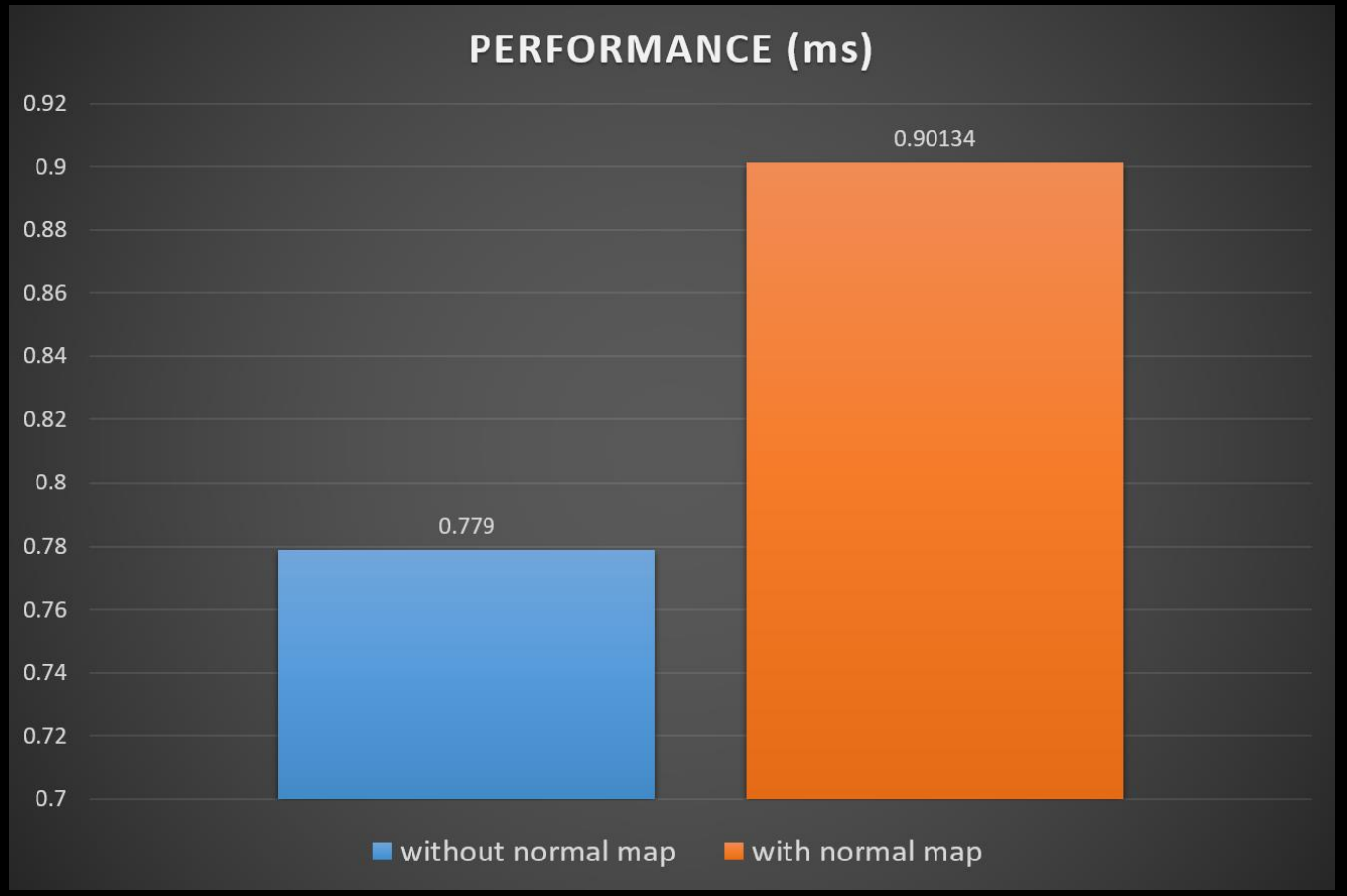
Brute force VS Pixel-projected SSR

Comparison with Brute Force Screen Space Reflections

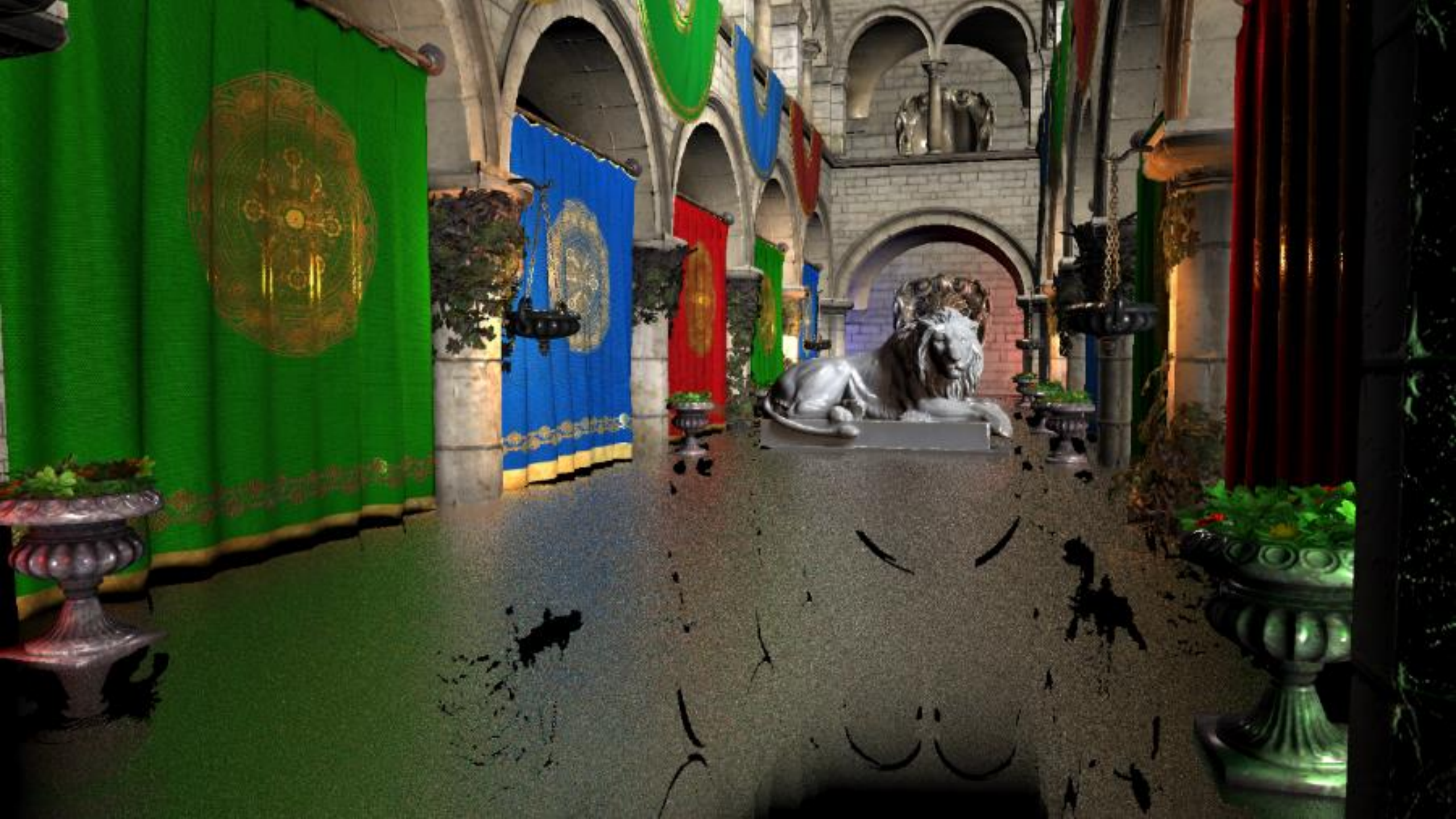


