Gerchberg–Saxton Holograms for Optical-Tweezer Arrays

How GS, weighted GS and phase-fixed WGS turn a phase-only LCOS SLM into a uniform 1,024-site trap array, with a Python demo, an SLM bit-depth study and spot-only solvers

David F. Hakala

21 October 2026

1. What the algorithm solves

An optical-tweezer array is a pattern of tightly focused spots in the focal plane of a high-NA objective. Such an array is a critical element in the design of a neutral atom quantum computer. The LCOS SLM sits in the pupil plane (or a plane conjugate to it), so the focal field is the Fourier transform of the field leaving the SLM:

$$E_f(\mathbf{u}) = \mathcal{F}\{A_{in}(\mathbf{x})\, e^{i\phi(\mathbf{x})}\}$$

A_{in} is the laser amplitude on the SLM, which is fixed. ϕ is the phase written by the SLM, which is the only free variable. The goal is to find ϕ such that $|E_f|^2$ equals the target: M equal spots at chosen positions $\mathbf{u}_m$ and darkness elsewhere. A phase-only device cannot set amplitude, so the problem has no direct inverse. Gerchberg–Saxton (GS) finds a good ϕ by iterating between the two planes.

2. The basic GS loop

1. Start with the true input amplitude A_{in} and a random phase ϕ_0.
2. Propagate to the focal plane using a Fourier Transform: $E_f = \mathcal{F}\{A_{in}e^{i\phi_k}\}$.
3. Keep the phase $\psi = \arg E_f$ and replace the amplitude with the target: $G = T(\mathbf{u})\, e^{i\psi}$, with $T = \sqrt{I_m}$ at the spots and 0 elsewhere.
4. Propagate back, using the inverse Fourier Transform: $g = \mathcal{F}^{-1}\{G\}$.
5. Keep the phase and discard the amplitude: $\phi_{k+1} = \arg g$ (the laser fixes the amplitude).
6. Repeat until the pattern stops improving, or an acceptable variance in the optical tweezer traps is found.

Each half-step projects the field onto one constraint set: the input amplitude in the SLM plane and the target amplitude in the focal plane. The squared error cannot increase from one iteration to the next. GS stalls in a local minimum, however, and for a large spot array it leaves the spot intensities spread by about 15–20% rms.

3. Refinements used for atom arrays

Weighted GS (WGS)

Trap depth sets each atom's light shift, loading probability and gate detuning, so the spots have to be equal to about 1%. WGS gives each spot a weight w_m that is updated every iteration from the field the hologram actually produces:

$$w_m^{(k+1)} = w_m^{(k)} \frac{\langle |E_f(\mathbf{u}_m)| \rangle}{|E_f(\mathbf{u}_m)|}, \qquad G(\mathbf{u}_m) = w_m \, T_m \, e^{i\psi_m}$$

Dim spots are asked for more light and bright spots for less. In the demo below, WGS brings uniformity from 16% to below 1%.

Phase-fixed WGS

In WGS the weights and the focal-plane phases ψ_m keep changing together, and convergence can wander. In phase-fixed WGS, after N_{fix} iterations (about 10–20) the spot phases are frozen, $\psi_m \equiv \psi_m^{(N_{fix})}$, and only the weights keep updating. With the phases fixed, each weight acts almost independently on its own spot, so uniformity drops sharply within a few iterations and then holds steady.

Spot-only ("gratings and lenses") methods

Instead of computing the field on a full FFT grid, the hologram can be built directly from one steering pattern per spot. This suits a few hundred to a few thousand spots, spots that don't sit on the FFT grid, 3D arrays, and per-spot aberration correction.

