PRORENDER HYBRID RENDERING IN PRACTICE

DMITRY KOZLOV, AMD

Agenda

- Requirements
- Hybrid engine modes
  - Raster
  - Hybrid
  - Biased path tracing
- Raytraced effects
  - Reflection / refraction
  - Ambient occlusion
  - Area lighting and shadowing
- Demo

Requirements

- Radeon ProRender
  - High quality unbiased path tracer
  - Good at final rendering and complex scenes / materials
  - Not that good at preview rendering and simple scenes / materials
  - Noise
  - High latency in an attempt to reduce variance as quick as possible
  - Dynamic geometry
- What to do?
  - We need something fast and noise free for viewport / quick preview
  - Realtime
  - As accurate as possible given time budget
  - Time budget differs for different GPUs models / generations

Hybrid engine

- Highly scalable rendering framework
  - Flexible and intuitive performance-quality tradeoff (game-like quality settings)
  - None to minimum amount of noise for all quality levels
  - Clever rendering – spend time on complex things, fill the rest using ML or deterministic filters
  - Covers broad range of AMD hardware
- Flexible presets
  - High performance and high quality raster preview – noise free
  - Hybrid ray-traced viewport – noise free, deterministically filtered
  - Biased Monte-Carlo path tracer – minimum noise, ML filtered
- Highly optimized for dynamic geometry / scene changes
  - Fast acceleration structure builds
  - Fast updates
  - Material / lighting changes

Hybrid engine

- Based on Vulkan 1.1
  - Works on any Vulkan 1.1 compatible hardware
  - Benefits from exclusive AMD hardware features:
    - Async compute
    - Shader ballot
    - Descriptor indexing
    - External memory
    - Spares resources
  - Uses RadeonRaysNext for ray casting

AMD

Radeon ProRender Building Blocks

- Visibility solvers
  - Hardware rasterization
  - Ray tracing
  - Voxel ray tracing / cone tracing
- Illumination solvers
  - Approximate PBR (shadow maps, split sum)
  - Monte-Carlo path tracing
  - Voxel gather
  - Dynamic lightmaps
- Denoisers
  - Deterministic denoisers (EAW, SVGF)
  - ML denoising autoencoder
  - FidelityFX upscaler

Radeon ProRender Raster

- Clustered deferred renderer
- Analytic and physical lights
- PBR uber material
  - GGX coating
  - GGX reflection
  - Lambert diffuse
  - Tranparency / refraction
- Reflections
  - Pre-computed IBL (split sum)
  - Screen space reflections
- Transparency
  - Weighted OIT
- GI
  - Dynamic lightmaps
  - Voxel cone tracing
- TAA

AMD

Radeon ProRender Hybrid

- Thin visibility buffer
  - Depth
  - Normal
  - Texcoord
  - Barycentrics
  - Object ID
  - Material ID
  - Derivatives
- Reflections
  - Stochastic ray tracing
  - Multistage spatiotemporal denoiser
- Refraction
  - Ray tracing
- Area lighting
- GI
  - Lightmaps
  - Voxel cone tracing

Radeon ProRender Path Tracer

- Wavefront path tracer
  - Based on RadeonRays
  - Aggressive compaction
- GI
  - Full global illumination
  - Multiple importance sampling
  - Lightmaps
  - Voxels
- Denoise
  - DirectML based denoising autoencoder

AMD

Radeon ProRender: Reflections requirements

- GGX BRDF lobe
- Variable roughness support
- Can’t trace more than 1spp at half resolution
- Need to preserve fine details

Radeon ProRender Usecase: Reflections

- Light transport equation

$$L(w_o) = \int_{\Omega} L_i(w_i) F(w_o, w_i) \cos\theta \, dw_i$$

- F is a complex material function
  - IOR or metalness modes
  - Up to 2 reflection layers

$$L(w_o) = \int_{\Omega} L_i(w_i)(C + R + D + RF) \cos\theta \, dw_i$$

- Denoise each layer separately
  - Use split-sum approx.
  - Precompute R * cos LUT
  - Calculate and denoise L at runtime