Comparison between the results With / Without Normal map

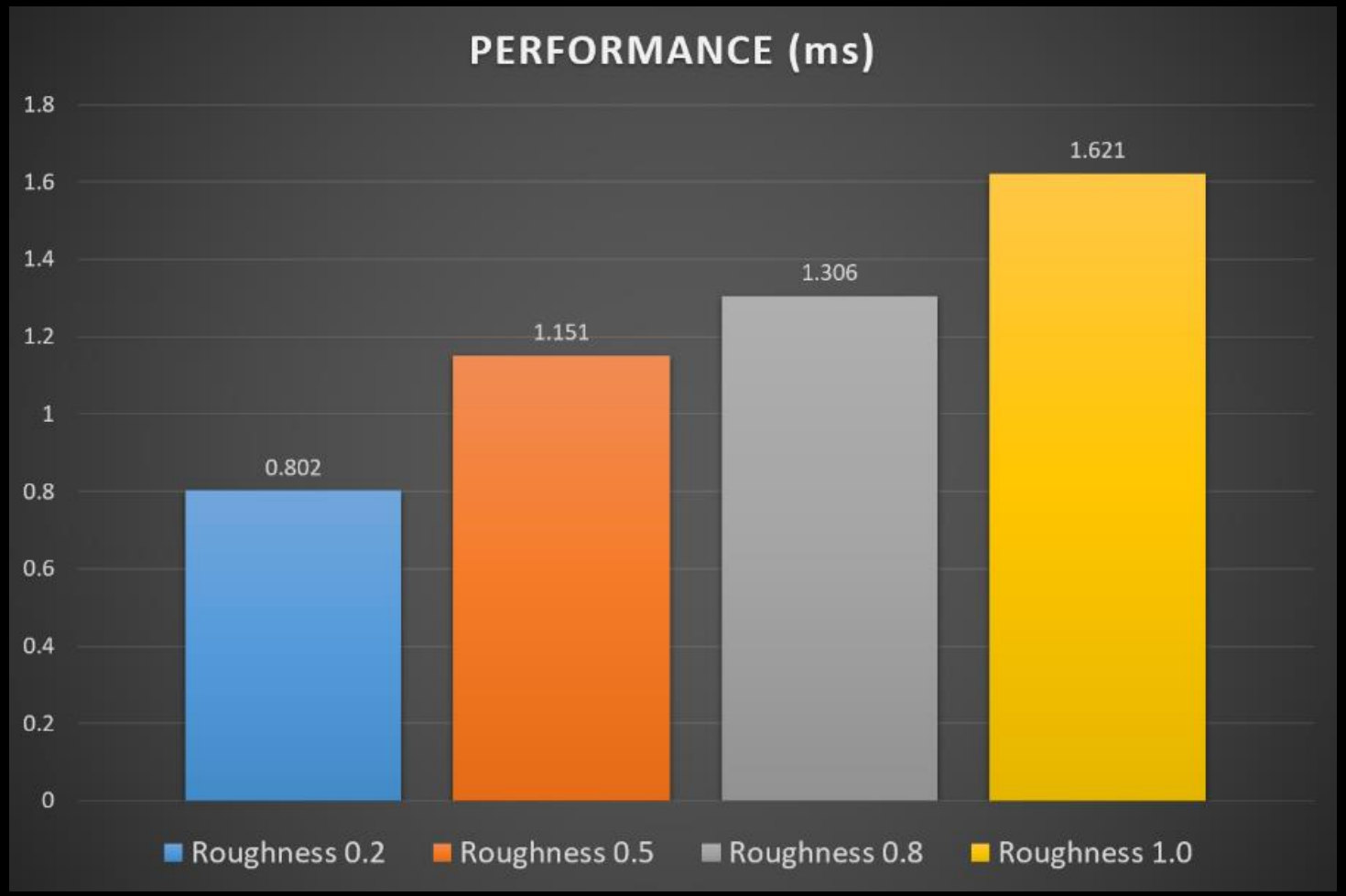
Comparison with and without normal map



Comparison between the results with different roughness

