TRAVERSE RESEARCH

Evolve

NEXT SLIDE

Introduction

TRAVERSE RESEARCH

- Applied Research and Development
  - Founded in 2019 by Jasper Bekkers
  - Located in Breda, The Netherlands
- Research
  - Try to run ahead of the industry to prove out concepts
  - Nimble codebase
- Benchmarking
  - Working with device manufacturers
  - Cutting edge features
- Collaborations
  - Strong university ties
  - Game industry background & veterans
- Sharing
  - Publish at conferences
  - Open source contributions
  - Host and attend meetups

TREE PILLARS

GPU BENCHMARKING

Evolve

We're developing a GPU benchmarking suite called "Evolve" that will focus in various graphics workloads

2024

GRAPHICS RESEARCH

Impact and longevity

Long and short term rendering research that has a primary focus on Ray Tracing and Machine Learning.

Framework built from the ground up for prototyping and developing forward looking graphics.

Ray tracing, path tracing, volumetrics, machine learning inference and training.

High paced R&D

DEVELOPMENT

Excellence

Together with software and hardware vendors we can help to build out and optimize their GPU drivers.

We provide direct services for workload generation, pre-/post-silicon validation and performance evaluation.

Expertise

EVOLVE

01. WHAT IS EVOLVE

Modern gpu benchmarking of new workloads.

- Vulkan + DirectX12 + Metal
- Ray tracing
- Path Tracing
- Complex scene
- Dynamic lighting
- Ray Query and Ray Pipeline based implementations of **all** techniques

02. WHY EVOLVE

Breakthrough & Innovation

- Scores for various workloads
- Open core
- Community
- Core features of Evolve and Breda framework
- Quality assurance
- Game developer backgrounds
- Realistic workloads

RAYTRACING

Showcase

Path tracing

We'll render this scene with various rendering algorithms, we have two path tracers and a hybrid renderer

Hybrid renderer

We've set up a state of the art hybrid renderer; designed from the ground up to support both RayQuery and Pipeline based ray tracing

On mobile platforms

Android is first class supported, just due to its extremely heavy workload, real-time performance is expected only on high-end devices.

Desktop

Desktop will release on all ray tracing capable devices

Lush vegetation

The point of this demo is to show how we can render (animated) vegetation really well. We'll need to test with alpha testing vs mesh geo

Breakthrough & Innovation

SCORES FOR VARIOUS WORKLOADS

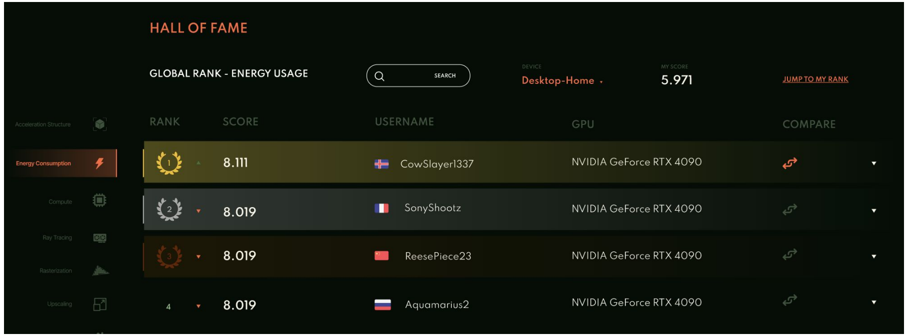
What makes EVOLVE **different** from other benchmarks that exist on the market

Instead of just giving an overall score EVOLVE can bring you more tailored and more practical scores

Our high-level scores include;

- Acceleration structure
- Ray Tracing
- Rasterization
- Compute
- Energy consumption
- Driver overhead

