Montecarlo-DGL: A C++20 Framework for Stochastic Integration and Metaheuristic Optimization on Constructive Solid Geometry Domains

Domenico De Giorgio, Giacomo Merlo, Leonardo Pelorosso
Advanced Methods for Scientific Computing

January 2026

Abstract

This report presents the design, implementation, and validation of **Montecarlo-DGL**, a computational framework developed to solve integration and optimization problems over high-dimensional, non-convex domains defined by Constructive Solid Geometry (CSG). The project addresses the limitations of deterministic quadrature methods in high dimensions (Curse of Dimensionality) by implementing Monte Carlo and Markov Chain Monte Carlo (MCMC) techniques. Furthermore, the library integrates metaheuristic optimizers (PSO, GA) to solve inverse engineering problems. Special attention is given to the implementation of thread-safe reproducibility via deterministic hashing and to the handling of stochastic noise in gradient-free optimization, validated through aerospace case studies including drone arm balancing and stress-proxy minimization.

1 Introduction

Numerical integration over complex domains is a fundamental problem in computational physics and engineering. While deterministic methods (e.g., Simpson's rule, Gaussian quadrature) are efficient in low dimensions ($D \leq 3$), their computational cost scales exponentially with dimensionality D. Monte Carlo (MC) methods offer a convergence rate of $O(N^{-1/2})$ independent of dimensionality, making them the standard for high-dimensional analysis.

However, applying MC methods to engineering geometries (like a drone arm) presents specific challenges:

1. **Complex Boundaries:** The domain is often a boolean combination of primitive shapes (Union, Intersection, Difference), making analytical bounds calculation impossible.
2. **Optimization on Noisy Functions:** When the objective function is itself the result of a stochastic integration, standard optimizers fail due to numerical noise.

This project leverages C++20 features (Concepts, Templates) to build a flexible, high-performance library that solves these issues without the overhead of mesh generation.

2 Mathematical Framework and Architecture

2.1 Domain Representation via CSG

Unlike Finite Element Analysis (FEA), which requires discretizing volume into meshes, we use an implicit representation based on Constructive Solid Geometry (CSG). A point $\mathbf{p} \in \mathbb{R}^D$ is considered "inside" the domain Ω based on a boolean evaluation tree.

Let Ω_A and Ω_B be two primitives (e.g., a HyperRectangle and a HyperSphere). The union is defined as:

$$\mathbb{I}_{\Omega_A \cup \Omega_B}(\mathbf{p}) = \mathbb{I}_{\Omega_A}(\mathbf{p}) \vee \mathbb{I}_{\Omega_B}(\mathbf{p}) \tag{1}$$

where $\mathbb{I}$ is the indicator function returning true/false. This allows exact boundary representation and requires negligible memory compared to storing mesh vertices.

2.2 Integration Strategies

The core estimator used is the classic Monte Carlo estimator:

$$I = \int_{\Omega} f(\mathbf{x}) d\mathbf{x} \approx V_{\text{box}} \cdot \frac{1}{N} \sum_{i=1}^{N} [f(\mathbf{x}_i) \cdot \mathbb{I}_{\Omega}(\mathbf{x}_i)] \tag{2}$$

where samples $\mathbf{x}_i$ are drawn uniformly from the bounding box V_{box}.

To improve convergence on "sparse" domains (where $V_{\Omega} \ll V_{\text{box}}$), we implemented **Importance Sampling** and **Metropolis-Hastings (MCMC)**. The latter constructs a Markov Chain that explores the domain volume adaptively, effectively concentrating samples inside the geometry rather than wasting them in the empty bounding box space.

2.3 Software Architecture and Design Patterns

The system follows a **layered modular design** to separate geometric definitions from the orchestration logic (Figure 1). Key architectural choices include:

1. **Template-Driven Dimensionality:** All primitives are templated on '<int dim>'. This enables compile-time loop unrolling and the use of 'std::array' for stack allocation, eliminating heap churn during the sampling of millions of points.

2. **Strategy Pattern for Sampling:** Proposal distributions (Uniform, Gaussian, Mixture) implement a common interface. This allows runtime switching between variance-reduction strategies without modifying the core integrator logic.

3. **Observer Pattern:** Visualization is handled via callbacks in the optimizers, allowing frame capture for animation (via Gnuplot) without coupling the optimization logic to the I/O system.