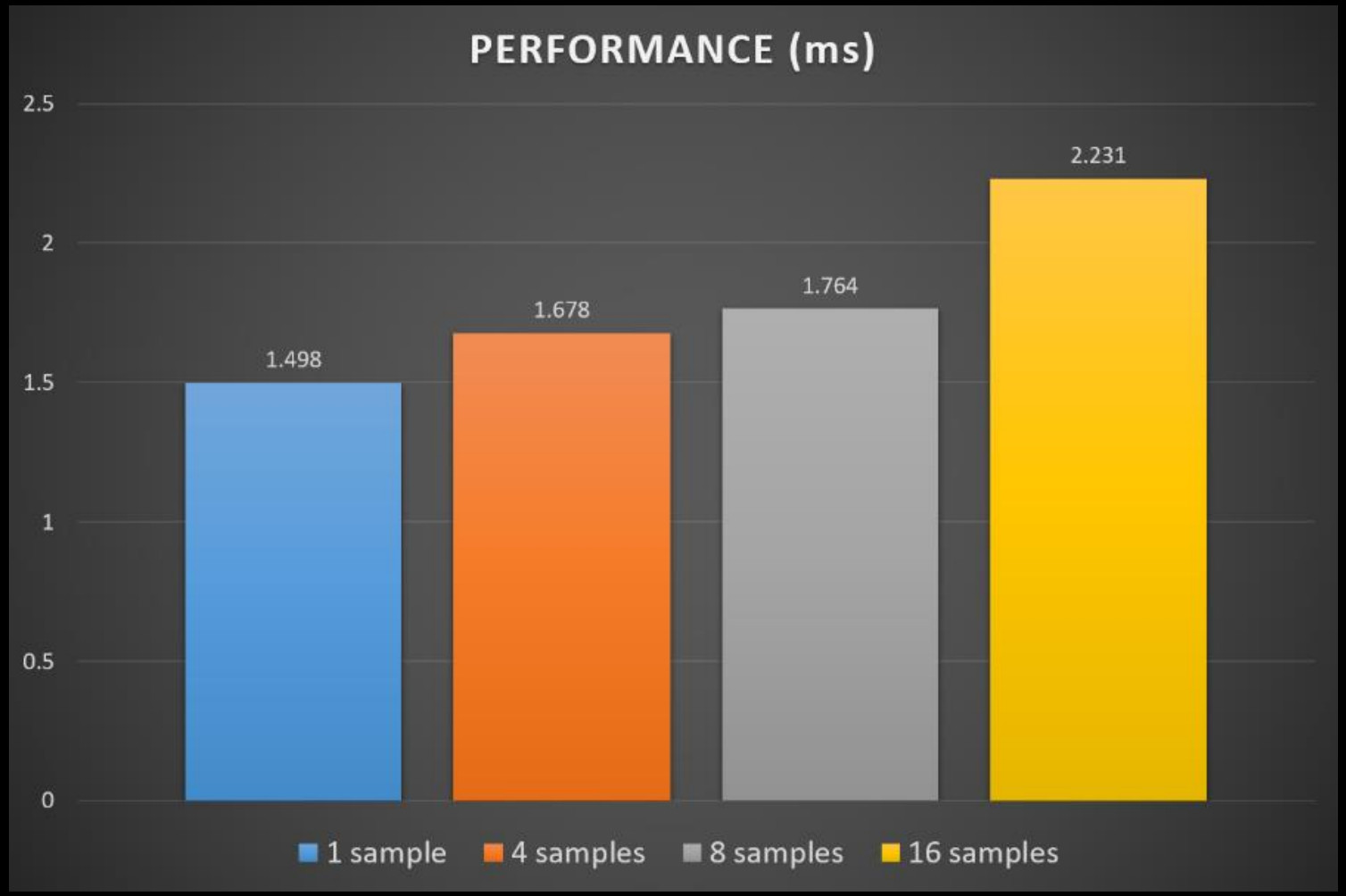


Comparison of the results with various Roughness



Comparison of the results with different number of Samples

