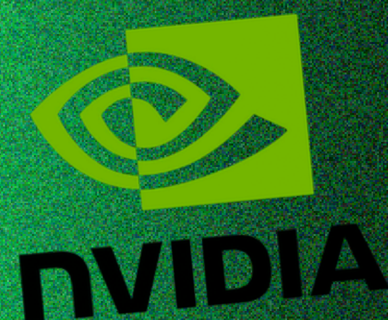
Presented by
Carolina Reis | 131193 & Eduardo lopes | 103070

deti
universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

2025/2026

CLE



Volumetric MIP with CUDA

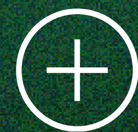
GPU-ACCELERATED MAXIMUM INTENSITY PROJECTION

Outline

1. **Problem & Dataset**
2. **Processing Pipeline**
3. **CUDA Kernels**
4. **Innovation: Multi-Frame Video**
5. **Speedup Analysis**
6. **Demo & Results**
7. **Conclusion**

1.

Problem & Dataset



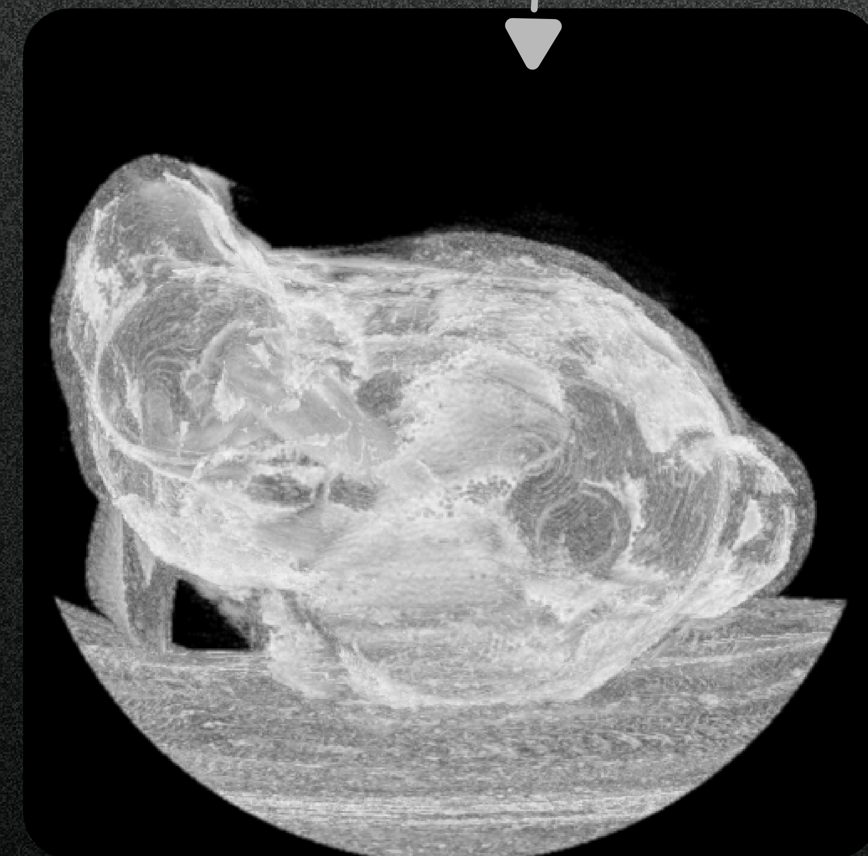
Stanford Bunny CT Scan

Dataset

- 361 grayscale slices, each 512 × 512 pixels
- Stacked ⇒ 3-D voxel grid 512 × 512 × 361
- Each voxel: uint16 t density value
- Total: ≈ 190 MiB in host RAM

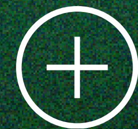
Goal

Render a 2-D image from any viewing angle interactively via GPU acceleration.



2.

