Particle Life on GPU CuPy port for long-form renders

Code: `render_seed_480_panoramic_gpu.py` (337 lines, single file)
Engine: CuPy + one custom CUDA splat kernel
Host: `ai.foxhop.net` (RTX 4090, sm_89, 24 GB)
Use: render seed-480 particle-life to MP4, MPEG-2 (DVD-Video), or HEVC for BD-R

What this script renders

Particle Life is a 2-D agent toy: N particles, K species, an asymmetric K×K force matrix, periodic torus boundary on both axes. One seed value fully determines the matrix, initial positions, & species assignments, so a given seed gives the same evolving universe every time on every GPU bit-for-bit. We picked **seed 480** because its matrix produces a visually active universe that never stalls into a fixed point or limit cycle for hours of simulation time.

Single CuPy file drives four output targets via one `MODE` switch. Each preset is a sealed dict at the top of our script — flip the constant, run, get our target file. No CLI args needed.

Output modes

mode	resolution	N particles	duration	target file
panoramic	7680 × 2160	115 000	200 s	h264 .mp4 (~2 GB)
dvd_mp4	720 × 480	2 500	7.5 h	h264 .mp4 on data DVD-R
dvd_video	720 × 480	2 500	2 h	MPEG-2 .mpg → real DVD-Video
bd_4k_6h	3840 × 2160	115 000	6 h	HEVC .mp4 on BD-R

`dvd_video` is our most-tested target: ffmpeg `-target ntsc-dvd` produces a spec-compliant MPEG-2 Program Stream with a silent AC-3 audio track that `dvdauthor` turns into a `VIDEO_TS/` folder. That folder gets `mkisofs -dvd-video` packed into an ISO & `growisofs -dvd-compat -Z` burnt to a Verbatim DVD-R blank. Bootable on cheap DVD players & Xbox 360, verified on both 2026-06-29.

The four optimisations that mattered

Naïve CuPy port of our CPU renderer hit OOM on our 4090 at N=22 000 (peak 19.4 GB of 24 GB). Four changes brought peak down to 2.8 GB at the same N & raised throughput an order of magnitude:

1. Custom CUDA splat kernel — one launch per frame.

The 5×5 soft-particle splat was 25 `np.add.at` ops in our CPU code, which became 25 sequential `cp.add.at` kernel launches per frame. We replaced our entire splat with one `cp.RawKernel` that takes `(pos, buf, kern, N, H, W)` & atomically adds the 25 contributions for every particle inside one launch. The whole 720×480 frame splat now costs one kernel dispatch instead of 25.

2. ``cp.fuse`` on the force formula.

The piecewise force `f(rn)` — repulsive below β, attractive between β & 1, zero beyond — was three array passes in our pure-CuPy version. Wrapping it in `@cp.fuse()` lets CuPy generate one fused elementwise CUDA kernel that does the whole branch on one read of `rn`, `Aij_pairs` and one write to the output.

3. Raw bytes straight into ffmpeg's stdin.

Our CPU script wrote PNGs to disk & let ffmpeg read them back. At 540 000 frames × 720×480 grayscale that would have been 186 GB of raw data on `/tmp` — wouldn't fit. We open ffmpeg as a subprocess with `stdin=PIPE`, send each frame as `frame.tobytes()`, & ffmpeg encodes in parallel with our simulation. Disk usage stays bounded at our final output size (~4 GB for the 2-hour DVD-Video).

4. CUDA async memory pool + periodic ``free_all_blocks``.

`cp.cuda.set_allocator(cp.cuda.MemoryAsyncPool().malloc)` lets the runtime overlap allocation with kernels. `mem_pool.free_all_blocks()` fires every 32 frames so peak allocation doesn't climb monotonically across our 216 000-frame render. The two together kept us under 3 GB peak across our entire 2-hour render.

