Quantum Light Simulation

A Numerical Solver for the Time-Dependent Schrödinger Equation
with Applications to Quantum Optical Imaging
Technical Reference Document

Aaron Gonsior

March 2026

Contents

1 Introduction **4**

2 Governing Equation **4**
- 2.1 The Time-Dependent Schrödinger Equation . . . 4
- 2.2 Hamiltonian Splitting . . . 4
- 2.3 Probability Density and Intensity . . . 4

3 Spatial Discretisation **5**
- 3.1 2D: Nine-Point Isotropic Laplacian . . . 5
- 3.2 3D: Nineteen-Point Isotropic Laplacian . . . 5

4 Time Integration: RK4 Solver **5**
- 4.1 Formulation . . . 5
- 4.2 Properties . . . 6
- 4.3 Stability: CFL Condition . . . 6

5 Time Integration: Split-Step Fourier Method (SSFM) **6**
- 5.1 Strang Splitting . . . 6
- 5.2 Algorithm . . . 6
- 5.3 Kinetic Phase Factor in Fourier Space . . . 6
- 5.4 Separability of the Kinetic Factor . . . 7
- 5.5 Properties . . . 7
- 5.6 GPU Memory Optimisation . . . 7

6 Complex Potential and Material Model **8**
- 6.1 Complex Potential . . . 8
- 6.2 Material Library . . . 8
- 6.3 Absorbing Boundary Layer . . . 9

7 Source Model: Incoherent Photons **9**
- 7.1 Gaussian Wave Packet . . . 9
- 7.2 Radial (Spherical/Circular) Wave Source . . . 9
- 7.3 Incoherent Averaging . . . 10

8 Scene Description System **10**
- 8.1 YAML-Based Scene Definition . . . 10
- 8.2 Supported Optical Elements . . . 11
- 8.3 3D Pinhole Camera Scene . . . 11
- 8.4 2D Torus Slice Scene . . . 12

9 Imaging Regime: Fresnel Number and Grid Constraints **12**
- 9.1 Fresnel Number . . . 12
- 9.2 Nyquist Constraint on the Simulation Frequency . . . 12
- 9.3 Material Opacity at High Frequency . . . 13
- 9.4 VRAM Budget and Maximum Grid Size . . . 13
- 9.5 Achievable Fresnel Number and Path to Geometric Imaging . . . 13

10 Rendering and Visualisation **14**
- 10.1 Sensor Plane Rendering . . . 14
- 10.2 Camera Frame . . . 14
- 10.3 Raytracing Scene Overview . . . 14

10.4 GIF Animation 14

11 Implementation Architecture **14**
11.1 Code Organisation 14
11.2 Parallelism 15
11.3 Data Flow: One Simulation Step 15
11.4 Cloud Deployment 15
11.5 Dynamic Simulation Parameters 15
11.6 Web Frontend 16

12 Comparison of Solvers **16**

13 Numerical Considerations **16**
13.1 Dispersion Relation 16
13.2 Norm Conservation 16
13.3 Boundary Conditions 16

14 Physical Phenomena Captured **16**

15 Known Limitations and Future Directions **17**

A Derivation: Strang Splitting Error **17**

B FFT Frequency Mapping **18**

C Nine-Point 2D Laplacian Stencil Derivation **18**

1 Introduction

Go Quantum Sim is a research-oriented numerical simulator that solves the time-dependent Schrödinger equation (TDSE) on regular Cartesian grids in two and three spatial dimensions. Its primary application is the study of quantum-optical phenomena—diffraction, interference, and wave–particle duality—through full-wave simulations of photons propagating through macroscopic optical scenes (pinhole cameras, parabolic reflectors, lenses, double slits, and compound instruments).

The simulator provides two complementary execution backends:

- A **CPU backend** written in Go, using goroutine-based parallelism.
- A **GPU backend** written in Python, using the Taichi framework for CUDA kernel generation and PyTorch for GPU-accelerated FFTs.

Both backends share the same scene description files (YAML) and implement the same physics, enabling cross-validation and flexible performance trade-offs.

Two time-integration schemes are provided: the classical fourth-order Runge–Kutta method (RK4) and the Split-Step Fourier Method (SSFM). This document presents the mathematical foundations, discretisation strategies, material models, source models, and implementation details of the simulator.

2 Governing Equation

2.1 The Time-Dependent Schrödinger Equation

The simulator solves the TDSE in natural units ($\hbar = 1$):

$$\boxed{i\,\frac{\partial \psi(\mathbf{r},t)}{\partial t} = \hat{H}\,\psi(\mathbf{r},t) = \Big[-\frac{1}{2}\nabla^2 + V(\mathbf{r})\Big]\,\psi(\mathbf{r},t),} \tag{1}$$

where

- $\psi(\mathbf{r},t) \in \mathbb{C}$ is the complex-valued wavefunction (probability amplitude),
- ∇^2 is the Laplacian operator encoding kinetic energy,
- $V(\mathbf{r}) = V_{\mathrm{re}}(\mathbf{r}) + i\,V_{\mathrm{im}}(\mathbf{r}) \in \mathbb{C}$ is the (generally complex) potential encoding material properties.

The formal solution over one time step Δt is

$$\psi(\mathbf{r}, t+\Delta t) = e^{-i\,\hat{H}\,\Delta t}\,\psi(\mathbf{r},t). \tag{2}$$

2.2 Hamiltonian Splitting

The Hamiltonian decomposes into kinetic and potential parts:

$$\hat{H} = \hat{T} + \hat{V}, \qquad \hat{T} = -\frac{1}{2}\nabla^2, \qquad \hat{V} = V(\mathbf{r}). \tag{3}$$

Since $[\hat{T}, \hat{V}] \neq 0$ in general, the exponential does not factor exactly. The two solvers exploit this decomposition differently (see Sections 4 and 5).

