Ray Tracing in Entertainment Industry

Tanaboon Tongbuasirilai

Dept. Computer Science

Kasetsart University

Week 10

Simple visual effects

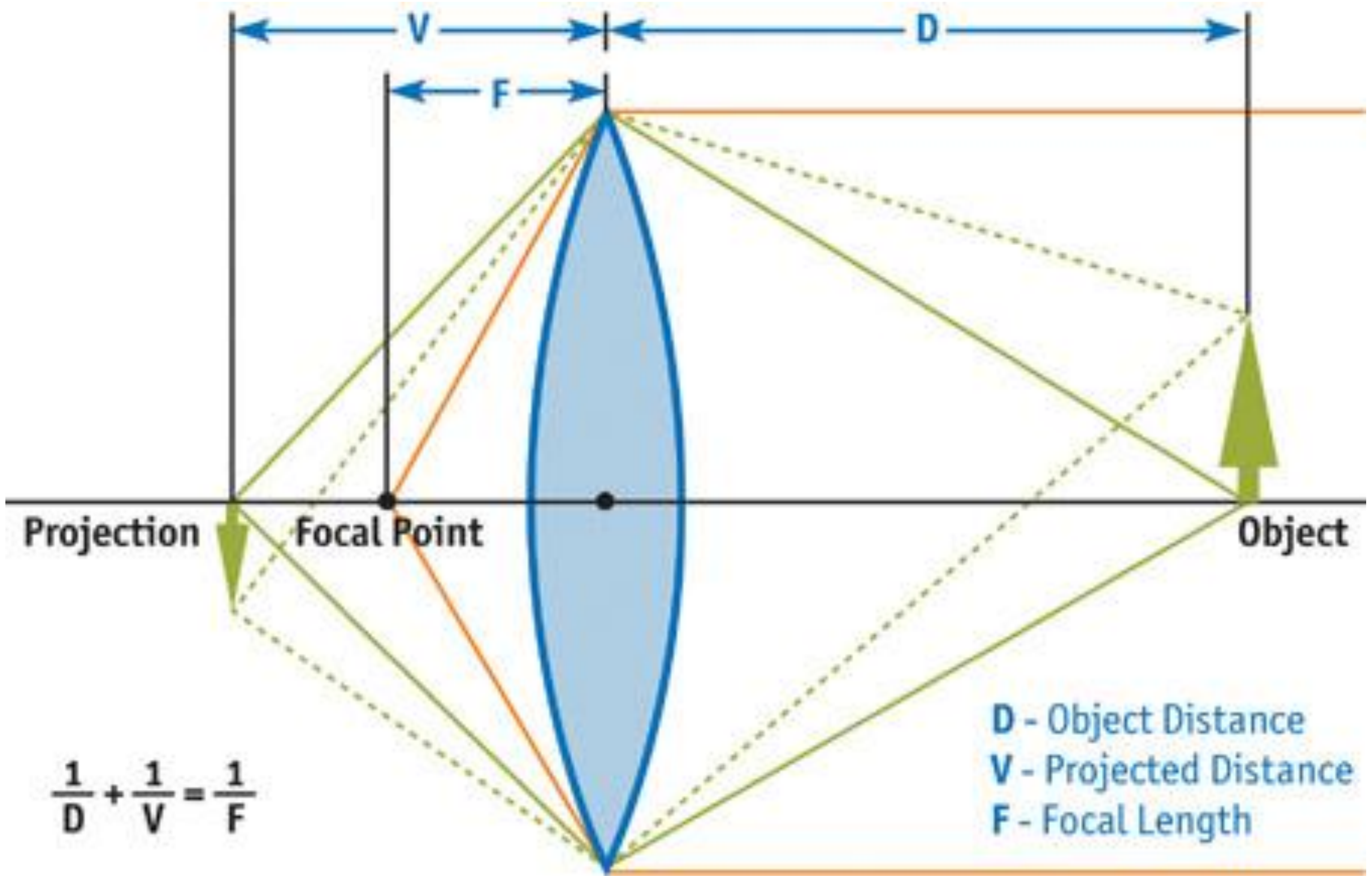
- Depth of Field (DoF)
- Motion blur

Depth of Field (DoF)

