# 需求文档

## 简介

你是一个经验丰富的消防安全工程师和代码工程师。我想开发一个网页端计算工具，它的目标是计算房屋在火灾时某些部分（例如外墙窗户，天窗，墙体）受到的热辐射以判断房屋设计是否安全。此计算工具的计算方法之参考文献为Walton (2002), “Calculation of Obstructed View Factors by Adaptive Integration”, NISTIR 6925。我现在想要检验Adaptive Integration的计算数值是否合理。

请用英文引导我一步步补充一份完整的需求文档内容，包括以下部分：

1. 功能简介：生成一个英文网页，供用户输入矩形emitter和receiver的尺寸，然后计算引擎通过Adaptive Integration方法计算后出receiver的local peak view factor，并同时使用计算出1Ai fixed grid, Monte Carlo，analytical等方法所算出的结果供用户参考。 理想状态下网页可选择生成图片供用户参考。
2. 主要用户和使用场景：消防工程师开发计算工具
3. 输入数据格式：矩形emitter和receiver的尺寸，间距。暂时假设两矩形互相平行，后续会加入矩形间的夹角以计算非平行矩形（或其他形状）间的局部最大view factors，最终会在emitter 和receiver间加入阻挡，并计算receiver之局部最大view factor
4. 输出内容要求：受检房屋部分所接受的热辐射
5. 功能流程：用户输入：emitter dimensions, receiver dimensions, setback, angle （optional）, occluder dimensions and relative location (optional), 用户点击计算，网页生成receiver的局部最大view factors ，并比较用不同方法（1AI adaptive，1AI fixed grid， Monte Carlo，analytical）所算出的结果。Optional: 生成表格/图片比较结果。
6. 异常处理：报错，提示用户联系软件工程师Roger
7. 验收标准：用1AI adaptive方法成功计算receiver的局部最大view factor，1AI adaptive 方法必须符合NISTIR 6925的计算方法。 其他方法只需供用户参考。
8. 如果用AI参与，请帮我草拟prompt模板

A screen shot of a product

AI-generated content may be incorrect.

## Product Requirement Document (PRD)

**Document ID:** FSE-RAD-VIEWFACTOR-PRD-v0.5  
**Date:** [TBC]  
**Product:** Local Python Tool – Radiation View Factor Validation  
**Owner:** Fire Safety Engineer (you)  
**Tech Support:** Roger (software engineer role)

### Function Overview

Create a **local Python-based tool** to calculate **local peak view factors** between rectangular surfaces (emitter → receiver) under fire conditions.

* **Primary method:** Adaptive Integration (1AI) per Walton (2002), *NISTIR 6925*.
* **Secondary methods (for cross-checking):**
  + Fixed grid integration
  + Monte Carlo sampling
  + Analytical formulas (where applicable)

**Definition of Local Peak VF:** the **maximum differential (point) view factor** over the receiver surface (not the area-averaged view factor). For parallel, centre-aligned E/R, the local peak occurs at the receiver centre.

Tool purpose: **methodology validation / sanity checks** only.

### Users & Use Cases

**Users:** You (Fire Safety Engineer / developer).

**Use Cases:**

* Validate 1AI implementation against NISTIR 6925.
* Compare across numerical methods to understand accuracy vs. runtime.
* Confirm values before embedding into other projects.

### Input Data Format

Inputs via **CLI** or **config file** (YAML/JSON).

**Required inputs:**

* **Emitter dimensions:** width (m), height (m)
* **Receiver dimensions:** width (m), height (m); **default = emitter dimensions**
* **Setback distance:** separation between emitter and receiver (m)
* **Directions:** emitter and receiver face **towards each other**
* **Relative orientation:** **centres align** with each other

**Optional inputs:**

* **Angle:** rotation (degrees, default 0 → parallel)
* **Occluder:** width, height, position (future)
* **Tuning knobs (see §6):** tolerances, refinement, iteration limits, etc.

**Example CLI:**

python viewfactor.py --emitter 5.1 2.1 --receiver 5.1 2.1 --setback 1.0 \

--method adaptive --rel-tol 3e-3 --max-depth 12

### Output Requirements

**Primary Output:**

* Local peak view factor (receiver) from **1AI**.

**Secondary Outputs:**

* Fixed grid VF
* Monte Carlo VF (+ uncertainty)
* Analytical reference (if applicable)

**Formats:**

* Console printout
* CSV in /results/
* Optional: flux map PNG via matplotlib

### Functional Flow

1. **User runs script** (CLI or config).
2. **Engine computes**: Adaptive (primary) + Fixed Grid + Monte Carlo + Analytical (secondary).
3. **Results** printed and saved to /results/.
4. **Optional plots** (flux distribution / convergence) saved to /results/plots/.

**Default geometric assumption:** facing each other, centres aligned; receiver dims = emitter dims unless specified.

### Performance & Tuning (Refinement Controls)

Goal: allow developer to trade accuracy vs. runtime. Defaults provided; can be tightened/loosened.

**6.1 Adaptive Integration (1AI)**

* **--rel-tol** (relative tolerance, default: 3e-3) → targets ≈ **99.7%** accuracy.
* **--abs-tol** (absolute tolerance, default: 1e-6).
* **--max-depth** (max recursion depth, default: 12).
* **--min-cell-area-frac** (min sub-rect area as fraction of emitter area, default: 1e-8).
* **--max-cells** (cap on number of sub-cells, default: 200000).
* **--time-limit-s** (wall-clock limit per solve, default: 60).
* **--init-grid** (initial partition, default: 4x4).

**Where to change:** src/adaptive.py (constants at top), or via CLI flags.  
**Lower cost:** relax --rel-tol (e.g., 5e-2 ≈ 95%), reduce --max-depth/--max-cells.  
**Higher accuracy:** tighten --rel-tol, raise --max-depth/--max-cells.

**6.2 Fixed Grid**

* **--grid-nx, --grid-ny** (default: 100, 100).
* **--quadrature** (centroid | 2x2), default: centroid.
* **--time-limit-s** (default: 60).

**Where to change:** src/fixed\_grid.py (defaults), or via CLI.  
**Lower cost:** reduce grid; **Higher accuracy:** increase grid or use 2x2 quadrature.

**6.3 Monte Carlo**

* **--samples** total rays (default: 200000).
* **--target-rel-ci** relative half-width of 95% CI (default: 0.02 → 2%).
* **--max-iters** batches (default: 50).
* **--seed** RNG seed for reproducibility (default: 42).
* **--time-limit-s** (default: 60).

**Where to change:** src/montecarlo.py.  
**Lower cost:** fewer samples / looser CI; **Higher accuracy:** more samples / tighter CI.

**Note:** Only **Adaptive** is required to meet the **±0.3%** target (§7). Fixed Grid & Monte Carlo are reference comparisons (trends & ballpark), unless you explicitly tighten them.

### Acceptance Criteria

* **Adaptive Integration (1AI)** reproduces benchmark cases within **±0.3% error (≈99.7% accuracy)** vs. references.
* **NISTIR 6925** validation cases match within tolerance.
* **Secondary methods** show consistent trends; exact ±0.3% not required unless tuned.
* Tool runs in **Python 3.12+** with numpy, scipy, matplotlib, pytest.
* **No hangs / infinite loops / unbounded growth** (see §8 safeguards).
* All runs end with a **clear status**: converged, reached\_limits, or failed.

### Non-Convergence & Hang Safeguards

**Apply to all methods:**

* **Global timeouts:** per-solve --time-limit-s (default 60s).
* **Iteration caps:** --max-depth, --max-cells, --max-iters, --samples.
* **Early stopping:** stop when tolerance or CI met.
* **Stagnation detection:** if improvement < 10% of tolerance for N consecutive steps (default N=5), stop with reached\_limits.
* **Numerical guards:**
  + Enforce positive dimensions; nonzero setback; clamp cosθ≥0.
  + Avoid division by ~0 (use EPS=1e-12).
  + Handle near-contact by switching to higher-precision path or analytical approximation.
* **Determinism:** fixed RNG seed for Monte Carlo unless overridden.
* **Graceful exit codes/messages** with diagnostics: iterations, achieved tolerance/CI, caps hit.
* **Unit tests** include: smoke tests, worst-case geometries, near-contact, and symmetry checks.

### Error Handling

* **Invalid input:**
* Error: Invalid geometry (non-positive dimensions or setback). Check inputs.
* **Non-convergence / limits reached:**
* Warning: Reached limits before tolerance. Result may be approximate.
* **Failure / NaN:**
* Error: Calculation failed (NaN/Inf). Contact Roger.

### Validation Test Cases

* **Sources included:**
* **NISTIR 6925 (Walton, 2002)** analytic obstructed case (unit squares with two 0.5×0.5 occluders; Table 1): analytic value and adaptive-integration results at several tolerances. [nvlpubs.nist.gov](https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir6925.pdf)
* **User hand calculations** (unobstructed, parallel, centre-aligned E/R).
* Note: The NIST case involves **obstruction**. Keep it in the suite (it’s the canonical benchmark for 1AI), but you can mark it **skip** until occluders are turned on.