2.3 Probability Density and Intensity

The measurable quantity is the probability density:

$$\rho(\mathbf{r},t) = |\psi(\mathbf{r},t)|^2 = \big[\mathrm{Re}\,\psi\big]^2 + \big[\mathrm{Im}\,\psi\big]^2. \tag{4}$$

The simulator accumulates a time-integrated intensity (analogous to a long camera exposure):

$$I(\mathbf{r}) = \sum_{n=0}^{N_{\mathrm{steps}}-1} |\psi(\mathbf{r}, t_n)|^2. \tag{5}$$

3 Spatial Discretisation

The domain is discretised on a uniform Cartesian grid with spacing $h = \Delta x = \Delta y = \Delta z$ (set by the `resolution` parameter, typically $h = 1.0$ in grid units).

3.1 2D: Nine-Point Isotropic Laplacian

In two dimensions the standard five-point stencil is anisotropic at high spatial frequencies. The simulator uses a nine-point isotropic stencil with weights

$$\nabla^2\psi\Big|_{i,j} \approx \frac{1}{h^2}\left[\frac{2}{3}\big(\psi_{i\pm1,j} + \psi_{i,j\pm1}\big) + \frac{1}{6}\big(\psi_{i\pm1,j\pm1}\big) - \frac{10}{3}\,\psi_{i,j}\right], \tag{6}$$

where the sum runs over the four nearest (face) neighbours with weight $2/3$ and the four diagonal neighbours with weight $1/6$. The weights satisfy the consistency condition

$$4 \times \frac{2}{3} + 4 \times \frac{1}{6} - \frac{10}{3} = 0, \tag{7}$$

and yield $\mathcal{O}(h^2)$ accuracy with improved angular isotropy compared to the five-point stencil.

3.2 3D: Nineteen-Point Isotropic Laplacian

In three dimensions an analogous isotropic stencil is used. It sums the six face neighbours and twelve edge neighbours (corner-diagonals are omitted):

$$\nabla^2\psi\Big|_{i,j,k} \approx \frac{1}{h^2}\left[\frac{1}{3}\sum_{6\text{ face}}\psi_{\text{face}} + \frac{1}{6}\sum_{12\text{ edge}}\psi_{\text{edge}} - 4\,\psi_{i,j,k}\right], \tag{8}$$

with the weight consistency condition

$$6 \times \frac{1}{3} + 12 \times \frac{1}{6} - 4 = 0. \tag{9}$$

Remark 3.1. *Both stencils reduce grid anisotropy ("starburst" artifacts along the Cartesian axes) at the cost of increased stencil width. In the RK4 solver the Laplacian is evaluated explicitly; in SSFM the Laplacian is bypassed entirely (the kinetic operator is applied exactly in Fourier space).*

4 Time Integration: RK4 Solver

4.1 Formulation

The TDSE (1) is rewritten as an ODE system:

$$\frac{d\psi}{dt} = f(\psi, t) = -i\,\hat{H}\,\psi. \tag{10}$$

The classical fourth-order Runge–Kutta method advances one time step via

$$\boxed{\begin{aligned} k_1 &= -i\,\hat{H}\,\psi^n, \\ k_2 &= -i\,\hat{H}\Big(\psi^n + \tfrac{\Delta t}{2}\,k_1\Big), \\ k_3 &= -i\,\hat{H}\Big(\psi^n + \tfrac{\Delta t}{2}\,k_2\Big), \\ k_4 &= -i\,\hat{H}\Big(\psi^n + \Delta t\,k_3\Big), \\ \psi^{n+1} &= \psi^n + \frac{\Delta t}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big). \end{aligned}} \tag{11}$$

4.2 Properties

- **Accuracy**: Local truncation error $\mathcal{O}(\Delta t^5)$; global error $\mathcal{O}(\Delta t^4)$.
- **Cost**: Four Hamiltonian evaluations per step, each requiring the full Laplacian stencil. In 3D, this means four passes over the $W \times H \times D$ grid.
- **Unitarity**: RK4 is *not* symplectic or unitary. The norm $\|\psi\|^2$ can drift over long simulations, which is a known limitation.
- **Storage (3D)**: Five auxiliary grids $(k_1, k_2, k_3, k_4, \text{tmp})$, each of size $W \times H \times D$ complex values.

4.3 Stability: CFL Condition

For the Schrödinger equation with the explicit finite-difference Laplacian, the stability condition takes the form

$$\Delta t \;\le\; \frac{(\Delta x)^2}{2\,k_{\max}^2}, \tag{12}$$

where $k_{\max} = 2\pi f_{\max}/c$ is the maximum wave number present in the simulation. With $\Delta x = 1.0$ and typical frequencies $f \approx 3$, a time step of $\Delta t \le 0.15$ is empirically safe.

5 Time Integration: Split-Step Fourier Method (SSFM)

5.1 Strang Splitting

The SSFM applies the *Strang splitting* [1] (symmetric operator splitting) to the propagator (2):

$$e^{-i(\hat{T}+\hat{V})\Delta t} \approx e^{-i\hat{V}\,\Delta t/2}\; e^{-i\hat{T}\,\Delta t}\; e^{-i\hat{V}\,\Delta t/2} + \mathcal{O}(\Delta t^3). \tag{13}$$

This is second-order accurate in Δt.