One spot: a grating plus a lens

The objective (or Fourier lens, focal length f) maps direction to position: a plane wave leaving the SLM tilted by angle θ focuses at $u \approx f\theta$. A phase-only device tilts light with a linear phase ramp, a **blazed grating** (a "prism"). To put a spot at (u_m, v_m):

$$\phi_{grating,m}(x, y) = \frac{2\pi}{\lambda f} (u_m x + v_m y) = \mathbf{k}_m \cdot \mathbf{x}$$

- $\mathbf{x} = (x, y)$ is the position of a pixel on the SLM, $\mathbf{x} = p$ (column,row) for pixel pitch p.
- $\mathbf{k}_m = \frac{2\pi}{\lambda f}(u_m, v_m)$ is the transverse wave vector, or spatial frequency, of that grating. Its direction sets which way the light is steered and its magnitude how far. Equivalently, the grating period is $\Lambda_m = 2\pi/|\mathbf{k}_m| = \lambda f/|u_m|$ and the deflection angle is $\theta_m \approx \lambda/\Lambda_m$: a spot twice as far off axis needs a grating with half the period.

To move a spot along the beam by z_m (3D arrays, or correcting field curvature), add a quadratic phase, a weak Fresnel **lens**:

$$\phi_{lens,m}(x,y) = \frac{\pi z_m}{\lambda f^2}\,(x^2+y^2) \equiv \alpha_m\,(x^2+y^2)$$

The sign depends on the convention for $+z$. The complete steering phase for spot m is

$$\Delta_m(\mathbf{x}) = \mathbf{k}_m \cdot \mathbf{x} + \alpha_m\,|\mathbf{x}|^2$$

Every spot gets its own grating, plus a lens if it's out of plane; hence the name. Per-spot Zernike terms can be added to Δ_m in the same way to correct aberrations that vary across the field.

Many spots: superpose the fields, not the phases

Adding the *phases* of two gratings just makes a third grating pointing somewhere else. Instead, add the complex fields each steering pattern would produce, and keep the phase of the sum:

$$\phi(\mathbf{x}) = \arg \sum_{m=1}^{M} w_m\; e^{i(\Delta_m(\mathbf{x})+\theta_m)}$$

The arg step is where the phase-only constraint bites. The true sum has an amplitude that varies across the SLM, and discarding it creates ghost spots and unequal spot brightness. That's why w_m and θ_m have to be chosen well.

w_m: the spot weights

w_m is the amplitude requested at spot m. With all w_m equal the spots still come out unequal after the arg step, so weighted versions iterate. The field actually produced at spot m is evaluated directly at that point:

$$V_m = \sum_{j=1}^{N} A_j\; e^{i\left(\phi(\mathbf{x}_j)-\Delta_m(\mathbf{x}_j)\right)}, \qquad w_m \leftarrow w_m\,\frac{\langle|V|\rangle}{|V_m|}$$

The sum runs over the N SLM pixels, with A_j the illumination amplitude on pixel j. Dim spots ask for more light and bright spots for less.

θ_m: the relative phase of each spot

θ_m is the optical phase of spot m's field in the focal plane. The atoms don't care about it, because each trap depends only on $|E|^2$ and the spots are too far apart to interfere. But it strongly affects hologram quality:

- **All θ_m equal.** For a regular grid the terms add in phase at a few places on the SLM and cancel elsewhere. The summed field is very non-uniform, the arg step discards most of it, and the result is poor efficiency and strong ghost orders. It's the same reason a regular array of in-phase emitters makes bright diffraction orders.
- **Random θ_m.** Spreads the summed field more evenly over the SLM. The simplest fix: decent efficiency, but large spot-to-spot power fluctuations.

- **Optimized θ_m.** Each iteration sets $\theta_m = \arg V_m$, so the hologram and the spot phases settle into a consistent pair. These are the same focal-plane phases ψ_m that phase-fixed WGS freezes after N_{fix} iterations. A smooth quadratic pattern of θ_m across the array is a common deterministic starting guess.

Practical limits on $\mathbf{k}_m$

- **Field of view.** The finest grating the SLM can display has a period of two pixels. That sets $|\mathbf{k}|_{max} = \pi/p$ and the field of view $|u|_{max} = \lambda f/(2p)$, referred to the SLM. Spots near the edge lose efficiency, roughly as sinc^2 of the pixel aperture, so in practice the inner part of the field is used.
- **Illustrative scale.** With a 0.3× relay from a 12.5 µm-pitch SLM into a 4 mm focal-length objective (example numbers, not a design), a 4 µm spot spacing at the atoms corresponds to a grating period of about 3.5 mm, roughly 280 pixels, on the SLM, and the usable field is several hundred µm across. A 32 × 32 array at 4 µm spans about 124 µm, so it uses only low spatial frequencies.
- **Zero order.** One common grating (or lens) term added to every $\mathbf{k}_m$ shifts the whole array off axis, and a block in the relay removes the unmodulated zero-order spot.