1. **10.1 NISTIR 6925 – Analytic Obstructed Test (unit squares with mid-span occluders)**

* **Geometry (per Fig. 13):**
* Surface 1: **1.0 × 1.0** square
* Surface 2: **1.0 × 1.0** square, **unit separation** from surface 1
* Surfaces 3 & 4: **0.5 × 0.5** squares, **back-to-back**, centred on the line between surfaces 1 and 2, located at **3/4 of the distance** from surface 1 toward surface 2.
* Visibility: Only **surface 1 is visible from surface 3**, only **surface 2 from surface 4**.
* **Analytic unobstructed** between 1 & 2 (if 3 & 4 absent): **F\*₁,₂ = 0.19982490**.
* **Analytic obstructed** (with 3 & 4): **F₁,₂ = 0.11562061**.  
  **Adaptive results (ε from eq. 9):**
* ε=1e-3 → **F₁,₂ = 0.11563653**, error **+0.00015924**, points **25**
* ε=1e-4 → **F₁,₂ = 0.11562055**, error **−0.00000006**, points **125**
* ε=1e-3 (project from 2) → **0.11473675**, error **−0.00088386**, points **525**
* ε=1e-4 → **0.11526465**, error **−0.00035596**, points **925**
* ε=1e-5 → **0.11553235**, error **−0.00008826**, points **2925**
* ε=1e-6 → **0.11560305**, error **−0.00001756**, points **8125**
* ε=1e-7 → **0.11561626**, error **−0.00000435**, points **18525**  
  (See NISTIR 6925 Table 1 and Fig. 13.) [nvlpubs.nist.gov](https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir6925.pdf)

1. **10.2 User Hand-Calc Cases (unobstructed; parallel; centres aligned; facing each other)**

* **5.1 × 2.1**, **5.1 × 2.1**, **setback 0.05**, angle **0°** → **F = 0.998805**
* **5.1 × 2.1**, **5.1 × 2.1**, **setback 1.0**, angle **0°** → **F = 0.70274**
* **5.0 × 2.0**, **5.0 × 2.0**, **setback 3.8**, angle **0°** → **F = 0.17735**
* **20.02 × 1.05**, **20.02 × 1.05**, **setback 0.81**, angle **0°** → **F = 0.54375**
* **20.02 × 1.05**, **20.02 × 1.05**, **setback 1.8**, angle **0°** → **F = 0.27931**
* **21 × 1**, **21 × 1**, **setback 3.67**, angle **0°** → **F = 0.13285**
* **Acceptance target:** Adaptive method within **±0.3%** (≈**99.7%** accuracy) of the respective reference **for each applicable case**. Fixed Grid & Monte Carlo provide trend/ballpark (looser tolerance unless tuned).

### Glossary

* **View Factor (F):** fraction of radiation from emitter reaching receiver directly.
* **Local Peak View Factor:** maximum pointwise (differential) VF on receiver surface.
* **Adaptive Integration (1AI):** recursively refines emitter subareas until error tolerance is met.
* **Fixed Grid:** uniform partition of emitter; accuracy scales with grid density.
* **Monte Carlo:** random ray sampling; accuracy ~ O(1/√N).
* **Analytical Solution:** closed-form for simplified geometries.

### Developer Notes (README pointers)

* **Where to tune:**
  + src/adaptive.py: REL\_TOL, ABS\_TOL, MAX\_DEPTH, MIN\_CELL\_AREA\_FRAC, MAX\_CELLS, TIME\_LIMIT\_S
  + src/fixed\_grid.py: GRID\_NX, GRID\_NY, QUADRATURE, TIME\_LIMIT\_S
  + src/montecarlo.py: SAMPLES, TARGET\_REL\_CI, MAX\_ITERS, SEED, TIME\_LIMIT\_S
* **CLI flags override** these defaults. Document flags in README.md.
* **Target accuracy:** keep **Adaptive** at rel-tol ≈ 3e-3 for 99.7%; if runtime too high, relax to 5e-2 (~95%).
* **Testing:** run pytest -q; review convergence plots under /results/plots/.
* **Reproducibility:** pin versions in requirements.txt.

# Useful Prompts

## PRD Stage – PRD (AI Orchestrator)

你是一个经验丰富的消防安全工程师，现在我要开发一个英文网页端计算工具，它的目标是计算房屋在火灾时某些部分（例如外墙窗户，天窗，墙体）受到的热辐射以判断房屋设计是否安全。

请引导我一步步补充一份完整的需求文档内容，包括以下部分：

1. 功能简介：生成一个网页, 通过模拟两块矩形间的热辐射传导计算房屋在火灾中是否会释放危机相邻建筑的热量导致火势传播，或者房屋中无防火措施的部分（例如非防火玻璃）是否会接收过多相邻建筑火灾所释放的热量，从而导致火势传播。该模拟需考虑某些情况下两矩形间可能有一块矩形阻挡。
2. 主要用户和使用场景：消防工程师在检视房屋设计安全时以及写消防报告时使用
3. 输入数据格式：理想状态下输入文件为PDF房屋设计平面图。若难度较高则首先尝试用户在网页端输入数据例如窗户尺寸，边界距离，窗户朝向，阻挡（若有）的尺寸等。
4. 输出内容要求：受检房屋部分所接受的热辐射
5. 功能流程：理想状态下用户上传房屋设计平面图至网页，AI检视后找到emitter 和receiver。若有阻挡则生成一个occluder。根据它们的位置和距离等参数AI计算receiver所接受的view factor. 其次根据用户定义的emitter 辐射参数（温度，emissivity 等）计算receiver所接受的radiation。
6. 异常处理：报错，提示用户联系软件工程师Roger
7. 验收标准：成功计算各个receiver所接受的radiation
8. 如果用AI参与，请帮我草拟prompt模板

## PRD Stage - PRD Breakdown (AI Orchestrator)

You are one of the top coding developers, break the PRD into executable steps and prompts that are copiable to Cursor, Codex or Claude Code.

---or---

用英文把的 PRD 拆解成 阶段性可执行任务，每个阶段包含目标、主要文件/代码块，以及你可以直接复制到 Cursor 或 Claude code 或Codex 的 prompt

## Coding Stage – Existing Projects (Agent / Developer)

You are one of the top coding developers. I am starting a new revision from VXX to VXX, log this change.

Thoroughly read the code and all technical documentation, and wait for my instructions

## Coding Stage – Commencement - Import PRD (Agent / Developer)

You are one of the top coding developers.

Create a DOCS folder to store technical documentation.

Create README for AI tools or developers to understand the project, and wait for my instruction for the PRD.

**Document ID:** FSE-RAD-VIEWFACTOR-PRD-v0.5  
**Date:** [TBC]  
**Product:** Local Python Tool – Radiation View Factor Validation  
**Owner:** Fire Safety Engineer (you)  
**Tech Support:** Roger (software engineer role)

1. Function Overview

Create a **local Python-based tool** to calculate **local peak view factors** between rectangular surfaces (emitter → receiver) under fire conditions.

* **Primary method:** Adaptive Integration (1AI) per Walton (2002), *NISTIR 6925*.
* **Secondary methods (for cross-checking):**
  + Fixed grid integration
  + Monte Carlo sampling
  + Analytical formulas (where applicable)

**Definition of Local Peak VF:** the **maximum differential (point) view factor** over the receiver surface (not the area-averaged view factor). For parallel, centre-aligned E/R, the local peak occurs at the receiver centre.

Tool purpose: **methodology validation / sanity checks** only.

1. Users & Use Cases

**Users:** You (Fire Safety Engineer / developer).

**Use Cases:**

* Validate 1AI implementation against NISTIR 6925.
* Compare across numerical methods to understand accuracy vs. runtime.
* Confirm values before embedding into other projects.

1. Input Data Format

Inputs via **CLI** or **config file** (YAML/JSON).

**Required inputs:**

* **Emitter dimensions:** width (m), height (m)
* **Receiver dimensions:** width (m), height (m); **default = emitter dimensions**
* **Setback distance:** separation between emitter and receiver (m)
* **Directions:** emitter and receiver face **towards each other**
* **Relative orientation:** **centres align** with each other

**Optional inputs:**

* **Angle:** rotation (degrees, default 0 → parallel)
* **Occluder:** width, height, position (future)
* **Tuning knobs (see §6):** tolerances, refinement, iteration limits, etc.

**Example CLI:**

python viewfactor.py --emitter 5.1 2.1 --receiver 5.1 2.1 --setback 1.0 \

--method adaptive --rel-tol 3e-3 --max-depth 12

1. Output Requirements

**Primary Output:**

* Local peak view factor (receiver) from **1AI**.