5.2 Algorithm

One SSFM time step proceeds as follows:

(1) Half potential:	$\psi \;\leftarrow\; \psi \cdot e^{-i\,V(\mathbf{r})\,\Delta t/2}$	(real space)
(2) Forward FFT:	$\hat{\psi} = \mathcal{F}[\psi]$	(to k-space)
(3) Full kinetic:	$\hat{\psi} \;\leftarrow\; \hat{\psi} \cdot e^{-i\,\lvert\mathbf{k}\rvert^2 \Delta t/2}$	(k-space)
(4) Inverse FFT:	$\psi = \mathcal{F}^{-1}[\hat{\psi}]$	(back to real space)
(5) Half potential:	$\psi \;\leftarrow\; \psi \cdot e^{-i\,V(\mathbf{r})\,\Delta t/2}$	(real space)

(14)

5.3 Kinetic Phase Factor in Fourier Space

The kinetic operator is diagonal in Fourier space. For a grid point (i, j, k) in the frequency domain, the discrete wave vector components are

$$k_\alpha = \frac{2\pi\,\tilde{n}_\alpha}{N_\alpha\,\Delta x}, \qquad \tilde{n}_\alpha = \begin{cases} n_\alpha, & 0 \le n_\alpha < \lceil N_\alpha/2 \rceil, \\ n_\alpha - N_\alpha, & \text{otherwise}, \end{cases} \tag{15}$$

where $\alpha \in \{x, y, z\}$, N_α is the grid size along axis α, and $n_\alpha \in \{0, 1, \ldots, N_\alpha - 1\}$ is the FFT bin index.

The full kinetic phase factor is

$$K(k_x, k_y, k_z) = \exp\Big(-\frac{i\,\Delta t}{2}\big(k_x^2 + k_y^2 + k_z^2\big)\Big). \tag{16}$$

5.4 Separability of the Kinetic Factor

Because the exponent is a sum of independent terms, the 3D kinetic factor factorises as a product of three 1D factors:

$$K(k_x, k_y, k_z) = \underbrace{e^{-i\Delta t\, k_x^2/2}}_{K_x(k_x)} \cdot \underbrace{e^{-i\Delta t\, k_y^2/2}}_{K_y(k_y)} \cdot \underbrace{e^{-i\Delta t\, k_z^2/2}}_{K_z(k_z)}. \tag{17}$$

This is exploited in the GPU implementation: instead of storing a full $W \times H \times D$ complex tensor ($\sim 1.6\,\text{GB}$ at $700 \times 560 \times 560$ complex64), only three 1D arrays of sizes W, H, D are stored. The step $\hat{\psi} \mathrel{*}= K$ is applied as three sequential in-place broadcasts:

$$\hat{\psi}_{i,j,k} \;\leftarrow\; \hat{\psi}_{i,j,k} \cdot K_x(k_x^{(i)}) \cdot K_y(k_y^{(j)}) \cdot K_z(k_z^{(k)}). \tag{18}$$

5.5 Properties

- **Accuracy**: Second-order (Strang splitting).
- **Kinetic operator**: Applied *exactly* in Fourier space—no finite-difference approximation to ∇^2. This eliminates grid-anisotropy artifacts inherent in all stencil-based Laplacians.
- **Unitarity**: Each phase-factor multiplication is unitary (multiplication by a complex number of unit modulus) when $V_{\text{im}} = 0$. SSFM thus preserves $\|\psi\|$ to machine precision in the lossless case.
- **Storage (3D GPU)**: No auxiliary grids (k_1–k_4, tmp) needed, saving $\sim 7\,\text{GB}$ VRAM compared to RK4.
- **FFT**: The Go CPU backend uses `gonum/dsp/fourier` (1D transforms applied sequentially along each axis). The GPU backend uses PyTorch's `torch.fft.fft` applied as three sequential 1D batched FFTs (avoids the large workspace of a monolithic 3D cuFFT plan).

5.6 GPU Memory Optimisation

The 3D GPU implementation (Taichi + PyTorch, targeting NVIDIA L40S with 48 GB VRAM) requires careful memory management. Three key optimisations are applied:

1. **Taichi in-place potential step.** The half-potential step $\psi \leftarrow \psi \cdot e^{-iV\Delta t/2}$ is computed by a Taichi CUDA kernel operating directly on the `psi` field, reading $V(\mathbf{r})$ from the existing `potential` field. This avoids storing a $W \times H \times D$ complex potential-half tensor ($\sim 1.6\,\text{GB}$).
2. **Separable kinetic factors.** As described in Section 5.4, the kinetic phase factor is stored as three 1D arrays ($\sim 44\,\text{KB}$ total) instead of a full 3D tensor.
3. **Sequential 1D FFTs.** The 3D FFT is decomposed into three sequential 1D `torch.fft.fft` calls (one per axis). PyTorch's caching allocator reuses freed blocks for each subsequent call without new `cudaMalloc` invocations. Note that `psi.to_torch()` followed by `view_as_complex` creates a storage reference chain, so the peak PyTorch allocation is $3\times$ the wavefunction size (to_torch buffer + complex view + FFT output).

The resulting peak VRAM budget is:

Table 1: GPU VRAM budget per cell for SSFM (after dtype optimisation).

Allocation	**Dtype**	**B/cell**	**Notes**
Taichi: `psi` (Vector2)	float32	8	wavefunction (must stay f32 for Laplacian precision)
Taichi: `accumulator`	float32	4	time-integrated $\|\psi\|^2$
Taichi: `potential` (Vector2)	float32	8	complex $V(\mathbf{r})$
Taichi: `materials`	int32	4	material type per voxel
Taichi subtotal		24	
`psi_t` (torch copy)	float32	8	GPU–GPU copy for FFT step
FFT input (`view_as_complex`)	complex64	8	storage alias of `psi_t` (not freed)
FFT output (1D pass)	complex64	8	reused by subsequent passes
Peak total		**48**	+ negligible cuFFT workspace (smooth sizes)

6 Complex Potential and Material Model

6.1 Complex Potential

Materials are modelled by a spatially varying complex potential

$$V(\mathbf{r}) = V_{\text{re}}(\mathbf{r}) + i\, V_{\text{im}}(\mathbf{r}). \tag{19}$$

Real part—refractive index. The real potential induces a local wave-vector shift. For a particle with kinetic energy $E = k^2/2$, the local effective wave number becomes

$$k_{\text{eff}} = \sqrt{2(E - V_{\text{re}})}, \tag{20}$$

yielding an effective refractive index

$$n(E) = \frac{k_{\text{eff}}}{k} = \sqrt{1 - \frac{V_{\text{re}}}{E}}. \tag{21}$$

- $V_{\text{re}} < 0$ (attractive): $n > 1$ — glass-like, increases phase velocity denominator $\Rightarrow$ focusing.
- $V_{\text{re}} > 0$ (repulsive): $n < 1$ for $V_{\text{re}} < E$ (partial reflection), or n imaginary for $V_{\text{re}} > E$ (total reflection / evanescent decay) — mirrors, walls.