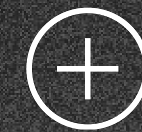
Processing Pipeline

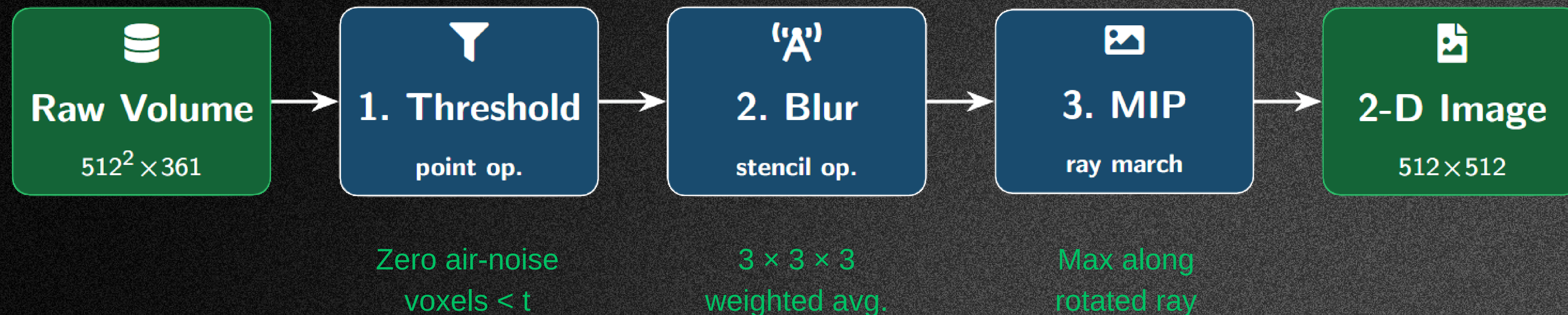


THREE-STAGE PIPELINE

Sequential Dependency

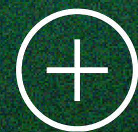
Each stage consumes the previous stage's output ⇒ three separate CUDA kernels, intermediate buffers stay on device.





3.

CUDA Kernels



STRATEGY

Point operation → every voxel is independent.

LAUNCH CONFIG

- 1-D grid, 256 threads/block
- ⌈94 896 128/256⌉ = 370 610 blocks
- Thread i checks bounds, zeroes if < t

```
__global__ void kernel_threshold (
    uint16_t * vol , int size ,
    uint16_t threshold )
{
    int i = blockIdx . x * blockDim . x + threadIdx . x ;

    if ( i < size && vol [ i ] < threshold )
        vol [ i ] = 0;
}
```

No inter-thread communication.
In-place on the device buffer.

Kernel 1: Threshold

‘kernel_threshold’

STRATEGY

Stencil → each thread computes one output voxel.

LAUNCH CONFIG

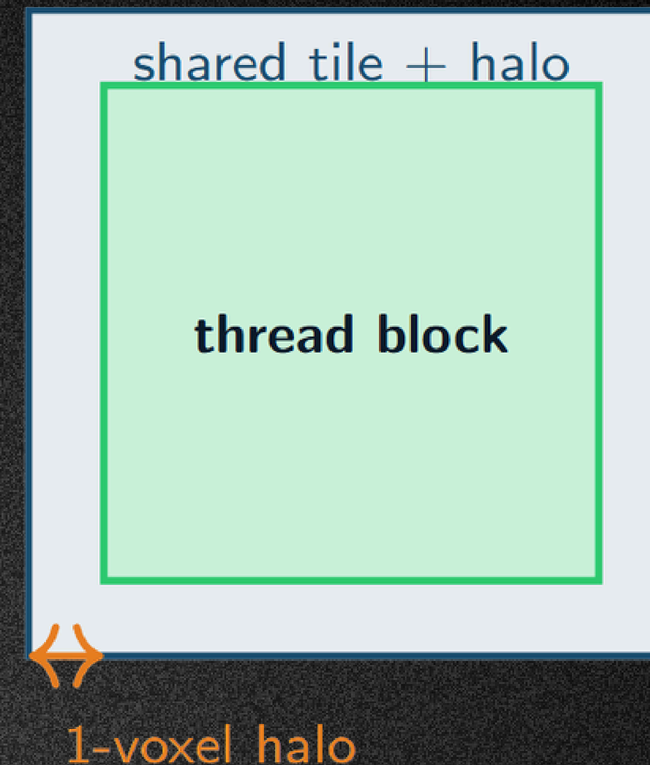
- 3-D block 8 × 8 × 4 = 256 threads
- Grid 64 × 64 × 91 blocks

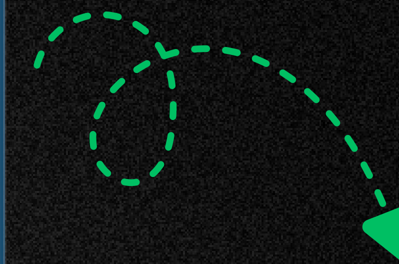
CONSTANT MEMORY

27 Gaussian weights pre-computed on CPU, uploaded via cudaMemcpyToSymbol.

SHARED MEMORY

Each block loads a (8+2) × (8+2) × (4+2) tile with 1-voxel halo ⇒ replaces 27 global reads per voxel with shared memory.





