EEE303 PROGRAMMING WITH PYTHON FOR ENGINEERS TERM PROJECT

BLACK HOLE SIMULATION USING PYTHON

CEM KUTAY NANÇIN-181112026

ENES SÖKMEN-23111012022

ÖMER FARUK KOLAYCA-23111012054

M. PUSAT ÖZÇELİK-23111012003

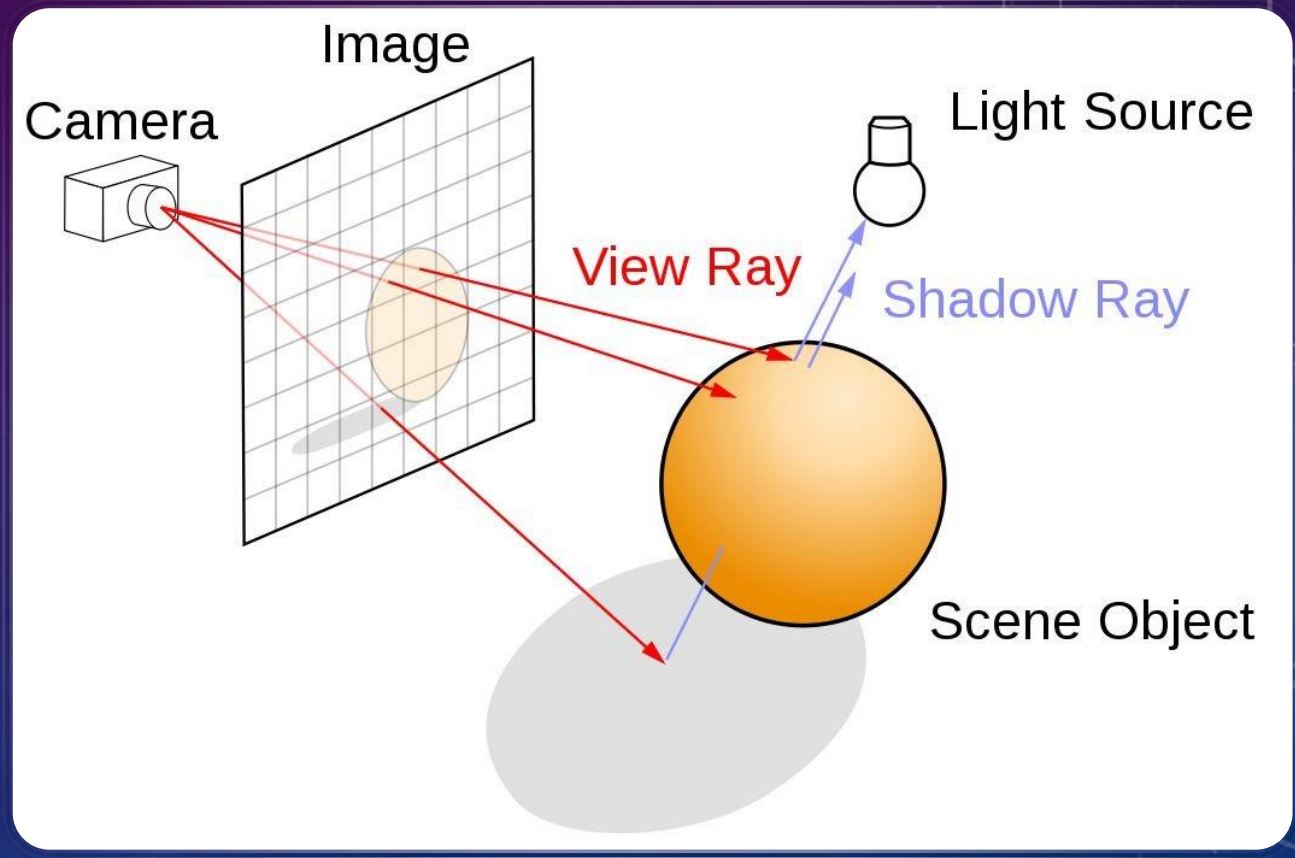
FURKAN KART-23111012042

LIBRARIES USED IN THE PROJECT

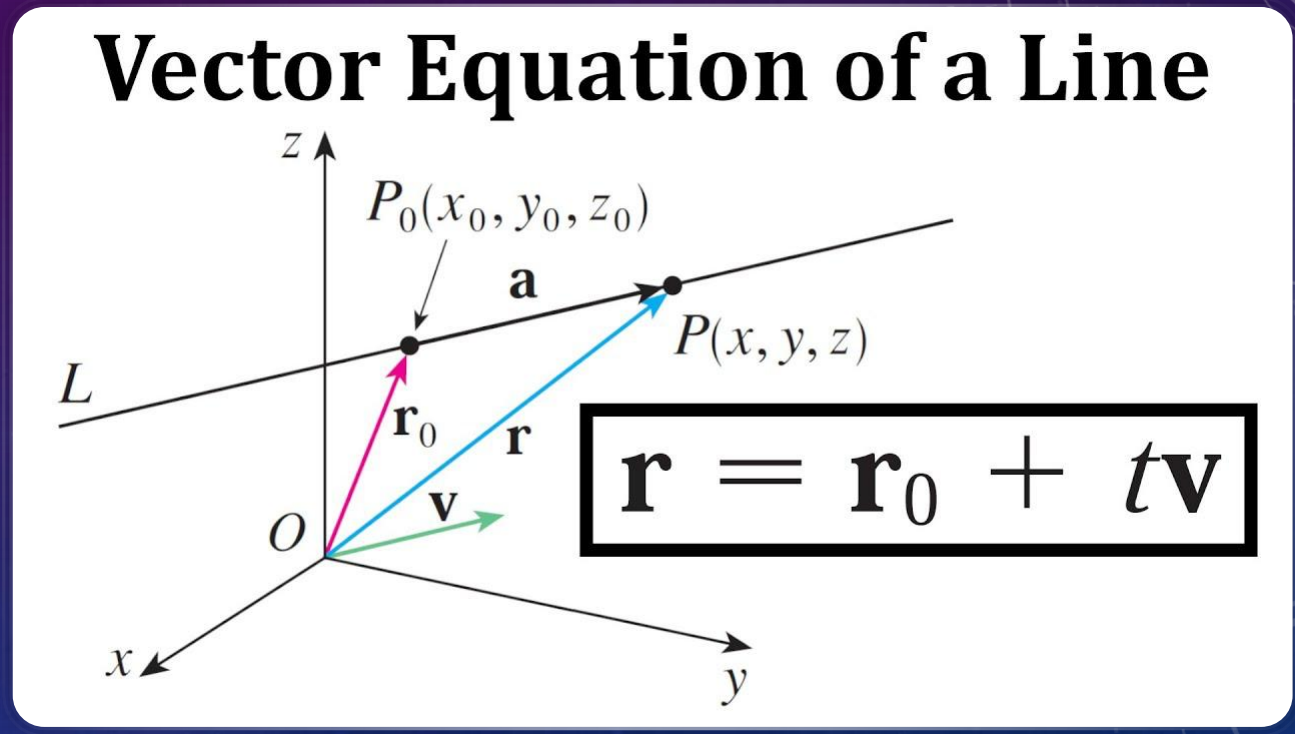
- NUMPY: Vector Algebra
- MATH: Mathematical Operations
- NUMBA: Running Calculations on GPU
- PILLOW: Image Recording Processes
- OPENCV: Image Processing
- TIME: Time Oriented Operations
- OS: File Operations

RAY TRACING

- It traces the path of light rays from the camera to objects to calculate realistic reflections and shadows.