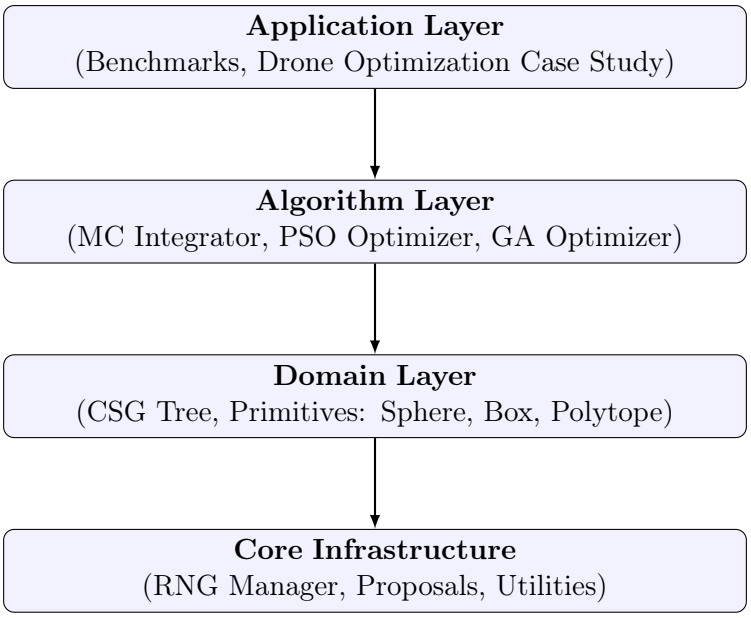


Figure 1: System Architecture: The separation allows swapping integration backends without changing the domain definition.

3 Class Specifications and Mathematical Models

3.1 Domains

Class: `Hypersphere<dim>`

Represents an n-dimensional ball centered at $\mathbf{c}$ with radius R. The analytical volume, implemented using `std::tgamma`, is:

$$V_{\text{sphere}}(R, D) = \frac{\pi^{D/2}}{\Gamma(\frac{D}{2} + 1)} R^D \tag{3}$$

where $\Gamma(\cdot)$ is the Euler gamma function. The 'isInside' check is optimized as $\|\mathbf{x} - \mathbf{c}\|^2 \leq R^2$ to avoid square roots.

Class: `HyperRectangle<dim>`

Represents an axis-aligned bounding box defined by intervals $[a_i, b_i]$ for each dimension $i = 1, \ldots, D$. The volume is:

$$V_{\text{rect}} = \prod_{i=1}^{D} (b_i - a_i) \tag{4}$$

Class: `Polytope<dim>`

Represents a convex polytope defined as the intersection of F half-spaces. A point $\mathbf{x}$ is inside if it satisfies linear inequalities derived from the plane normals $\mathbf{n}_i$ and offsets d_i:

$$\Omega_{\text{poly}} = \{\mathbf{x} \in \mathbb{R}^D \mid \mathbf{n}_i \cdot \mathbf{x} + d_i \leq 0, \quad \forall i = 1, \ldots, F\} \tag{5}$$

This class relies on the implementation of the dot product in $\mathbb{R}^D$ and has a complexity of $O(F)$ per query.

Class: `HyperCylinder<dim>`

Generalizes a cylinder to D dimensions. It imposes a circular constraint on the first two dimensions (principal axes) and bounded intervals on the remaining $D-2$ dimensions (heights).

$$\text{isInside}(\mathbf{x}) \iff (x_1^2 + x_2^2 \leq R^2) \wedge (a_j \leq x_j \leq b_j, \forall j = 3 \dots D) \tag{6}$$

The analytical volume is computed as the area of the base circle multiplied by the lengths of the orthogonal sides:

$$V_{\text{cyl}} = (\pi R^2) \cdot \prod_{j=3}^{D} (b_j - a_j) \tag{7}$$

3.2 Estimators

Class: `ISMeanEstimator` **(Importance Sampling Mean Estimator)**

The `ISMeanEstimator` implements an importance sampling estimator for the mean of a function $f : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ using a proposal distribution $q(\mathbf{x})$.

Given N samples $\{\mathbf{x}_i\}_{i=1}^N$ drawn independently from q,

$$\mathbf{x}_i \sim q(\mathbf{x}),$$

the integral of f over the domain Ω is estimated as

$$\int_\Omega f(\mathbf{x})\, d\mathbf{x} = \int_\Omega \frac{f(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x})\, d\mathbf{x} \approx \hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{q(\mathbf{x}_i)}. \tag{8}$$

Only samples satisfying $\mathbf{x}_i \in \Omega$ and $q(\mathbf{x}_i) > 0$ are retained; let N_{in} denote the number of valid samples.