Computing it efficiently

Because Δ_m is a sum of an x-term and a y-term, $e^{i\Delta_m}$ factorizes per spot into a row vector times a column vector. Both the hologram synthesis and the spot fields V_m then become two dense matrix products of size $(M \times N_y)(N_y \times N_x)$, valid for any spot positions, on or off the FFT grid and in 3D. For 1,024 spots on a 512 × 512 SLM, one iteration takes about 60 ms in numpy, about 6 s for 100 iterations.

4. Extra terms added to the hologram

The phase actually sent to the SLM is

$$\phi_{SLM} = \left[\phi_{GS} + \phi_{grating} + \phi_{lens} + \phi_{Zernike} + \phi_{flat}\right] \bmod 2\pi \xrightarrow{\text{LUT}} \text{drive levels}$$

- A blazed grating and/or weak lens term moves the array away from the undiffracted zero-order spot, which a block in the 4f relay then removes.
- Zernike terms correct aberrations of the SLM, relay and objective.
- ϕ_{flat} is the vendor's flatness-correction map. The LUT converts phase to drive level and is calibrated at the operating wavelength.

5. Closing the loop on hardware

1. **Camera feedback.** Image the spots at a plane conjugate to the atoms. Estimate each spot's optical gain as measured ÷ model-predicted intensity, $g_m = I_m^{meas}/I_m^{model}$, averaged over rounds to suppress camera noise. Then re-aim the hologram at targets $T_m \propto 1/\sqrt{g_m}$. After each new target, let the spot phases re-adapt for a few

iterations before freezing them again: phases frozen for a uniform array can't reach a tilted target. A few rounds remove fixed errors of the optics, such as vignetting, aberrations and SLM nonuniformity.

2. **Atom feedback.** Load atoms and measure each site's trap depth from its light shift, then use those values in place of P_m. This corrects what the camera can't see, such as polarization and the optics after the camera pick-off. It's needed to reach the < 0.5% depth uniformity target in the design draft.

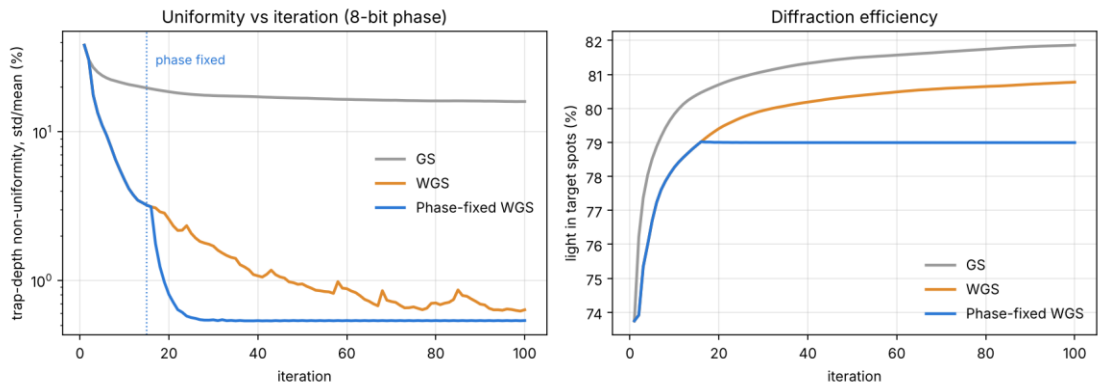
6. Python demo results

The script `wgs_tweezer_demo.py` uses the following model:

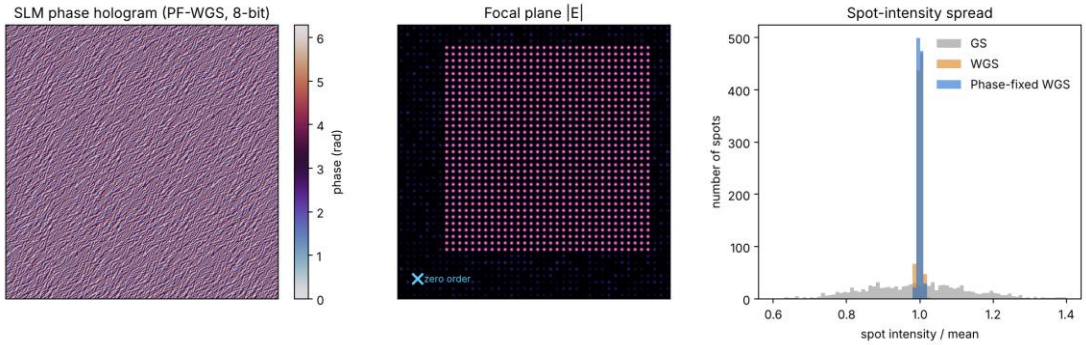
- **SLM:** 512 × 512 pixels with Gaussian illumination ($1/e^2$ radius 0.4 × the aperture width), zero-padded to a 1024 × 1024 FFT.
- **Hologram:** quantized to 8 bits when evaluated.
- **Target:** a 32 × 32 array (1,024 spots) at an 8-pixel pitch, offset 160 pixels from the zero order.
- **Uniformity metric:** std/mean of the peak intensity at each site, which is proportional to trap depth. Efficiency counts the light within a 3 × 3 pixel window around each spot.
- **Run:** 100 iterations per method from the same random starting phase, with phase fixing at iteration 15.

Method	Trap-depth non-uniformity (std/mean)	Light in target spots	Runtime (CPU, numpy)
GS	15.9%	81.9%	23 s
WGS	0.63%	80.8%	24 s
Phase-fixed WGS	0.54%	79.0%	23 s
Phase-fixed WGS, quantization-aware refine (40 more iterations)	0.34%	80.0%	+10 s

"Quantization-aware" means the weights are computed from the 8-bit phase the SLM will actually display, so they absorb the rounding error, rather than rounding a finished hologram.



Convergence: phase-fixed WGS drops below 1% within about 8 iterations of fixing the phase; WGS gets there more slowly and less steadily, and GS stalls near 16%.

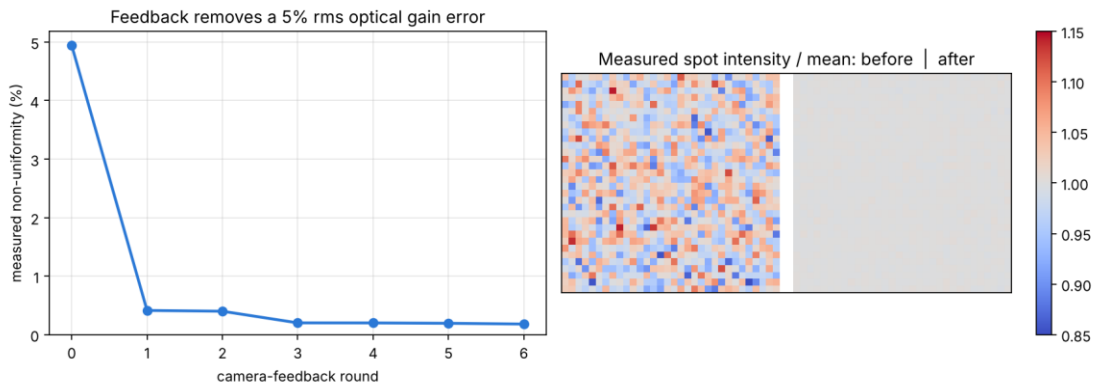


Left: the 8-bit phase hologram. Centre: the focal-plane field, with the array well clear of the zero order. Right: the spread of spot intensities for each method.

The camera-feedback test adds an unknown, fixed 5% rms gain error per spot, standing in for the optics after the SLM. The measured uniformity then goes:

4.9% → 0.41% → 0.40% → 0.20% → 0.20% → 0.20% → 0.18%

over six rounds of feedback.



Camera feedback: measured non-uniformity per round (left) and the spot map before and after (right).