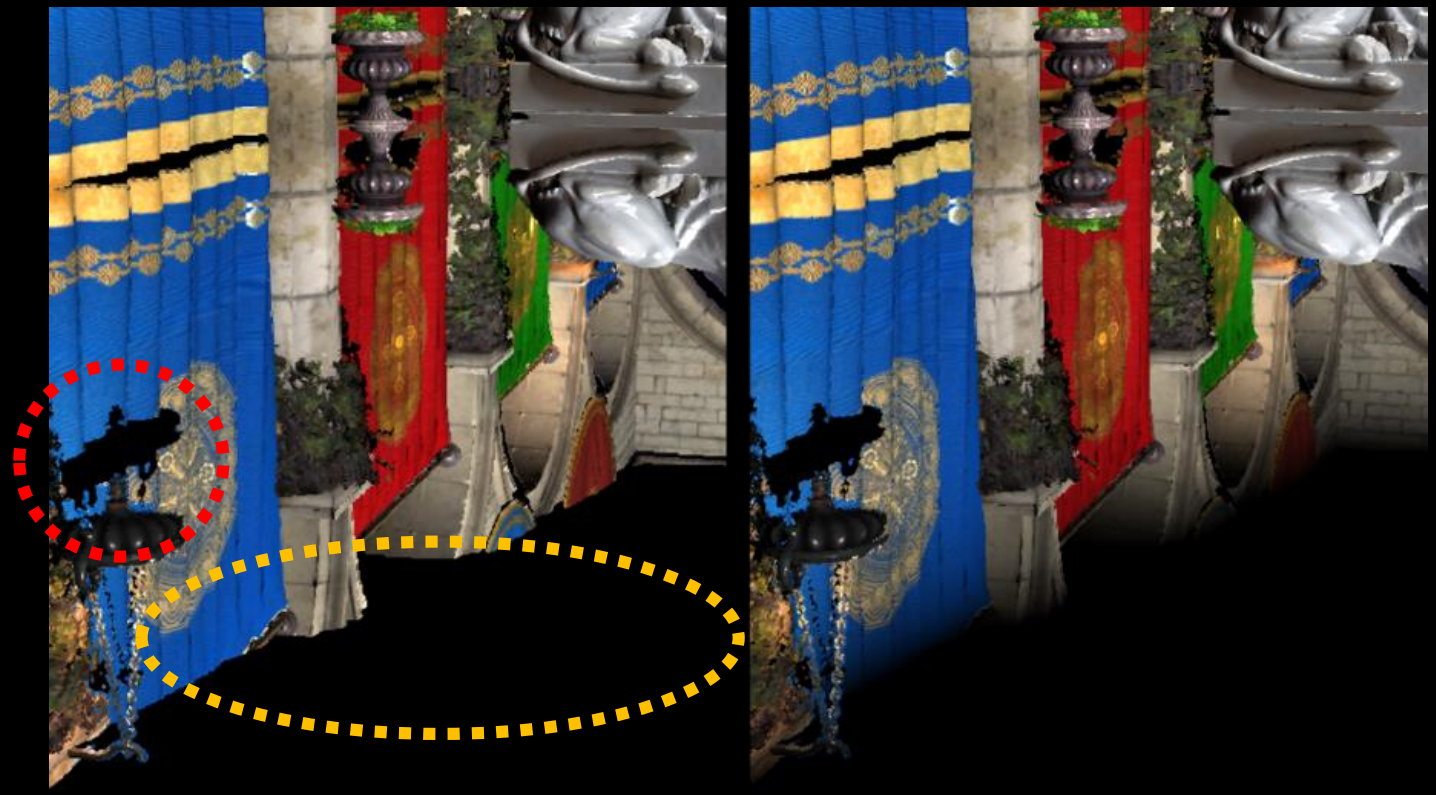
Comparison of the results with different number of Samples



Limitations

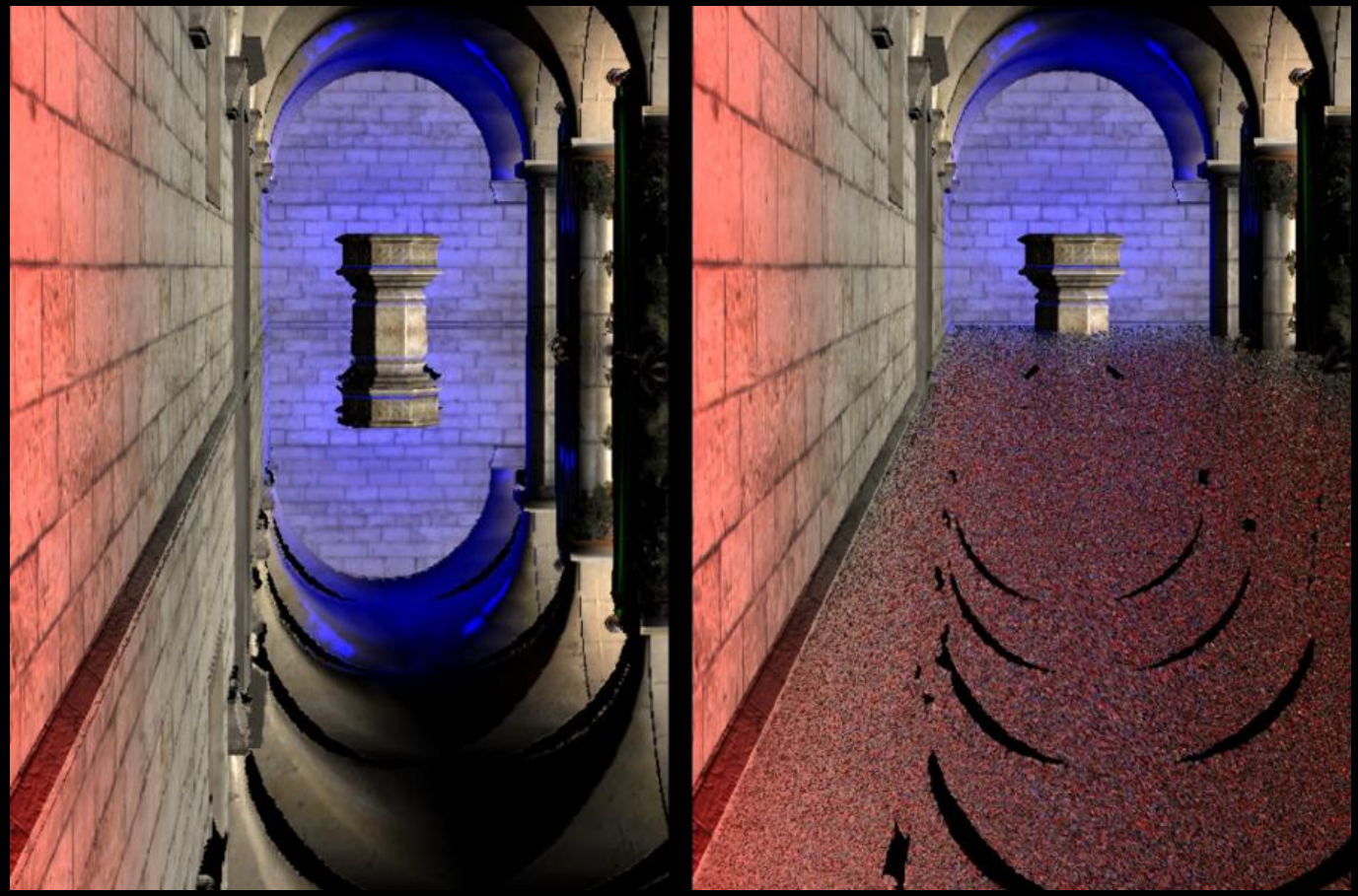
- Missing data
  - Occluded by other geometries
  - Out of Screen Space Boundary
- Color bleeding artifacts
- No glossy reflections
- Only simple geometries can be used for reflectors