- Depth of field is the effect in which objects within some range of distances in a scene appear in focus, and objects nearer or farther than this range appear out of focus.
- Depth of field is frequently used in photography and cinematography to direct the viewer's attention within the scene, and to give a better sense of depth within a scene.

https://developer.nvidia.com/gpugems/gpugems/part-iv-image-processing/chapter-23-depth-field-survey-techniques

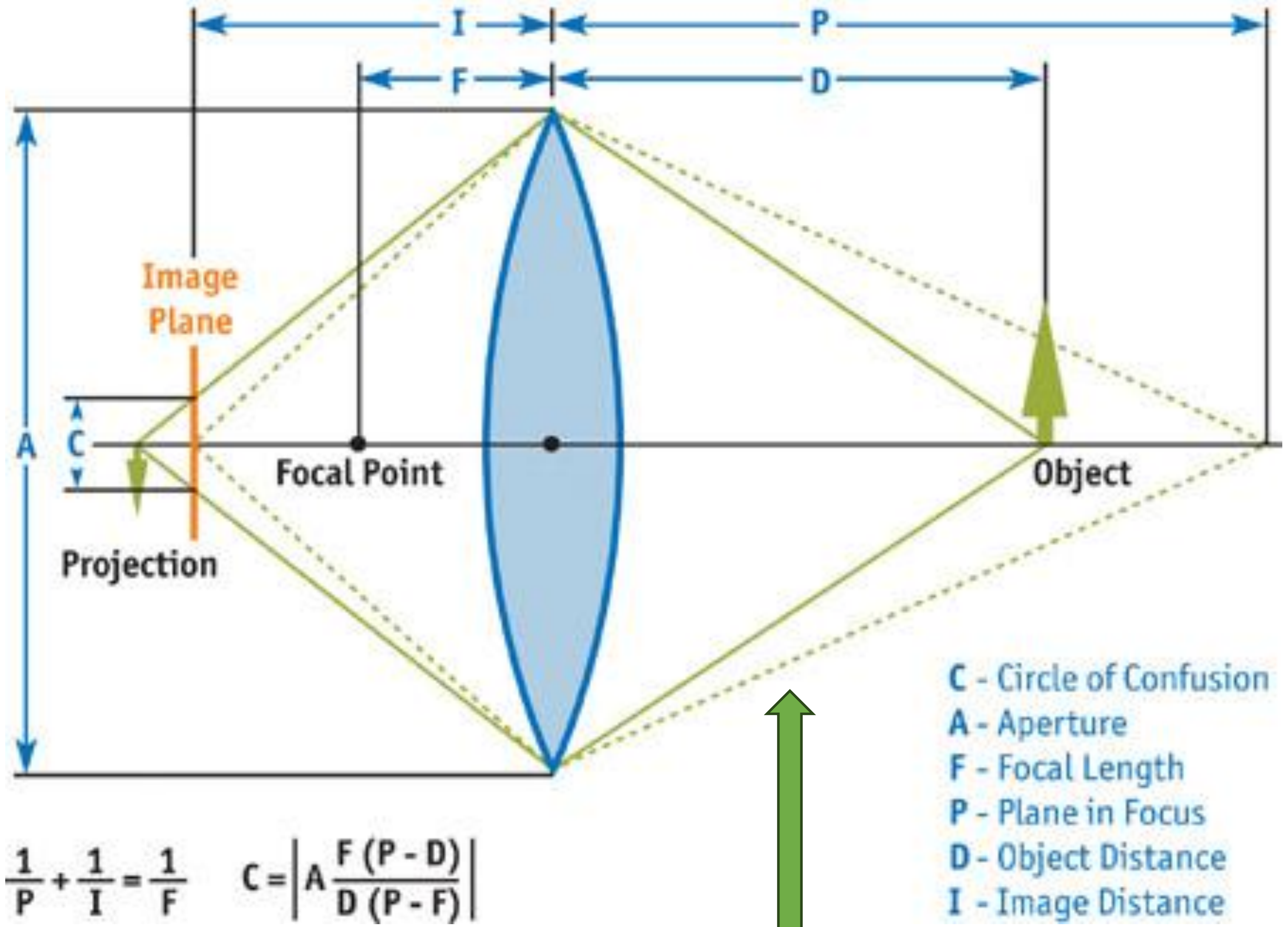
Thin Lens



https://developer.nvidia.com/gpugems/gpugems/part-iv-image-processing/chapter-23-depth-field-survey-techniques