STATS FOR NERDS



01. Measure more

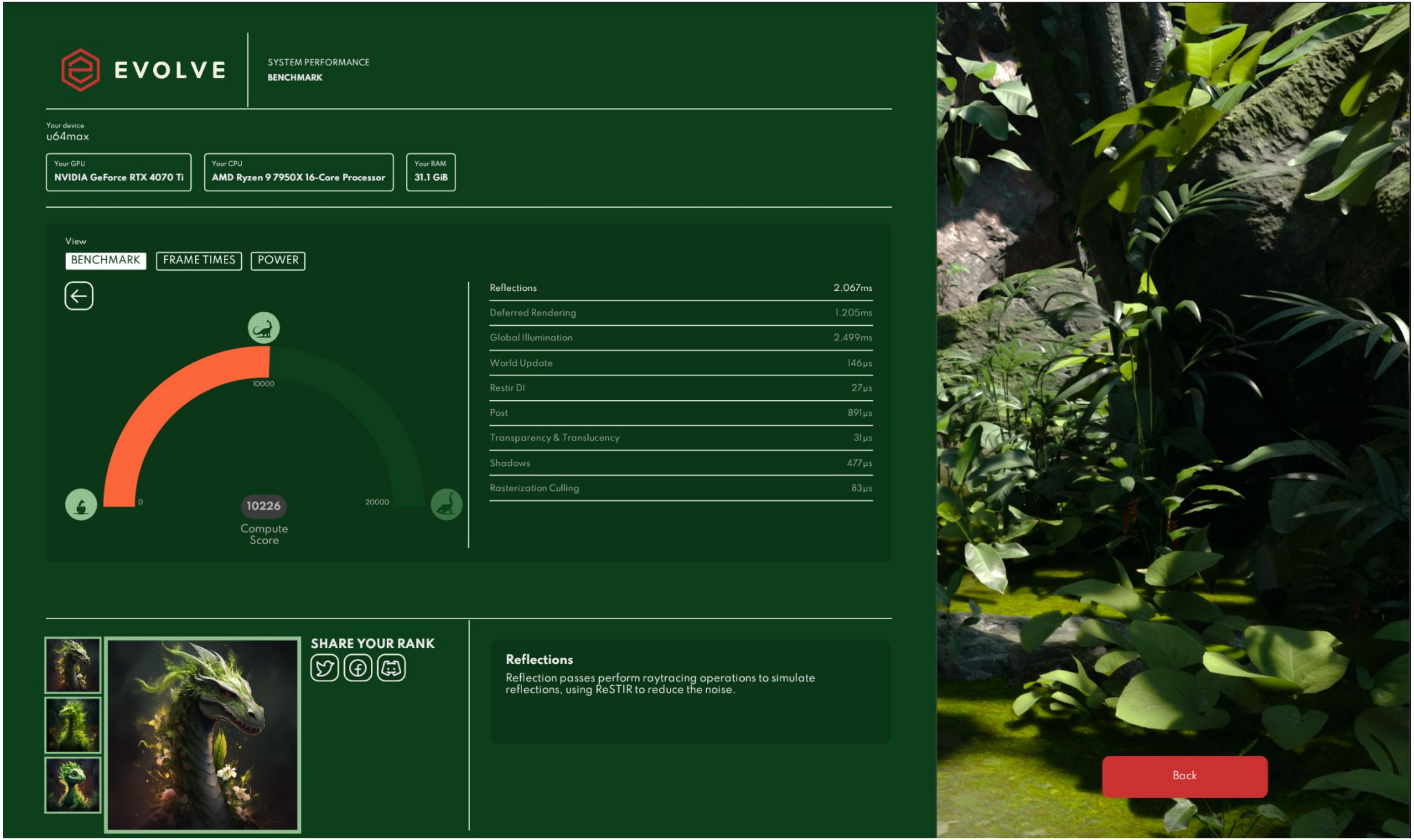
Show and share detailed information about TLAS / BLAS performance, trace performance etc

02. Educate our audience

We'll need to educate our users about what all the statistics mean in laymans terms and we'll work with press to get this information right

03. Leaderboards

For our end-users we'll have leaderboards that can be sorted and segmented. In app we'll award users their scores as well