Global memory traffic reduced ≈ 27× for stencil neighbours.

Kernel 2: Gaussian Blur

‘kernel_gaussian_blur’

STRATEGY

Ray marching → one thread per output pixel.

LAUNCH CONFIG

- 2-D block 16 × 16 = 256 threads
- Grid 32 × 32 blocks
- Each thread marches 2 × r steps

$$r = \lfloor \sqrt{512^2 + 512^2 + 361^2} \div 2 \rfloor + 1 = 405$$

```
__constant__ float c_R[9];

// For pixel (x , y), march along w:
float u = x - N / 2.f , v = y - N / 2.f;
uint16_t maxV = 0;

for ( int w = -r ; w < r ; w ++) {
    float vx = c_R[0] * u + c_R[1] * v + c_R[2] * w + N / 2.f;
    // ... sample volume ...
    if ( val > maxV ) maxV = val;
}
image [ y * N + x ] = maxV;
```

No inter-thread communication.

CONSTANT MEMORY

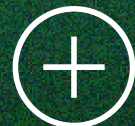
Rotation matrix R (9 floats) broadcast to all warps with zero bandwidth cost.

Kernel 3: Rotated MIP

‘kernel_mip’

4.

Innovation: Multi-Frame Video

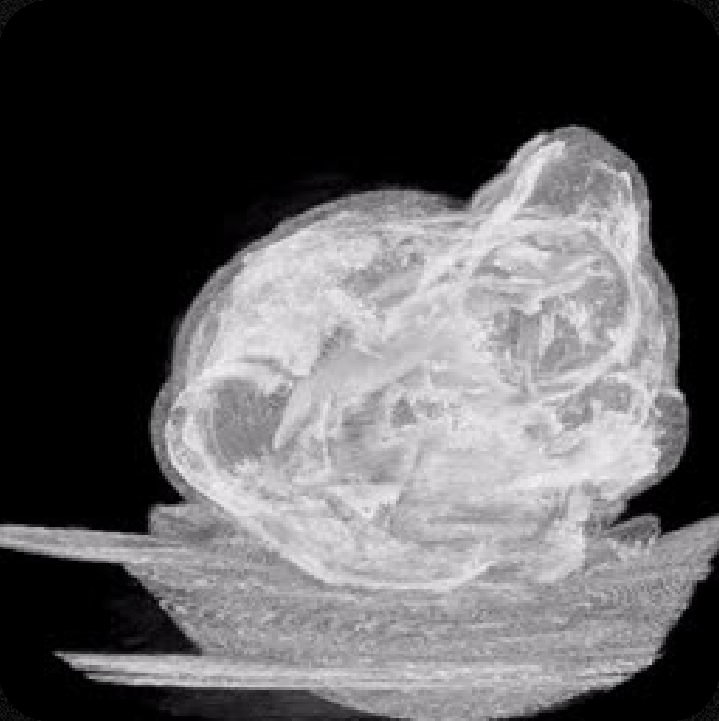


Innovation: N Rotations in One Kernel Call

IDEA

extend the 2-D output grid with a third frame dimension.

- Threshold + blur run **once**; blurred volume reused by all N frames
- N rotation matrices pre-computed on CPU, uploaded as a single flat array
- ffmpeg encodes the PGM frames into MP4/GIF

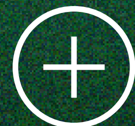


./bunnyMIP --frames 360

	kernel_mip	kernel_mip_multi
Grid	(32,32,1)	(32,32,N)
Frame	const c_R	blockIdx.z
Output	512^2	$N \times 512^2$