The estimator variance is computed from the weighted samples

$$w_i = \frac{f(\mathbf{x}_i)}{q(\mathbf{x}_i)},$$

leading to an estimate of the standard error

$$\text{stderr} = \sqrt{\frac{1}{N_{\text{in}}(N_{\text{in}} - 1)} \sum_{i=1}^{N_{\text{in}}} (w_i - \hat{\mu})^2}. \tag{9}$$

Class: `MCMeanEstimator` **(Uniform Monte Carlo Mean Estimator)**

The `MCMeanEstimator` implements the classic uniform Monte Carlo estimator for the mean of a function $f : \Omega \subset \mathbb{R}^d \to \mathbb{R}$.

A total of N sample points are drawn uniformly from the bounding box $\mathcal{B} \supset \Omega$,

$$\mathbf{x}_i \sim \mathcal{U}(\mathcal{B}), \qquad i = 1, \ldots, N,$$

and only points satisfying $\mathbf{x}_i \in \Omega$ contribute to the estimator.

The empirical mean is computed as

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i)\, \mathbb{I}[\mathbf{x}_i \in \Omega]\,, \tag{10}$$

where $\mathbb{I}[\cdot]$ denotes the indicator function. Points outside the domain are treated as contributing zero to the sum.

For numerical integration over Ω, the integral is approximated by

$$\int_{\Omega} f(\mathbf{x})\, d\mathbf{x} \approx V_{\Omega}\, \hat{\mu}, \tag{11}$$

where V_{Ω} is the volume of the integration domain.

The variance is estimated using the second moment

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^{N} \left(f(\mathbf{x}_i)\, \mathbb{I}[\mathbf{x}_i \in \Omega] \right)^2 - \hat{\mu}^2,$$

leading to the standard error

$$\text{stderr} = \frac{\hat{\sigma}}{\sqrt{N}}. \tag{12}$$

Algorithmic Properties

- Uniform sampling over a bounding box enclosing the domain.
- Domain membership enforced via an indicator function.
- Parallelized sampling and accumulation using OpenMP.
- Simple, unbiased estimator with variance decreasing as $O(N^{-1})$.

Remarks While robust and easy to implement, uniform Monte Carlo sampling becomes inefficient for high-dimensional domains or geometries with small volume-to-box ratios. In such cases, importance sampling provides superior variance reduction.

Class: `VolumeEstimatorMC` (Hit-or-Miss Monte Carlo Volume Estimator)

The `VolumeEstimatorMC` implements a hit-or-miss Monte Carlo estimator for the volume of a possibly complex domain $\Omega \subset \mathbb{R}^d$.

A total of N points are sampled uniformly from a bounding box $\mathcal{B} \supset \Omega$,

$$\mathbf{x}_i \sim \mathcal{U}(\mathcal{B}), \qquad i = 1, \ldots, N,$$

and the number of *hits*

$$N_{\text{hits}} = |\{\mathbf{x}_i \in \mathcal{B} \ : \ \mathbf{x}_i \in \Omega\}|$$

is recorded.

The domain volume is estimated as

$$\hat{V}_\Omega = V_\mathcal{B} \cdot \hat{p}, \qquad \hat{p} = \frac{N_{\text{hits}}}{N}, \tag{13}$$

where $V_\mathcal{B}$ denotes the volume of the bounding box and $\hat{p}$ is the empirical probability of a point falling inside Ω.

Since each sample corresponds to a Bernoulli trial, the variance of $\hat{p}$ is given by

$$\text{Var}(\hat{p}) = \frac{\hat{p}(1-\hat{p})}{N},$$

leading to the standard error of the volume estimate

$$\text{stderr} = V_\mathcal{B}\sqrt{\frac{\hat{p}(1-\hat{p})}{N}}. \tag{14}$$

Algorithmic Properties

- Uniform hit-or-miss sampling in a bounding box enclosing the domain.
- Unbiased estimator for arbitrary geometries and dimensions.
- Straightforward parallelization using OpenMP.
- Variance governed by the Bernoulli process associated with domain membership.

Remarks The estimator becomes inefficient for domains with a small volume-to-box ratio, as most samples are rejected. For this reason, it is primarily used as a building block for more advanced integration schemes, such as two-stage or Metropolis–Hastings-based Monte Carlo integrators.