What the numbers mean

- At 8 bits, phase quantization is the main limit in this model: with continuous phase the same optimizer reaches 0.07% (Section 7). A real system adds LUT error, pixel crosstalk (fringing fields) and flicker. That's why atom feedback is the final step.
- Weighting costs only about 1–3 points of efficiency compared with plain GS. The light that doesn't reach the spots goes into ghost orders and a weak speckle background between the spots.
- Runtime is about 0.24 s per iteration in numpy on a CPU. On a GPU, 1–2k FFTs run in well under a millisecond, so a 30-iteration hologram takes tens of milliseconds. The limit on changing the array is the SLM's LC response and refresh rate (tens of Hz), not computation. That's why the SLM writes the static grid and AODs do the rearranging.

7. Effect of SLM bit depth

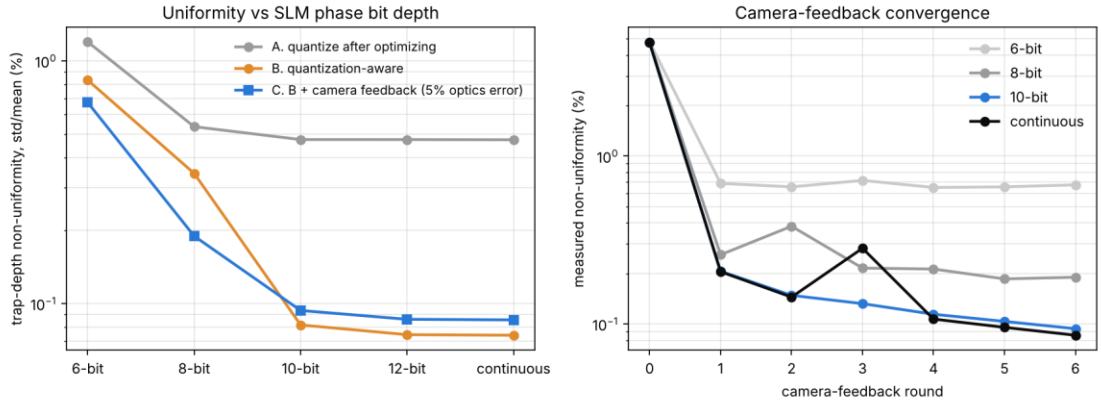
The script `wgs_bitdepth_sweep.py` repeats the 32 × 32 case at 6, 8, 10 and 12 bits and with continuous phase. It compares:

- **A.** rounding a finished hologram to the SLM's levels
- **B.** quantization-aware optimization
- **C.** B plus six rounds of camera feedback against a 5% rms optics error, with 0.2% camera noise

SLM phase	Phase step	rms rounding error	A. Quantize after	B. Quantization-aware	C. B + camera feedback
6-bit	98 mrad	28 mrad	1.20%	0.83%	0.67%
8-bit	24.5 mrad	7.1 mrad	0.54%	0.34%	0.19%
10-bit	6.1	1.8 mrad	0.47%	0.082%	0.094%

SLM phase	Phase step	rms rounding error	A. Quantize after	B. Quantization-aware	C. B + camera feedback
	mrad				
12-bit	1.5 mrad	0.44 mrad	0.47%	0.074%	0.086%
Continuous	—	0	0.47%	0.074%	0.086%

The rms rounding error is step/$\sqrt{12}$.



Left: trap-depth non-uniformity vs bit depth for the three approaches. Right: camera-feedback convergence at each bit depth.

What 10 bits buys:

- **Open-loop uniformity improves about 4×** with quantization-aware optimization: 0.34% at 8 bits to 0.082% at 10 bits. Removing the optimizer's own floor (0.074%) in quadrature, the part due to quantization falls from about 0.33% to 0.035%. That's roughly the factor of 4 per two bits you'd expect from the step size.
- **With camera feedback the gain is about 2×,** 0.19% to 0.09%. At 10 bits and above, the result is limited by the 0.2% camera noise and the number of feedback rounds, not by the SLM.
- **Beyond 10 bits there's no further gain** in this model. 12-bit and continuous phase give the same result.
- **Rounding a finished hologram wastes most of the benefit.** In column A all depths above 8 bits sit at the optimizer floor of that run (0.47%). Optimizing against the quantized phase is what lets extra bits help.
- **Efficiency is unaffected.** The loss from quantization is about $(\pi/2^b)^2/3$, which is 0.005% at 8 bits.

