Numerical Optimization Method for Satellite Constellation Control in Area Surveying Problems Using a GPU-Accelerated Simulator

Pavel Andreevich Chertovskikh Andrey Mikhailovich Boiko Ilya Yurievich Popov

ITMO University
Kronverksky Pr. 49, 197101 St. Petersburg, Russia
{pachertovskikh, amboiko, iupopov}@itmo.ru

SpacSec 2026
International Conference on Cyber Warfare, Security & Space Computing
26–28 Oct 2026

Presentation Outline

1. Remote Sensing: Area Surveying vs. Point Set Surveying
2. GPU Parallelism in Aerospace Mission Planning
3. Mathematical Model & Simulator Demands
4. Simulator Architecture (SVK & RSO)
5. Proof-of-Concept Optimization Benchmark & Results
6. Conclusion & Future Work

Remote Sensing: Point vs. Area Surveying

Core Focus: Area Target Observation (ATO)

- **Point Target Observation (PTO):** Classical scheduling of isolated target coordinates using precomputed Visible Time Windows (VTW).
- **Area Target Observation (ATO):** Continuous, dynamic surveying of extended geographical regions.
- **ATO Subsumes PTO:** Our continuous formulation naturally encompasses both extended geographical areas and localized point targets.

The Impact of Mega-Constellations

The rise of modern mega-constellations (hundreds of smallsats, e.g., Planet FLOCK) makes the ATO problem both operationally more relevant and computationally much more complex.

Why PTO Approaches Fail for Area Surveying

Classical PTO formulates observation planning as an NP-hard graph problem [1]:

Grid Discretization → Combinatorial Explosion [2]

- Continuous territory is divided into a discrete grid of independent target points subjected to classical PTO solvers.
- Graph-visiting solutions scale exponentially with the number of vertices.
- High-resolution grids (millions of nodes) trigger a **combinatorial explosion**.

GPU Parallelism in Aerospace Mission Planning

- Utilizing GPU compute is a novel direction in **aerospace mission planning**.
- **SGP4 & *jaxsgp4*:** SGP4 is the standard analytical model for propagating LEO satellite states [3]. Recently, ***jaxsgp4*** [4] achieved a **1500× GPU speedup** over C++ CPU code for propagating mega-constellations (e.g., Starlink).

The Research Gap

- While SGP4 propagation handles satellite trajectories, evaluating geometric surface coverage remains an research challenge.
- 2D texture shortcuts like Mercator Projection Superposition (MPS) [5] introduce map distortions and ignore flat CMOS sensor perspective.
- **To our knowledge, no prior work has thoroughly applied full 3D GPU parallel integration to continuous area survey optimization.**

Mathematical Model (1/3): Coverage Quality Functional

Defining Continuous Area Coverage as an Optimization Problem:

Terminal Quality Functional (J_T)

$$J_T = \int_S w(r)\,\alpha(T, r)\,dr \longrightarrow \min$$

Definitions:

- S: Target geographical survey region on the Earth's surface.
- r: Surface coordinate vector.
- T: Terminal time of the observation window.
- $w(r) \geq 0$: Priority weight of coordinate r.
- $\alpha(t, r) \in [0, 1]$: **Measure of data incompleteness** (*coverage gap*) at coordinate r and time t ($\alpha = 1$: unobserved; $\alpha = 0$: fully surveyed).

Mathematical Model (2/3): Dynamics of Coverage Gap α

Time Evolution of Surface Data Incompleteness:

Differential Coverage Equation (ODE)

$$\dot{\alpha}(t,r) = \epsilon(t,r) - \alpha(t,r)\Big(\epsilon(t,r) + \Omega(q,r)\,\upsilon(t,r)\Big)$$

Definitions:

- $\epsilon(t,r) \geq 0$: Natural data obsolescence (degradation) rate over time.
- $\upsilon(t,r) \in [0,1]$: Visibility factor accounting for solar illumination and cloud cover at coordinate r and time t.
- $q(t)$: Combined state of both satellites and their cameras.
- $\Omega(q,r) \geq 0$: **Information flow density** (per area patch) at coordinate r with constellation state q.