**Secondary Outputs:**

* Fixed grid VF
* Monte Carlo VF (+ uncertainty)
* Analytical reference (if applicable)

**Formats:**

* Console printout
* CSV in /results/
* Optional: flux map PNG via matplotlib

1. Functional Flow
2. **User runs script** (CLI or config).
3. **Engine computes**: Adaptive (primary) + Fixed Grid + Monte Carlo + Analytical (secondary).
4. **Results** printed and saved to /results/.
5. **Optional plots** (flux distribution / convergence) saved to /results/plots/.

**Default geometric assumption:** facing each other, centres aligned; receiver dims = emitter dims unless specified.

1. Performance & Tuning (Refinement Controls)

Goal: allow developer to trade accuracy vs. runtime. Defaults provided; can be tightened/loosened.

**6.1 Adaptive Integration (1AI)**

* **--rel-tol** (relative tolerance, default: 3e-3) → targets ≈ **99.7%** accuracy.
* **--abs-tol** (absolute tolerance, default: 1e-6).
* **--max-depth** (max recursion depth, default: 12).
* **--min-cell-area-frac** (min sub-rect area as fraction of emitter area, default: 1e-8).
* **--max-cells** (cap on number of sub-cells, default: 200000).
* **--time-limit-s** (wall-clock limit per solve, default: 60).
* **--init-grid** (initial partition, default: 4x4).

**Where to change:** src/adaptive.py (constants at top), or via CLI flags.  
**Lower cost:** relax --rel-tol (e.g., 5e-2 ≈ 95%), reduce --max-depth/--max-cells.  
**Higher accuracy:** tighten --rel-tol, raise --max-depth/--max-cells.

**6.2 Fixed Grid**

* **--grid-nx, --grid-ny** (default: 100, 100).
* **--quadrature** (centroid | 2x2), default: centroid.
* **--time-limit-s** (default: 60).

**Where to change:** src/fixed\_grid.py (defaults), or via CLI.  
**Lower cost:** reduce grid; **Higher accuracy:** increase grid or use 2x2 quadrature.

**6.3 Monte Carlo**

* **--samples** total rays (default: 200000).
* **--target-rel-ci** relative half-width of 95% CI (default: 0.02 → 2%).
* **--max-iters** batches (default: 50).
* **--seed** RNG seed for reproducibility (default: 42).
* **--time-limit-s** (default: 60).

**Where to change:** src/montecarlo.py.  
**Lower cost:** fewer samples / looser CI; **Higher accuracy:** more samples / tighter CI.

**Note:** Only **Adaptive** is required to meet the **±0.3%** target (§7). Fixed Grid & Monte Carlo are reference comparisons (trends & ballpark), unless you explicitly tighten them.

1. Acceptance Criteria

* **Adaptive Integration (1AI)** reproduces benchmark cases within **±0.3% error (≈99.7% accuracy)** vs. references.
* **NISTIR 6925** validation cases match within tolerance.
* **Secondary methods** show consistent trends; exact ±0.3% not required unless tuned.
* Tool runs in **Python 3.12+** with numpy, scipy, matplotlib, pytest.
* **No hangs / infinite loops / unbounded growth** (see §8 safeguards).
* All runs end with a **clear status**: converged, reached\_limits, or failed.

1. Non-Convergence & Hang Safeguards

**Apply to all methods:**

* **Global timeouts:** per-solve --time-limit-s (default 60s).
* **Iteration caps:** --max-depth, --max-cells, --max-iters, --samples.
* **Early stopping:** stop when tolerance or CI met.
* **Stagnation detection:** if improvement < 10% of tolerance for N consecutive steps (default N=5), stop with reached\_limits.
* **Numerical guards:**
  + Enforce positive dimensions; nonzero setback; clamp cosθ≥0.
  + Avoid division by ~0 (use EPS=1e-12).
  + Handle near-contact by switching to higher-precision path or analytical approximation.
* **Determinism:** fixed RNG seed for Monte Carlo unless overridden.
* **Graceful exit codes/messages** with diagnostics: iterations, achieved tolerance/CI, caps hit.
* **Unit tests** include: smoke tests, worst-case geometries, near-contact, and symmetry checks.

1. Error Handling

* **Invalid input:**
* Error: Invalid geometry (non-positive dimensions or setback). Check inputs.
* **Non-convergence / limits reached:**
* Warning: Reached limits before tolerance. Result may be approximate.
* **Failure / NaN:**
* Error: Calculation failed (NaN/Inf). Contact Roger.

1. Validation Test Cases

**Sources included:**

* **NISTIR 6925 (Walton, 2002)** analytic obstructed case (unit squares with two 0.5×0.5 occluders; Table 1): analytic value and adaptive-integration results at several tolerances. [nvlpubs.nist.gov](https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir6925.pdf)
* **User hand calculations** (unobstructed, parallel, centre-aligned E/R).
* Note: The NIST case involves **obstruction**. Keep it in the suite (it’s the canonical benchmark for 1AI), but you can mark it **skip** until occluders are turned on.

10.1 NISTIR 6925 – Analytic Obstructed Test (unit squares with mid-span occluders)

* **Geometry (per Fig. 13):**
* Surface 1: **1.0 × 1.0** square
* Surface 2: **1.0 × 1.0** square, **unit separation** from surface 1
* Surfaces 3 & 4: **0.5 × 0.5** squares, **back-to-back**, centred on the line between surfaces 1 and 2, located at **3/4 of the distance** from surface 1 toward surface 2.
* Visibility: Only **surface 1 is visible from surface 3**, only **surface 2 from surface 4**.
* **Analytic unobstructed** between 1 & 2 (if 3 & 4 absent): **F\*₁,₂ = 0.19982490**.
* **Analytic obstructed** (with 3 & 4): **F₁,₂ = 0.11562061**.  
  **Adaptive results (ε from eq. 9):**
* ε=1e-3 → **F₁,₂ = 0.11563653**, error **+0.00015924**, points **25**
* ε=1e-4 → **F₁,₂ = 0.11562055**, error **−0.00000006**, points **125**
* ε=1e-3 (project from 2) → **0.11473675**, error **−0.00088386**, points **525**
* ε=1e-4 → **0.11526465**, error **−0.00035596**, points **925**
* ε=1e-5 → **0.11553235**, error **−0.00008826**, points **2925**
* ε=1e-6 → **0.11560305**, error **−0.00001756**, points **8125**
* ε=1e-7 → **0.11561626**, error **−0.00000435**, points **18525**  
  (See NISTIR 6925 Table 1 and Fig. 13.) [nvlpubs.nist.gov](https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir6925.pdf)

10.2 User Hand-Calc Cases (unobstructed; parallel; centres aligned; facing each other)

* **5.1 × 2.1**, **5.1 × 2.1**, **setback 0.05**, angle **0°** → **F = 0.998805**
* **5.1 × 2.1**, **5.1 × 2.1**, **setback 1.0**, angle **0°** → **F = 0.70274**
* **5.0 × 2.0**, **5.0 × 2.0**, **setback 3.8**, angle **0°** → **F = 0.17735**
* **20.02 × 1.05**, **20.02 × 1.05**, **setback 0.81**, angle **0°** → **F = 0.54375**
* **20.02 × 1.05**, **20.02 × 1.05**, **setback 1.8**, angle **0°** → **F = 0.27931**
* **21 × 1**, **21 × 1**, **setback 3.67**, angle **0°** → **F = 0.13285**
* **Acceptance target:** Adaptive method within **±0.3%** (≈**99.7%** accuracy) of the respective reference **for each applicable case**. Fixed Grid & Monte Carlo provide trend/ballpark (looser tolerance unless tuned).

1. Glossary

* **View Factor (F):** fraction of radiation from emitter reaching receiver directly.
* **Local Peak View Factor:** maximum pointwise (differential) VF on receiver surface.
* **Adaptive Integration (1AI):** recursively refines emitter subareas until error tolerance is met.
* **Fixed Grid:** uniform partition of emitter; accuracy scales with grid density.
* **Monte Carlo:** random ray sampling; accuracy ~ O(1/√N).
* **Analytical Solution:** closed-form for simplified geometries.

## Coding Stage – Initialisatio – Create Rules

As a top coding programmer, apply the following coding principles for AI tools or developers to understand the project.