Imaginary part—absorption. A negative imaginary potential causes exponential damping:

$$\psi(t) \;\propto\; e^{-i\,(V_{\text{re}} + i\,V_{\text{im}})\,t} = e^{V_{\text{im}}\,t}\; e^{-i\,V_{\text{re}}\,t}. \tag{22}$$

For $V_{\text{im}} < 0$, the amplitude decays as $|\psi| \propto e^{V_{\text{im}}\,t}$, modelling absorbing materials and detector surfaces.

6.2 Material Library

All scenes share a common material definition file (`materials.yaml`). Potentials are calibrated at a reference frequency $f_{\text{ref}} = 2.0$ and are scaled at runtime by $f_{\text{centre}}/f_{\text{ref}}$ to preserve reflection coefficients across wavelengths.

Table 2: Material definitions and their physical roles.

Material	**mat_id**	V_{re}	**Physical role**
Vacuum	0	0.0	Free propagation ($n = 1$)
Glass	1	-1.5	Refractive lens ($n > 1$)
Wall	2	$+20.0$	Opaque hard barrier (total reflection)
Mirror	3	$+10.0$	Reflective surface
Absorbing Wall	4	0.0	Impedance-matched absorber (V_{im} added dynamically)
Partial Mirror	5	$+1.5$	Semi-transparent scatterer ($V < E$)
Scatter	6	$+6.0$	Diffuse opaque surface ($V \gg E$, total reflection)
Milky Glass	7	$+2.5$	Translucent frosted glass (partial transmission)

6.3 Absorbing Boundary Layer

To prevent spurious reflections from the grid edges, a smooth absorbing boundary ramp is applied in a border region of width b cells. The imaginary potential ramps linearly:

$$V_{\text{im}}^{(\text{border})} = -\gamma \, \frac{d}{b}, \tag{23}$$

where $d = \max(b - x, x - (N - b), \dots)$ is the distance to the nearest grid edge (clamped to zero inside the interior) and γ is the absorption strength ($\gamma = 0.5$ in 2D, $\gamma = 2.0$ in 3D with $b = 10{-}20$ cells).

This is a simple *complex absorbing potential* (CAP) approach, which is effective but not perfectly impedance-matched (some residual reflection persists, unlike Perfectly Matched Layers).

7 Source Model: Incoherent Photons

7.1 Gaussian Wave Packet

Individual photons are injected as Gaussian wave packets:

$$\psi_{\text{add}}(\mathbf{r}) = A \, \exp\Big(-\frac{|\mathbf{r} - \mathbf{r}_0|^2}{2\sigma^2}\Big) \, \exp(i \, \mathbf{k} \cdot \mathbf{r}), \tag{24}$$

where $\mathbf{r}_0$ is the source position, σ is the spatial width, $\mathbf{k} = (k_x, k_y, k_z)$ is the momentum (wave vector), and A is a normalisation amplitude:

$$A = \frac{I_0}{\sigma \sqrt{\pi}}, \qquad \text{such that } \int |\psi_{\text{add}}|^2 \, d\mathbf{r} = I_0^2. \tag{25}$$

7.2 Radial (Spherical/Circular) Wave Source

For an incoherent point source, the phase is radial rather than planar:

$$\psi_{\text{add}}(\mathbf{r}) = \frac{A}{\max(r, \epsilon)^{(d-1)/2}} \, \exp\Big(-\frac{r^2}{2\sigma^2}\Big) \, \exp(i \, f \, r), \tag{26}$$

where $r = |\mathbf{r} - \mathbf{r}_0|$, f is the frequency, d is the spatial dimension (2 or 3), and $r^{-(d-1)/2}$ provides the correct amplitude falloff for a spherical ($1/r$) or cylindrical ($1/\sqrt{r}$) wave.

7.3 Incoherent Averaging

To simulate a thermal (non-coherent) light source, at each time step the simulator injects N_{ph} photons with *random* frequencies and *random* global phase offsets:

$$\text{For each photon } j = 1, \ldots, N_{\text{ph}}: \qquad \begin{cases} f_j \sim \mathcal{U}(f_{\min}, f_{\max}), \\ \varphi_j \sim \mathcal{U}(0, 2\pi). \end{cases} \tag{27}$$

Each photon is injected as a *spatially coherent* wave packet—the phase offset φ_j is constant across all voxels within one injection:

$$\psi_{\text{add}}(\mathbf{r}) = \frac{A}{\max(r, \epsilon)} \exp\Big(-\frac{r^2}{2\sigma^2}\Big) \exp\big(i\,(f_j\, r + \varphi_j)\big). \tag{28}$$

Spatial coherence is essential: the parabolic mirror can only collimate a wave whose wavefronts share a consistent phase relationship. A per-voxel random phase would scatter energy across all wave vectors, preventing beam formation and producing an expanding noise cloud instead of a directed wave.