Mathematical Model (3/3): Information Flow Ω & Jacobian

Evaluating $\Omega(q, r)$ from Perspective Camera Physics:

1. **Uniform Sensor Distribution:** We assume information flow is distributed uniformly across each flat CMOS camera matrix.
2. **Jacobian Projection:** Using the analytical Jacobian determinant J_i derived from camera i's perspective projection matrix, we find the impact of a selected camera at surface coordinate r:

$$dS_{NDC,i} = J_i \, dS$$

3. **Superposition Across Constellation:** Summing individual camera impacts weighted by each sensor's specific strength $\Omega_{\text{strength},i}$:

$$\Omega(q, r) = \sum_{i=1}^{N} \Omega_{\text{strength},i} \, J_i(q_i, r)$$

From Mathematical Model to Simulator

The Need for a High-Performance Simulator:

- The proposed mathematical model holds standalone theoretical value.
- However, once we start proposing **optimization algorithms**, we need a testbed to evaluate and compare them — thus the necessity for a numerical simulator.

Technical Demands for the Simulator:

- **Grid Resolution:** Sub-3 km node spacing ($M \geq 57$M cells).
- **Constellation Size:** Simultaneous evaluation of up to $N = 512$ satellites.

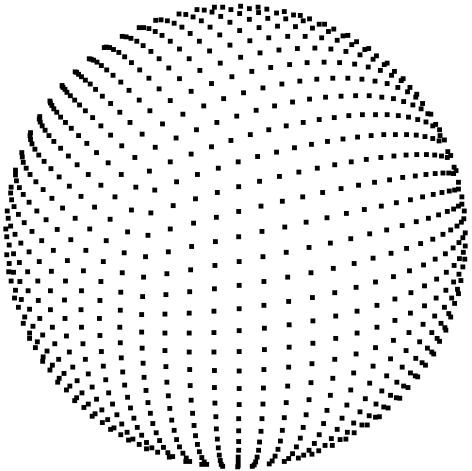
The Ultimate Goal: Faster-Than-Real-Time Optimization

High simulation throughput is essential for **operational viability**: the entire optimization cycle (thousands of simulation runs) needs to finish before the satellites move in real life.

Simulator Architecture: Equiangular Cube-Sphere Grid

Why Cube-Sphere Topology [6]?

- Eliminates polar singularities and reduces geometric aberrations compared to Lat-Lon grids.
- **Flexible Mesh Sizing:** Unlike an icosphere — which forces doubling node counts at each subdivision — the cube-sphere allows arbitrary edge resolution.
- Provides a practical compromise between geometric complexity and precision.



Cube-sphere mesh with an edge resolution of 20 nodes.

Simulator Architecture: Economical Memory Bandwidth

Memory Bandwidth is the Primary Performance Bottleneck:

- Modern GPUs have massive compute throughput, but transferring states for millions of grid cells every time step saturates the global VRAM bus.
- To achieve high occupancy, we must be strictly economical with memory bandwidth.

In-Place Backward (Implicit) Euler Integration

$$\alpha_{n+1} = \frac{f \cdot \alpha_n + \epsilon}{f + \epsilon + \Omega \cdot \upsilon}, \quad f = \frac{1}{\Delta t}$$

- Unconditionally stable for arbitrary time steps Δt and sensor strengths Ω.
- Updates state vector α **in-place**, eliminating the need to store previous states.

Core Architecture: Shared Workgroup L1 Cache

Leveraging On-Chip Workgroup Memory (M mesh cells, N satellites):

- **Compact Satellite Packing:** Camera/satellite parameters are compressed to 80 bytes per agent ($N = 512 \implies$ 40 KB total).
- **Workgroup & Shared L1 Cache:** In our Slang compute shader [7], threads are organized into 256-thread workgroups. Before evaluating surface cells, all threads in a workgroup cooperatively preload the entire N-satellite array into fast on-chip **Shared L1 Cache**.