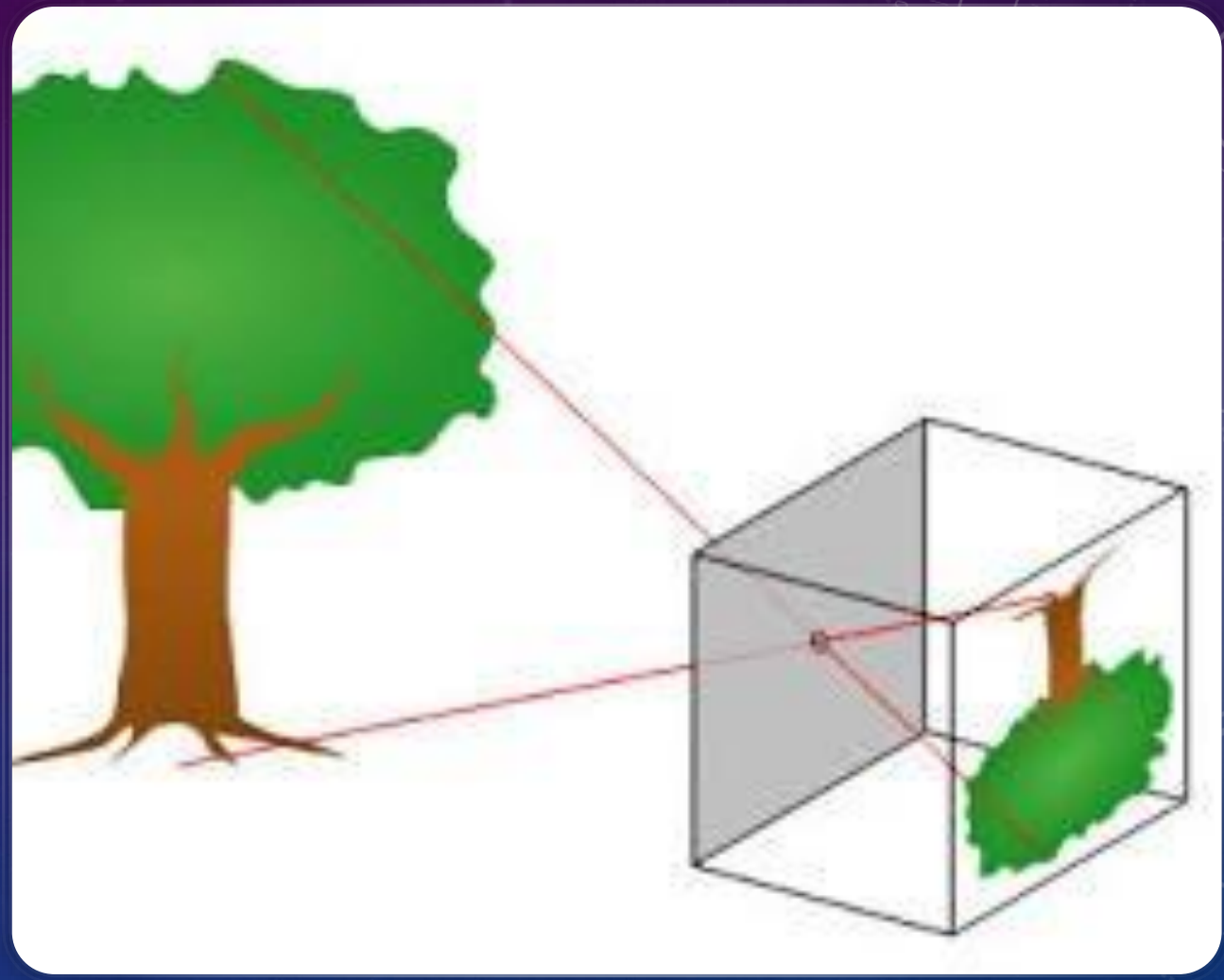


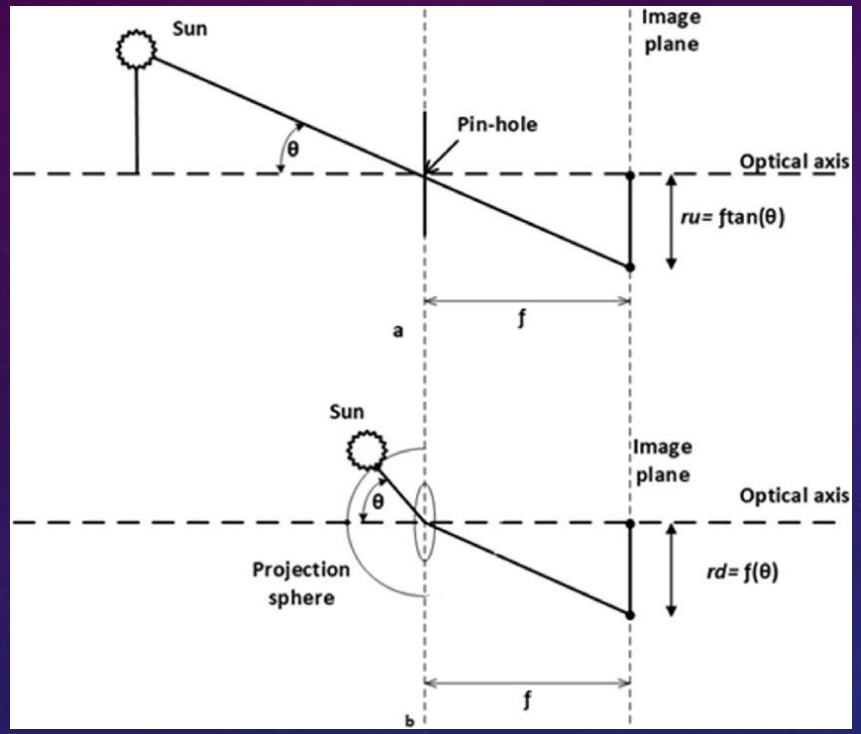
- This is the equation used to find the line equation of the casted rays.