Benchmarks

Wall-clock fps on RTX 4090, seed 480, our four presets:

mode	resolution	N	wall fps
CPU baseline (panoramic)	2880 × 1080	22 000	6 – 7 fps
GPU panoramic	7680 × 2160	115 000	33 fps
GPU dvd_video	720 × 480	2 500	326 fps

The 2-hour DVD-Video (216 000 frames) renders in about **11 minutes wall time** on our 4090. The CPU-version equivalent at the same resolution & N would have taken roughly **9 hours**.

Simulation kernel (short recap)

Per particle, per frame:

1. **Spatial hash** — bin all N particles by `floor(pos / rmax)` into a grid; sort by cell index; `cp.searchsorted` gives a `cell_start[cell_id]` array in one pass.
2. **Candidate pairs** — for each particle, look up its 3×3 neighbour cells; concatenate every particle in those cells as a candidate for force interaction. O(N) instead of O(N^2).
3. **Distance + torus wrap** — apply periodic boundary in both x & y by `d -= S * round(d / S)`.
4. **Force** — fused kernel returns the piecewise force value; multiply by direction & species-pair coefficient `A[typ_i, typ_j]`; scatter back to `acc` via `cp.add.at`.
5. **Integrate** — `vel = vel * friction + acc * dt` then `pos = (pos + vel * dt) % S`.
6. **Render** — decay buffer, splat 5×5 gaussian per particle, log normalise, emit one uint8 grayscale frame to ffmpeg stdin.

`friction = 0.85`, `dt = 0.6`, `fs = 0.8`, `rmax = 88`, `beta = 0.3`, `K = 5`. Buffer decays at 0.80 between frames which gives the visible motion trails.

Burn pipeline (DVD-Video target)

Once `MODE = 'dvd_video'` produces `animation.mpg` (~3.9 GB), four shell steps cut a bootable DVD:

```
# 1. author VIDEO_TS/ from the .mpg
cd $OUT/480_dvd_video
mkdir -p dvd_root
VIDEO_FORMAT=NTSC dvdauthor -o dvd_root -T
VIDEO_FORMAT=NTSC dvdauthor -o dvd_root \
    -f animation.mpg -t

# 2. stage extras alongside VIDEO_TS/
mkdir -p dvd_root/EXTRAS
cp -r path/to/source dvd_root/EXTRAS/

# 3. pack ISO
```

```
mkisofs -dvd-video -V "PARTICLE_LIFE" \
    -r -J -o disc.iso dvd_root/

# 4. burn single-session, finalised
growisofs -dvd-compat -Z /dev/sr0=disc.iso
```

`-dvd-compat` finalises our disc on first write — single session, no multi-session firmware quirks. Tested cheap-DVD-player & Xbox 360 playback 2026-06-29 on Verbatim MCC 03RG20 (Mitsubishi AZO) blanks.

Why one file

Whole pipeline lives in 337 lines of one Python file. No build system, no config files, no separate tokenizer/encoder/renderer split. `cupy` & `ffmpeg` are our only runtime requirements; `dvdauthor` / `mkisofs` / `growisofs` only enter our pipeline at burn time, not render time. Every render mode is one literal dict near the top — to add a 4K Blu-ray or a vertical-phone aspect, copy & edit one preset.

The renderer & our authoring shell steps together fit on one printed page. That printed page is in our v2 disc's `EXTRAS/py/` directory.

