Russian light

1. Imports 

Import all external libraries required for the program to exist in reality.

- `cv2` handles webcam access and image processing
- `pygame` handles rendering, timing, and display
- `numpy` is used for fast array operations on image masks
- `math` provides trigonometric functions
- `random` injects chaos in controlled doses

No library here is optional. remove one and the system collapses.

2. Configuration values 

This block defines all global parameters that shape the behavior of the system.

- screen dimensions (`w, h`) define the rendering resolution
- `too_bright` sets the brightness threshold for detecting light sources
- `spawn_rate` and `particle_limit` control particle density and performance safety
- `angle_noise` introduces directional variation to avoid rigid patterns
- `anti_gravity` applies a subtle upward force, violating physics politely

These values are separated to allow easy tuning without touching the logic, which is already fragile enough.

3. Setup and initialization 

This section establishes contact with both reality and the simulation.

- the webcam is opened using opencv
- pygame is initialized
- a window is created
- a clock object is instantiated to regulate time

This is where the program decides it is in fact happening.

4. Particle definition (thing class) 

This class represents a single visual particle emitted from a detected light source.

Initialization

Each particle:

- spawns at a bright pixel location
- calculates its movement direction based on its position relative to the light center
- applies random angular deviation to avoid symmetry
- receives a random speed, lifespan, size, and color

The result is a particle that behaves like light escaping but thinking about it first.

Update method

This method is called every frame and handles:

- velocity decay (friction)
- vertical force application (anti-gravity)
- position updates
- lifespan reduction

It simulates the transition from energetic ray to drifting dust.

Draw method

This method renders the particle as a small surface with alpha transparency based on remaining life.

Particles fade out instead of disappearing abruptly, which is both visually pleasing and emotionally considerate.

5. Program state 

This section defines mmutable state used by the main loop.

- particles stores all active particle objects
- running controls the main loop execution

This is the memory of the system. it forgets aggressively.

6. main loop 

This loop runs once per frame and is the core of the application.

Frame acquisition

A new frame is captured from the webcam, flipped for mirror consistency, resized, and converted to rgb for pygame compatibility.

Light detection

The frame is converted to grayscale and thresholded.

Pixels exceeding the brightness threshold are extracted and treated as valid light sources.

If no bright pixels are found, nothing spawns. the universe remains quiet.

Particle spawning

If bright pixels exist:

- their average position is computed
- new particles are spawned at random bright pixel locations
- each particle receives directionality relative to the average center

This produces a ray-like emission effect rather than a radial explosion.

Rendering layers

Two layers are rendered each frame:

1. The webcam frame (base reality)
2. A transparent overlay containing all particles

The overlay is blended additively, causing light accumulation and glow effects.

Event handling

Pygame events are processed to allow window closure.

Without this, the program would need to be terminated with force, which is rude.

Timing

The clock enforces a 60 fps limit, preserving visual stability and preventing the cpu from achieving enlightenment.

Cleanup

When the loop ends:

- The webcam is released
- Pygame shuts down gracefully

This ensures the system returns borrowed resources and pretends nothing happened.