EVOLVE
SYSTEM PERFORMANCE
BENCHMARK
Your device
u64max
Your GPU
NVIDIA GeForce RTX 4070 Ti
Your CPU
AMD Ryzen 9 7950X 16-Core Processor
Your RAM
31.1 GiB
View
BENCHMARK
FRAME TIMES
POWER
10000
0
20000
10226
Compute Score
Reflections 2.067ms
Deferred Rendering 1.205ms
Global Illumination 2.499ms
World Update 146μs
Restir DI 27μs
Post 891μs
Transparency & Translucency 31μs
Shadows 477μs
Rasterization Culling 83μs
SHARE YOUR RANK
Reflections
Reflection passes perform raytracing operations to simulate reflections, using ReSTIR to reduce the noise.
Back

CORE FEATURES

01. Path tracer

Wavefront-style path tracer with support for both ray-queries and pipeline traces. Optional recursive pipeline path tracer.

02. Hybrid renderer

RT Global illumination, RT Reflections, RT Shadows as well as a fitted material model.

03. Flexible Engine

Default usage of a render graph, enabling quick iteration times while remaining efficient. **Breda** guarantees resource lifetimes outlive the GPU timeline; no explicit syncing or lifetime management is needed.

04. Development

All code is written in Rust. Shaders are written in HLSL and are identical code-wise between graphics APIs.

PATH TRACER

Path tracer features

- Both ray-queries and pipeline tracing are available
- Wavefront style path tracer
- Pipeline style recursive path tracer
- Radiance caching

HYBRID RENDERER

Caches

- Clipmap irradiance cache
- Hash grid radiance cache
- Neural radiance cache

HYBRID RENDERER

Hybrid renderer features

- ReSTIR based dynamic GI
- ReSTIR based Reflections
- RT Refraction
- RT Soft shadows
- Fitted conductor materials

ENGINE & DEVELOPMENT

Workload setup

- Very small amount of code
- No explicit resource/pass syncing
- Optimal barriers and grouping is handled internally
- Resources are guaranteed to outlive GPU timeline

```
compute:
    wireframe:
        cs:
            filename: "evolve::wireframe.cs.hlsl"
            entry_point: "main"
```

```
ComputePass::new("Wireframe pass", render_graph)
    .constants_buffer(&wireframe_constants)
    .read(vbuffer_indices)
    .write(&wireframe_output)
    .dispatch(
        &shader_db.get_pipeline("wireframe"),
        render_width.div_ceil(8),
        render_height.div_ceil(8),
        1,
    );
```

ENGINE & DEVELOPMENT

Resources & Bindless

```rust
#[derive(Copy, Clone, Eq, PartialEq, Hash)]
#[repr(transparent)]
pub struct RenderResourceHandle(u32);
impl RenderResourceHandle {
    pub fn new(version: u8, tag: RenderResourceTag, index: u32, access_type: AccessType) -> Self {
        let version: u32 = version as u32;
        let tag: u32 = tag as u32;
        let index: u32 = index;
        let access_type: u32 = access_type.is_read_write() as u32;
        Self(version << 26 | access_type << 25 | tag << 23 | index)
    }
```

ENGINE & DEVELOPMENT

Shader abstraction

- Resources are abstracted
- Flexible usage
- Aimed to work for all APIs
- No API specific user code
- *ResourceDescriptorHeap* abstraction for vulkan

```
struct RwTexture {
    RenderResourceHandle handle;

    template < typename RWTextureValue > RWTextureValue load1D(uint pos) {
        validateResource(kWritable, kTextureResourceTag, this.handle);
        RWTexture1D<RWTextureValue> texture = DESCRIPTOR_HEAP(RWTexture1DHandle<RWTextureValue>, this.handle.writeIndex());
        return texture.Load(pos);
    }

    template < typename RWTextureValue > RWTextureValue load2D(uint2 pos) {
        validateResource(kWritable, kTextureResourceTag, this.handle);
        RWTexture2D<RWTextureValue> texture = DESCRIPTOR_HEAP(RWTexture2DHandle<RWTextureValue>, this.handle.writeIndex());
        return texture.Load(pos);
    }

    template < typename RWTextureValue > RWTextureValue load3D(uint3 pos) {
        validateResource(kWritable, kTextureResourceTag, this.handle);
        RWTexture3D<RWTextureValue> texture = DESCRIPTOR_HEAP(RWTexture3DHandle<RWTextureValue>, this.handle.writeIndex());
        return texture.Load(pos);
    }
```