3.3 Integrators

Class: `MCIntegrator` (Classic Monte Carlo)

Implements the standard Monte Carlo estimator. Samples are drawn uniformly from the domain's bounding box V_{box}. The integral is approximated as:

$$I \approx V_{\text{box}} \cdot \frac{1}{N} \sum_{i=1}^{N} [f(\mathbf{x}_i) \cdot \mathbb{I}_\Omega(\mathbf{x}_i)] \tag{15}$$

This class serves as the baseline for performance and accuracy comparisons.

Class: `ISIntegrator` (Importance Sampling)

Implements Variance Reduction via Importance Sampling. Instead of sampling uniformly, samples $\mathbf{x}$ are drawn from a proposal distribution $q(\mathbf{x})$ that approximates $|f(\mathbf{x})|$. The estimator becomes:

$$I_{IS} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i) \cdot \mathbb{I}_\Omega(\mathbf{x}_i)}{q(\mathbf{x}_i)}, \quad \mathbf{x}_i \sim q(\mathbf{x}) \tag{16}$$

Class: `MHIntegrator` (Markov Chain Monte Carlo)

Implements the Metropolis-Hastings algorithm to sample adaptively from the domain indicator function $\mathbb{I}_\Omega(\mathbf{x})$. A candidate $\mathbf{x}' \sim q(\mathbf{x}'|\mathbf{x}_t)$ is accepted with probability:

$$A(\mathbf{x}', \mathbf{x}_t) = \min\left(1, \frac{\mathbb{I}_\Omega(\mathbf{x}')}{\mathbb{I}_\Omega(\mathbf{x}_t)}\right) \tag{17}$$

This allows the sampler to "stay inside" the geometry, effectively concentrating samples in the region of interest.

3.4 MCMC

Class: `MetropolisHastingsSampler` (Metropolis–Hastings MCMC Sampler)

The `MetropolisHastingsSampler` generates samples from a target density $\pi(\mathbf{x})$ defined on a domain $\Omega \subset \mathbb{R}^d$ using the Metropolis–Hastings Markov Chain Monte Carlo algorithm with a symmetric random-walk proposal.

Starting from a valid initial state $\mathbf{x}_0 \in \Omega$, at each step a candidate point is proposed as

$$\mathbf{y} = \mathbf{x}_n + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_d),$$

where σ is the proposal deviation. Because the proposal is symmetric, the acceptance probability reduces to

$$\alpha(\mathbf{x}_n, \mathbf{y}) = \min\left(1, \frac{\pi(\mathbf{y})}{\pi(\mathbf{x}_n)}\right). \tag{18}$$

The proposal is accepted if $u < \alpha$, with $u \sim \mathcal{U}[0, 1]$; otherwise the chain remains at $\mathbf{x}_n$.

Proposals for which $\pi(\mathbf{y}) \leq 0$ or $\pi(\mathbf{y})$ is non-finite are automatically rejected. The sampler tracks the number of accepted moves and reports the acceptance rate

$$\text{acc\_rate} = \frac{n_{\text{accept}}}{n_{\text{steps}}},$$

which is used in practice to tune the proposal deviation for efficient exploration of the target distribution.

3.5 Optimizers

Class: `PSO` (Particle Swarm Optimization)

The swarm consists of N_p particles. At iteration t, the velocity $\mathbf{v}_i$ and position $\mathbf{x}_i$ of particle i are updated as:

$$\mathbf{v}_i^{t+1} = \omega \mathbf{v}_i^t + c_1 r_1 (\mathbf{p}_{best,i} - \mathbf{x}_i^t) + c_2 r_2 (\mathbf{g}_{best} - \mathbf{x}_i^t) \tag{19}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{20}$$

where ω is the inertia weight, c_1, c_2 are cognitive and social coefficients, and $r_1, r_2 \sim \mathcal{U}(0,1)$. To handle constraints, we implement a **damping reflection** boundary condition: if $x_{i,d}$ exits bounds, $v_{i,d} \leftarrow -0.5 v_{i,d}$.

Class: `GA` (Genetic Algorithm)

The algorithm evolves a population via:

- **Selection:** k-Tournament selection (default $k = 4$).
- **Crossover:** Uniform crossover where offspring genes come from parent A or B with 50% probability.
- **Mutation:** Gaussian perturbation added to genes with probability p_m:

$$x'_d = x_d + \mathcal{N}(0, \sigma_{mut}^2) \tag{21}$$

3.6 Proposals

Class: `UniformProposal` (Uniform Domain Proposal)

The `UniformProposal` defines a uniform proposal distribution over the integration domain $\Omega \subset \mathbb{R}^d$.