Incoherence between photons arises from two mechanisms:

- **Random frequency**: different photons carry different f_j, so their interference fringes oscillate at beat frequencies and average out in the accumulator over a coherence time $\tau_c \sim 1/(f_{\max} - f_{\min})$.
- **Random global phase**: φ_j prevents constructive build-up between photons that happen to share the same frequency.

Over many time steps, the accumulated intensity $I(\mathbf{r})$ converges to the incoherent sum of many single-photon diffraction patterns, reproducing physically realistic imaging and diffraction phenomena without requiring perfect temporal coherence.

8 Scene Description System

8.1 YAML-Based Scene Definition

Scenes are defined in YAML files under `scenes/2D/` and `scenes/3D/`, shared between the Go and Python backends. All spatial parameters use *fractional coordinates*:

$$x_{\text{phys}} = x_{\text{fw}} \times W, \qquad y_{\text{phys}} = y_{\text{fh}} \times H, \qquad z_{\text{phys}} = z_{\text{fd}} \times D, \tag{29}$$

where W, H, D are the grid dimensions. This makes scene definitions resolution-independent.

8.2 Supported Optical Elements

Table 3: Primitive and compound optical elements.

Element	Dim.	Description
`box` / `rect`	2D/3D	Axis-aligned rectangular solid or slab
`circle` / `sphere`	2D/3D	Spherical region
`cylinder`	3D	X-axis-aligned cylinder
`torus`	3D	YZ-plane torus (parametric: major & minor radius)
`parabolic_mirror`	2D	Parabolic reflector with arbitrary orientation
`absorbing_plane`	3D	Disc-shaped absorber with arbitrary normal
`pinhole_camera_3d`	3D	Compound: box with pinhole aperture, optional hood
`pinhole_camera_2d`	2D	Compound: 2D box with slit aperture, rotatable via `target`
`spotlight`	2D/3D	Compound: parabolic mirror + focal-point source + absorbing sleeve
`sensor`	3D	Disc-shaped sensor for monitoring intensity at arbitrary position
`sundisc`	3D	Clean parallel-ray source (alternative to spotlight)

8.3 3D Pinhole Camera Scene

The primary 3D scene models a pinhole camera observing a reflective torus. The geometric magnification of the pinhole system is

$$M = \frac{v}{u}, \tag{30}$$

where u is the object-to-pinhole distance and v is the pinhole-to-sensor distance. The projected image diameter on the sensor plane is $D_{\text{proj}} = D_{\text{obj}} \times M$.

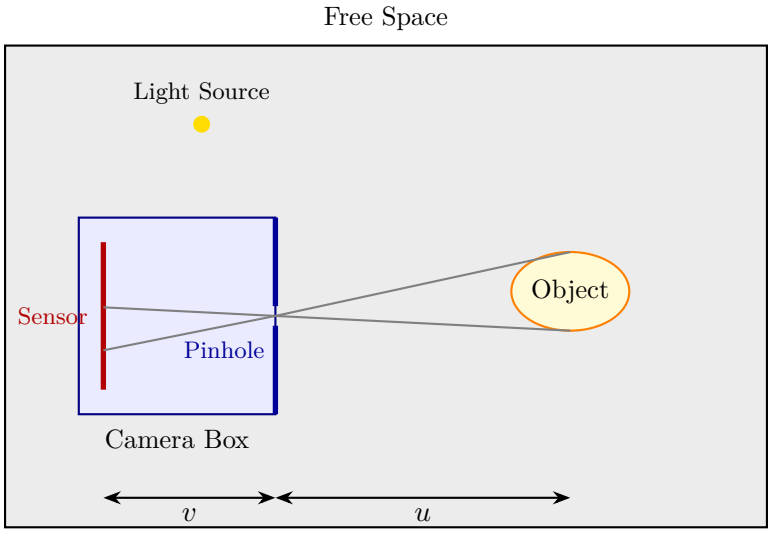

Figure 1: Schematic of the 3D pinhole camera scene. Rays from the object converge through the pinhole and form an inverted image on the sensor plane. Magnification $M = v/u$.

8.4 2D Torus Slice Scene

Scene 04 (`scenes/2D/scene_04.yaml`) is a 2D cross-section of the 3D pinhole-camera setup, modelling an X–Y slice through the torus. It features:

- Two `partial_mirror` circles representing the ring cross-sections (top and bottom of the torus slice).
- A rotated `pinhole_camera_2d` aimed at the torus centre via the `target` parameter; the sensor pixel list rotates with the camera body.
- A `spotlight` compound source with focal point, parabolic mirror, and absorbing sleeve for beam collimation.

This scene validates the 3D optics in a cheaper 2D simulation (SSFM, Taichi GPU) before committing to a full 3D cloud run.

9 Imaging Regime: Fresnel Number and Grid Constraints

9.1 Fresnel Number

The sharpness of the simulated pinhole camera image is characterised by the **Fresnel number**

$$N_F = \frac{a^2}{\lambda L}, \tag{31}$$

where a is the pinhole radius (pixels), $\lambda = 2\pi/k$ is the wavelength (pixels), and L is the pinhole-to-sensor depth (pixels).

Table 4: Imaging regime as a function of Fresnel number.

N_F	**Regime**	**Image appearance**
< 1	Fraunhofer (far-field)	Overlapping Airy rings; object outline unresolvable
1–5	Fresnel (near-field)	Partially resolved; ring structure begins to emerge
≥ 5	Geometric optics	Sharp, geometrically inverted image of the object

9.2 Nyquist Constraint on the Simulation Frequency

The parameter `freq` in the scene YAML files and in the GPU injection kernel is the *angular wavenumber* k in units of rad/px:

$$\phi_{\text{prop}}(\mathbf{r}) = f \cdot r = k \cdot |\mathbf{r} - \mathbf{r}_0|, \tag{32}$$

confirmed by the kernel source `prop_phase_arg = freq * dist`. With grid spacing $\Delta x =$ `resolution` $= 1.0\,\text{px}$, the **Nyquist limit** is

$$k_{\max} = \frac{\pi}{\Delta x} \approx 3.14\ \text{rad/px} \quad \Longrightarrow \quad \lambda_{\min} = 2\ \text{px}. \tag{33}$$

Any `freq` $> \pi$ creates a sub-pixel wavelength that is aliased on the grid and cannot be faithfully simulated. Scene 01 uses `freq_min=2.5`, `freq_max=3.1`, corresponding to $\lambda \approx 2.0$–$2.5\,\text{px}$ — at the Nyquist boundary.