Full source

Self-contained — only stdlib + `numpy` + `cupy`. No project-local imports. Runtime deps: a `cupy-cudaXX` wheel matching our CUDA toolkit & an `ffmpeg` binary on `$PATH`. `dvdauthor` / `mkisofs` / `growisofs` only enter for the DVD burn path, not for render.

```
#!/usr/bin/env python3
"""GPU port of render_seed_480_panoramic.py using CuPy — optimised.

Streams raw uint8 frames directly into ffmpeg's stdin (no intermediate
raw file on disk) so very long renders are bounded by GPU memory, not
storage. At 540,000 frames × 720×480 the un-streamed raw bytes would
have been 186 GB — would not fit on /tmp.

Two top-level parameter sets at the bottom: one for the
panoramic-pan-piece (4K-wide aspect, short duration) and one for the
disc-fill target (TV native aspect, multi-hour duration).  Flip the
`MODE` constant to select.
"""
import os
import subprocess
import sys
from pathlib import Path
import numpy as np
import cupy as cp


try:
    cp.cuda.set_allocator(cp.cuda.MemoryAsyncPool().malloc)
except Exception:
    pass


# --- which render to do ------------------------------------------------
# 'panoramic'  : 7680×2160 × 4000 frames  (200 sec, pan-piece, ~2 GB)
# 'dvd_mp4'    : 720×480 × 540,000 frames (7.5 hours, data DVD-R 4.7 GB)
# 'dvd_video'  : 720×480 × 216,000 frames (2 hours, real DVD-Video, any player)
# 'bd_4k_6h'   : 3840×2160 × 432,000 frames (6 hours, fills BD-R 25 GB)
MODE = 'dvd_video'


_PRESETS = {
    'panoramic': dict(width=7680, height=2160, N=115000, iters=4000,
                      crf=23, aspect=None),
    # DVD-R MP4 fill (data disc, NOT DVD-Video).  ~1.4 Mbps avg h264.
    'dvd_mp4':   dict(width=720,  height=480,  N=2500,   iters=540_000,
                      bitrate='1400k', maxrate='2500k',
```

```
                      bufsize='5000k', aspect='16:9'),
    # Real DVD-Video — bootable in any DVD player + Xbox 360.
    # 720×480 NTSC, ~30 fps, MPEG-2 + silent AC-3 audio, DVD-PS muxer.
    # Uses ffmpeg's `-target ntsc-dvd` which fixes codec/bitrate/GOP/mux
    # to spec.  Output is a .mpg (MPEG-2 Program Stream); dvdauthor
    # turns that into the VIDEO_TS/ folder you burn to disc.
    # DVD-R single layer = 4.7 GB marketed = 4.38 GiB binary.  Target
    # final file at ~4.2 GiB so dvdauthor's IFO/BUP overhead still fits.
    # `-target ntsc-dvd` defaults to 6 Mbps video + 448 kbps audio
    # (5.8 GB total over 2 hr — too big).  Audio is SILENT so 192 kbps
    # is plenty; the savings go to the video budget.
    'dvd_video': dict(width=720, height=480, N=2500,
                      iters=216_000, fps=30,
                      dvd_target='ntsc-dvd',
                      video_bitrate='4400k',
                      audio_bitrate='192k',
                      aspect='16:9',
                      out_ext='.mpg', add_silent_audio=True),
    # BD-R fill (~25 GB) at 3840×2160, 6 hr → ~9.3 Mbps avg, HEVC.
    'bd_4k_6h':  dict(width=3840, height=2160, N=115000, iters=432_000,
                      bitrate='9000k', maxrate='15000k',
                      bufsize='30000k', aspect=None, codec='libx265'),
}

PARAMS = _PRESETS[MODE]


# --- splat kernel (one CUDA launch per frame) ----------------------------
_SPLAT_KERNEL = cp.RawKernel(r"""
extern "C" __global__
void splat(const float* __restrict__ pos,
           float* __restrict__ buf,
           const float* __restrict__ kern,
           const int N, const int H, const int W) {
    int p = blockIdx.x * blockDim.x + threadIdx.x;
    if (p >= N) return;
    float px = pos[p * 2 + 1];
    float py = pos[p * 2];
    int ix = ((int)px % W + W) % W;
    int iy = ((int)py % H + H) % H;
    #pragma unroll
    for (int dy = -2; dy <= 2; ++dy) {
        int ky = ((iy + dy) % H + H) % H;
        #pragma unroll
        for (int dx = -2; dx <= 2; ++dx) {
            int kx = ((ix + dx) % W + W) % W;
            float w = kern[(dy + 2) * 5 + (dx + 2)];
            atomicAdd(&buf[ky * W + kx], w);
        }
    }
}
""", 'splat')


_KERN_HOST = np.array([
    [0.05, 0.20, 0.30, 0.20, 0.05],
    [0.20, 0.60, 0.85, 0.60, 0.20],
    [0.30, 0.85, 1.00, 0.85, 0.30],
    [0.20, 0.60, 0.85, 0.60, 0.20],