Samples are drawn uniformly from the domain,

$$\mathbf{x} \sim \mathcal{U}(\Omega), \tag{22}$$

and the corresponding probability density function is constant over Ω,

$$q(\mathbf{x}) = \frac{1}{V_\Omega}, \qquad \mathbf{x} \in \Omega, \tag{23}$$

where V_Ω denotes the volume of the domain.

Sampling Uniform sampling is implemented by drawing points uniformly from a bounding box enclosing the domain and accepting only those that satisfy $\mathbf{x} \in \Omega$. This guarantees an exact uniform distribution over Ω.

Probability Density Evaluation The probability density function is defined as a constant over the domain,

$$q(\mathbf{x}) = \frac{1}{V_\Omega},$$

and is evaluated only for points inside Ω. Evaluating the PDF outside the domain is undefined and must be avoided by the caller.

Algorithmic Properties

- Exact uniform proposal over arbitrary integration domains.
- Simple baseline proposal for importance sampling.
- Deterministic and unbiased PDF evaluation.
- Naturally compatible with both uniform Monte Carlo and importance sampling estimators.

Remarks While straightforward and robust, uniform proposals are inefficient when the integrand exhibits strong spatial variation. They are primarily used as a baseline or as components in mixture proposals, where they provide global coverage of the domain.

Class: `GaussianProposal` (Diagonal Multivariate Gaussian Proposal)

The `GaussianProposal` defines a diagonal multivariate Gaussian proposal distribution $q(\mathbf{x})$ over the full space $\mathbb{R}^d$, without any domain truncation.

Given a mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and standard deviations $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_d)$, the proposal density is

$$q(\mathbf{x}) = \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi}\,\sigma_i} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}\right), \qquad \mathbf{x} \in \mathbb{R}^d. \tag{24}$$

Sampling is performed directly from the Gaussian distribution,

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2)), \tag{25}$$

without rejection or truncation.

Domain Handling The proposal distribution is intentionally defined on the full space $\mathbb{R}^d$. Domain constraints $\mathbf{x} \in \Omega$ are *not* enforced at the proposal level. Instead, domain membership is handled explicitly by the estimator or integrator via

$$\mathbb{I}[\mathbf{x} \in \Omega],$$

ensuring that the `sample()` and `pdf()` methods remain fully coherent.

This design choice guarantees that importance weights of the form

$$w(\mathbf{x}) = \frac{f(\mathbf{x})}{q(\mathbf{x})}$$

are always mathematically consistent, even for samples falling outside the domain.

Normalization For numerical stability, the logarithm of the normalization constant is precomputed,

$$\log Z = -\frac{d}{2}\log(2\pi) - \sum_{i=1}^{d} \log \sigma_i,$$

allowing efficient evaluation of the probability density function.

Algorithmic Properties

- Diagonal multivariate Gaussian with independent components.
- Exact sampling in $\mathbb{R}^d$ without rejection.
- Fully compatible with importance sampling and MCMC estimators.
- Clear separation between proposal distribution and domain constraints.

Remarks This proposal is particularly effective when the Gaussian parameters $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ approximate the shape of the target integrand, yielding substantial variance reduction in importance sampling and improved mixing in Metropolis–Hastings algorithms.

Class: `MixtureProposal` (Mixture Importance Sampling Proposal)

The `MixtureProposal` defines a mixture proposal distribution composed of K component proposals $\{q_k(\mathbf{x})\}_{k=1}^{K}$, each implementing the `Proposal` interface.

The mixture density is defined as

$$q(\mathbf{x}) = \sum_{k=1}^{K} w_k \, q_k(\mathbf{x}), \qquad w_k \geq 0, \quad \sum_{k=1}^{K} w_k = 1, \tag{26}$$

where $\mathbf{w} = (w_1, \ldots, w_K)$ are normalized mixture weights.

Sampling Sampling from the mixture is performed in two stages:

1. Draw a component index
$$k \sim \text{Categorical}(\mathbf{w}),$$
2. Draw a sample from the selected component
$$\mathbf{x} \sim q_k(\mathbf{x}).$$

This procedure guarantees that the resulting samples are distributed according to the mixture density $q(\mathbf{x})$.

Probability Density Evaluation The probability density function of the mixture is evaluated as the weighted sum of component densities,

$$q(\mathbf{x}) = \sum_{k=1}^{K} w_k \, q_k(\mathbf{x}), \tag{27}$$

ensuring full consistency between the `sample()` and `pdf()` methods.