Caveats for real hardware. These figures cover addressing resolution only. On a real LCOS, the effective phase precision may be set instead by:

- the drive scheme's phase flicker
- LUT calibration error

- fringing-field crosstalk between neighbouring pixels
- thermal drift

If flicker is larger than the 8-bit rounding error (7 mrad rms), 10-bit addressing won't deliver the 4× gain. Two side benefits of 10 bits are robust, though:

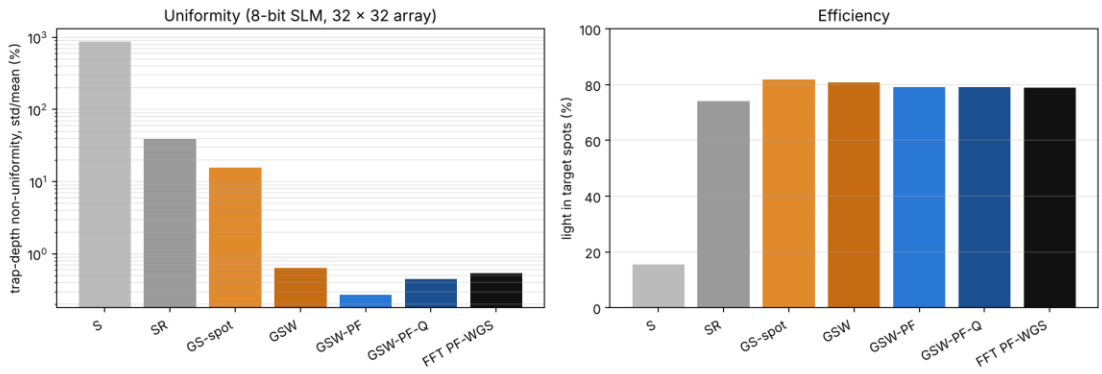
- **Finer LUT linearization.** When the LC stroke exceeds 2π at the working wavelength, only part of the drive range is used for a 2π cycle, so 8 bits leaves fewer than 256 usable levels.
- **More room for overlays.** The grating, lens and Zernike corrections are added to the hologram without losing phase resolution.

Measure the flicker spectrum and the effective phase resolution on the actual device before relying on the simulated numbers.

8. Spot-only solver results

The script `wgs_spot_solver.py` builds holograms for the same 32 × 32 targets with the spot-only methods and scores them exactly like the FFT demo: 8-bit phase, FFT-propagated, trap-depth uniformity from peak intensity, and efficiency from the light within 3 × 3 pixels of the spots. Method names follow Di Leonardo et al.

Method	What it does	Trap-depth non-uniformity	Light in target spots
S	Superposition, all $\theta_m = 0$, one shot	874%	15.5%
SR	Superposition, random θ_m, one shot	39%	74.0%
GS-spot	Iterates $\theta_m = \arg V_m$, equal weights	15.8%	81.9%
GSW	Iterates θ_m and the weights	0.64%	80.7%
GSW-PF	GSW with θ_m frozen after 15 iterations, damped weights	0.27%	79.1%
GSW-PF-Q	GSW-PF computed on the 8-bit phase	0.45%	79.1%
FFT PF-WGS (reference)	The FFT demo's phase-fixed WGS	0.54%	79.0%

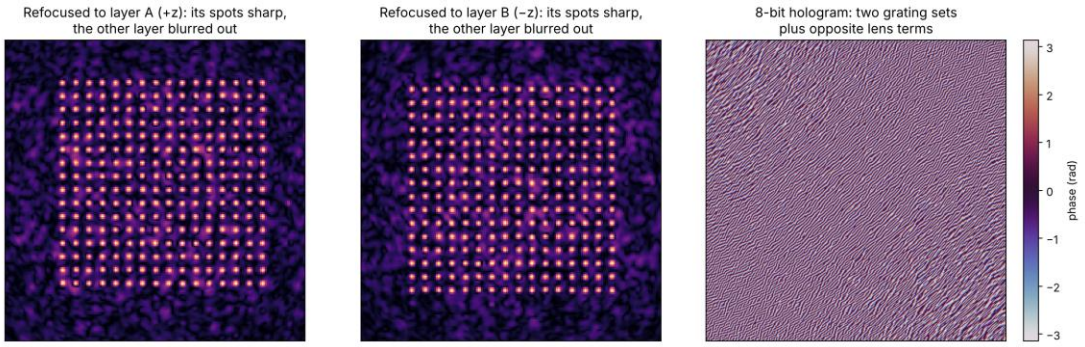


Spot-only methods compared with the FFT phase-fixed WGS on the same targets.