### 1. \*\*SOLID Principles\*\*: All code must follow SOLID principles

### 2. \*\*DRY\*\*: Avoid code duplication

### 3. \*\*KISS\*\*: Keep implementations simple and clear

### 4. \*\*Google Standards\*\*: Follow Google's C# style guide

### 5. \*\*Australian English\*\*: Use Australian English spelling and terminology throughout

### 6. \*\*Testing\*\*: All new code must include comprehensive tests

### 7. \*\*Documentation\*\*: All public APIs must be documented

### 8. \*\*Safety First\*\*: All changes must maintain safety-critical reliability

### 9. \*\*Performance\*\*: Must meet PRD performance requirements (≤3s single case)

### 10. \*\*Comments\*\*: Complex algorithms must include inline comments

### 11. \*\*Single Responsibility\*\*: Each class and method must have a single, well-defined purpose

### 12. \*\*Clear Naming\*\*: Use descriptive, self-documenting names for all variables, methods, and classes

### 13. \*\*No Hard-coded Secrets\*\*: Never store passwords, API keys, or sensitive data in source code

### 14. \*\*Comments First\*\*: Write comments before implementing complex logic

### 15. \*\*Test-Driven\*\*: Write tests before or alongside implementation (TDD/BDD approach)

### 16. \*\*Explicit Error Handling\*\*: All methods must include proper exception handling with specific error types

### 17. \*\*Secure by Default\*\*: Implement security measures as the default, not as an afterthought

### 18. \*\*Performance Boundaries\*\*: Define and enforce performance limits and monitoring

### 19. \*\*Minimal Dependencies\*\*: Keep external dependencies to a minimum and justify each one

### 20. \*\*YAGNI (You Aren't Gonna Need It)\*\*: Don't implement features until they are actually needed

### 21. \*\*Code for Readers\*\*: Write code that is easy to understand and maintain by other developers

### 22. \*\*Explicit Types\*\*: Every exported value must have an explicit type or interface

### 23. \*\*No "Any" Types\*\*: Avoid using `any` type unless explicitly whitelisted and documented

### 24. \*\*Internationalisation (i18n)\*\*: Support multiple languages with proper localisation

### 25. \*\*Code Length Limit\*\*: No single file should exceed 700 lines of code

## Coding Stage – Prompts from Agent

## Step 1 — Project scaffold

**Prompt to Cursor:**

You are setting up a local Python project for a radiation VIEW FACTOR validation tool.

Rules:

- Only create/modify files inside the repo root.

- Use relative imports and type hints.

- Target Python 3.12+.

Create this structure:

/src/\_\_init\_\_.py

/src/cli.py

/src/geometry.py

/src/io\_yaml.py

/src/analytical.py

/src/fixed\_grid.py

/src/adaptive.py

/src/montecarlo.py

/tests/\_\_init\_\_.py

/tests/test\_smoke.py

/docs/validation\_cases.yaml # (content will be added in Step 3)

/results/.gitkeep

main.py

README.md

requirements.txt

pyproject.toml # basic metadata

requirements.txt should include:

numpy

scipy

matplotlib

pyyaml

pytest

main.py:

- Entry point that calls src.cli:main().

- If invoked with no args, print a short usage help and exit(2).

README.md:

- Short description (local Python tool for local peak view factor validation).

- Note default geometry assumptions: facing each other, centres aligned; receiver dims default to emitter dims.

Add a minimal tests/test\_smoke.py:

- Asserts that Python imports all modules without error.

Run 'pytest -q' should pass.

**Accept when:**

pytest -q **passes** **and** python main.py **shows usage help.**

## Step 2 — CLI skeleton (+ geometry flags)

**Prompt to Cursor:**

Implement a robust CLI in src/cli.py and wire it in main.py.

Requirements:

- Use argparse with these flags:

--method {analytical,fixedgrid,adaptive,montecarlo}

--emitter W H

--receiver W H (default = emitter dims if omitted)

--setback S

--angle DEG (default 0; parallel)

--cases PATH (YAML test suite; optional)

--plot (generate plots)

--outdir PATH (default ./results)

Tuning flags (pass-through; not enforced yet):

# Adaptive

--rel-tol (default 3e-3)

--abs-tol (default 1e-6)

--max-depth (default 12)

--min-cell-area-frac (default 1e-8)

--max-cells (default 200000)

--time-limit-s (default 60)

--init-grid (default "4x4")

# Fixed grid

--grid-nx (default 100)

--grid-ny (default 100)

--quadrature {centroid,2x2} (default centroid)

# Monte Carlo

--samples (default 200000)

--target-rel-ci (default 0.02)

--max-iters (default 50)

--seed (default 42)

Behavior:

- Parse args, normalize defaults (receiver dims = emitter dims if not provided).

- For now, just echo the parsed args and exit 0.

- Ensure clean error messages for missing/invalid arguments.

- Keep centres aligned and facing each other as default assumptions.

Add unit tests in tests/test\_smoke.py that call cli.main() via argparse.Namespace to verify parsing and defaults.

**Accept when:** python main.py --method analytical --emitter 5 2 --setback 1 prints parsed config; pytest -q passes.

## Step 3 — YAML loader + --cases wiring

**Prompt to Cursor (copy–paste):**

You are implementing Step 3 for the local Python radiation view factor tool.

Goals:

1) Add a YAML loader (PyYAML) to read validation cases.

2) Wire a --cases PATH option in the CLI to iterate cases and write a summary CSV.

3) Add the full docs/validation\_cases.yaml provided below.

4) Add tests for the loader and the case runner.

Rules:

- Only modify files within the repo.

- Keep type hints and docstrings.

- Don’t implement solvers here; they can return a dummy value for now.

- All commands must pass pytest -q.

Files to create/modify:

--------------------------------------------------------------------------------

A) src/io\_yaml.py (new or complete)

--------------------------------------------------------------------------------

from \_\_future\_\_ import annotations

from typing import Any, Dict, List

import yaml

import os

class YamlError(Exception):

pass

REQUIRED\_GEOM\_KEYS = {"emitter", "receiver", "setback", "angle"}

def load\_cases(path: str) -> List[Dict[str, Any]]:

"""

Load a YAML file of validation cases.

Returns a list of cases (dicts). Raises YamlError on problems.

"""

if not os.path.isfile(path):

raise YamlError(f"YAML not found: {path}")

try:

with open(path, "r", encoding="utf-8") as f:

data = yaml.safe\_load(f)

except Exception as e:

raise YamlError(f"Failed to parse YAML: {e}") from e

if not isinstance(data, dict) or "cases" not in data or not isinstance(data["cases"], list):

raise YamlError("Invalid YAML structure: top-level 'cases' list required")

return data["cases"]

def validate\_case\_schema(case: Dict[str, Any]) -> None:

"""

Minimal schema check for a case.

Required: id (str), enabled (bool), geometry/emitter/receiver/setback/angle

"""

if "id" not in case or not isinstance(case["id"], str):

raise YamlError("Case missing 'id' (str)")

if "enabled" not in case or not isinstance(case["enabled"], bool):

raise YamlError(f"Case {case.get('id')} missing 'enabled' (bool)")

if "geometry" not in case or not isinstance(case["geometry"], dict):

raise YamlError(f"Case {case['id']}: 'geometry' dict required")

geom = case["geometry"]

for k in REQUIRED\_GEOM\_KEYS:

if k not in geom:

raise YamlError(f"Case {case['id']}: geometry missing key '{k}'")

for surf in ("emitter", "receiver"):

if surf not in geom or not isinstance(geom[surf], dict):

raise YamlError(f"Case {case['id']}: geometry.{surf} dict required")

for dim in ("w", "h"):

if dim not in geom[surf]:

raise YamlError(f"Case {case['id']}: geometry.{surf}.{dim} missing")

def coerce\_case\_to\_cli\_kwargs(case: Dict[str, Any]) -> Dict[str, Any]:

"""

Convert a YAML case dict into kwargs compatible with the CLI execution layer.

"""

method = case.get("method", "adaptive")

geom = case["geometry"]

k = {

"method": method,

"emitter": (float(geom["emitter"]["w"]), float(geom["emitter"]["h"])),

"receiver": (float(geom["receiver"]["w"]), float(geom["receiver"]["h"])),

"setback": float(geom["setback"]),

"angle": float(geom.get("angle", 0.0)),

"expected": (case.get("expected", {}) or {}).get("F12"),

"expected\_tol": (case.get("expected", {}).get("tolerance", {}) or {}),

"id": case["id"],

}

# Optional per-method overrides

if "method\_overrides" in case and isinstance(case["method\_overrides"], dict):

k["overrides"] = case["method\_overrides"]

else:

k["overrides"] = {}

return k

--------------------------------------------------------------------------------

B) src/cli.py (augment to support --cases and a basic runner)

--------------------------------------------------------------------------------

# In your existing argparse setup, ensure there is:

# parser.add\_argument("--cases", type=str, help="Path to YAML bundle of cases")

# After parsing args:

# - If args.cases is provided, call a new function run\_cases(args.cases, outdir)

# and then return.

# - Otherwise, proceed with single-run echo/placeholder.

# Add this helper at bottom of cli.py (or a new module if you prefer):

import csv

import time

from .io\_yaml import load\_cases, validate\_case\_schema, coerce\_case\_to\_cli\_kwargs

def run\_cases(cases\_path: str, outdir: str) -> int:

cases = load\_cases(cases\_path)

os.makedirs(outdir, exist\_ok=True)

summary\_csv = os.path.join(outdir, "cases\_summary.csv")

with open(summary\_csv, "w", newline="", encoding="utf-8") as f:

writer = csv.writer(f)

writer.writerow(["id","method","vf","expected","rel\_err","status","notes"])

for case in cases:

try:

validate\_case\_schema(case)

if not case["enabled"]:

writer.writerow([case.get("id"), case.get("method","adaptive"), "", "", "", "skipped", "disabled"])

continue

kw = coerce\_case\_to\_cli\_kwargs(case)

except Exception as e:

writer.writerow([case.get("id","<unknown>"), case.get("method","adaptive"), "", "", "", "invalid", str(e)])

continue

# Placeholder solve (replace in later steps)

# For now, produce a deterministic dummy vf

start = time.time()

vf = 0.123456

status = "pending"

# Compare against expected if present

expected = kw.get("expected")

rel\_err = ""

if expected is not None:

rel\_err = abs(vf - expected)/expected if expected != 0 else ""

writer.writerow([kw["id"], kw["method"], f"{vf:.8f}", expected if expected is not None else "", rel\_err, status, "placeholder"])

print(f"Wrote: {summary\_csv}")

return 0

# In main() / dispatch:

# if args.cases:

# return run\_cases(args.cases, args.outdir)

--------------------------------------------------------------------------------

C) docs/validation\_cases.yaml (full content)

--------------------------------------------------------------------------------

# EXACT FILE CONTENT BEGINS

version: 1

notes: >

Validation cases for local peak view factor.