9.3 Material Opacity at High Frequency

All material real potentials scale linearly with frequency at runtime ($V_{\text{eff}} = V_{\text{base}} \times f_c/f_{\text{ref}}$), but the photon kinetic energy scales quadratically ($E = k^2/2$). Consequently the opacity ratio V_{eff}/E *decreases* with frequency:

$$\frac{V_{\text{eff}}}{E} = \frac{2\,V_{\text{base}}}{f_{\text{ref}}\,f_c}. \tag{34}$$

Total reflection ($V_{\text{eff}} > E$) is maintained only while $f_c < 2V_{\text{base}}/f_{\text{ref}}$. For the scatter material ($V_{\text{base}} = 6$, $f_{\text{ref}} = 2$), this gives $f_c < 6\,\text{rad/px}$ — comfortably above the Nyquist limit of $\pi \approx 3.14\,\text{rad/px}$. All scene-01 materials therefore remain opaque throughout the full usable frequency range.

9.4 VRAM Budget and Maximum Grid Size

The GPU script scales the grid as $W = \lfloor W_{\text{YAML}} \cdot s/6 \rfloor$, etc., where s is `sizeFactor`. Peak memory per voxel is 48 bytes (SSFM, including PyTorch FFT overhead) or 64 bytes (RK4 with five auxiliary grids). For scene 01 (YAML: $2400 \times 800 \times 800$):

Table 5: Maximum `sizeFactor` on an NVIDIA L40S (48 GB, AWS g6e.xlarge). Grid dimensions must be cuFFT-smooth (factors of $2, 3, 5, 7$ only).

Solver	**Max sf**	**Grid**	**Cells**	**Peak VRAM**
RK4	4.5	$1800 \times 600 \times 600$	648 M	$\sim$39 GB
SSFM	4.86	$1944 \times 648 \times 648$	816 M	$\sim$37 GB

9.5 Achievable Fresnel Number and Path to Geometric Imaging

For the current scene-01 geometry (`pinhole_radius_fh` $= 0.10$, $\Delta x_{\text{cam}} = 0.199W$), the Fresnel number scales linearly with `sizeFactor` at fixed k:

$$N_F(s) \approx 1.12\,s \qquad (k = \pi \text{ rad/px, Nyquist}). \tag{35}$$

Table 6: Achievable Fresnel number within Nyquist and VRAM constraints (`pinhole_radius_fh` $= 0.10$).

Solver	**sf**	N_F at $k = \pi$	**Regime**	**Notes**
RK4	4.5	5.0	Geometric (borderline)	Grid $1800 \times 600 \times 600$
SSFM	4.86	5.4	Geometric	Grid $1944 \times 648 \times 648$

With the current pinhole radius (`pinhole_radius_fh` $= 0.10$), both solvers reach the geometric-imaging regime ($N_F \geq 5$) at their maximum grid sizes. At the scene's centre frequency $k_c \approx 2.55\,\text{rad/px}$ the Fresnel number is $N_F \approx 4.4$ (SSFM, $s = 4.86$), placing the simulation in the near-geometric regime with a clearly resolvable torus outline.

Geometric blur. The pinhole geometric blur on the sensor is $\sigma_{\text{geom}} = a \times v/u \approx 27\,\text{px}$ at $s = 4.86$, compared to the projected torus ring width of $\sim$53 px. The Airy disc radius at $k = \pi$ is $\sim$8 px, confirming that pinhole blur rather than diffraction is the limiting factor.

Three ring-suppression strategies are active:

1. Geometric regime N_F (large pinhole relative to wavelength).
2. Wide spectral bandwidth ($\Delta k/k \approx 43\%$) for incoherent ring washing.

3. Strong torus reflection (scatter material, $V = 6$) to brighten the geometric core relative to diffraction rings.

The monitoring script `compute_reflection_eta.ps1` (called by `check_simulation.bat`) computes and displays N_F, the imaging regime, and the Nyquist utilisation at every polling interval, allowing rapid assessment of scene parameters before committing to a full cloud run.

10 Rendering and Visualisation

10.1 Sensor Plane Rendering

For 3D simulations, the sensor is a 2D plane embedded in the 3D grid. A camera model maps output pixel coordinates $(u, v) \in [-1, 1]^2$ to physical grid coordinates using an orthonormal frame:

$$\mathbf{r}_{\text{sensor}}(u, v) = \mathbf{r}_c + u \cdot \hat{\mathbf{r}} \cdot \frac{S}{2} + v \cdot \hat{\mathbf{u}} \cdot \frac{S}{2}, \tag{36}$$

where $\mathbf{r}_c$ is the sensor centre, $\hat{\mathbf{r}}$ and $\hat{\mathbf{u}}$ are the right and up basis vectors (computed via Gram–Schmidt from the camera forward direction and a world-up reference), and S is the physical sensor size.

10.2 Camera Frame

The camera frame vectors are derived from a position $\mathbf{P}$ and a look-at target $\mathbf{T}$:

$$\hat{\mathbf{f}} = \frac{\mathbf{T} - \mathbf{P}}{|\mathbf{T} - \mathbf{P}|}, \qquad \hat{\mathbf{r}} = \hat{\mathbf{f}} \times \hat{\mathbf{y}}_{\text{up}}, \qquad \hat{\mathbf{u}} = \hat{\mathbf{r}} \times \hat{\mathbf{f}}, \tag{37}$$

where $\hat{\mathbf{y}}_{\text{up}} = (0, 1, 0)$ is the world up vector.