RECTILINEAR PROJECTİON

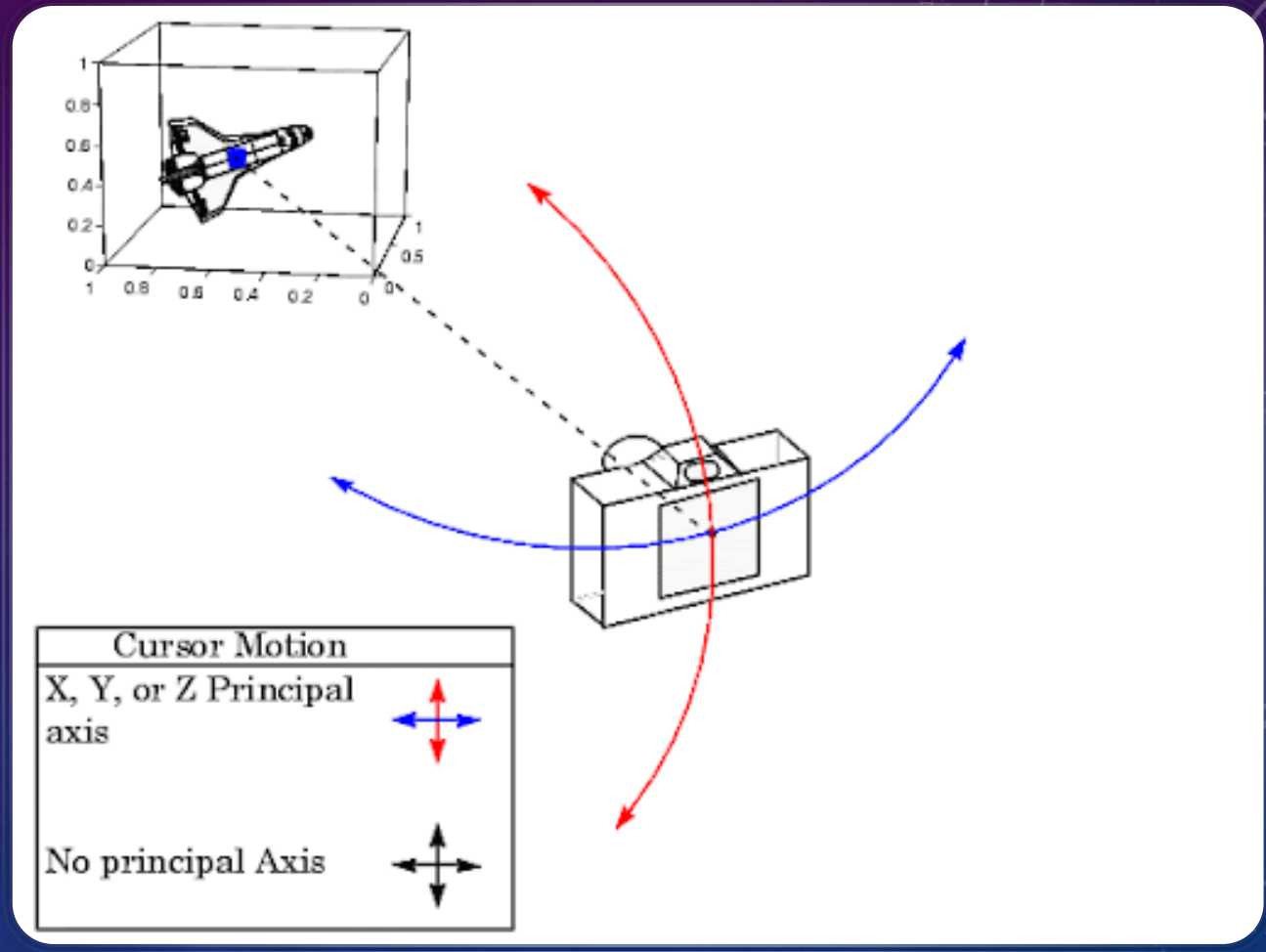
- Rectilinear projection preserves straight lines from the 3D scene onto the 2D image plane.