VRAM Access Complexity

- **Naive Global Fetch ($\mathcal{O}(M \cdot N)$):** Cycle over every cell for every satellite.
- **Shared L1 Cache ($\mathcal{O}(\mathrm{M} + \mathrm{N})$):** Load every satellite, then cycle over every cell.

Software Implementation: SVK Engine & RSO

1. SVK (Simple VulKan) Engine

- Resource-oriented C++20 engine for scientific computation with optional rudimentary visualization.
- Built strictly on the **RAII** idiom using `Vulkan-Hpp` (`vk::raii`).
- **Command Baking:** *"Record once — execute repeatedly"*, saving CPU bandwidth by repeatedly submitting the same commands.

2. RSO (RemoteSensingOptimizer)

- Built on top of **SVK**, utilizes the renowned `libsgp4` library [3], and downloads real TLE orbital data from **CelesTrak**.
- Implements state packing and the L1-cached coverage integration pipeline with baked compute command buffers.

RSO & SVK: `https://github.com/alwaysmad/RemoteSensingOptimizer`
Vulkan-Hpp: `https://github.com/KhronosGroup/Vulkan-Hpp`
libsgp4: `https://github.com/dnwrnr/sgp4`

Benchmarking Performance: PoC Optimization Problem

Structural Subset Selection Problem

Given an initial constellation U_{full} of N nadir-pointing satellites, find a subset U_{opt} of size K that minimizes coverage incompleteness $J_T(U)$.

Why This Problem for Our Benchmark?

- **Simple yet Realistic:** Operational constellations like Planet FLOCK rely on strict nadir-pointing scanning profiles.
- **Fitting Throughput Stress-Test:** Requires $\mathcal{O}(N^2)$ full 3D numerical simulations, making it a fitting Proof-of-Concept to benchmark whether our simulator is fast enough for iterative optimization.

Modified Backward Greedy Optimization Algorithm

*Remove the least impactful satellite at every step; any satellite with **zero impact** is removed immediately.*

Algorithm Steps ($U_{current} \leftarrow U_{full}$):

1. Compute current baseline coverage functional: $J_{current} \leftarrow J_T(U_{current})$.
2. Loop over each satellite $q_i \in U_{current}$ and run RSO for $U_{test} = U_{current} \setminus \{q_i\}$:
   - **Immediate Drop:** If $J_T(U_{test}) == J_{current}$, drop q_i right away and start the next step.
   - Otherwise, track the satellite q_{drop} whose removal causes the smallest increase in J_T.
3. Remove q_{drop} and repeat until $|U_{current}| = K$.

Experimental Setup: Planet FLOCK over France

Benchmark Scenario [8]:

- **Constellation:** Planet FLOCK ($N = 111$ SuperDove satellites), 3-hour window (20 Jun 2026, 10:00–13:00 UTC).
- **Mesh Resolution:** Edge resolution of 4000 nodes (96M cells in a full mesh), restricted here to **Metropolitan France**.
- **Illumination Cutoff:** FLOCK satellites are only active when the solar elevation angle is at least 10°:

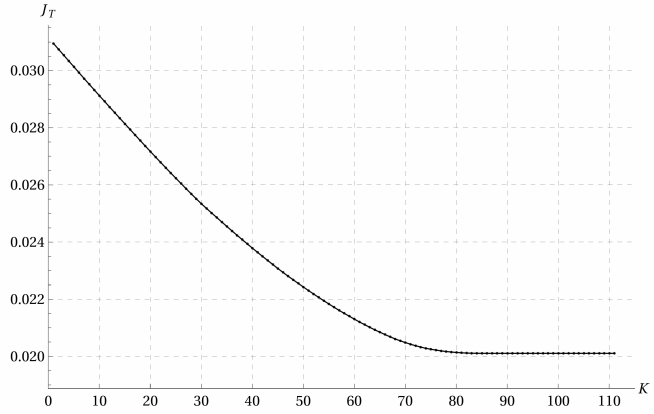
$$h_{\odot} \geq h_{min} = 10^{\circ}$$

- **Hardware:** NVIDIA Tesla P100 GPU.



Final coverage gap state α over France.