Limitations



Without / with edge fade effect

Limitations



Color bleeding

Conclusion

- Better quality
  - More accurate reflection images
- Better performance
  - The fastest approach for performance vs quality
- Require reflectors
- Limitations to apply normal and roughness

Future work

- More efficient ray tracing solutions
  - Complex geometry
  - Faster acceleration structure
  - Microsoft’s DirectX Raytracing

References

[Akenine18] Tomas Akenine-Möller, Eric Haines, Naty Hoffman, Angelo Pesce, Michael Iwanicki: Real-Time Rendering, 4th Edition.

[Appel68] Arthur Appel: Some techniques for shading machine renderings of solids. AFIPS Conference Proc. 32 pp.37-45

[Bruce07] Bruce Walter, Steve Marschner, Hongsong Li, Ken Torrance: Microfacet Models for Refraction through Rough Surfaces.

[Cichocki17] Adam Cichocki: Optimized Pixel-Projected Reflections for Planar Reflectors, SIGGRAPH 2017.

[Giacaone16] Michele Giacalone: Screen Space Reflections in The Surge.

[Gregory14] Jason Gregory: Game Engine Architecture 2nd Edition.

[Lapinski17] Pawel Lapinski: Vulkan Cookbook.

[Le17] Trung Le: Evaluating BVH splitting strategies.

[Luna16] Frank D. Luna: 3D Game Programming with DirectX 12.

[Mansouri16] Jalal El Mansouri: Rendering Rainbow Six Siege, GDC, 2016.

[McGuire14] Morgan McGuire, Michael Mara: Efficient GPU Screen-Space Ray racing

[Pharr17] Matt Pharr, Wenzel Jakob, Greg Humphreys: Importance Sampling, Physically Based Rendering 3rd edition. pp.794-796.

[Raytracing18] “Wikipedia, Ray tracing (graphics)”. URL: < https://en.wikipedia.org/wiki/Ray_tracing_(graphics)>

[Stachowiak15] Tomasz Stachowiak, Yasin Uludag: Stochastic Screen-Space, SIGGRAPH 2015.

[Uludag14] Yasin Uludag: Hi-Z Screen-Space Cone-Traced Reflections, GPU Pro 5.

[Valient13] Michal Valient: Killzone Shadow Fall Demo Postmortem.

[Valient14] Michal Valient: Reflections and Volumetrics of Killzone: Shadow Fall, SIGGRAPH, 2014.

[Vulkan17] “Vulkan Tutorial”. < https://vulkan-tutorial.com>

[Wronski14] Bart Wronski: Assassin’s Creed 4: Road to Next-gen Graphics, GDC, 2014.