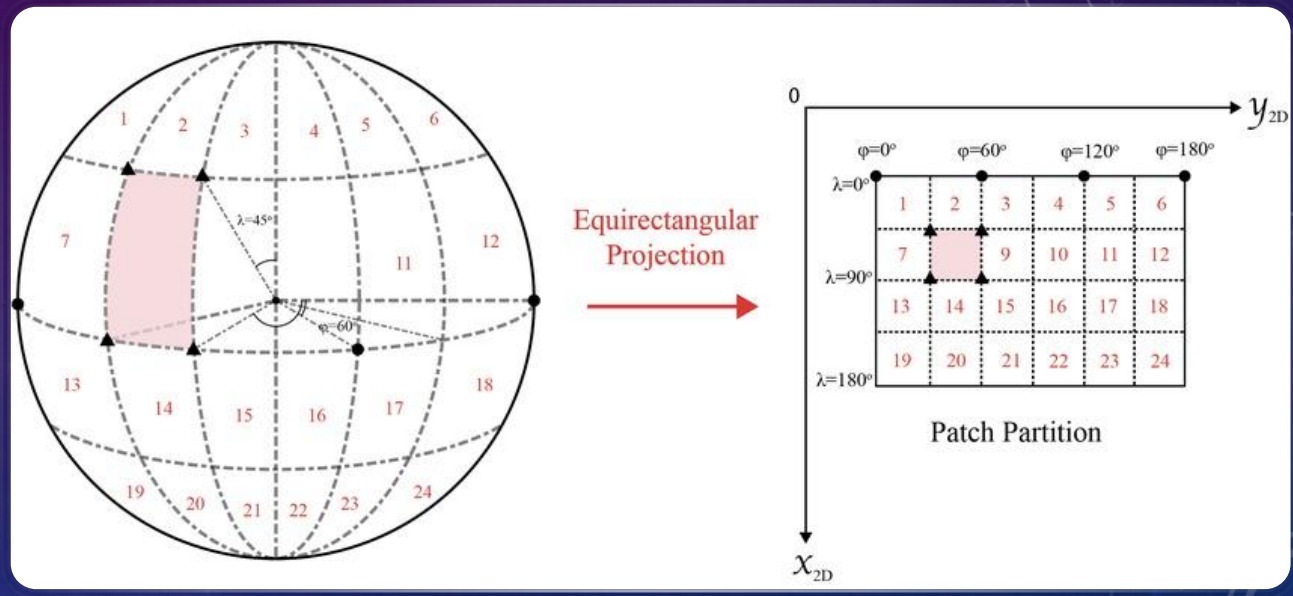

```python
def find_ray_direction(self, h, w):
    forward = spherical_to_cartesian(self.direction)
    forward /= np.linalg.norm(forward)

    global_up = np.array([0, 0, 1.0])
    if abs(np.dot(forward, global_up)) > 0.99:
        global_up = np.array([0, 1.0, 0])

    right = np.cross(forward, global_up)
    right /= np.linalg.norm(right)

    up = np.cross(right, forward)

    scale = math.tan(math.radians(self.fov) / 2.0)

    px = (2 * (w + 0.5) / self.width - 1) * scale * self.aspect_ratio
    py = (1 - 2 * (h + 0.5) / self.height) * scale

    d = forward + px * right + py * up
    return d / np.linalg.norm(d)
```

- The camera operates in orbital mode, rotating around a fixed target object.

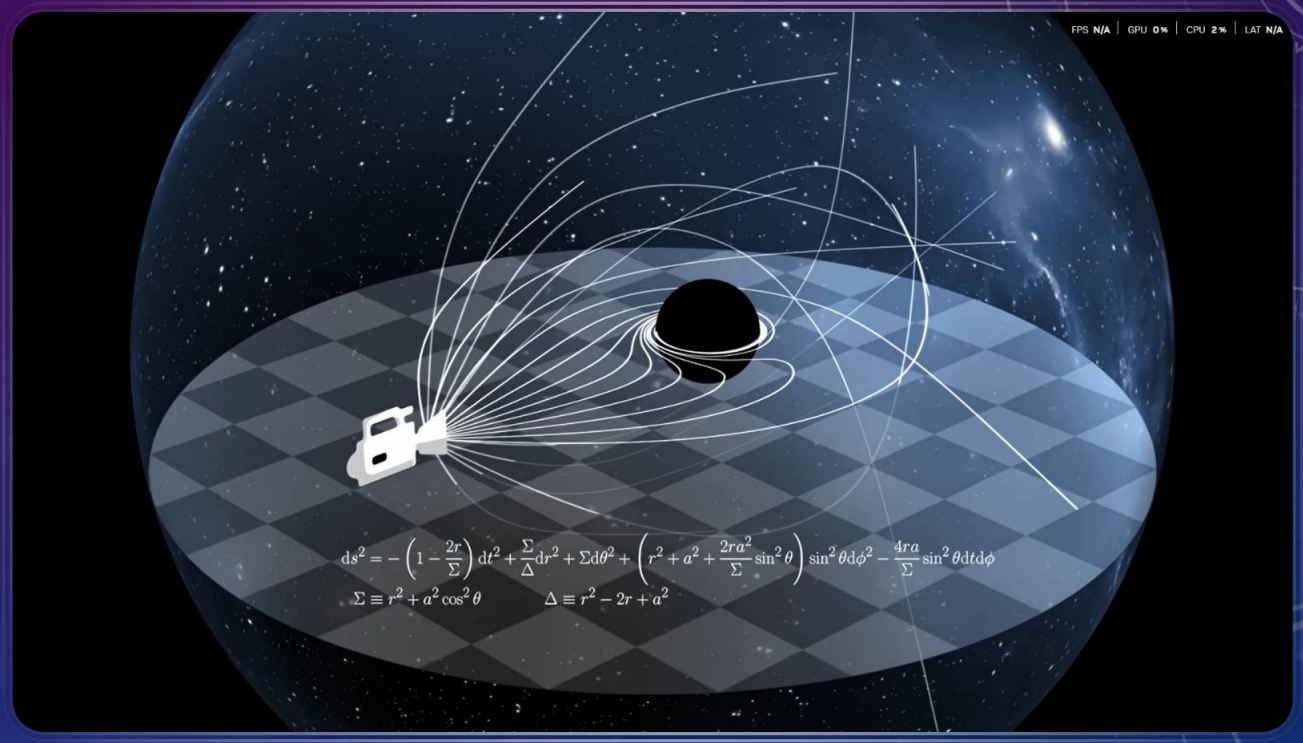


EQUIRECTANGULAR PROJECTION

- Mapping the rectangular image onto a spherical surface using azimuth and elevation angles of the casted rays.