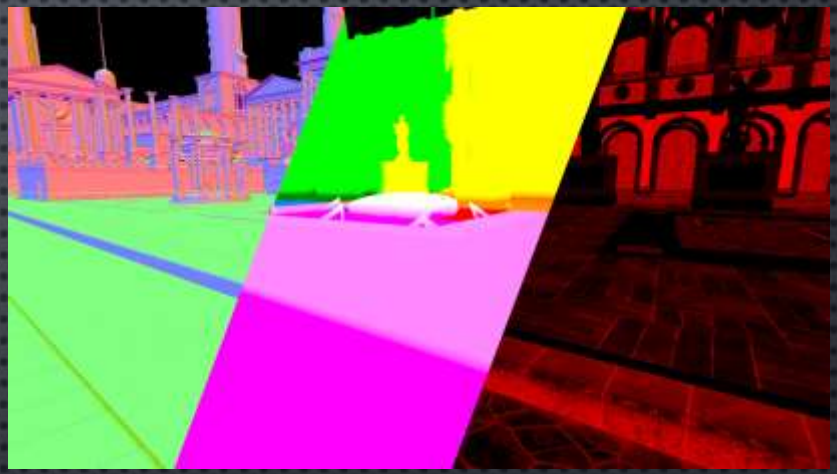
$$L_R(w_o) = \int_{\Omega} L_i(w_i) R(w_o, w_i) \cos\theta \, dw_i \approx$$

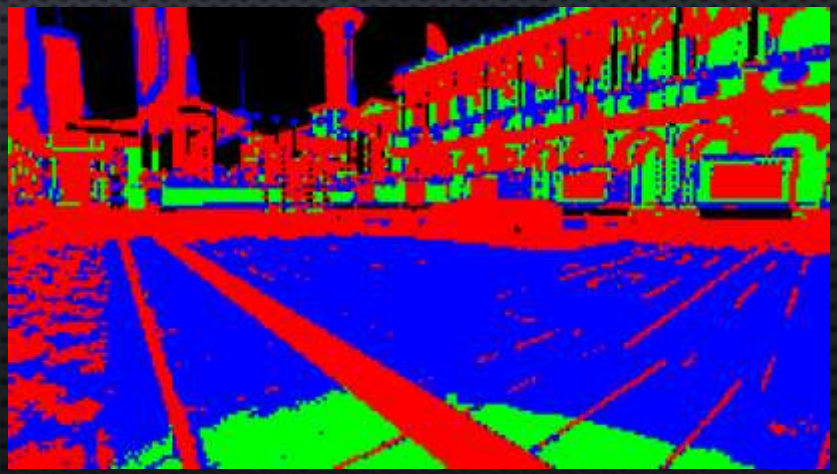
$$\int_{\Omega} L_i(w_i) \, dw_i \int_{\Omega} R(w_o, w_i) \cos\theta \, dw_i$$