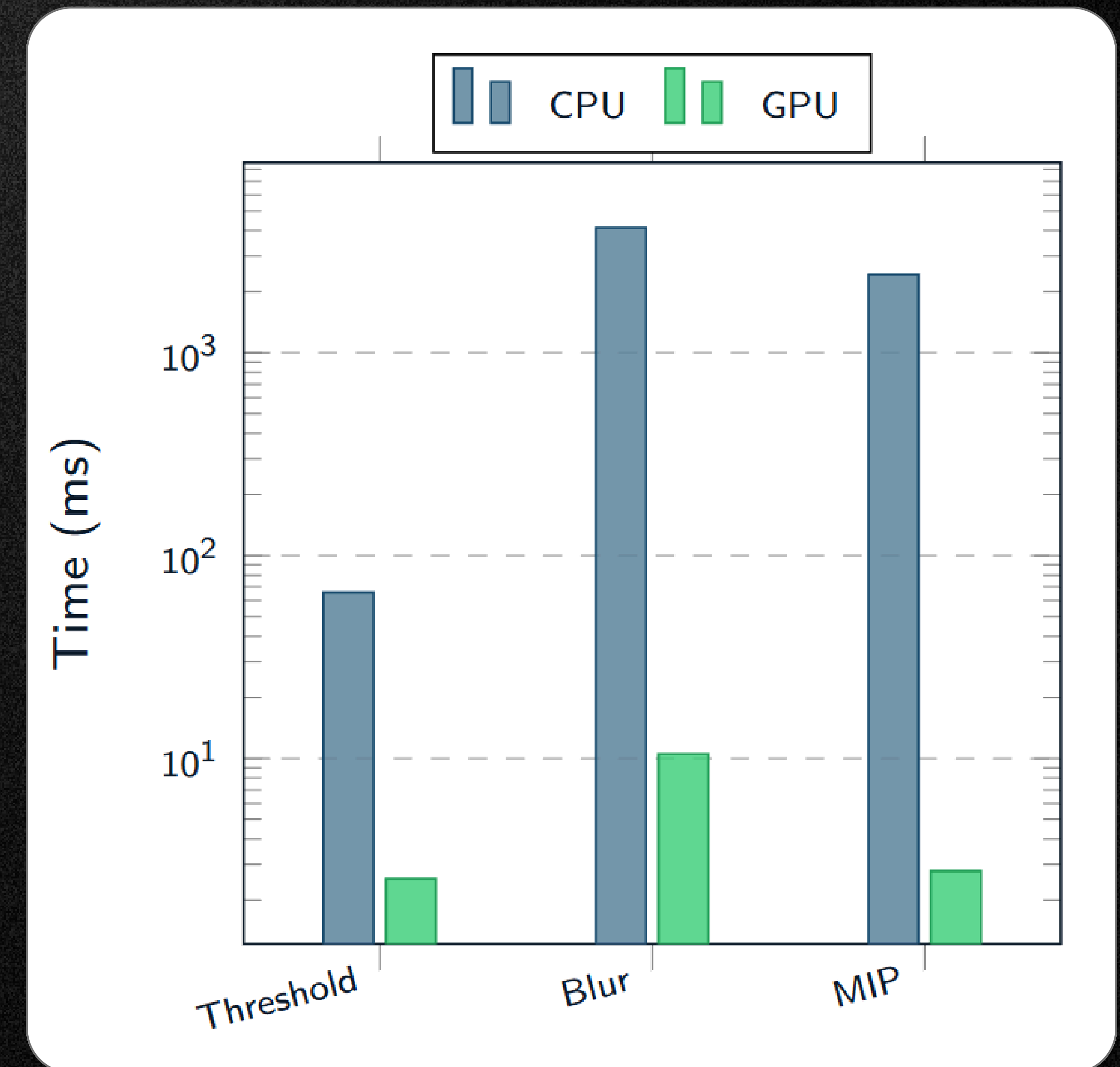
5.

Speedup Analysis



Per-kernel Speedup

Stage	CPU	GPU	Speedup
Threshold	65.85	2.55	**25.8x**
Gaussian Blur	4128.93	10.51	**392.9x**
MIP Projection	2429.63	2.79	**870.8x**
Total	**6624.41**	**15.85**	**≈418x**