Default geometry assumption: parallel, centres aligned, facing each other.

Units in meters; angle in degrees.

defaults:

method: adaptive

rel\_tol: 3.0e-3 # ≈99.7% target

abs\_tol: 1.0e-6

max\_depth: 12

min\_cell\_area\_frac: 1.0e-8

max\_cells: 200000

time\_limit\_s: 60

cases:

# ---------- NISTIR 6925 analytic obstructed benchmark (Table 1) ----------

- id: nist\_analytic\_obstructed\_unit\_squares

enabled: false # set true once occluders are implemented

description: >

NISTIR 6925 Table 1 analytic obstructed case.

Two 1x1 squares separated by 1 unit; two 0.5x0.5 occluders at 3/4 distance.

Analytic F12 (obstructed) = 0.11562061.

geometry:

emitter: { w: 1.0, h: 1.0 }

receiver: { w: 1.0, h: 1.0 }

setback: 1.0

angle: 0

centres\_aligned: true

facing\_each\_other: true

occluders:

- { w: 0.5, h: 0.5, position: { along\_axis: 0.75, lateral: 0.0 }, back\_to\_back\_pair: true }

expected:

F12: 0.11562061

tolerance:

type: relative

value: 3.0e-3

references:

- source: NISTIR 6925, Table 1 & Fig. 13

url: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir6925.pdf

method\_overrides:

rel\_tol: 1.0e-3

# ---------- User hand-calculation (unobstructed) ----------

- id: hc\_5p1x2p1\_s0p05

enabled: true

description: Unobstructed; parallel; centres aligned.

geometry: { emitter: { w: 5.1, h: 2.1 }, receiver: { w: 5.1, h: 2.1 }, setback: 0.05, angle: 0 }

expected: { F12: 0.998805, tolerance: { type: relative, value: 3.0e-3 } }

- id: hc\_5p1x2p1\_s1p0

enabled: true

description: Unobstructed; parallel; centres aligned.

geometry: { emitter: { w: 5.1, h: 2.1 }, receiver: { w: 5.1, h: 2.1 }, setback: 1.0, angle: 0 }

expected: { F12: 0.70274, tolerance: { type: relative, value: 3.0e-3 } }

- id: hc\_5x2\_s3p8

enabled: true

description: Unobstructed; parallel; centres aligned.

geometry: { emitter: { w: 5.0, h: 2.0 }, receiver: { w: 5.0, h: 2.0 }, setback: 3.8, angle: 0 }

expected: { F12: 0.17735, tolerance: { type: relative, value: 3.0e-3 } }

- id: hc\_20p02x1p05\_s0p81

enabled: true

description: Unobstructed; parallel; centres aligned.

geometry: { emitter: { w: 20.02, h: 1.05 }, receiver: { w: 20.02, h: 1.05 }, setback: 0.81, angle: 0 }

expected: { F12: 0.54375, tolerance: { type: relative, value: 3.0e-3 } }

- id: hc\_20p02x1p05\_s1p8

enabled: true

description: Unobstructed; parallel; centres aligned.

geometry: { emitter: { w: 20.02, h: 1.05 }, receiver: { w: 20.02, h: 1.05 }, setback: 1.8, angle: 0 }

expected: { F12: 0.27931, tolerance: { type: relative, value: 3.0e-3 } }

- id: hc\_21x1\_s3p67

enabled: true

description: Unobstructed; parallel; centres aligned.

geometry: { emitter: { w: 21.0, h: 1.0 }, receiver: { w: 21.0, h: 1.0 }, setback: 3.67, angle: 0 }

expected: { F12: 0.13285, tolerance: { type: relative, value: 3.0e-3 } }

# EXACT FILE CONTENT ENDS

--------------------------------------------------------------------------------

D) tests/test\_yaml.py (new)

--------------------------------------------------------------------------------

from src.io\_yaml import load\_cases, validate\_case\_schema, coerce\_case\_to\_cli\_kwargs

def test\_load\_and\_validate\_cases():

cases = load\_cases("docs/validation\_cases.yaml")

assert isinstance(cases, list) and len(cases) >= 2

for c in cases:

validate\_case\_schema(c)

def test\_coerce\_kwargs\_shape():

cases = load\_cases("docs/validation\_cases.yaml")

c\_enabled = next(c for c in cases if c.get("enabled"))

kw = coerce\_case\_to\_cli\_kwargs(c\_enabled)

assert {"method","emitter","receiver","setback","angle","id"}.issubset(kw.keys())

assert isinstance(kw["emitter"], tuple) and len(kw["emitter"]) == 2

--------------------------------------------------------------------------------

E) tests/test\_cases\_runner.py (new)

--------------------------------------------------------------------------------

import os

from src.cli import run\_cases

def test\_run\_cases\_creates\_summary(tmp\_path):

outdir = tmp\_path / "results"

rc = run\_cases("docs/validation\_cases.yaml", str(outdir))

assert rc == 0

summary = outdir / "cases\_summary.csv"

assert summary.exists()

text = summary.read\_text(encoding="utf-8")

assert "id,method,vf,expected,rel\_err,status,notes" in text

--------------------------------------------------------------------------------

After implementing the above:

- Run: pytest -q

- Then: python main.py --cases docs/validation\_cases.yaml --outdir results

Expected: results/cases\_summary.csv is created with rows for all cases (enabled ones not 'skipped').

Do not implement the actual solvers in this step. Keep the placeholder VF constant; we’ll replace it in later steps.

**Accept when:**

pytest -q   
python main.py --cases docs/validation\_cases.yaml --outdir results

**produces results/cases\_summary.csv**

## Step 4 — Analytical references (sanity baselines)

**Prompt to Cursor:**

Implement analytical baselines in src/analytical.py for unobstructed, parallel, centre-aligned rectangles.

Deliverables:

1) src/analytical.py

- Function:

local\_peak\_vf\_analytic\_approx(em\_w: float, em\_h: float,

rc\_w: float, rc\_h: float,

setback: float) -> float

Notes:

\* Compute the point-to-point (differential) view factor at the receiver center.

\* Use the standard formula dF = (cosθ1 \* cosθ2) / (π \* r^2) dA\_emitter,

integrated over the emitter by a modest fixed grid (e.g., 200x200) just for the baseline.

\* For parallel planes aligned normal to each other: cosθ1 = cosθ2 = setback / r.

\* Clamp denominators with EPS=1e-12, and ensure result is within [0, 1].

\* Docstring must clearly state this is a baseline approximation, not the Walton 1AI.

2) Wire CLI:

- When --method analytical is chosen, call local\_peak\_vf\_analytic\_approx(...)

- Print the value and save a one-line CSV to ./results/analytical.csv with columns:

emitter\_w,emitter\_h,receiver\_w,receiver\_h,setback,angle,vf

3) Tests:

- tests/test\_analytical.py:

\* Validate 0 ≤ F ≤ 1

\* F decreases as setback increases (monotonic sanity check)

\* Symmetry: when emitter == receiver, swapping dims doesn’t change F

\* Runtime under ~5s on defaults

4) Keep everything deterministic and within the current repo. No external deps beyond numpy.

**Accept when:**

pytest -q

python main.py --method analytical

**OR**

python main.py --method analytical --emitter 5.1 2.1 --setback 1

outputs reasonable F; tests pass.

## Step 5 — Fixed-grid method (reference)

**Prompt to Cursor:**

Implement src/fixed\_grid.py:

Goal:

- Compute local peak VF by sampling the receiver on a grid (e.g., Nx×Ny where Ny can equal Nx), and for each receiver point, integrate over emitter with a fixed emitter grid using centroid quadrature.