AMD

Radeon ProRender: Reflections

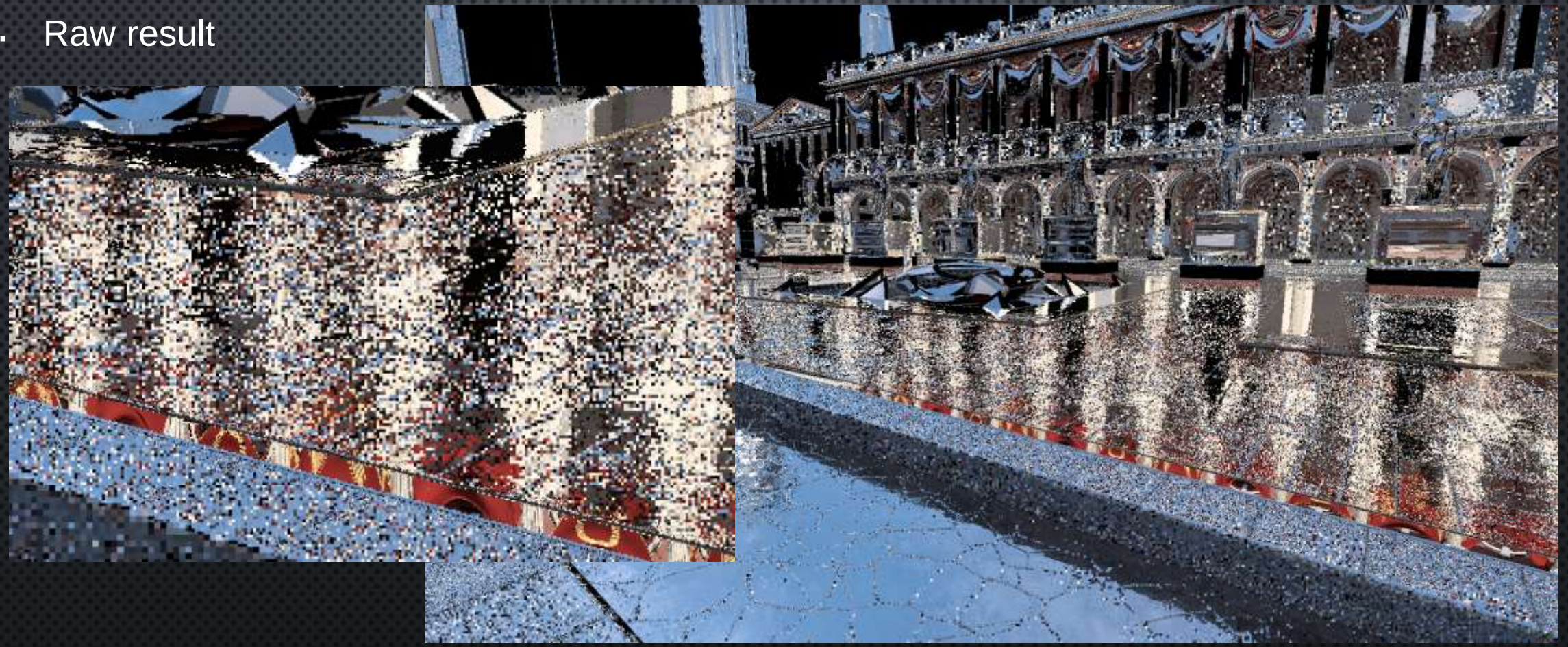
- Rasterize primary visibility buffer:
  - Thin G-buffer: depth, normal, barycentrics, primitive ID, shape ID, derivatives
- Spawn rays from G-buffer:
  - Use reconstructed world space position, normal and GGX roughness
  - Use importance sampling using distribution of visible normal (Heitz 2014)
- Variable rate ray generation and shading:
  - Split G-buffer in tiles
  - Spawn different number of rays based on estimated reflection importance in a tile
  - Red: 1spp, blue: 1/4spp, green: 1/16 spp