ENGINE & DEVELOPMENT

PREDECLARED RESOURCE IDENTIFIERS

```
template <typename T> struct Texture1DHandle { uint internalIndex; };
template <typename T> struct Texture2DHandle { uint internalIndex; };
template <typename T> struct Texture3DHandle { uint internalIndex; };
template <typename T> struct RWTexture1DHandle { uint internalIndex; };
template <typename T> struct RWTexture2DHandle { uint internalIndex; };
template <typename T> struct RWTexture3DHandle { uint internalIndex; };
```

EMULATION STRUCT

```
struct VulkanResourceDescriptorHeapInternal {
    ByteAddressBuffer operator[](ByteBufferHandle identifier) {
        return g_ByteAddressBuffer[NonUniformResourceIndex(identifier.internalIndex)];
    }
```

ENGINE & DEVELOPMENT

Shader development

- Load any resource from a buffer
- Robust bindless resource usage validation
- Resource lifetime validation
- Built-in breadcrumbs system
- Runtime shader printing/asserts

```
struct Bindings {
    SimpleBuffer constants;
    Texture vbufferIndices;
    RwTexture output;
};

[numthreads(8, 8, 1)] void main(uint2 dispatchThreadId
                                : SV_DispatchThreadID) {
    Bindings bnd = loadBindings<Bindings>();
    // For demonstration purposes just output 1.0
    float someOutput = 1.0;
    bnd.output.store2D<float>(dispatchThreadId, someOutput);
}
```

ENGINE & DEVELOPMENT

VERSION MISMATCH FAILURE

```
[breda_render_backend_api::shader_logging][ERROR]
Compute Shader GPU resource validation failed:
Resource version mismatch in `gpu_validation` RenderResourceHandle of type `Buffer` has version: `0` Expected version: `1`.
Possible causes:
- A `RenderResourceHandle` was unsafely extracted by the user, where the handle outlived the resource.
- User copied raw `RenderResourceHandle` within a shader to a buffer for later reuse, this is not allowed!
```

RESOURCE TYPE MISMATCH

```
[breda_render_backend_api::shader_logging][ERROR]
Compute Shader GPU resource validation failed:
Resource access mismatch in `gpu_validation` handle is of type: `Texture`, Expected handle of type: `Buffer`.
```

WRITABILITY FAILURE

```
[breda_render_backend_api::shader_logging][ERROR]
Compute Shader GPU resource validation failed:
Tried writing to resource that is read-only in `gpu_validation` RenderResourceHandle has AccessType of: `ReadOnly`.
```

Foundation of evolve

BREDA FRAMEWORK

Enjoy unprecedented ease of build, use, and code retrieval

- Breda will be fully open source under a permissive license

- 100% source code and assets become available to licensees.

- Easy cross-platform builds due to our usage of Rust and Cargo

CORE FEATURES OF EVOLVE AND BREDA FRAMEWORK

Performance and Compatibility

- Cross-Platform Support: Windows, Android, Linux, SteamOS/SteamDeck, and upcoming support for MacOS and iOS.
- Hardware Ray Tracing
- Efficient GPU Utilization
- Render Graph System
- Written by industry experts
- Raytracing natively on Galaxy S23 and S24, Mali, and Qualcomm mobile GPUs as well as Nvidia, Amd and Intel

Animation and Dynamic Elements

- GPU Skinning Animation System Dynamic Elements & TLAS/BLAS
- Alpha Testing/Vegetation Rendering
- Large scale raytraced crowd rendering
- Deterministic from run to run

VOLUMETRIC RENDERING

Vdb volumes

After ray tracing hardware got introduced, it was quite evident that building and maintaining acceleration structures became feasible for solid geometry.