Design and Lifetime Semantics The mixture stores *non-owning* pointers to its component proposals. Consequently, it does not manage their lifetime: all component proposals must outlive the `MixtureProposal` instance.

Weights provided at construction are validated, normalized, and internally stored in normalized form. Invalid configurations (negative weights, null components, or zero total weight) are rejected.

Algorithmic Properties

- Flexible combination of heterogeneous proposal distributions.
- Exact sampling via categorical selection without rejection.
- Fully compatible with importance sampling and MCMC estimators.
- Clear separation between proposal composition and domain handling.

Remarks Mixture proposals are particularly effective when the target distribution exhibits multiple modes or when combining global exploration proposals with locally adaptive components. They provide a robust variance-reduction strategy in importance sampling and improve state-space coverage in Metropolis–Hastings algorithms.

4 Implementation Challenges

4.1 Parallelism with OpenMP

Monte Carlo is "embarrassingly parallel". We utilized OpenMP for multi-threading. However, standard ‘rand()‘ is not thread-safe. We employed C++11 ‘std::mt19937‘ with a separate instance per thread, preventing false sharing and race conditions.

4.2 Optimizer Tuning for Reproducibility

To ensure convergence on multimodal landscapes (such as the Rastrigin function or complex CSG interactions), specific tuning of the PSO hyperparameters was required. Table 1 details the "Hard Configuration" used for the presented results.

Table 1: PSO "Hard" Configuration for Multimodal Problems

Parameter	**Value**
Population Size	500 particles
Max Iterations	1000
Inertia Weight (ω)	0.729 (Clerc's constriction)
Cognitive Coefficient (c_1)	1.49
Social Coefficient (c_2)	1.49

5 Applied Case Studies

This section presents two end-to-end engineering applications that validate **Montecarlo-DGL** as a unified framework for (i) integration over complex domains defined by Constructive Solid Geometry (CSG) and (ii) gradient-free optimization in the presence of stochastic estimators. The case studies are deliberately chosen to highlight complementary aspects of the library: a **3D mass-properties inverse design** problem driven by Monte Carlo integration (Drone Arm), and a **2D stochastic expectation maximization** problem driven by Metropolis-Hastings (Wind Farm).

5.1 Case Study 1: Drone Arm Center-of-Mass Targeting via Spherical Void Optimization

Engineering goal

The objective is to shift the center of mass (CM) of a drone arm toward a target point $\mathbf{T} = (1.0, 0, 0)$ by carving a spherical void (hole) inside the structure. The design vector is:

$$\mathbf{z} = (h_x, h_y, h_z, r) \in \mathbb{R}^4, \tag{28}$$

where (h_x, h_y, h_z) is the hole center and r is its radius. The optimization minimizes the Euclidean CM error:

$$J(\mathbf{z}) = \|\mathbf{CM}(\mathbf{z}) - \mathbf{T}\|_2 , \tag{29}$$

subject to implicit feasibility constraints: the hole center must lie inside the body, and the hole must not remove essentially all material.

Geometry modeling with CSG domains

The drone arm is modeled as a CSG union of primitives in $\mathbb{R}^3$:

$$\Omega_{\text{drone}} = \Omega_{\text{arm}} \cup \Omega_{\text{motor}} \cup \Omega_{\text{cabin}}, \tag{30}$$

implemented through a custom domain class (`DroneArmDomain`) that inherits from `IntegrationDomain<3>`. Concretely:

- Ω_{arm} is a `HyperRectangle<3>` representing the main beam.
- Ω_{motor} is a `HyperCylinder<3>` placed at the arm tip through an offset transformation.
- Ω_{cabin} is an `optional PolyTope<3>` loaded from external hull data (points, facet normals and offsets). If the assets are not available, the cabin term is safely ignored, demonstrating graceful degradation of geometry complexity.

This approach showcases the library's core design principle: **exact implicit boundaries** without mesh generation, evaluated through fast `isInside` queries.

Monte Carlo estimation of CM under hole subtraction

The center of mass is computed from volume integrals of mass density and first moments. Assuming unit density, the continuous definitions are:

$$M(\mathbf{z}) = \int_{\Omega_{\text{drone}} \setminus \Omega_{\text{hole}}(\mathbf{z})} 1 \, d\mathbf{x}, \qquad \mathbf{m}(\mathbf{z}) = \int_{\Omega_{\text{drone}} \setminus \Omega_{\text{hole}}(\mathbf{z})} \mathbf{x} \, d\mathbf{x}, \tag{31}$$

$$\mathbf{CM}(\mathbf{z}) = \frac{\mathbf{m}(\mathbf{z})}{M(\mathbf{z})}. \tag{32}$$

In the application, these integrals are estimated by hit-or-miss sampling on a bounding box returned by `getBounds()`, accumulating only points that satisfy: (i) `domain.isInside(p)` and (ii) `p` lies outside the spherical hole. This produces a single-pass estimator for (M, m_x, m_y, m_z), which is computationally cheap enough to be used inside PSO.