    [0.05, 0.20, 0.30, 0.20, 0.05],
], dtype=np.float32)
KERN_GPU = cp.asarray(_KERN_HOST.ravel())

_DY_OFFSETS = cp.asarray(np.array([-1, -1, -1, 0, 0, 0, 1, 1, 1],
                                  dtype=np.int32))
_DX_OFFSETS = cp.asarray(np.array([-1, 0, 1, -1, 0, 1, -1, 0, 1],
                                  dtype=np.int32))


def render_points(buf, pos, S):
```

```
    buf *= 0.80
    H, W = int(S[0]), int(S[1])
    threads = 256
    blocks = (pos.shape[0] + threads - 1) // threads
    _SPLAT_KERNEL((blocks,), (threads,),
                  (pos.astype(cp.float32), buf, KERN_GPU,
                   pos.shape[0], H, W))
    buf_max = float(buf.max())
    bright = cp.clip(cp.log1p(buf) / cp.log1p(buf_max + 1e-9) * 255,
                     0, 255).astype(cp.uint8)
    return cp.asnumpy(bright)


@cp.fuse()
def _force_value(rn, Aij_pairs, beta):
    rep = rn / beta - 1.0
    att = Aij_pairs * (1.0 - cp.abs(2.0 * rn - 1.0 - beta) / (1.0 - beta))
    return cp.where(rn < beta, rep, cp.where(rn < 1.0, att, 0.0))


def particle_life_stream(width, height, iters, ffmpeg_stdin,
                         N=2000, K=5, seed=480):
    """Run the simulation; write each uint8 grayscale frame directly
    to the supplied open file/pipe (so encoding can happen in parallel
    with simulation and no large raw file is materialised)."""
    rng = np.random.default_rng(seed)
    S_host = np.array([height, width], dtype=np.float32)
    S = cp.asarray(S_host)

    pos = cp.asarray(rng.random((N, 2)).astype(np.float32)) * S
    vel = cp.zeros((N, 2), dtype=cp.float32)
    typ = cp.asarray(rng.integers(0, K, N).astype(np.int32))
    A = cp.asarray(rng.uniform(-1, 1, (K, K)).astype(np.float32))

    rmax = cp.float32(88.0)
    rmax_val = 88.0
    beta = cp.float32(0.3)
    friction = cp.float32(0.85)
    fs = cp.float32(0.8)
    dt = cp.float32(0.6)

    buf = cp.zeros((height, width), dtype=cp.float32)
    cell_size = rmax_val
    grid_h = int(np.ceil(height / cell_size))
    grid_w = int(np.ceil(width / cell_size))
    n_cells = grid_h * grid_w

    print(f"Rendering {iters} frames at {width}×{height}, N={N} on GPU...",
          flush=True)
    mem_pool = cp.get_default_memory_pool()

    import time
    t_start = time.time()
    last_report = t_start

    for t in range(iters):
        cell_y = (pos[:, 0] / cell_size).astype(cp.int32) % grid_h
        cell_x = (pos[:, 1] / cell_size).astype(cp.int32) % grid_w
        cell_idx = cell_y * grid_w + cell_x

        sort_order = cp.argsort(cell_idx)
        cell_idx_sorted = cell_idx[sort_order]

        all_cells = cp.arange(n_cells, dtype=cp.int32)
        cell_start_all = cp.searchsorted(cell_idx_sorted, all_cells,
                                         side='left')
        cell_end_all = cp.searchsorted(cell_idx_sorted, all_cells,
                                       side='right')
        cell_start = cp.where(cell_end_all > cell_start_all,
                              cell_start_all, -1)
```

```
cell_end = cell_end_all

ny = (cell_y[:, None] + _DY_OFFSETS[None, :]) % grid_h
nx = (cell_x[:, None] + _DX_OFFSETS[None, :]) % grid_w
neighbour_ids = (ny * grid_w + nx).astype(cp.int32)
flat_ne = neighbour_ids.reshape(-1)
starts = cell_start[flat_ne]
ends = cell_end[flat_ne]
counts = cp.where(starts >= 0, ends - starts, 0).astype(cp.int32)

total = int(counts.sum())

per_particle = counts.reshape(N, 9).sum(axis=1)
idx_i = cp.repeat(cp.arange(N, dtype=cp.int32), per_particle)

cum_counts = cp.concatenate(
    [cp.zeros(1, dtype=cp.int64),
     cp.cumsum(counts.astype(cp.int64))])
positions = cp.arange(total, dtype=cp.int64)
pair_idx = cp.searchsorted(cum_counts[1:], positions,
                           side='right').astype(cp.int32)
local_offset = (positions - cum_counts[pair_idx]).astype(cp.int32)
sort_idx = starts[pair_idx] + local_offset
idx_j = sort_order[sort_idx].astype(cp.int32)

keep = idx_i != idx_j
idx_i = idx_i[keep]
idx_j = idx_j[keep]

d = pos[idx_j] - pos[idx_i]
cp.subtract(d[:, 0], S[0] * cp.round(d[:, 0] / S[0]), out=d[:, 0])
cp.subtract(d[:, 1], S[1] * cp.round(d[:, 1] / S[1]), out=d[:, 1])
dist = cp.sqrt((d * d).sum(axis=1))