We've focussed research on acceleration structures for volumes, light transport for volumes and physics / animation of volume data.

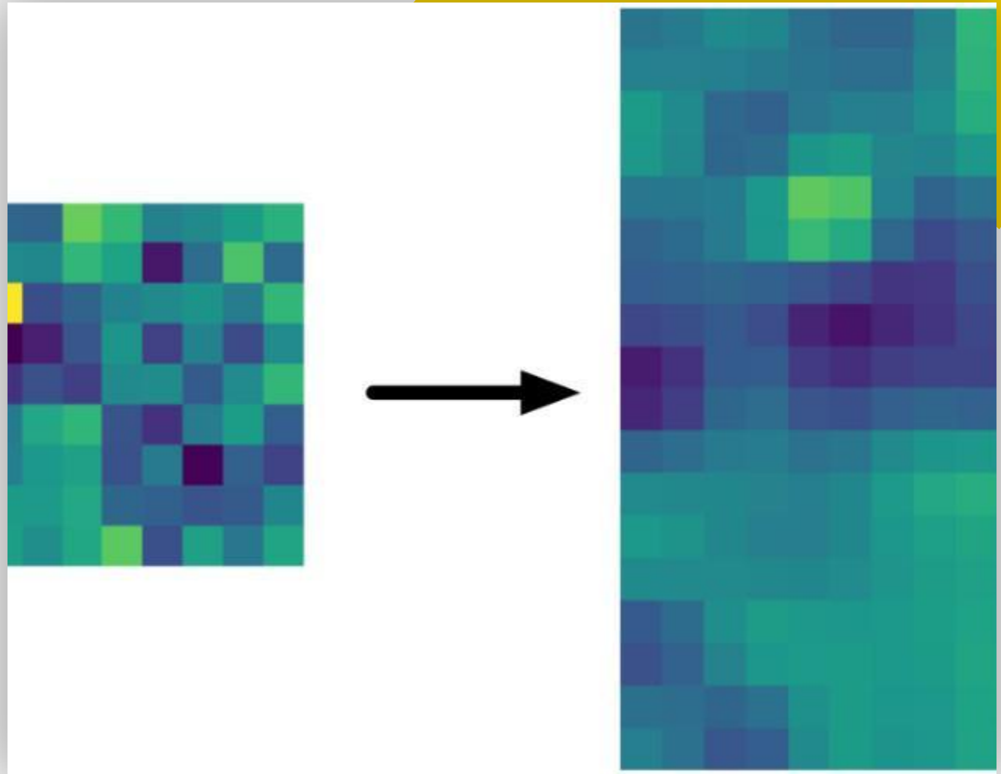
for games

MACHINE LEARNING

- Obvious candidates for mobile
  - Upsampling
  - Denoising
  - Frame interpolation
- Features often driven by hardware manufacturers
- Easy to run on “seperate” TPU
- Relatively easy to integrate
- Frequently CUDA only

Engine built for Machine Learning and Ray Tracing

Invest in tools and processes for growth.



BREDA-NN

Successor of breda-inference.

Compute only inference and **training** cross-platform (vulkan + dx12).
Focus on real-time and tight integration with our rendering pipeline.

01. Render graph

Built on top of our render graph, all kernels in hlsl.

02. fp16 inference & training

The framework supports inference and training at full fp16 precision. (few operations are implemented using mixed precision)

03. extended ONNX

Our models are imported as ONNX. Wider support of onnx operator and support for loading and executing (inference/training) a graph defined in ONNX.