10.3 Raytracing Scene Overview

A simple forward-marching raytracer renders static overviews of the 3D scene geometry. For each output pixel, a ray is marched from the camera through the volume:

$$\mathbf{p}(s) = \mathbf{P} + s\,\hat{\mathbf{d}}, \qquad s = 0, \Delta s, 2\Delta s, \ldots, s_{\max}. \tag{38}$$

The first intersection with a non-vacuum material determines the pixel colour.

10.4 GIF Animation

Output frames (PNG images of the accumulated intensity) are collected and assembled into animated GIFs at 10 frames per second (100 ms per frame).

11 Implementation Architecture

11.1 Code Organisation

The project consists of three main packages:

Table 7: Package structure.

Package / Script	Responsibility
`simulation/`	2D CPU backend (Go): grid, physics, RK4, SSFM, rendering
`simulation3d/`	3D CPU backend (Go): grid, physics, RK4, SSFM, camera, rendering
`gpu_sim.py`	3D GPU backend (Python/Taichi/PyTorch)
`gpu_sim_2d.py`	2D GPU backend (Python/Taichi)
`scene_loader.py`	Shared Python YAML scene loader
`main.go`	CLI entry point and simulation loop orchestration

11.2 Parallelism

Go CPU backend. Both 2D and 3D simulations use a `parallelLoop` abstraction that distributes loop iterations across 256 goroutines (configurable). The RK4 Hamiltonian evaluation, source injection, intensity accumulation, and rendering are all parallelised over the outermost spatial index.

Python GPU backend. Taichi compiles Python kernel functions into CUDA code at runtime. Each Taichi `@ti.kernel` function launches a GPU grid covering all field elements. The SSFM kinetic step uses PyTorch's `torch.fft.fft` (backed by cuFFT) for the Fourier transforms.

11.3 Data Flow: One Simulation Step

The main simulation loop (common to both backends) follows this pattern per time step:

1. **Source injection**: Add N_{ph} incoherent Gaussian pulses, each with a random frequency and random global phase offset.
2. **Intensity accumulation**: $\texttt{acc}[\mathbf{r}] \mathrel{+}= |\psi(\mathbf{r})|^2$.
3. **Frame output** (every n-th step): Render sensor plane image and/or full-field visualisation to PNG.
4. **Time evolution**: Advance ψ by one step via RK4 or SSFM.

11.4 Cloud Deployment

Large 3D simulations are deployed to AWS EC2 GPU instances—NVIDIA T4 via `g4dn.xlarge` for smaller grids, or NVIDIA L40S via `g6e.xlarge` (48 GB VRAM) for large SSFM runs—using Terraform for infrastructure provisioning and Windows batch scripts for orchestration. Simulation outputs are synced periodically to the local machine via SCP and to S3 for disaster recovery.

11.5 Dynamic Simulation Parameters

The deployment configuration supports step-wise schedules for both the time step and the source intensity, enabling multi-phase simulations within a single run:

- **Dynamic Δt**: A piecewise-constant schedule maps step ranges to Δt values (e. g. large Δt for initial propagation, reduced Δt near the sensor). For SSFM the kinetic phase factors are recomputed at each transition.
- **Dynamic source intensity**: The injection amplitude can be reduced to zero after a target number of steps, allowing already-injected wavefronts to propagate without further source perturbation.

11.6 Web Frontend

A static web frontend (`frontend/index.html`) serves as a showcase for simulation results, presenting animated GIFs and raytraced scene overviews for all five principal scenes (2D parabolic mirror, 2D pinhole, 2D torus slice, 3D pinhole torus, and 3D large-grid pinhole torus). An accompanying configuration tool (`frontend/config.html`) provides interactive parameter exploration with real-time Fresnel number, memory budget, and arrival-time estimates.

12 Comparison of Solvers

Table 8: Comparison of the RK4 and SSFM time-integration schemes.

Property	**RK4**	**SSFM**
Temporal accuracy	$\mathcal{O}(\Delta t^4)$	$\mathcal{O}(\Delta t^2)$ (Strang)
Spatial accuracy	Stencil-limited ($\mathcal{O}(h^2)$)	Spectral (exact kinetic)
Grid isotropy	Stencil-dependent (9/19-point helps)	Perfect (Fourier)
Norm preservation	No (non-unitary)	Yes (unitary for real V)
Hamiltonian evaluations	4 per step	0 (operator splitting)
FFTs per step	0	2 (forward + inverse)
Auxiliary storage (3D)	5 full grids (k_1–k_4, tmp)	1 torch copy + cache reuse
VRAM per cell (3D GPU)	64 B	48 B

13 Numerical Considerations

13.1 Dispersion Relation

The simulator uses the free-particle dispersion relation $\omega = k^2/2$ ("Schrödinger dispersion"), which differs from the linear photon dispersion $\omega = c\,k$. This means the group velocity $v_g = d\omega/dk = k$ is wave-number-dependent; wave packets disperse over time. For the imaging applications considered (moderate propagation distances, narrow frequency bands), this approximation is acceptable.

13.2 Norm Conservation

With real potentials ($V_{\text{im}} = 0$), the exact Schrödinger evolution preserves $\|\psi\|^2$. SSFM preserves this to machine precision (each half-step is a unitary multiplication). RK4, being a non-symplectic explicit method, allows slow drift; this is mitigated by using sufficiently small Δt.

When absorbing potentials are present ($V_{\text{im}} < 0$), the norm deliberately decreases, modelling photon absorption.