What the numbers show:

- **θ_m matters enormously.** With all spot phases equal, the regular grid concentrates the summed field in a few places on the SLM. Most of the light goes into ghost orders (15% efficiency), and the spread in spot brightness is about nine times the mean. Random phases recover most of the efficiency, and iterating them recovers the rest.
- **Weights set the uniformity.** Without weights, GS-spot stalls at about 16%, the same as the FFT GS. With weights it reaches below 1%, and phase-fixing with damped weights gives the best result, 0.27%.
- **Same physics as the FFT method.** For targets on the FFT grid, the spot-only and FFT versions solve the same problem and reach the same regime. The small differences come from iteration details: the FFT reference runs undamped weights.
- **Quantization-aware is not automatically better on a regular grid.** In 2D, computing the weights from the 8-bit phase gave 0.2–0.5% depending on iteration count, not a steady improvement. A likely cause is that the hologram of a regular array is nearly periodic, so its rounding errors are periodic too and diffract onto the spot lattice itself, where the weights can't cleanly remove them. In the 3D example below, the lens terms break that periodicity and the same method reaches 0.01%. Treat this as a likely explanation, not a proven one.

3D example. Two 16 × 16 layers are placed in two planes by giving them opposite lens terms ±α. The lens phase is 10 waves at the SLM edge, far more than the depth of focus, and the second layer is offset by half a pitch. With quantization-aware GSW-PF, each layer is uniform to about 0.01% when refocused to its own plane, while the other layer is blurred into a weak background.



Left and centre: the hologram refocused to each layer's plane. Right: the 8-bit hologram, two sets of gratings plus opposite lens terms.

When to use which. The FFT method is simplest for large regular grids. The spot-only method is better when spots sit off the FFT grid, the array is 3D, each spot needs its own aberration correction, or positions change slightly between holograms. In every case, choose the θ_m iteratively and iterate the weights.

9. Running and changing the scripts

```
pip install numpy matplotlib
python wgs_tweezer_demo.py
python wgs_bitdepth_sweep.py
python wgs_spot_solver.py
```

The demo runs in about 2 minutes, the sweep in about 5 minutes and the spot solver in about 1 minute on a CPU. The sweep and the spot solver reuse the model and functions in the demo script, so keep all three in the same folder.

The figures and `wgs_summary.json` are written to `wgs_out/`. All parameters are at the top of the script:

- `N_SLM` and `N_FFT` set the SLM size and padding.
- `N_SPOTS` and `PITCH` set the array.
- `OFFSET` sets the distance from the zero order.
- `N_FIX` sets when the phase is frozen.
- `BITS` sets phase quantization (0 = continuous).
- `KAPPA` damps the weight updates after phase fixing.
- `CAM_NOISE` sets the camera noise in the feedback test.
- `GAIN_RMS` and `FB_ROUNDS` control the feedback test.

To model the X15213's 1280 × 1024 format, set `N_SLM` to the short side and increase `N_FFT` to 2048. Each iteration then takes about 4× longer.

References

- R. W. Gerchberg and W. O. Saxton, “A practical algorithm for the determination of phase from image and diffraction plane pictures,” *Optik* 35, 237 (1972).
- R. Di Leonardo, F. Ianni and G. Ruocco, “Computer generation of optimal holograms for optical trap arrays,” *Opt. Express* 15, 1913 (2007). Introduces weighted GS.
- D. Kim et al., “Large-scale uniform optical focus array generation with a phase spatial light modulator,” *Opt. Lett.* 44, 3178 (2019). Introduces phase-fixed WGS for atom arrays.