Feasibility guards and penalties

To preserve optimizer stability and avoid pathological solutions, the implementation enforces geometric guards:

- **Ghost-hole penalty:** if the hole center lies outside Ω_{drone}, the objective returns a large penalty (soft constraint).
- **Mass-removal penalty:** if the retained mass estimate is numerically near zero, a large penalty is returned to prevent solutions that delete the entire body.

These guards illustrate a common inverse-design pattern: constraints are enforced at the objective level to keep the optimizer's search in physically meaningful regions.

Deterministic seeding to stabilize stochastic optimization

A key difficulty is that the objective is stochastic: repeated evaluations of the same (h_x, h_y, h_z, r) would normally yield different CM estimates due to sampling noise, generating a non-stationary landscape for PSO. The application resolves this by a **parameter-based hashing** strategy:

- a hash function maps $\mathbf{z}$ to a 32-bit seed;
- the local seed is combined with the global library seed, making the objective **pointwise deterministic**.

As a result, identical candidate solutions return identical objective values, converting Monte Carlo noise into a reproducible (pseudo-deterministic) surface. This design is essential for fair benchmarking across thread counts and for debugging convergence behavior.

Two-stage evaluation: fast optimization vs high-precision verification

The workflow explicitly separates **exploration** from **validation**:

1. **Fast inner-loop objective:** $\sim 2\times10^4$ samples per evaluation during PSO to keep iteration time low.

2. **High-precision verification:** $\sim 10^6$ correlated samples in a single pass to validate the final CM and error with significantly reduced variance.

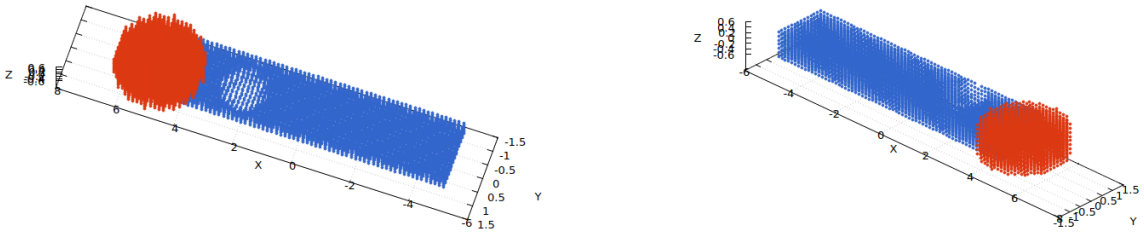
In addition, the application computes a hybrid reference by combining:

- a high-precision Monte Carlo baseline for the *full body* (no hole), and
- an *analytic* subtraction for the spherical hole volume and first moments,

yielding a robust ground-truth proxy for cross-checking the stochastic estimator. This demonstrates how Montecarlo-DGL supports not only computation but also **scientific validation and error auditing**.

Artifacts and visualization

Finally, the application exports a sampled point-cloud representation of the optimized geometry (arm and motor with the hole excluded) and auto-generates a Gnuplot script. These artifacts provide a reproducible visualization pipeline independent of the numerical backend.



(a) Hole. (b) General view.

Figure 2: Resulting hole disposition in the drone arm.

5.2 Case Study 2: Wind Farm Layout Optimization via MH-MCMC Expectation and PSO/GA Comparison

Engineering goal

The goal is to maximize the expected farm power by optimizing turbine placement inside a rectangular farm. The decision vector is:

$$\mathbf{x} = (x_1, y_1, \ldots, x_{N_T}, y_{N_T}) \in \mathbb{R}^{2N_T}, \tag{33}$$

under box bounds and minimum-distance constraints. Since the optimizers operate in minimization mode, the implemented objective is:

$$J(\mathbf{x}) = -\widehat{\mathbb{E}[P(\mathbf{x})]} + \Pi(\mathbf{x}), \tag{34}$$

where $\widehat{\mathbb{E}[P(\mathbf{x})]}$ is an estimated expected power and Π is a proximity penalty for violations of the minimum spacing requirement.