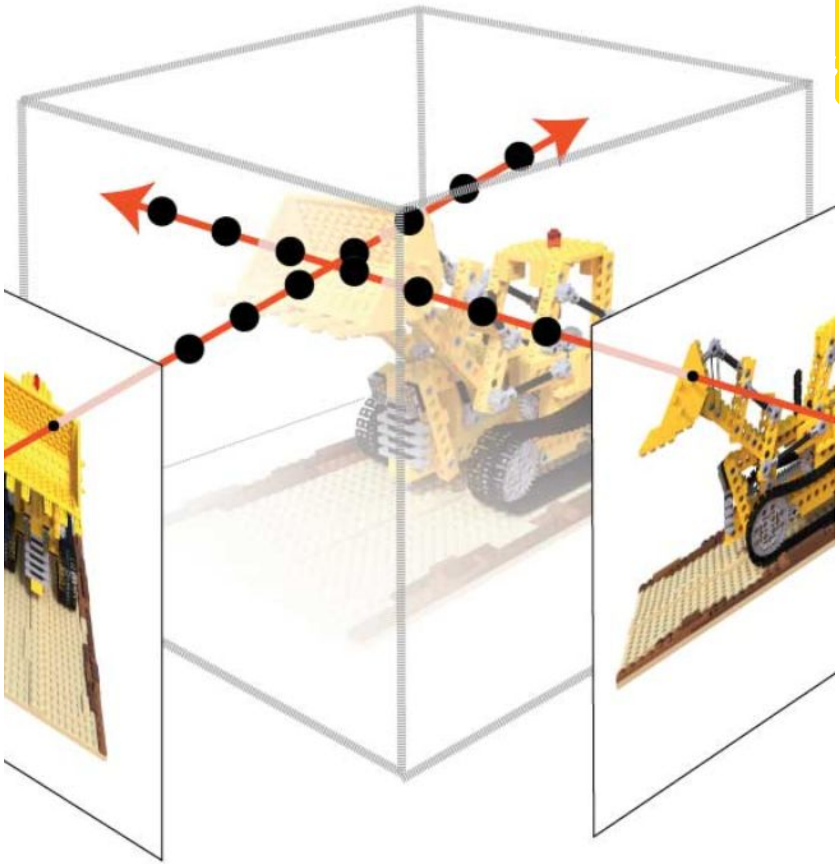
04. Features

Nerf, OIDN denoiser;
nn-operations: grid encoders (dense + hash instant ngp), linear layer, fully fused mlp, Conv2d, pool2d, upsample2d, losses, spherical harmonics encoder, activation

breda-nn

NERF

Based on [Instant-NGP] and using fully fused MLP.
Support for loading and rendering NeRFs trained using Instant-NGP.



01. Next Steps

- Training in Breda (ongoing)
- Integration in ray-tracer/path-tracer:
    - Relightable NeRF (light)
    - Integration with textured meshes
- Integration with NeRF studio

breda-nn

NEURAL MATERIAL

Published in Siggraph Asia 22: NeuBTF
Implemented in both our path-tracer and ray-tracer.

They can be obtained from captures of

- real-life materials
- rendered materials

Applied to a mesh like “textures”