- Mapping the direction of escaping rays to the environment map to retrieve the corresponding pixel color."



EINSTEIN FIELD EQUATIONS-GEODESIC EQUATION

- The Einstein Field Equations define how matter curves spacetime, while the Geodesic Equation describes how objects move within that curvature.

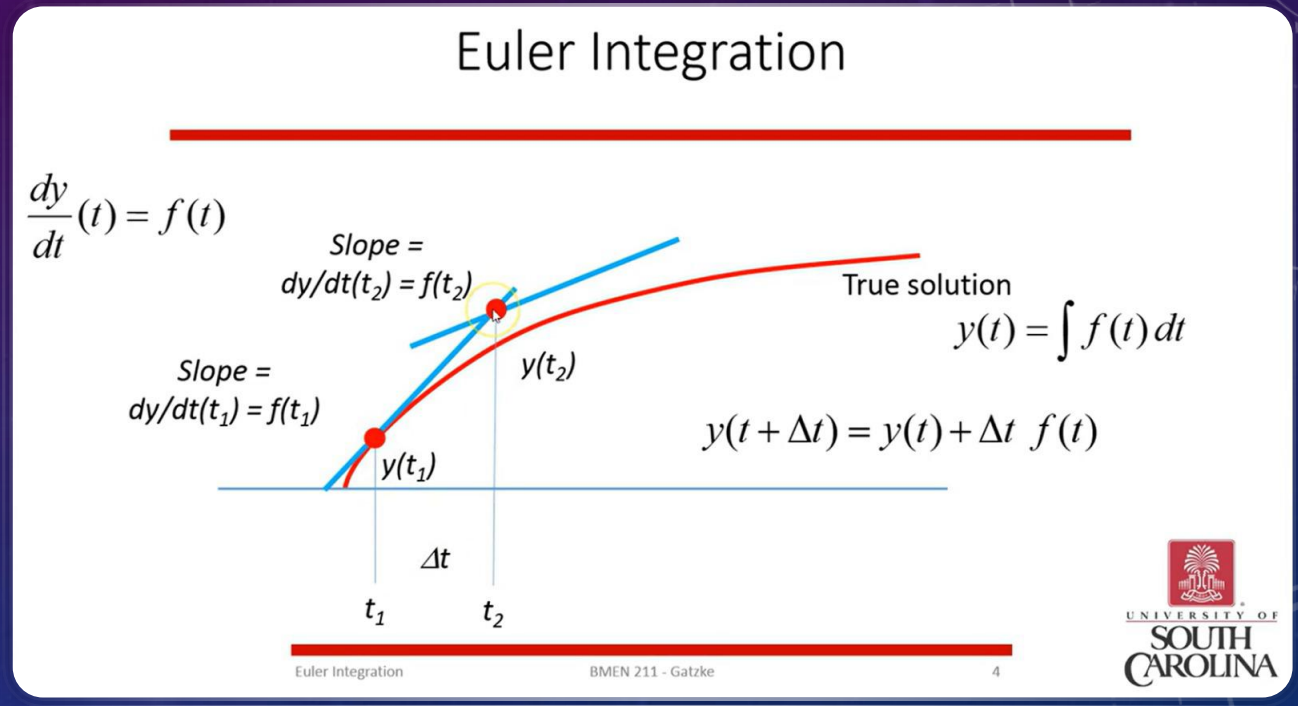
```
class BlackHole():

    def __init__(self, scene, position, rs):
        scene.add(self)
        self.position = np.array(position, dtype=float)
        self.rs = rs

    def geodesic(self, ray, dλ):
        dλ = dλ

        #ray = [t, r, theta, phi, dt, dr, dtheta, dphi]
        t = ray[0]
        r = ray[1]
        theta = ray[2]
        phi = ray[3]

        dt = ray[4]
        dr = ray[5]
        dtheta = ray[6]
        dphi = ray[7]

        rs = self.rs

        ddt = -1*((rs / (r*(r-rs))) * dr * dt)
        ddr = -1*((((rs*(r-rs))/(2*r**3))*dt**2) - ((rs/(2*r*(r-rs)))*dr**2) - ((r-rs)*dtheta**2) - ((r-rs)*(math.sin(theta)**2)*dphi**2))
        ddtheta = -1*(((2/r)*dr*dtheta) - (math.sin(theta)*math.cos(theta)*dphi**2))
        ddphi = -1*((2/r)*dr*dphi + 2*(math.cos(theta) / math.sin(theta))*dtheta*dphi)

        dt = dt + ddt*dλ
        dr = dr + ddr*dλ
        dtheta = dtheta + ddtheta*dλ
        dphi = dphi + ddphi*dλ

        t = t + dt*dλ
        r = r + dr*dλ
        theta = theta + dtheta*dλ
        phi = phi + dphi*dλ

        return np.array([t, r, theta, phi, dt, dr, dtheta, dphi])
```