Results (1/3): Coverage Incompleteness vs. Constellation Size



Optimization Outcome:

- Identified **26 redundant satellites**.
- Removing agents below 85 induces a monotonic penalty in J_T.

Results (2/3): Hardware Utilization Efficiency

Profiling Across 4,960 Full 3-Hour Simulation Runs:

99% GPU Compute Utilization

- GPU remains completely saturated throughout execution and never stalls waiting for data or commands from the CPU.

31% Single-Core CPU Load

- Runs on just **1 CPU core**, which is practically idling while waiting on GPU synchronization.
- SGP4 propagation is performed on the CPU.
- Command baking saves CPU load.

Results (3/3): Optimization Speed vs. Real Satellite Motion

Benchmark Timing Summary

- **Total Simulator Invocations:** 4,960 full 3-hour numerical integrations.
- **Single 3-Hour Simulation Time:** ≈ 8.58 seconds.
- **Total Optimization Runtime: 11 hours, 49 minutes, 34 seconds** (for a **3-hour** real-life pass).

Performance Assessment:

- We are still slightly behind real-time operational speed ($\sim$ 12 h compute vs. 3 h orbital window).
- Consequently, the current throughput is **not yet fast enough for heavy gradient-based optimization** (such as continuously optimizing camera roll angles).
- However, this gap is small ($\sim 4\times$) and can readily be bridged with further optimizations.

Conclusion & Future Work

Conclusion

We presented a new approach to numerical ATO optimization, comprising a continuous mathematical model and a GPU-accelerated simulator (SVK/RSO). Our performance benchmarking indicates that our approach is promising, warranting future work.

Future Work to Bridge the Speed Gap:

1. **Aggressive Data Packing:** Switching to **FP16 coordinates** (16-bit half-precision floats) to halve memory bandwidth pressure.
2. **Compute Cycle Optimization:** Implementing **meshlet culling** to discard out-of-frustum surface regions before per-cell evaluation.
3. **Advanced Optimization Methods:** Proposing specialized/modified **gradient-based optimization algorithms** for agile attitude control (roll angles).

References

1. Constraint Programming Committee (ed.): *Proceedings of the 31st International Conference on Principles and Practice of Constraint Programming (CP 2025)*. Springer, Glasgow, UK (2025).
2. Xu, Y., Liu, X., He, R., Chen, Y.: Multi-satellite scheduling framework and algorithm for very large area observation. *Acta Astronautica* **167**, 93–107 (2020).
3. Werner, D.: SGP4: C++ implementation of most recent NORAD PROPAGATION Spacetrack Report No. 3. `https://github.com/dnwrnr/sgp4`
4. Priestley, C., Handley, W.: jaxsgp4: GPU-accelerated mega-constellation propagation with batch parallelism. *arXiv preprint arXiv:2603.27830* (2026).
5. Feng, G., Lv, L., Li, W.: Mercator Projection Superposition: A Computationally Efficient Alternative to Grid-Based Coverage Analysis for LEO Mega-Constellations. *Applied Sciences* **15**(19), 10610 (2025).
6. Balaji, V., Liang, Z.: Gridspec: A standard for the description of grids used in Earth System models. *NOAA*, Princeton, NJ (2007).
7. Slang: A shading language with advanced modularity and extensibility. `https://shader-slang.org/`
8. Planet Documentation: PlanetScope Imagery Product Specifications. `https://docs.planet.com/data/imagery/planetscope/`

Numerical Optimization Method for Satellite Constellation Control in Area Surveying Problems Using a GPU-Accelerated Simulator

Pavel Andreevich Chertovskikh, Andrey Mikhailovich Boiko, Ilya Yurievich Popov
ITMO University, St. Petersburg, Russia
{pachertovskikh, amboiko, iupopov}@itmo.ru

SpacSec 2026 — International Conference on Cyber Warfare, Security & Space Computing

Open-Source Code Repository

The source code for the SVK graphics engine, Slang compute shaders, and RemoteSensingOptimizer is publicly available at:

https://github.com/alwaysmad/RemoteSensingOptimizer