StarTrailCUDA: GPU-Accelerated Rendering of Night-Sky Star Trails Video

Team Members: Zijun Yang <zijuny@andrew.cmu.edu> and Jiache Zhang <jiachez@andrew.cmu.edu>

Project Home Page: https://blog.zjyang.dev/StarTrailCUDA/

https://www.bilibili.com/video/BV1Q64y1a7FE/?share_source=copy_web&vd_source=248bf19a901960bb7bbfb1705c664b9c&t=79

SUMMARY

We will build a GPU-accelerated rendering pipeline that converts a large sequence of fixed-camera night-sky photographs into a star-trail time-lapse video using CUDA. Our implementation will parallelize pixel-level operations across thousands of GPU threads, focusing on efficient kernel design, memory hierarchy utilization, and overlapping data transfers with computation to overcome I/O and PCIe bandwidth constraints.

BACKGROUND

Star-trail videos differ fundamentally from conventional time-lapse photography in their requirement for per-frame image accumulation. While time-lapse simply plays back captured frames at an accelerated rate, star-trail videos must composite each new frame onto a growing accumulation to create visible star motion trails. Based on the maximum number of images in the stack, three generation methods exist: unlimited stacking where all historical frames contribute indefinitely, limited stacking where only the most recent N frames contribute, and limited stacking of recent N frames with linear luminance fading.

```
// Method 1: Unlimited stacking (no fading)
A = load_image(frame_0)
save_image(A, output_0)
for t = 1 to T-1:
    F = load_image(frame_t)
    for each pixel (x,y):
        A[x,y] = max(A[x,y], F[x,y])
    save_image(A, output_t)

// Method 2: Limited stacking (recent N frames, no fading)
window_size = N
buffer = circular_buffer(N)
for t = 0 to T-1:
    F = load_image(frame_t)
    buffer.push(F)

    A = zeros()
    for each frame in buffer:
        for each pixel (x,y):
            A[x,y] = max(A[x,y], frame[x,y])
    save_image(A, output_t)

// Method 3: Limited stacking with linear luminance fading
window_size = N
buffer = circular_buffer(N)
weights = compute_linear_weights(N) // [1.0, (N-1)/N, ..., 1/N]
for t = 0 to T-1:
    F = load_image(frame_t)
    buffer.push(F)

    A = zeros()
    for i = 0 to buffer.size()-1:
        weight = weights[i]
        frame = buffer[(i + t) % N]
        for each pixel (x,y):
            weighted_pixel = frame[x,y] * weight
            A[x,y] = max(A[x,y], weighted_pixel)
    save_image(A, output_t)
```

Although mature and simple tools exist for creating static star-trail images, producing star-trail videos remains technically challenging for average users, motivating our GPU-accelerated solution. This workload is naturally data-parallel across pixels, as each pixel's new value depends only on its own history and the current frame's pixel, not on neighboring pixels. This maps directly to CUDA threads operating on independent pixels. The per-pixel computations are simple arithmetic operations, making them ideal for GPU acceleration.

THE CHALLENGE

The primary challenge stems from the sheer scale of the problem. We target high-resolution astrophotography exceeding 10 million pixels per frame: a two order of magnitude increase from our previous circle rendering assignment. This project is substantially more compute-intensive and vastly more memory-intensive. The limited stacking method exemplifies this: maintaining a sliding window of N frames requires buffering multiple high-resolution images simultaneously, then iterating through all N frames for every output timestep. With N potentially

reaching hundreds, we face memory footprints exceeding several gigabytes, far surpassing the 48KB of fast shared memory available per streaming multiprocessor. This severe constraint forces difficult decisions about memory placement: what data merits precious shared memory, what must reside in high-latency global memory, and how to structure memory accesses to maximize coalescing. Furthermore, as N grows large, the computational cost of iterating through the frame stack becomes non-negligible relative to memory access time. The naive approach of simply loading and processing each frame in sequence risks becoming compute-bound, requiring sophisticated algorithmic optimizations and heuristics to balance the workload.

Compounding the memory challenge is the immense I/O volume. A typical night-long capture generates thousands of raw frames, each tens of megabytes, resulting in total dataset sizes approaching dozens or hundreds of gigabytes. Disk I/O speed emerges as a critical bottleneck that can easily saturate even high-performance SSDs. The challenge therefore lies in structuring the pipeline to overlap computation with I/O, ensuring the disk-CPU-GPU connection never idles while waiting for processing. We must design a streaming architecture that keeps the disk constantly reading, and the PCIe bus fully utilized. This requires careful orchestration of asynchronous memory copies and prefetching strategies to hide processing time.

RESOURCES AND PLATFORM CHOICE

We will use GHC machines with NVIDIA RTX 2080 GPUs as our primary development platform. These machines provide accessible GPU resources integrated with our course environment and enable debugging and profiling. If needed, especially given the tight disk storage quota limitations on GHC machines, we may also use AWS GPU instances.