Integrated with “bespoke” materials/scenes

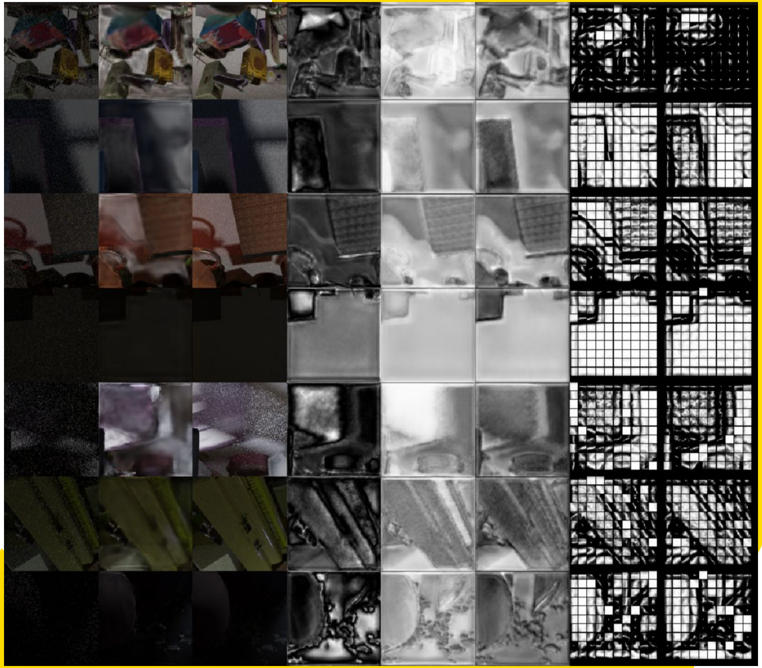
breda-nn

PATH SPACE DENOISING

In modern rendering, denoising is increasingly handled by machine learning rather than traditional filters. A neural network predicts a pixel-specific filtering kernel, utilizing additional per-pixel features like world normals, camera distance, and material properties. For real-time rendering efficiency, an encoded kernel version is initially predicted and later expanded to its full form, ensuring both performance and enhanced image quality

Applied to a mesh like "textures"

Integrated with "bespoke" materials/scenes

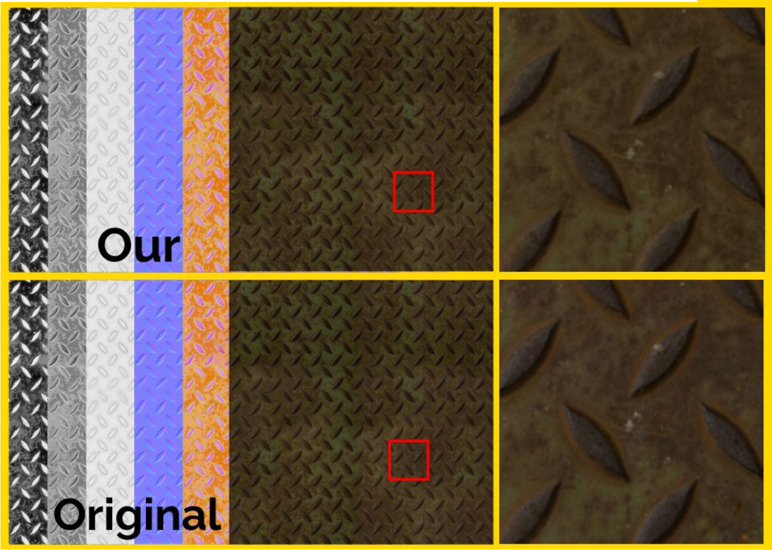


breda-nn

TEXTURE COMPRESSION

Vaidyanathan et al. published first, but we've independently researched a similar technique.

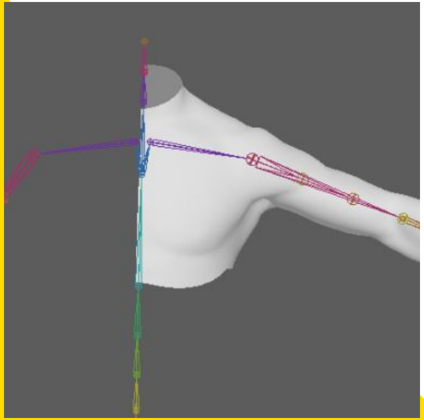
As modern streaming demands high-quality visual content at low bandwidths, efficient texture compression methods are becoming increasingly essential. Our approach for compressing Physically-Based Rendering (PBR) textures into Machine Learning models ensures high-quality texture reconstruction while reducing storage and streaming requirements.



What’s possible with ML?

ML DEFORMERS

Use machine learning to
blend between animations



Content generation

Upcoming project: we’ll dive into
content generation in
stable-diffusion type techniques

Water simulations

Use machine learning inference to
run water simulations in real time

DEEPCLOTH

Use machine learning to
animate cloth in real time

CONTACT US

jasper@traverse.nl / darius@traverse.nl

https://evolvebenchmark.com

Jasper Bekkers & Darius Bouma