Radeon ProRender: Reflections

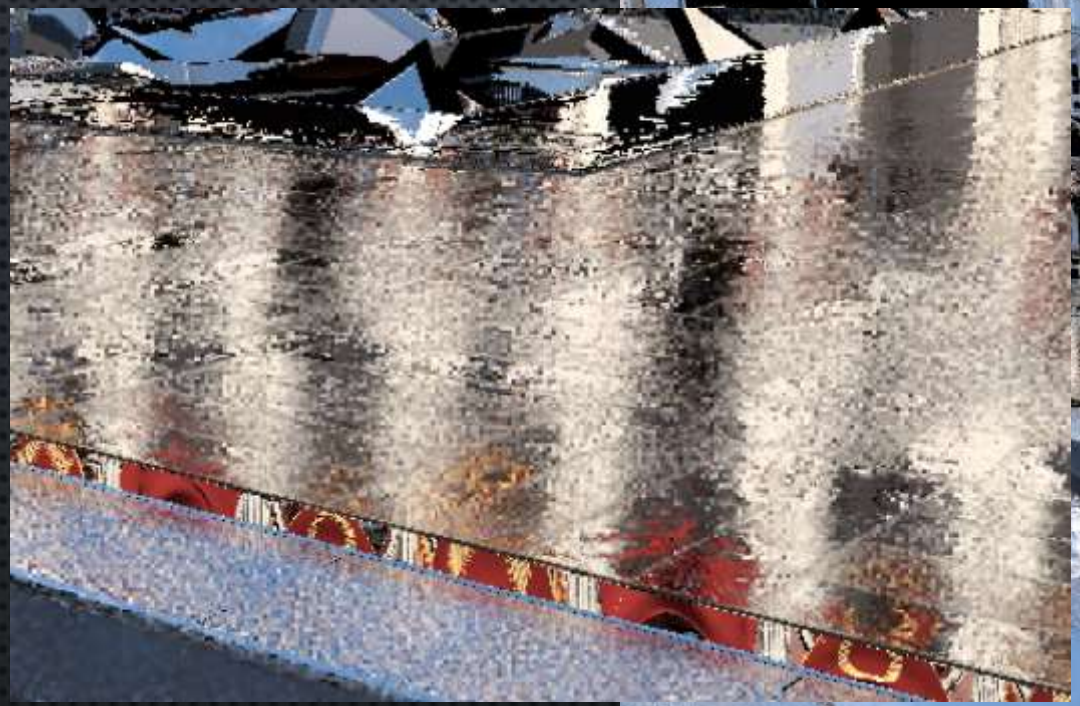
- Raw result

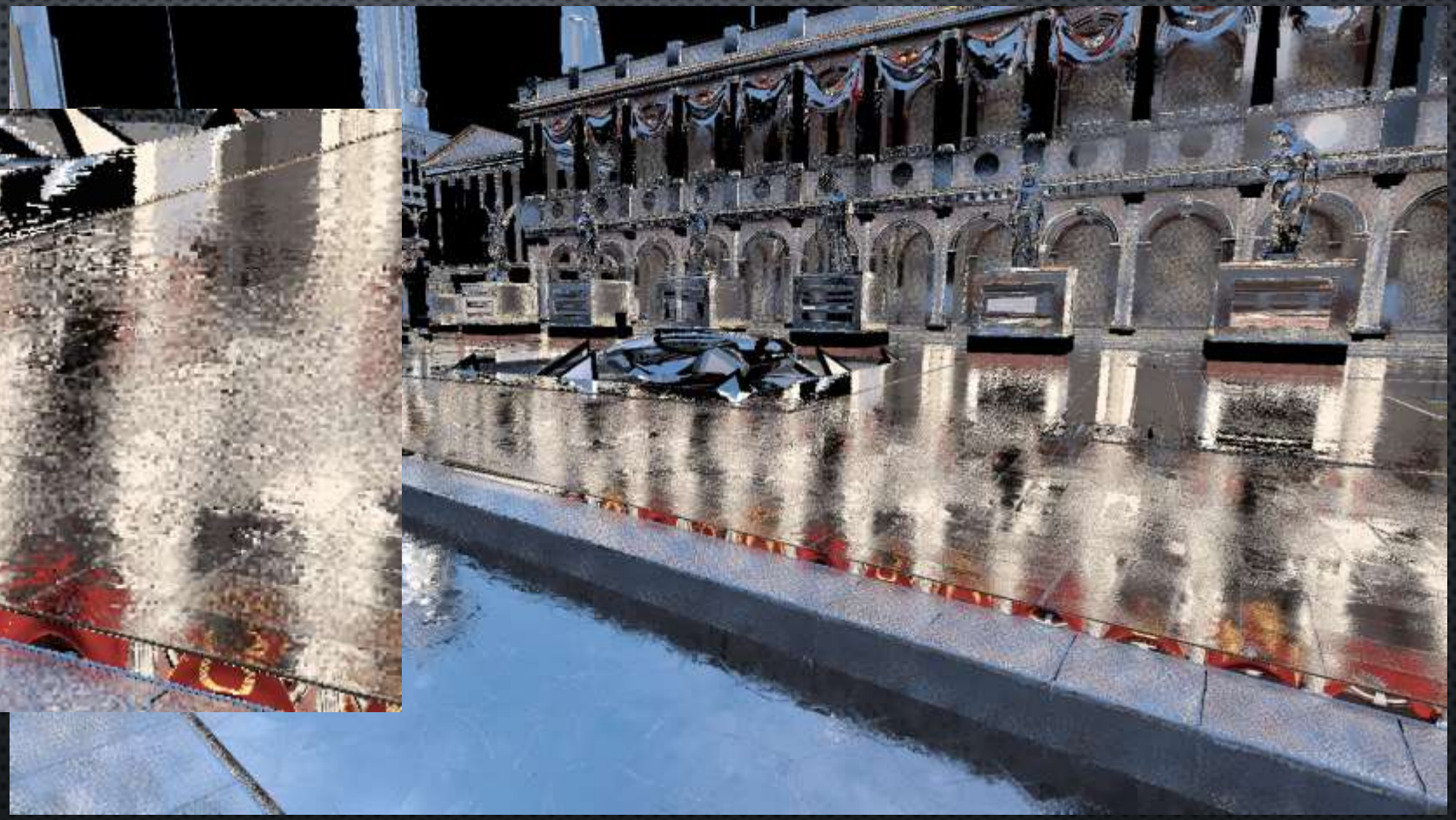


AMD

Radeon ProRender: Reflections

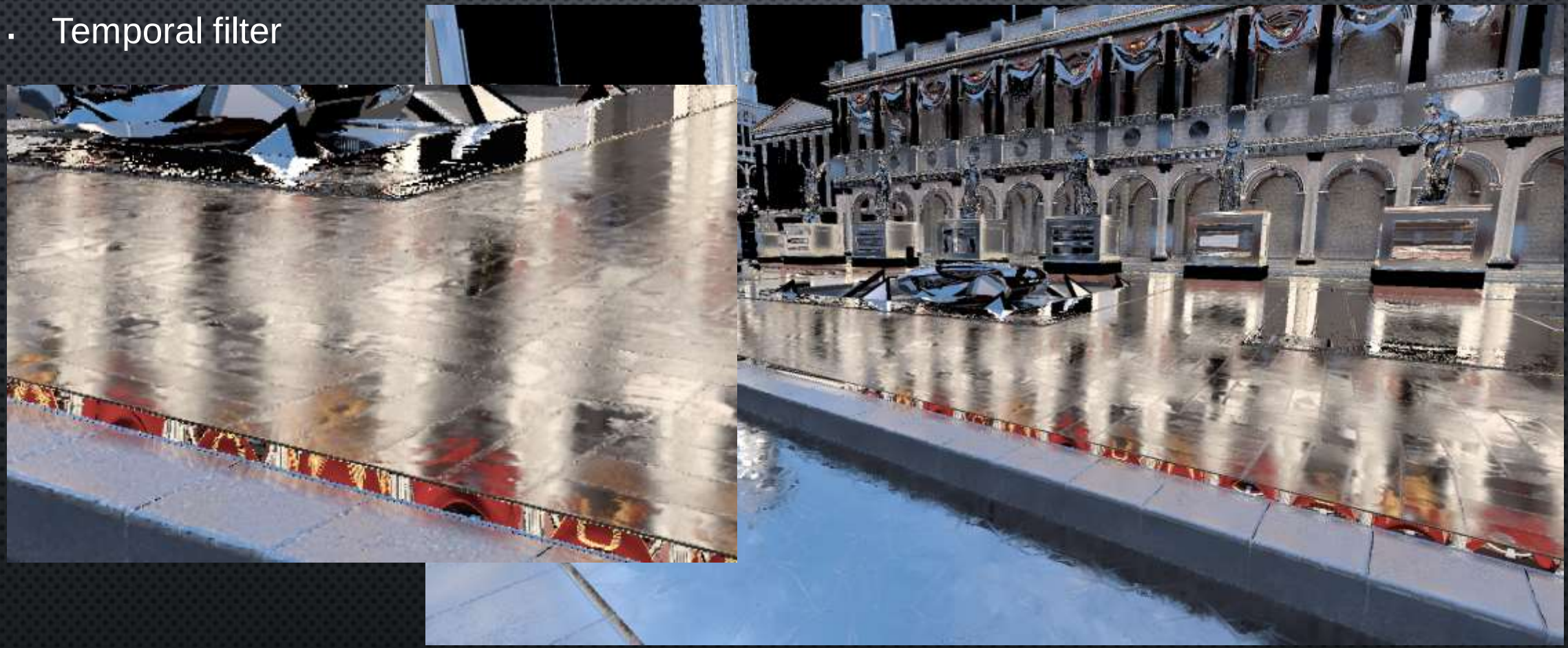
- Spatial denoise





Radeon ProRender: Reflections

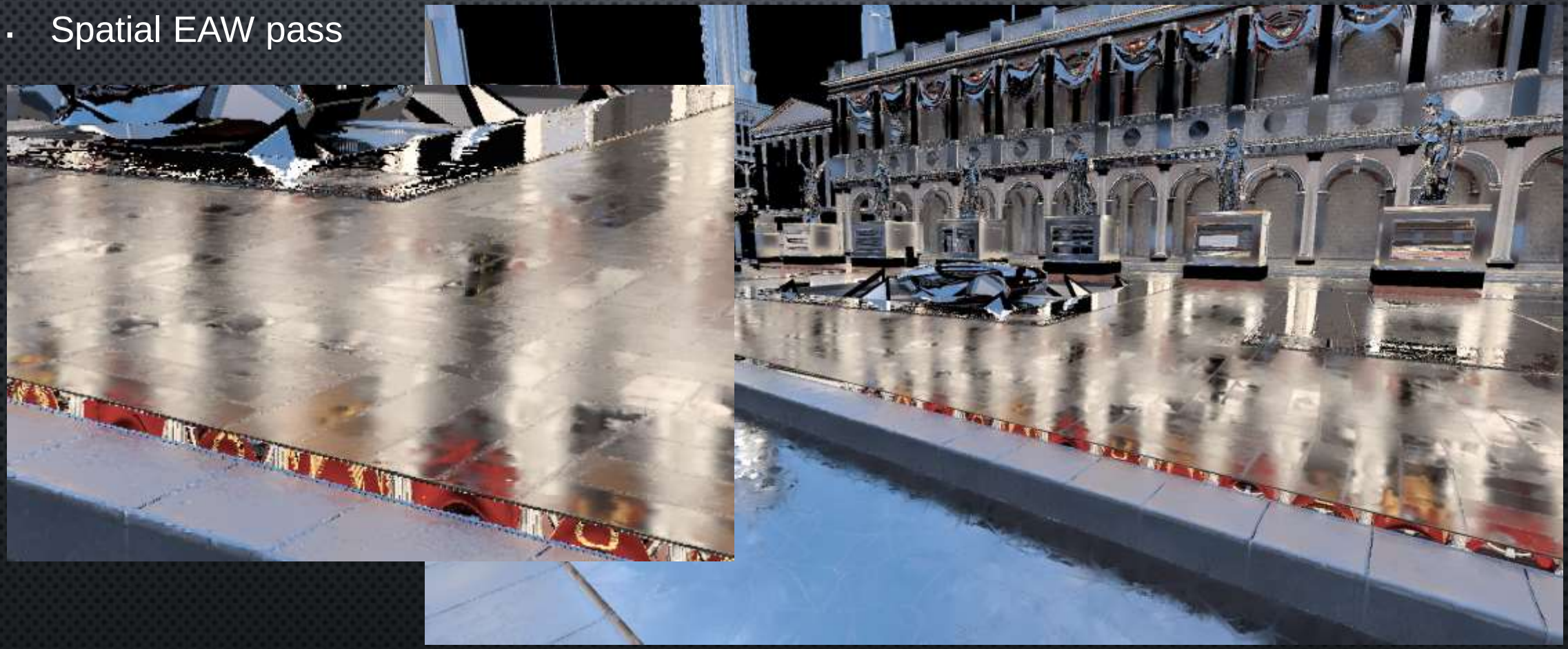
- Temporal filter



AMD

Radeon ProRender: Reflections

- Spatial EAW pass



Radeon ProRender: Reflections

- Final result

AMD

Radeon ProRender: Ambient occlusion

- Use cosine distribution to spawn 1spp at half resolution