API:

vf\_fixed\_grid(

em\_w, em\_h, rc\_w, rc\_h, setback,

grid\_nx=100, grid\_ny=100, quadrature="centroid",

time\_limit\_s=60, eps=1e-12

) -> dict

Return:

{

"vf": float, # max pointwise VF over receiver grid

"status": "converged" | "reached\_limits" | "failed",

"samples\_emitter": int,

"samples\_receiver": int

}

Implementation notes:

- Use the differential exchange formula dF = (cosθ1 cosθ2)/(π r^2) dA\_emitter.

- For parallel, cosθ1=cosθ2= setback / r; ensure eps guard on r^2.

- Compute pointwise integrals over emitter grid for several receiver points; take the max.

- Add a simple wall-clock timeout check.

CLI:

- --method fixedgrid routes to this function.

- Save one-line CSV: geometry, grid\_nx, grid\_ny, vf, status.

Tests:

- tests/test\_fixed\_grid.py

\* Regression: vs analytical approx within 5–10% for a few geometries

\* Monotonicity: increasing grid density should not worsen result (within small tolerance)

\* Runtime: completes under ~10s on default params

**Accept when:** fixed-grid runs and produces plausible values; tests pass.

python -m pytest -q tests/test\_fixed\_grid.py

## Step 6 — Adaptive Integration (1AI) with tuning & safety

**Prompt to Cursor:**

Implement Walton-style Adaptive Integration in src/adaptive.py.

API:

vf\_adaptive(

em\_w, em\_h, rc\_w, rc\_h, setback,

rel\_tol=3e-3, abs\_tol=1e-6, max\_depth=12,

min\_cell\_area\_frac=1e-8, max\_cells=200000, time\_limit\_s=60,

init\_grid="4x4", eps=1e-12

) -> dict

Return:

{

"vf": float,

"status": "converged" | "reached\_limits" | "failed",

"iterations": int,

"achieved\_tol": float

}

Implementation outline:

- Evaluate local VF at candidate receiver points (start with center; optionally a small receiver grid).

- For each receiver point, adaptively subdivide the emitter into sub-rectangles:

\* Estimate contribution per sub-rect via a cheap rule (e.g., centroid)

\* Estimate error via corner sampling or 2×2 rule

\* Subdivide the highest-error cells until global error <= max(rel\_tol\*VF, abs\_tol), OR limits reached.

- Accumulate VF; track cells, depth, time.

- Numerical guards for near-contact, r^2, cosines.

- Stop criteria: achieved tolerance, cell/iteration cap, or time limit.

CLI:

- --method adaptive routes here; write CSV to ./results/adaptive.csv (include achieved\_tol and status).

- Respect all tuning flags from the CLI.

Tests:

- tests/test\_adaptive.py

\* Parametric tests loaded from docs/validation\_cases.yaml (use io\_yaml loader).

\* For enabled hand-calc cases: require ±0.3% relative error.

\* Mark the NIST obstructed case xfail (or skip) until occluders are implemented.

\* Include non-convergence guard tests (tiny setback; extreme aspect ratio) that must exit with status != failed/hung.

**Accept when:** adaptive passes hand-calc cases within ±0.3% and never hangs.

python -m pytest -q tests/test\_adaptive.py

## Step 7 — Monte Carlo (reference + CI)

**Prompt to Cursor:**

Implement src/montecarlo.py.

API:

vf\_montecarlo(

em\_w, em\_h, rc\_w, rc\_h, setback,

samples=200000, target\_rel\_ci=0.02, max\_iters=50, seed=42, time\_limit\_s=60,

eps=1e-12

) -> dict

Return:

{

"vf\_mean": float,

"vf\_ci95": float, # half-width

"status": "converged" | "reached\_limits" | "failed",

"samples": int

}

Method:

- Randomly sample points on emitter and the matching receiver point(s) for local VF; for local peak, either:

(a) sample across receiver points and keep max estimate per batch, or

(b) focus on center as a proxy (document choice).

- Use batch sampling; after each batch, compute standard error and 95% CI = 1.96 \* s/√N.

- Stop when CI/mean <= target\_rel\_ci or limits hit.

CLI:

- --method montecarlo routes here; write CSV to ./results/montecarlo.csv.

Tests:

- tests/test\_montecarlo.py

\* Convergence downwards with sample size

\* Loose agreement vs adaptive/fixedgrid (±5–10%)

\* Determinism with seed

**Accept when:** runs within time limits, returns CI, tests pass.

python -m pytest -q tests/test\_montecarlo.py

## Step 8 — Unified results, plotting, and --cases execution

**Prompt to Cursor:**

Enhance CLI to:

- If --cases is provided, iterate cases from docs/validation\_cases.yaml:

\* For each case:

- Choose method (default adaptive) unless overridden.

- Run solver with case-specific tolerance overrides.

- Compare to expected.F12 if provided; compute relative error.

- Record status: converged | reached\_limits | failed.

- Append to results/cases\_summary.csv

- If --plot is set:

\* Generate a 2D heatmap of pointwise VF across receiver (for fixedgrid/adaptive sample points).

\* Save to results/plots/{case\_or\_single}\_{method}.png

\* No custom color styling needed.

- Ensure CSV headers include: id, method, vf (or vf\_mean), ci95(optional), expected, rel\_err, status, iterations/samples, achieved\_tol, notes.

Add tests/test\_cases\_runner.py:

- Use a tiny YAML with 2 simple cases; assert that summary CSV is created and fields are sane.

**Accept when:** python main.py --cases docs/validation\_cases.yaml --method adaptive --plot **produces a summary CSV and plots for enabled cases.**

## Step 9 — Safeguards & robustness (no hangs)

**Prompt to Cursor:**

Harden all solvers against hangs/non-convergence:

- Add per-call wall-clock guards (time\_limit\_s) checked inside main loops.

- Add iteration/cell caps; return status="reached\_limits" (not "failed") when limits triggered.

- Stagnation detection: if best error improvement < 10% of tolerance for 5 consecutive refinement steps, stop with reached\_limits.

- Validate inputs: positive dimensions, nonzero setback; ValueError otherwise.

- EPS clamps for denominators and angles.

Improve logging:

- Print a single-line summary per run:

[method=adaptive] vf=..., achieved\_tol=..., status=..., depth=..., cells=..., time=...s

- Suppress verbose logs by default; add --verbose to enable.

Add tests/test\_safety.py:

- Extreme geometries (very small setback, huge aspect ratio).

- Enforce that no test exceeds 10s runtime and none hangs.

**Accept when:** extreme tests terminate gracefully with sensible statuses.

## Step 10 — Final polish

**Prompt to Cursor:**

Polish & docs:

- README.md:

\* Quickstart with examples:

python main.py --method adaptive --emitter 5.1 2.1 --setback 1

python main.py --cases docs/validation\_cases.yaml --method adaptive --plot

\* Explain local peak VF vs area-averaged VF.

\* Document ALL tuning knobs and where to change defaults:

src/adaptive.py: REL\_TOL, ABS\_TOL, MAX\_DEPTH, MIN\_CELL\_AREA\_FRAC, MAX\_CELLS, TIME\_LIMIT\_S, INIT\_GRID

src/fixed\_grid.py: GRID\_NX, GRID\_NY, QUADRATURE, TIME\_LIMIT\_S

src/montecarlo.py: SAMPLES, TARGET\_REL\_CI, MAX\_ITERS, SEED, TIME\_LIMIT\_S

\* State acceptance target: ±0.3% (≈99.7%) for adaptive on enabled validation cases.

\* Note NIST obstructed case exists but is disabled until occluders are implemented.

- Ensure all modules have docstrings and type hints.

- Ensure 'pytest -q' passes and default run produces outputs into ./results.

**Accept when:** README is clear, tests pass, and demo commands work.

### Final Note: One-liner to run full suite later

python main.py --cases docs/validation\_cases.yaml --method adaptive --plot

If you want, I can now supply a **tiny starter docs/validation\_cases.yaml** (2 enabled hand-calc cases + 1 disabled NIST case) so Cursor can paste it directly in Step 3.

## Coding Stage – Second Iterations

## Coding Stage –Future Iterations

* When implementing non-concentric E/Rs, make sure to tell cursor/agent/developer to consider the location of the local peak VF, as it is not at the centroid anymore.
  + For example: off-centered; non-parallel; obstructed

## Coding Stage - Run and Test (Agent / Developer)

* How to run the project?
* Test and run the project

## Coding Stage – Project Clean up

You are a top-tier Senior Repo Curator. Your tasks:

1) Summarise and consolidate all technical documentation into a single, well-structured DOCS/ folder.

2) Clean the repo: remove redundant/duplicate files, merge near-duplicates, and optimise project size.

3) DO NOT remove critical documentation or anything needed to understand, build, or run the project.

Operating rules (must follow)

- FILE-BOUNDARY: Never write outside repo root. Use relative paths only.

- DRY-RUN FIRST: Perform a non-destructive analysis, produce a plan with diffs and file lists. Only proceed after writing the plan to DOCS/cleanup-plan.md.

- NON-INTERACTIVE: No prompts. If ambiguous, keep the file and note it in the plan’s “Needs Review” section.