TAYLOR SERIES

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \ldots.$$

EULER INTEGRATION

- Euler's method is effectively a first-order Taylor series approximation used for numerical integration.



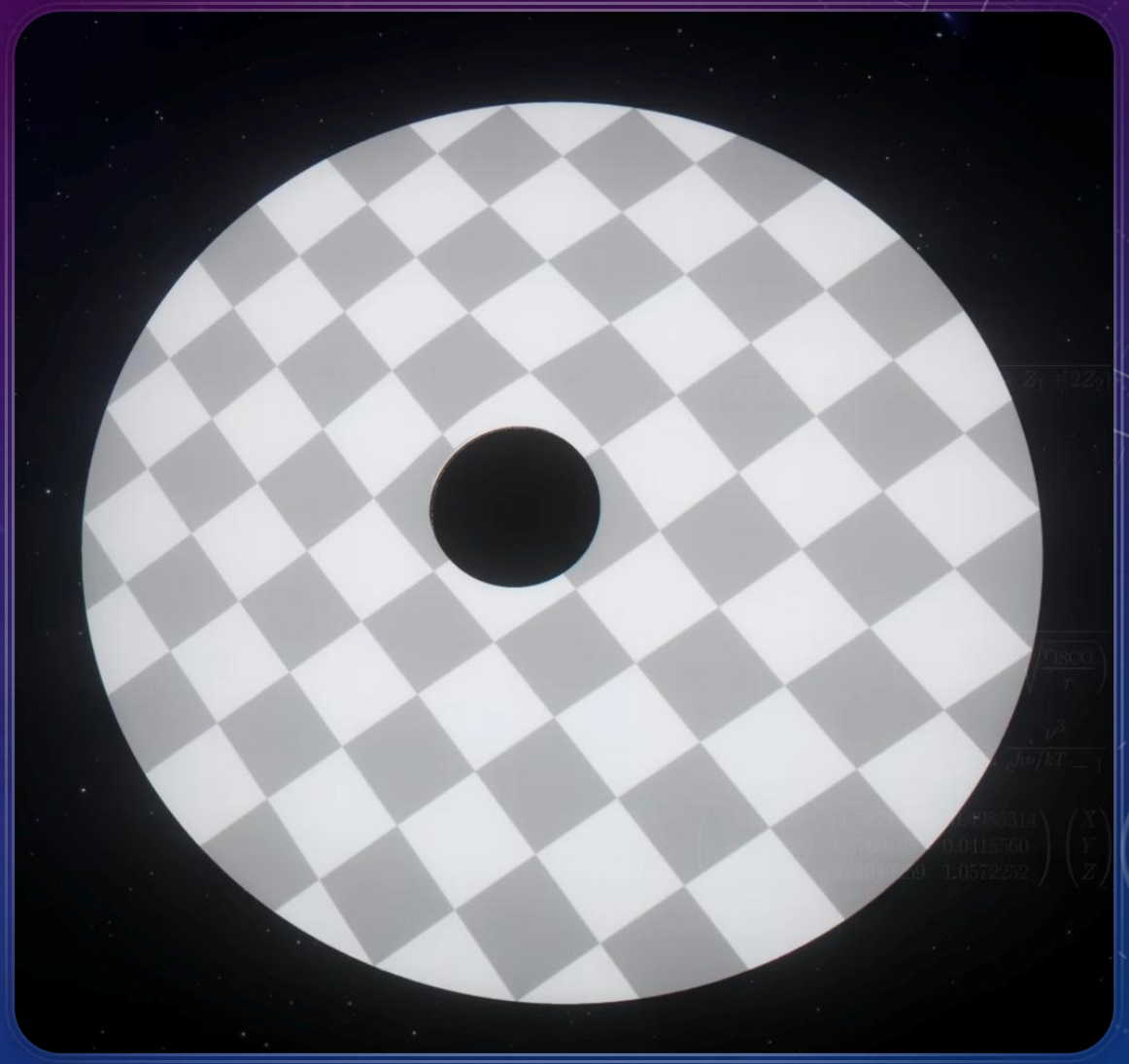
RK4 ALGORITHM

- Estimating the next value by taking a weighted average of four slopes calculated within the step interval.

```
def rk4_step(self, state, h):
    # h: Adım büyüklüğü (step size)
    k1 = h * self.get_derivatives(state)
    k2 = h * self.get_derivatives(state + 0.5 * k1)
    k3 = h * self.get_derivatives(state + 0.5 * k2)
    k4 = h * self.get_derivatives(state + k3)

    new_state = state + (k1 + 2*k2 + 2*k3 + k4) / 6.0
    # Bu kısım kabaca durumlar içinden ağırlıklı ortalama alır ve yeni durumu hesaplar
    return new_state
```