Stochastic wind model and power evaluation

Wind uncertainty is modeled by two variables:

- wind speed v distributed according to a Weibull law (shape k, scale λ),
- wind direction θ assumed uniform on $[0, 2\pi)$.

For each sample (v, θ), the application evaluates the sum of turbine powers with a simplified wake-loss model, where downstream turbines experience reduced effective wind speed depending on relative geometry and projected upstream direction. The resulting instantaneous farm power $P(\mathbf{x}; v, \theta)$ is integrated over wind variables to obtain the expected performance metric.

Metropolis-Hastings integration in a centered 2D domain

The expectation over (v, θ) is estimated using `MHMontecarloIntegrator<2>` on a centered `HyperRectangle<2>`. A mapping converts centered coordinates $\mathbf{w}$ to physical variables:

$$v = w_0 + c_v, \qquad \theta = w_1 + c_\theta, \tag{35}$$

with (c_v, c_θ) chosen as midpoints of the physical bounds. The MH target density is proportional to the Weibull *p.d.f.* in v and constant in θ, while proposals are generated via `GaussianProposal<2>` with physically meaningful step sizes $(\sigma_v, \sigma_\theta)$. This illustrates how the framework enables:

- switching from uniform MC to MCMC sampling,
- configuring burn-in, thinning, and volume-estimation budgets,
- reusing the same objective function interface expected by the optimizers.

Outer-loop optimization and algorithmic comparison

The case study compares two metaheuristics available in Montecarlo-DGL:

- **PSO** (swarm-based continuous search),
- **GA** (population-based evolutionary search).

Both optimizers call the same objective function, which triggers an MH-based estimator at each evaluation. To maintain stability under stochasticity, the application uses deterministic hashing for per-evaluation seeds (and optionally mixes the thread id under OpenMP), achieving reproducible comparisons between PSO and GA under identical settings.

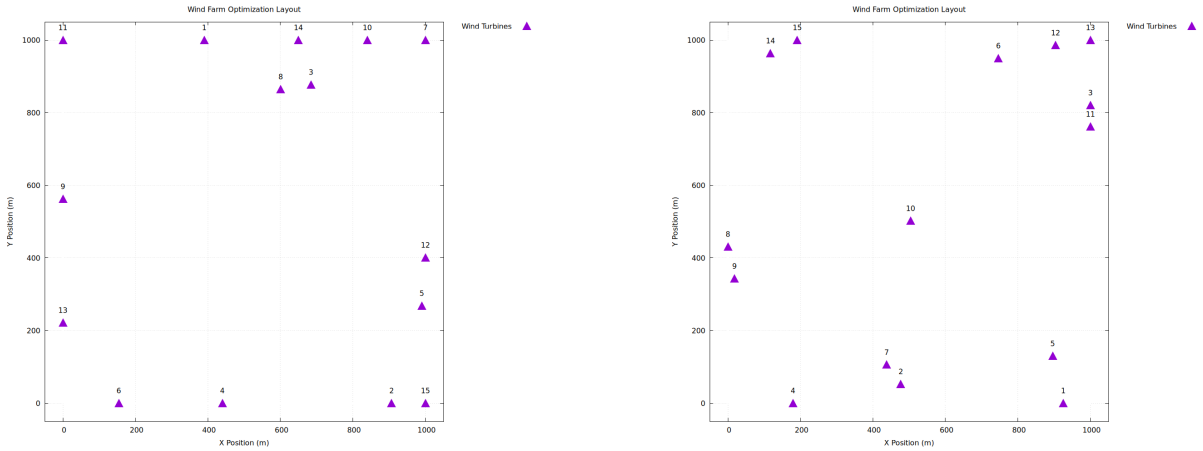
Outputs for benchmarking and reproducibility

The application produces machine-readable artifacts and clean logs:

- progress printed at fixed cadence (e.g., every 10 iterations/generations),
- final summaries reporting runtime, best expected power (MW), and minimum turbine spacing (m),

- result files containing turbine coordinates and Gnuplot scripts for layout visualization.

This closes the experimental loop: each algorithm run yields reproducible data suitable for reporting, visualization, and regression testing.



(a) GA optimization method.

(b) PSO optimization method.

Figure 3: Resulting disposition of wind turbines (violet triangles) in the wind farm.

What this case study validates

Together, the wind farm application validates the library's capability to: (i) treat expensive stochastic integrals as black-box objectives, (ii) run robust gradient-free optimization, and (iii) provide deterministic, thread-safe reproducibility mechanisms required for HPC-grade experimentation.