within = dist <= rmax
idx_i = idx_i[within]
idx_j = idx_j[within]
d = d[within]
dist = dist[within]

rn = dist / rmax
dirv = d / (dist[:, None] + cp.float32(1e-9))

Aij_pairs = A[typ[idx_i], typ[idx_j]]
F_val = _force_value(rn, Aij_pairs, beta)
force_contrib = fs * F_val[:, None] * dirv

acc = cp.zeros((N, 2), dtype=cp.float32)
cp.add.at(acc, idx_i, force_contrib)

vel = vel * friction + acc * dt
pos = (pos + vel * dt) % S

frame_host = render_points(buf, pos, S)
ffmpeg_stdin.write(frame_host.tobytes())

if (t & 0x1F) == 0:
    mem_pool.free_all_blocks()

# Progress report every ~10 sec wall clock
now = time.time()
if now - last_report > 10:
    elapsed = now - t_start
    fps = (t + 1) / elapsed
    eta = (iters - t - 1) / fps
    print(f"  frame {t + 1}/{iters}  "
          f"({(t + 1) / iters * 100:5.1f}%)  "
          f"fps={fps:5.1f}  "
          f"eta={eta / 60:5.1f} min", flush=True)
    last_report = now
```

```
    print(f"Completed {iters} frames in {time.time() - t_start:.0f} sec")


if __name__ == "__main__":
    seed = 480
    P = PARAMS
    width, height = P['width'], P['height']
    iters, N = P['iters'], P['N']
    outdir = (f"/home/fox/Downloads/py/output/even_more/particle_life/"
              f"{seed}_{MODE}")
    Path(outdir).mkdir(parents=True, exist_ok=True)
    fps = P.get('fps', 20)
    out_ext = P.get('out_ext', '.mp4')
    mp4_path = Path(outdir) / f"animation{out_ext}"

    print(f"=== Particle Life Rendering: MODE={MODE} ===")
    print(f"Seed: {seed}   Resolution: {width}×{height}   "
          f"Duration: {iters / fps:.1f} sec ({iters} frames @ {fps} fps)")
    print(f"N: {N}   GPU: "
          f"{cp.cuda.runtime.getDeviceProperties(0)['name'].decode()}")
    print()

    # ffmpeg command construction.  Two distinct flavours: real DVD-Video
    # (-target ntsc-dvd, MPEG-2 PS, silent AC-3 audio track) vs the
    # h264/x265 .mp4 path.  Both stream video frames from stdin so no
    # giant raw file is materialised.
    if P.get('dvd_target'):
        # DVD-Video pipeline — outputs a .mpg you feed to dvdauthor.
        cmd = [
            'ffmpeg', '-y',
            '-f', 'rawvideo', '-pix_fmt', 'gray',
            '-s', f'{width}x{height}',
            '-framerate', str(fps),
            '-i', '-',
        ]
        if P.get('add_silent_audio'):
            cmd += ['-f', 'lavfi',
                    '-i', 'anullsrc=channel_layout=stereo:sample_rate=48000']
        cmd += ['-target', P['dvd_target']]
        # Override video bitrate that -target ntsc-dvd would default to.
        # The DVD-Video max combined bitrate is 10.08 Mbps; we deliberately
        # stay well under so we hit a specific final file size.
        if P.get('video_bitrate'):
            cmd += ['-b:v', P['video_bitrate']]
        if P.get('audio_bitrate'):
            cmd += ['-b:a', P['audio_bitrate']]
        if P.get('aspect'):
            cmd += ['-aspect', P['aspect']]
        cmd += ['-shortest', str(mp4_path)]
    else:
        cmd = [
            'ffmpeg', '-y',
            '-f', 'rawvideo', '-pix_fmt', 'gray',
            '-s', f'{width}x{height}',
            '-framerate', str(fps),
            '-i', '-',
            '-c:v', P.get('codec', 'libx264'),
            '-pix_fmt', 'yuv420p',
            '-preset', 'medium',
        ]
        if P.get('bitrate'):
            cmd += ['-b:v', P['bitrate']]
            if P.get('maxrate'):
                cmd += ['-maxrate', P['maxrate'], '-bufsize', P['bufsize']]
        elif P.get('crf') is not None:
            cmd += ['-crf', str(P['crf'])]
        if P.get('aspect'):
            cmd += ['-aspect', P['aspect']]
        cmd += [str(mp4_path)]
```

```
print(f"ffmpeg cmd: {' '.join(cmd)}")
print()
ff = subprocess.Popen(cmd, stdin=subprocess.PIPE,
                      stderr=subprocess.DEVNULL)
try:
    particle_life_stream(width, height, iters, ff.stdin,
                         N=N, seed=seed)
finally:
    ff.stdin.close()
    ff.wait()

sz = mp4_path.stat().st_size if mp4_path.exists() else 0
print(f"\nMP4 saved: {mp4_path} ({sz / 1024**3:.2f} GB, "
      f"{sz / 1024**2:.1f} MB)")
print(f"\n=== Complete ===")
```