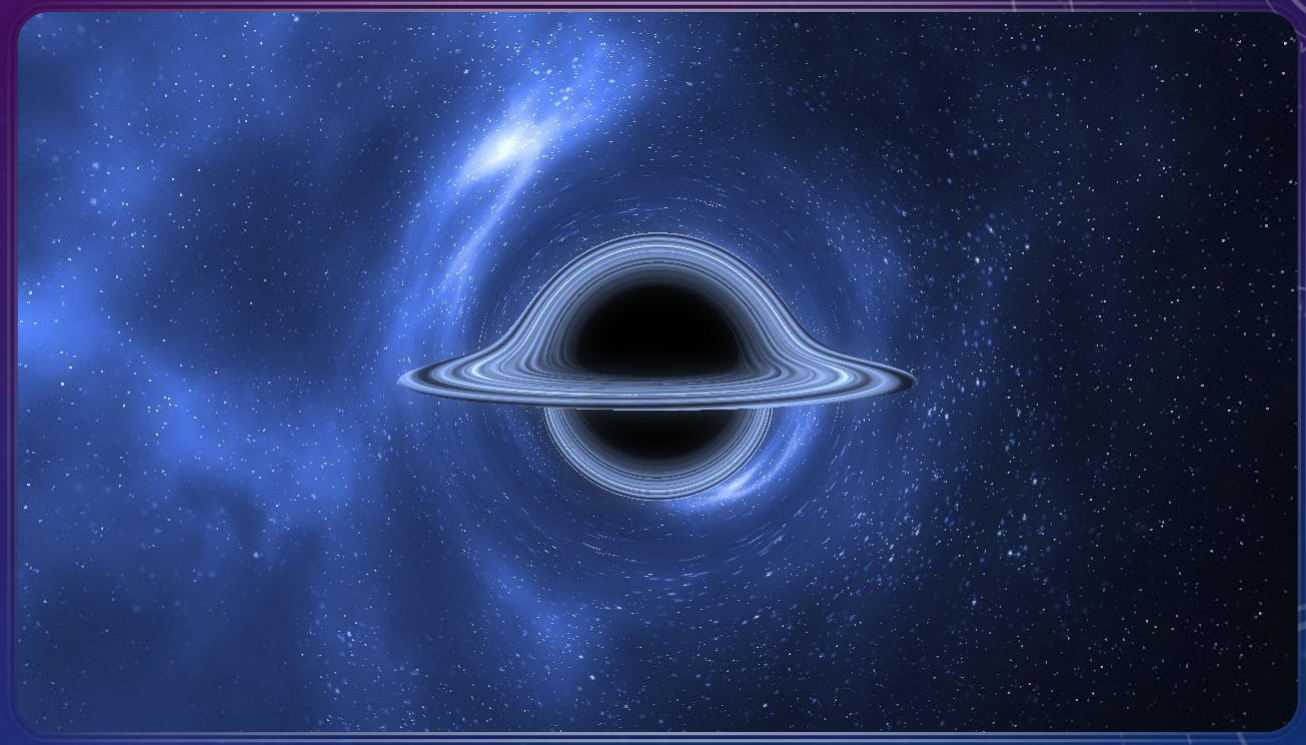
ACCRETION DISK

- The simulation models the disk as a flat 2D plane, using a checkerboard pattern to visualize the coordinate system. Due to gravitational lensing, light rays from behind the disk are bent around the black hole, making the far side visible above and below the event horizon.



- The rendering logic detects ray intersections with the accretion disk's plane and calculates the emitted color based on temperature gradients and procedural noise. Crucially, it applies relativistic Doppler beaming to simulate the characteristic brightness asymmetry caused by the disk's rapid rotation.

- Final output of the simulation. The geometry of the black hole and the gravitational and warping effects it creates on the spacetime can be clearly observed.



GPU ACCELERATION & PARALLEL COMPUTING

- Ray Casting requires independent calculations for millions of pixels. While a CPU processes these rays sequentially (Serial Processing), a GPU utilizes thousands of cores to compute them simultaneously (Parallel Processing). This massive parallelism and natural speed of GPU reduced the simulation time from estimated days to mere minutes.