- PRESERVE HISTORY: Do not rewrite git history. Only add/modify/rename/delete files in a normal commit.

- NO CODE BREAKAGE: Do not move source files required by builds/tests. If in doubt, keep and mark for review.

Scope

- Source tree likely includes: web/, server/, engine/, scripts/, tests/, data/, artifacts/, logs/, DOCS/ (may exist), README, LICENSE, etc.

- “Technical documentation” includes: READMEs, design docs, API specs, schemas, comments exported as docs, installation/build guides, troubleshooting, ADRs/decisions, CHANGELOG, test strategy, runbooks, diagrams, and any \*.md/\*.rst/\*.adoc under repo.

Deliverables

*A) DOCS/ structure (create if missing):*

- DOCS/README.md → docs landing page with nav

- DOCS/00-index.md → table of contents

- DOCS/01-architecture.md → high-level system & diagrams (text-based mermaid where possible)

- DOCS/02-build-and-run.md → build matrix (Windows/macOS/Linux), local dev, CI, scripts, non-interactive rules

- DOCS/03-apis.md → REST endpoints, request/response samples

- DOCS/04-data-models.md → schemas (VS3, Controls, etc.), validation rules

- DOCS/05-engine-and-cli.md → view3d/viewht usage, runners, timeouts, dummy inputs

- DOCS/06-testing.md → test strategy (unit/integration), pytest flags, non-hanging guidance

- DOCS/07-decisions-adr.md → one-file ADR catalog (append dated entries)

- DOCS/08-troubleshooting.md → common errors (%1 invalid Win32, PATH, watchers/UNC, etc.)

- DOCS/99-glossary.md → acronyms & terms

*B) Summaries*

- Convert scattered docs into concise summaries under the appropriate file above.

- Where multiple READMEs exist, consolidate into sections and link back to source components.

- Preserve key context for AI agents & new devs.

*C) Cleanup & dedup*

- Identify duplicate/near-duplicate docs (e.g., README copies, old specs). Keep the most up-to-date version, merge differences into it, and remove the redundant file.

- For partially overlapping docs: merge content into the canonical DOCS/\* file, then replace the old doc with a pointer line:

"This content has moved to DOCS/<path>. See git history for previous versions."

- Do NOT delete: LICENSE, SECURITY, CODE\_OF\_CONDUCT, CONTRIBUTING, top-level README, CHANGELOG (if present).

*D) Optimise size (safe)*

- Remove generated logs, caches, tmp, coverage, .pytest\_cache, node\_modules in snapshots, large raw artifacts under artifacts/ or data/ that are re-generatable. Keep a tiny “golden” sample if tests/docs depend on it.

- If large binaries are truly required, move them under a single folder (e.g., third\_party/ or bin/) and document purpose + source; otherwise note them in “Needs Review”.

- Do not remove sample inputs required by smoke tests (e.g., data/dummy.vs3).

*E) Index & cross-links*

- Ensure DOCS/README.md links to every DOCS/\* page.

- Insert backlinks from component READMEs to relevant DOCS pages.

*F) Reports (must produce)*

- DOCS/cleanup-plan.md → DRY-RUN plan: files to merge/delete/keep; rationale & risk.

- DOCS/cleanup-report.md → After changes: exact file actions, bytes saved, remaining review items.

- DOCS/doc-inventory.json → Machine-readable inventory (path, size, hash, dedup-group).

*G) Safety net*

- If a file seems critical but large/duplicated, KEEP it and list in “Needs Review” with justification.

- Never delete code/config or test fixtures unless clearly generated or redundant per the plan.

***Execution steps***

**Step 1 — Inventory (DRY-RUN)**

*- Walk the tree, gather all candidate docs: \*\*\*.md, \*.rst, \*.adoc, \*.txt, /docs, /DOCS\*\*, README-like files.*

*- Compute size, hash, similarity (shingling or rough cosine on text) to detect duplicates/near-duplicates.*

- Produce DOCS/doc-inventory.json and a human summary in DOCS/cleanup-plan.md:

*- “To Merge” groups with canonical target path*

*- “To Remove” (redundant/generated) with reason*

*- “To Keep” critical docs*

*- “Needs Review” ambiguous items*

**Step 2 — Consolidation**

*- Create/refresh all DOCS/\* files listed above.*

*- For each duplicate group, merge content into the canonical DOCS file, preserving unique details, then:*

*- Replace old files with a 1–2 line pointer to the canonical DOCS page (unless they can be safely deleted).*

**Step 3 — Optimisation**

*- Remove redundant files listed in the plan (logs, caches, generated artifacts).*

*- If extremely large binary docs remain, relocate to third\_party/ or bin/ and document in DOCS/05-engine-and-cli.md or 08-troubleshooting.md.*

**Step 4 — Index & Links**

*- Update DOCS/README.md and 00-index.md with a clickable ToC.*

*- From top-level README.md, add a “Documentation” section linking into* DOCS/00-index.md.

**Step 5 — Reports & Commit**

*- Write DOCS/cleanup-report.md with:*

*- Counts & sizes before/after*

*- List of file actions (moved/merged/deleted/kept)*

*- Outstanding “Needs Review”*

*- Stage and commit changes:*

*- Commit message: "docs: consolidate into DOCS/, deduplicate, and optimize (no code changes)"*

*- Print a short summary to stdout.*

*Constraints & correctness*

*- Cross-platform only (Windows/macOS/Linux). Use Node/Python for FS ops, no shell-specific flags.*

*- Don’t touch build/test scripts beyond doc paths.*

*- All links must be relative and valid.*

*- All moves/Deletes must be reflected in the plan & report.*

*Output expectation*

*- New/updated DOCS/ files as specified.*

*- doc-inventory.json, cleanup-plan.md (pre), cleanup-report.md (post).*

*- A single commit with only documentation/cleanup changes (no source code edits).*

Begin now with the DRY-RUN (Step 1) and write the plan to DOCS/cleanup-plan.md, then proceed through Steps 2–5.

## Coding Stage – Phase Wrap Up (before rev up)

You are a top-tier Senior CI Maintainer & Release Scribe. Run a comprehensive verification of the project, fix what you safely can, and if everything passes, finalize this revision, log critical changes with high-quality notes for the next iteration.

**OPERATING RULES (must follow)**

- FILE-BOUNDARY: Never write outside repo root. Relative paths only.

- NON-INTERACTIVE: No prompts; use flags/env to avoid pauses.

- CROSS-PLATFORM: Prefer Node/Python for FS/process. Avoid bash-only flags.

- SAFE FIXES FIRST: Autofix only issues that don’t change external behavior (lint autofix, import paths, flaky tests isolation, missing exports, racey awaits, path separators). For risky changes, open TODOs and keep tests marked xfail/skip with reason.

- NO LONG-RUNNING HANGS: Start dev servers detached with readiness probes and then return. Ensure all child processes/streams close.

**SCOPE OF TESTS**

1) Static checks:

- Prettier/ESLint (web, scripts)

- TypeScript `tsc --noEmit` (web)

- MyPy or pyright (server/engine if present)

2) Python tests:

- `pytest -q --maxfail=10 --disable-warnings`

3) Node tests:

- `npm test` or `pnpm test` in web/

4) Build checks:

- `pnpm run build` (web)

- Package import sanity (server)

5) Smoke tests:

- Start web dev server \*\*detached\*\* via `.\\start-web-fixed.ps1 -WaitReady` (Windows) or equivalent Node/Python script; probe `http://localhost:3000` then stop via `.\\stop-web.ps1`.

- If `view3d(.exe)` exists, run a tiny smoke: `./bin/view3d\* ./data/dummy.vs3 ./artifacts/smoke.vf` with 60s timeout; do NOT hang on stdin; log size of output file.

**EXECUTION PLAN**

A) Prepare

- Ensure `scripts/bootstrap.(mjs|py)` has created `logs/`, `artifacts/`, `data/`, `DOCS/`.

- Ensure dummy inputs exist (e.g., `data/dummy.vs3`).

B) Run checks (bounded & logged)

- Route all stdout/stderr to `./logs/\*.log` and print a short console summary for each step.

- Use timeouts:

- individual test command ≤ 10 minutes

- smoke steps ≤ 60 seconds

- Order:

1. Lint & format checks (autofix where safe; re-run quick lint to confirm)

2. Type checks (TS, Py)

3. Python tests (pytest)

4. Node tests (web)

5. Build (web)

6. Smoke: start web (detached, readiness probe), run a simple HTTP GET, then stop

7. Smoke: run view3d binary (if present)

C) Auto-fix loop (bounded)

- If a step fails:

- Attempt \*\*one\*\* safe autofix cycle (e.g., missing export re-export, broken import path, Windows path separators, test using dev server → replace with mocked HTTP).

- Re-run only the failed step.

- If still failing, document clearly in the report with suggested fix and mark test `xfail`/`skip` with reason \*only if\* it is flaky or blocked by environment; otherwise leave as FAIL.