We will write the core pipeline from scratch in C++/CUDA. For image I/O, we will use libraries like stb_image, libpng, or OpenCV with CUDA support.

We will prepare input data in two ways: artificially created star field images with parameterized star positions, brightness, and motion for easy development like the CUDA circle rendering assignment, and real night-sky time-lapse sequences from astronomy observatories and astrophotography communities. For output video generation, we will use Python or ffmpeg to convert image sequences into videos. We may optionally leverage NVIDIA Performance Primitives (NPP) for optimized image operations.

GOALS AND DELIVERABLES

Plan to Achieve:

25% Goal (Pipeline Infrastructure): By the end of Week 1 (Nov 23), we will have a complete serial pipeline that can import PNG/JPEG images, output processed frames, and convert frame sequences into videos using ffmpeg. We will implement a pseudo star image generator that creates parameterized star fields with controlled positions, brightness, and motion vectors. We will also source real night-sky photograph datasets. This infrastructure provides the foundation for algorithm development and correctness validation.

75% Goal (Basic GPU Implementation): By the end of Week 2 (Nov 30), we will implement functional CUDA kernels for unlimited stacking (no fading) and limited stacking with linear luminance fading. The implementation will use synchronous memory transfers and basic global memory access. We will also profile the program, find bottlenecks and propose ways to achieve potential performance improvements.

100% Goal (Optimized Real-World Pipeline): By the end of Week 3 (Dec 7), we will deliver a polished CUDA implementation with optimized memory access patterns, supporting both unlimited stacking and limited stacking with linear luminance fading. The pipeline will work on real night-sky photographs.

Hope to Achieve:

125% Goal (Additional Stacking Mode): If ahead of schedule in Week 4, we will implement the limited stacking (recent N frames, no fading) method using a circular buffer on the GPU. This mode composites only the most recent N frames without luminance fading. This requires more complex memory management.

150% Goal (Advanced Overlapping): As our stretch goal, we will implement CUDA streams to overlap PCIe data transfers with kernel computation, hiding I/O latency.

Team Division: Honestly we will work together on most parts, especially during the optimization phase (which should be the heaviest part), but here's a rough division. Zijun Yang will be responsible for pipeline development (synthetic data generation, image I/O integration, video output generation) and the limited stacking with linear luminance fading method. Jiache Zhang will be responsible for baseline development and the unlimited stacking (no fading) method. Both members will collaborate on algorithm design, optimization, debugging, and final performance analysis.

PLATFORM CHOICE

CUDA on NVIDIA GPUs is ideal for this workload because the problem exhibits massive data parallelism across pixels, with each pixel operation being independent. GPUs provide thousands of concurrent threads that can process millions of pixels simultaneously, far exceeding CPU core counts. The availability of GHC GPU workstations provides accessible development and profiling tools, making CUDA the natural choice for accelerating this image processing pipeline.

SCHEDULE

Week 1 (Nov 17-Nov 23): We will implement the serial reference pipeline capable of importing PNG/JPEG images, outputting processed frames, and generating videos via ffmpeg. Concurrently, we will develop a parameterized pseudo star image generator with controlled positions, brightness, and motion vectors while sourcing real night-sky photograph datasets. This week also includes designing CUDA kernel specifications, thread block configurations, and memory layout strategies for both unlimited and limited stacking algorithms.

Week 2 (Nov 24-Nov 30): We will implement functional CUDA kernels for unlimited stacking (no fading) and limited stacking with linear luminance fading, utilizing synchronous memory transfers and basic global memory access. The host code for memory management and pinned memory allocation will be developed alongside integrated timing instrumentation to separate I/O, transfer, and compute times. We will profile the application to identify bottlenecks and propose concrete performance improvements.

Week 3 (Dec 1-Dec 7): We will deliver a polished CUDA implementation with optimized memory access patterns for coalescing, supporting both unlimited and limited stacking with linear luminance fading. The pipeline will process real night-sky photographs, validating the system on authentic data. Systematic performance experiments will be conducted on extensive synthetic frame sequences at multiple resolutions, collecting speedup and time-breakdown metrics to quantify optimization gains.

Week 4 (?-Dec 8): If ahead of schedule, we will implement the advanced limited stacking mode using a circular buffer on the GPU to composite the most recent N frames without fading, and add CUDA streams to overlap PCIe transfers with kernel computation. Comprehensive experiments will compare all trail modes and parameter settings across both synthetic and real datasets. We will generate final star-trail videos, prepare performance visualizations for the poster, and complete the final project report integrating all experimental results and analysis.

Team Division from previous section:

> **Team Division**: Honestly we will work together on most parts, especially during the optimization phase (which should be the heaviest part), but here's a rough division. Zijun Yang will be responsible for pipeline development (synthetic data generation, image I/O integration, video output generation) and the limited stacking with linear luminance fading method. Jiache Zhang will be responsible for baseline development and the unlimited stacking (no fading) method. Both members will collaborate on algorithm design, optimization, debugging, and final performance analysis.