13.3 Boundary Conditions

The grid uses Dirichlet boundary conditions ($\psi = 0$ at the edges) combined with the absorbing boundary ramp (Section 6.3). The ramp suppresses reflections but introduces a thin non-physical absorption layer. An improvement would be to implement Perfectly Matched Layers (PML).

14 Physical Phenomena Captured

The simulator reproduces several fundamental quantum-optical effects:

Diffraction
: Wave spreading through apertures (pinholes, slits). The Airy pattern from a circular pinhole and the double-slit interference fringes are both observable in the accumulated intensity.

Interference
: Constructive and destructive superposition of wavefronts from coherent and partially coherent sources, producing fringe patterns behind slits and at lens focal planes.

Reflection and Refraction
: Waves encountering a potential step experience partial reflection (Fresnel coefficients) and transmitted-wave phase shift. Glass ($V < 0$) bends wavefronts (lens focusing); mirrors ($V \gg E$) reflect them.

Absorption
: Imaginary potentials exponentially damp the wavefunction amplitude, modelling absorbing surfaces and detector planes.

Geometric Imaging
: The pinhole camera produces a geometrically inverted image of the object, with the magnification given by (30). Diffraction through the small aperture broadens the image, demonstrating the resolution–brightness trade-off in pinhole optics.

Incoherent Averaging
: Accumulation of many random-frequency, random-direction photons produces smooth, physically realistic intensity patterns that lack the speckle of coherent illumination.

15 Known Limitations and Future Directions

1. **Non-relativistic**: The Schrödinger equation does not describe photons rigorously (photons obey the Maxwell equations or the relativistic wave equation). The simulator treats light as a scalar quantum-mechanical wave.
2. **Scalar wavefunction**: No polarisation or spin degrees of freedom.
3. **Non-linear dispersion**: $\omega \propto k^2$ rather than the correct photon dispersion $\omega = ck$.
4. **Approximate boundaries**: CAP rather than PML.
5. **Non-unitary RK4**: Norm drift over very long runs.
6. **Piecewise-constant time stepping**: Dynamic Δt schedules are supported (step-wise changes with phase-factor recomputation for SSFM), but fully adaptive error-based time stepping is not yet implemented.

Potential improvements include symplectic/unitary integrators (Crank–Nicolson, Chebyshev propagation), vectorial wave simulation (polarisation), perfectly matched layers, error-based adaptive time stepping, and real-time GPU visualisation.

A Derivation: Strang Splitting Error

The Baker–Campbell–Hausdorff formula gives

$$e^A e^B = e^{A+B+\frac{1}{2}[A,B]+\cdots}, \tag{39}$$

where $[A, B] = AB - BA$ is the commutator. For the symmetric (Strang) splitting with $A = -i\hat{V}\,\Delta t/2$ and $B = -i\hat{T}\,\Delta t$:

$$\begin{aligned} e^A e^B e^A &= e^{2A+B+\frac{1}{2}[A,B]+\frac{1}{2}[A+B,A]+\cdots} \\ &= e^{-i(\hat{T}+\hat{V})\Delta t+\mathcal{O}(\Delta t^3\,[\hat{T},[\hat{T},\hat{V}]])}. \end{aligned} \tag{40}$$

The leading error term is third-order in Δt, so the local truncation error per step is $\mathcal{O}(\Delta t^3)$ and the global error is $\mathcal{O}(\Delta t^2)$.

B FFT Frequency Mapping

For an N-point DFT with grid spacing Δx, bin index $n \in \{0, 1, \ldots, N-1\}$ corresponds to angular frequency

$$k_n = \frac{2\pi}{N\,\Delta x} \begin{cases} n, & 0 \le n < \lceil N/2 \rceil, \\ n - N, & n \ge \lceil N/2 \rceil. \end{cases} \tag{41}$$

This wraps the negative frequencies to the upper half of the bin range, matching the convention used by NumPy's `fftfreq`, PyTorch's `torch.fft.fftfreq`, and the Go implementation in this project.

C Nine-Point 2D Laplacian Stencil Derivation

The standard five-point 2D Laplacian stencil

$$\nabla^2 \psi \approx \frac{1}{h^2}(\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} - 4\psi_{i,j}) \tag{42}$$

has anisotropic leading-order error: the coefficient of the $\partial^4/\partial x^2 \partial y^2$ term differs from the $\partial^4/\partial x^4$ term. The nine-point isotropic stencil corrects this by adding the diagonal neighbours. One systematic derivation is to form the weighted average

$$\nabla^2_{\text{iso}} = \alpha\, \nabla^2_{\text{cross}} + \beta\, \nabla^2_{\text{diag}}, \tag{43}$$

where ∇^2_{cross} is the standard five-point stencil and ∇^2_{diag} is a "rotated" five-point stencil on the diagonal grid (with spacing $h\sqrt{2}$). Requiring the $\mathcal{O}(h^2)$ error to be isotropic yields $\alpha = 2/3$, $\beta = 1/3$, producing the weights given in (6).

References

[1] G. Strang, "On the construction and comparison of difference schemes," *SIAM J. Numer. Anal.*, vol. 5, no. 3, pp. 506–517, 1968.

[2] T. R. Taha and M. J. Ablowitz, "Analytical and numerical aspects of certain nonlinear evolution equations. II. Numerical, nonlinear Schrödinger equation," *J. Comput. Phys.*, vol. 55, no. 2, pp. 203–230, 1984.

[3] D. J. Griffiths and D. F. Schroeter, *Introduction to Quantum Mechanics*, 3rd ed. Cambridge University Press, 2018.

[4] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, 2007.

[5] Y. Hu *et al.*, "Taichi: A language for high-performance computation on spatially sparse data structures," *ACM Trans. Graph.*, vol. 38, no. 6, art. 201, 2019.