D) Finalize

- If any FAIL remains: write a clear \*\*DOCS/test-report.md\*\* describing:

- failing steps, stack traces (summarized), and proposed patch plan

- list of files changed by autofixes

- next actions checklist

- exit non-zero

- If all PASS:

- Generate \*\*DOCS/revision-notes.md\*\* and \*\*DOCS/CHANGELOG.md\*\* entries:

- “What changed in this revision” (bullets)

- “Why it matters”

- “How to verify locally (commands)”

- “Known limitations / TODOs for next rev”

- Update \*\*DOCS/00-index.md\*\* if new docs were added.

- Stage and commit doc/test fixes only (no unrelated code moves):

- Commit message: `chore(ci): comprehensive test pass, safe autofixes, and docs for next revision`

- Print a final summary to stdout with counts & durations.

CONCRETE COMMANDS (example defaults; adapt to actual scripts if different)

- Lint (root): `node ./scripts/run-lint.mjs` OR `pnpm -w run lint --if-present`

- TypeScript (web): `pnpm --prefix web run typecheck --if-present` or `tsc --noEmit`

- Py type check: `pyright` or `mypy server engine`

- Python tests: `python -m pytest -q --maxfail=10 --disable-warnings`

- Node tests (web): `pnpm --prefix web test --if-present`

- Build (web): `pnpm --prefix web run build --if-present`

- Web smoke:

- start: `powershell -NoProfile -ExecutionPolicy Bypass -File ".\start-web-fixed.ps1" -WaitReady`

- probe: HTTP GET `http://localhost:3000` (3 retries, 1s backoff)

- stop: `powershell -NoProfile -ExecutionPolicy Bypass -File ".\stop-web.ps1"`

- view3d smoke (if exists):

- win: `.\bin\view3d.exe .\data\dummy.vs3 .\artifacts\smoke.vf`

- posix: `./bin/view3d ./data/dummy.vs3 ./artifacts/smoke.vf`

OUTPUTS (must produce)

- `DOCS/test-report.md` → summary of the run, pass/fail table, durations, environment matrix (OS, Node, Python)

- `DOCS/revision-notes.md` → only if all PASS: concise summary for devs/AI agents (what changed, why, how to run)

- `logs/\*.log` → full logs for each step

- `artifacts/smoke.vf` → if view3d smoke ran

- Updated docs/tests with safe fixes committed as a single commit

BEGIN now. First, print the discovered scripts/commands you will run, then execute them in order with timeouts and logging. If something is missing, create a tiny adapter script under `./scripts/` to keep the flow cross-platform. No hanging, no interactivity.

# Temp Log

## Clean up, rev up

## To do

1. Mark what Cursor says and do it later: The R&D spike has successfully demonstrated the feasibility of building View3D as both a static library and WebAssembly module. The static library approach provides immediate 3-17x performance improvements with 69-75% memory reduction, while the WebAssembly approach enables client-side computation in web browsers.

2. Generate some test cases and give me the inputs so I can validate the program

## To rerun test

As a top-tier Senior CI Maintainer & Release Scribe, you previously ran a comprehensive verification of the project but ran into errors. Now you have fixed some if not all of them, so rerun the test, fix what you safely can, and if everything passes, finalize this revision with high-quality notes for the next iteration.  
  
OPERATING RULES (must follow)  
- FILE-BOUNDARY: Never write outside repo root. Relative paths only.  
- NON-INTERACTIVE: No prompts; use flags/env to avoid pauses.  
- CROSS-PLATFORM: Prefer Node/Python for FS/process. Avoid bash-only flags.  
- SAFE FIXES FIRST: Autofix only issues that don’t change external behavior (lint autofix, import paths, flaky tests isolation, missing exports, racey awaits, path separators). For risky changes, open TODOs and keep tests marked xfail/skip with reason.  
- NO LONG-RUNNING HANGS: Start dev servers detached with readiness probes and then return. Ensure all child processes/streams close.  
  
SCOPE OF TESTS  
1) Static checks:  
- Prettier/ESLint (web, scripts)  
- TypeScript `tsc --noEmit` (web)  
- MyPy or pyright (server/engine if present)  
2) Python tests:  
- `pytest -q --maxfail=10 --disable-warnings`  
3) Node tests:  
- `npm test` or `pnpm test` in web/  
4) Build checks:  
- `pnpm run build` (web)  
- Package import sanity (server)  
5) Smoke tests:  
- Start web dev server \*\*detached\*\* via `.\\start-web-fixed.ps1 -WaitReady` (Windows) or equivalent Node/Python script; probe `http://localhost:3000` then stop via `.\\stop-web.ps1`.  
- If `view3d(.exe)` exists, run a tiny smoke: `./bin/view3d\* ./data/dummy.vs3 ./artifacts/smoke.vf` with 60s timeout; do NOT hang on stdin; log size of output file.  
  
EXECUTION PLAN  
A) Prepare  
- Ensure `scripts/bootstrap.(mjs|py)` has created `logs/`, `artifacts/`, `data/`, `DOCS/`.  
- Ensure dummy inputs exist (e.g., `data/dummy.vs3`).  
  
B) Run checks (bounded & logged)  
- Route all stdout/stderr to `./logs/\*.log` and print a short console summary for each step.  
- Use timeouts:  
- individual test command ≤ 10 minutes  
- smoke steps ≤ 60 seconds  
- Order:  
1. Lint & format checks (autofix where safe; re-run quick lint to confirm)  
2. Type checks (TS, Py)  
3. Python tests (pytest)  
4. Node tests (web)  
5. Build (web)  
6. Smoke: start web (detached, readiness probe), run a simple HTTP GET, then stop  
7. Smoke: run view3d binary (if present)  
  
C) Auto-fix loop (bounded)  
- If a step fails:  
- Attempt \*\*one\*\* safe autofix cycle (e.g., missing export re-export, broken import path, Windows path separators, test using dev server → replace with mocked HTTP).  
- Re-run only the failed step.  
- If still failing, document clearly in the report with suggested fix and mark test `xfail`/`skip` with reason \*only if\* it is flaky or blocked by environment; otherwise leave as FAIL.  
  
D) Finalize  
- If any FAIL remains: write a clear \*\*DOCS/test-report.md\*\* describing:  
- failing steps, stack traces (summarized), and proposed patch plan  
- list of files changed by autofixes  
- next actions checklist  
- exit non-zero  
- If all PASS:  
- Generate \*\*DOCS/revision-notes.md\*\* and \*\*DOCS/CHANGELOG.md\*\* entries:  
- “What changed in this revision” (bullets)  
- “Why it matters”  
- “How to verify locally (commands)”  
- “Known limitations / TODOs for next rev”  
- Update \*\*DOCS/00-index.md\*\* if new docs were added.  
- Stage and commit doc/test fixes only (no unrelated code moves):  
- Commit message: `chore(ci): comprehensive test pass, safe autofixes, and docs for next revision`  
- Print a final summary to stdout with counts & durations.  
  
CONCRETE COMMANDS (example defaults; adapt to actual scripts if different)  
- Lint (root): `node ./scripts/run-lint.mjs` OR `pnpm -w run lint --if-present`  
- TypeScript (web): `pnpm --prefix web run typecheck --if-present` or `tsc --noEmit`  
- Py type check: `pyright` or `mypy server engine`  
- Python tests: `python -m pytest -q --maxfail=10 --disable-warnings`  
- Node tests (web): `pnpm --prefix web test --if-present`  
- Build (web): `pnpm --prefix web run build --if-present`  
- Web smoke:  
- start: `powershell -NoProfile -ExecutionPolicy Bypass -File ".\start-web-fixed.ps1" -WaitReady`  
- probe: HTTP GET `http://localhost:3000` (3 retries, 1s backoff)  
- stop: `powershell -NoProfile -ExecutionPolicy Bypass -File ".\stop-web.ps1"`  
- view3d smoke (if exists):  
- win: `.\bin\view3d.exe .\data\dummy.vs3 .\artifacts\smoke.vf`  
- posix: `./bin/view3d ./data/dummy.vs3 ./artifacts/smoke.vf`  
  
OUTPUTS (must produce)  
- `DOCS/test-report.md` → summary of the run, pass/fail table, durations, environment matrix (OS, Node, Python)  
- `DOCS/revision-notes.md` → only if all PASS: concise summary for devs/AI agents (what changed, why, how to run)  
- `logs/\*.log` → full logs for each step  
- `artifacts/smoke.vf` → if view3d smoke ran  
- Updated docs/tests with safe fixes committed as a single commit  
  
BEGIN now. First, print the discovered scripts/commands you will run, then execute them in order with timeouts and logging. If something is missing, create a tiny adapter script under `./scripts/` to keep the flow cross-platform. No hanging, no interactivity.

## Temp Adp Rad Calc

Create another test case with a difference emitter size.

21m \* 1 m (w\*d), test the adaptive method for view factors when the setbacks between emitter and receive are 0.68, 1.67, 3.67, and 6.67m