Related pages

- 3-GPU mesh — hardware context for the 4090 this script targets.

Updated 2026-06-29.

fox@neoblanka:~/git/uncloseai-cli$

> '-preset', 'medium',] if P.get('bitrate'): cmd += ['-b:v', P['bitrate']] if P.get('maxrate'): cmd += ['-maxrate', P['maxrate'], '-bufsize', P['bufsize']] elif P.get('crf') is not None: cmd += ['-crf', str(P['crf'])] if P.get('aspect'): cmd += ['-aspect', P['aspect']] cmd += [str(mp4_path)]
>
> print(f"ffmpeg cmd: {' '.join(cmd)}") print() ff = subprocess.Popen(cmd, stdin=subprocess.PIPE, stderr=subprocess.DEVNULL) try: particle_life_stream(width, height, iters, ff.stdin, N=N, seed=seed) finally: ff.stdin.close() ff.wait()
>
> sz = mp4_path.stat().st_size if mp4_path.exists() else 0 print(f"nMP4 saved: {mp4_path} ({sz / 1024*3:.2f} GB, " f"{sz / 1024*2:.1f} MB)") print(f"n=== Complete ===")

Related pages

- 3-GPU mesh — hardware context for the 4090 this script targets.

Source: https://foxhop.net/0b6d520b-7414-11f1-9565-040140774501/particle-life-on-gpu-cupy-port-for-long-form-renders
Snapshot: 2026-06-30T04:04:05Z
Generator: Remarkbox `dba4024`