  - Blue noise sampling
- Use occlusion query instead of a closest hit
- Denoise result:
  - Reprojection
  - Bilateral filter
  - Temporal accumulation

Radeon ProRender: Area lighting and shadowing

- Split area light integral into two parts:

$$L_{area}(\omega_o) = \int_A L_i(\omega_i) G(\omega_i) f(\omega_o, \omega_i) V(\omega_i) \mathrm{d}A_i$$

$$\approx \int_A L_i(\omega_i) f_t(\omega_o, \omega_i) G(\omega_i) \mathrm{d}A_i \int_A V(\omega_i) \mathrm{d}A_i$$

- Use LTC (Heitz et al. 16) to compute lighting $\int_A L_i(\omega_i) f_t(\omega_o, \omega_i) G(\omega_i) \mathrm{d}A_i$
- Use raytracing + denoising to compute visibility $\int_A V(\omega_i) \mathrm{d}A_i$

Radeon ProRender: Area lighting and shadowing

AMD

Radeon ProRender: Reflections + AO + area lighting

AMD

Conclusion

- New Radeon ProRender backend
  - Vulkan 1.1
  - Performance-quality tradeoff
  - Optimized for AMD hardware
  - Cross platform
  - Cross vendor
  - Available for Blender 2.8+

- Visit https://www.amd.com/en/technologies/radeon-prorender-downloads

AMD