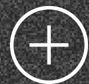


End-to-end

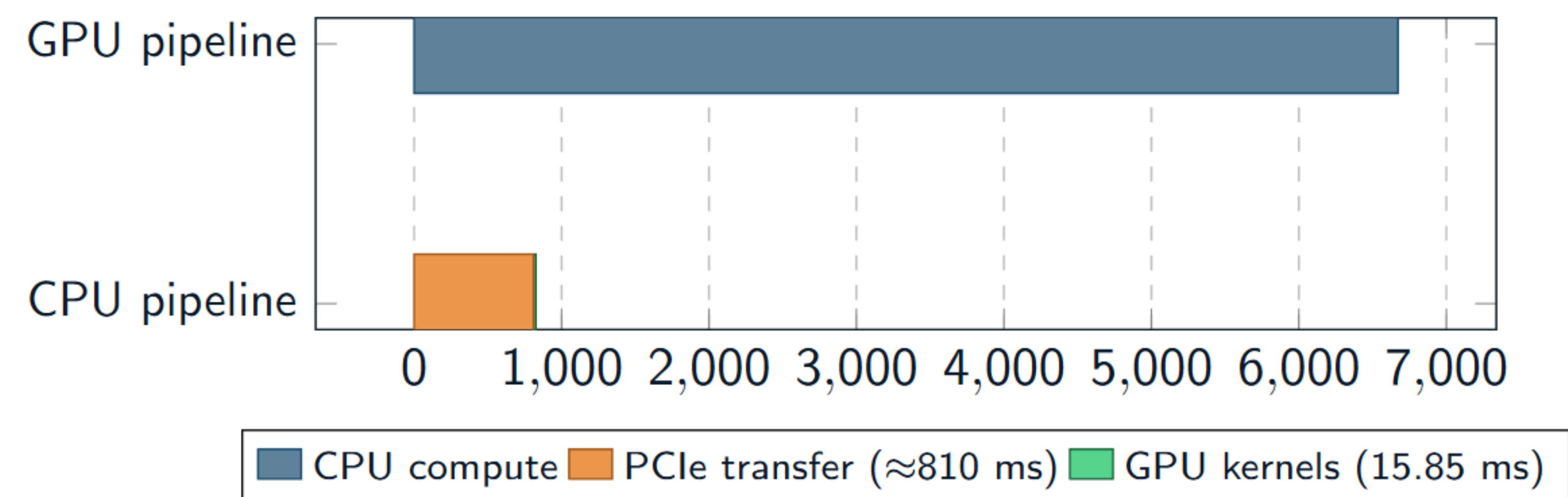
CPU: 6671 ms | **GPU:** 826 ms

Speedup: **8.08x**

Why End-to-End ≠ Kernel Speedup?

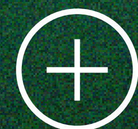


- PCIe transfer of 190 MiB volume ≈ **810 ms** → dominates GPU total time
- In video mode (**--frames N**), transfer paid **once** for all N frames ⇒ effective speedup approaches ≈ **418x**



6.

Demo & Results



SINGLE-FRAME MODE

- CPU + GPU pipelines run in parallel
- Pixel-by-pixel comparison (tolerance ± 2)
- Outputs: bunnyMIP cpu.pgm and bunnyMIP gpu.pgm

VIDEO MODE

`./bunnyMIP --frames 360`

- 360 PGM frames written to output/
- ffmpeg encodes to MP4/GIF

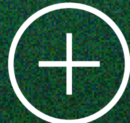
Output & Validation

✓ CPU output ≡ GPU output

pixel diff ≤ 2 on > 99.9% of pixels

7.

Conclusion



SUMMARY

- ✓ 3 CUDA kernels: threshold, Gaussian blur, MIP
- ✓ Shared memory tiling on blur (≈ 27× fewer global reads)
- ✓ Constant memory for weights and rotation matrix
- ✓ Innovation: N frames in one kernel call + ffmpeg
- ✓ End-to-end speedup 8x; kernel-only 418x
- ✓ Full CLI (--threshold, --sigma, --yaw/pitch/roll, --frames)

Key takeaway

The bottleneck is PCIe transfer, not computation.

Keeping the volume resident on the GPU (video mode) amortises the transfer cost and reveals the true GPU throughput.

References

1. Stanford University Computer Graphics Laboratory, The Stanford volume data archive, http://graphics.stanford.edu/data/voldata/

2. NVIDIA Corporation, CUDA C++ Programming Guide (Legacy), https://docs.nvidia.com/cuda/cuda-c-programming-guide/

3. NVIDIA Corporation, CUDA C++ Best Practices Guide, https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/

4. FFmpeg Project, FFmpeg Documentation, https://ffmpeg.org/documentation.html

Thank you!

QUESTIONS?