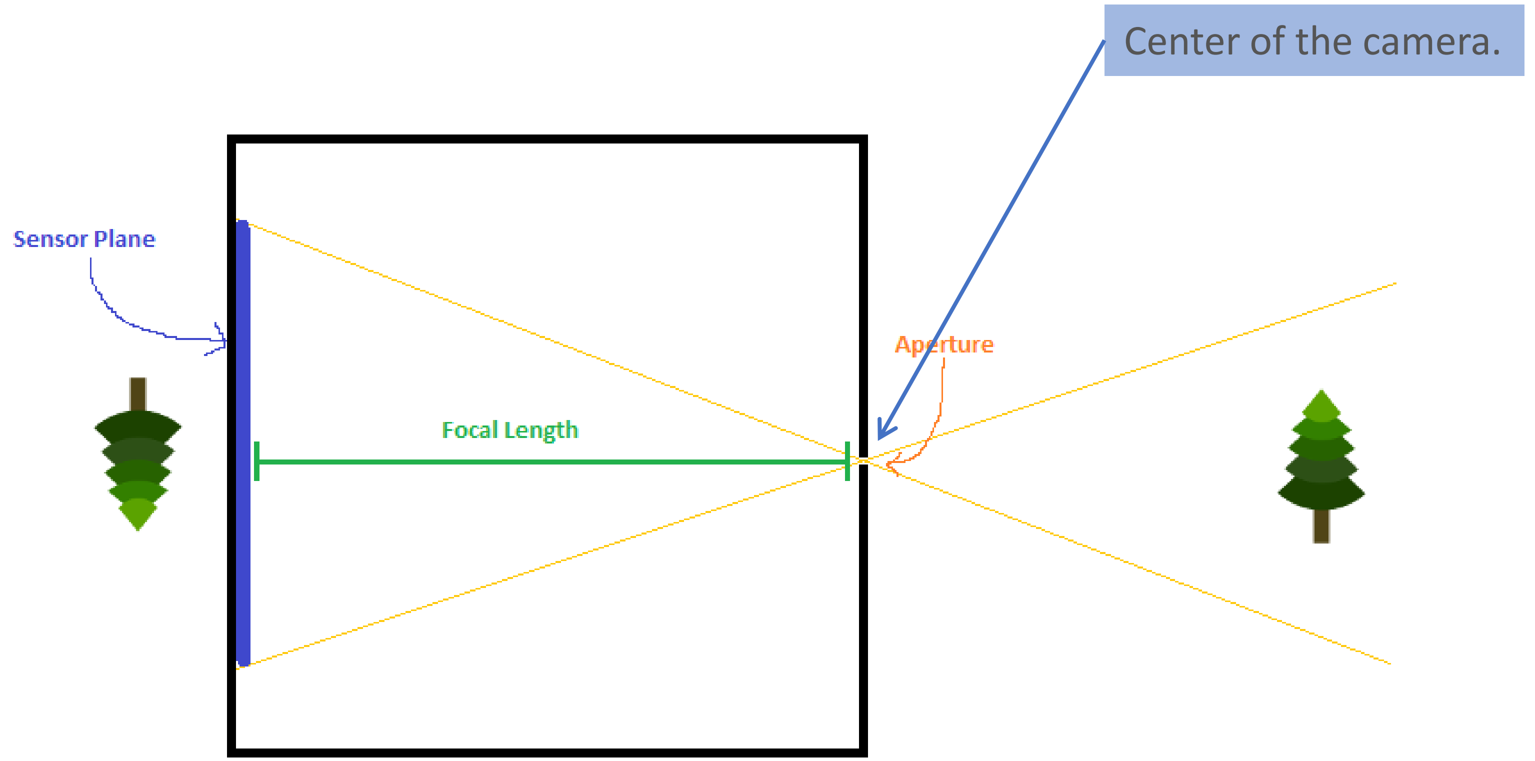
Circle of Confusion

When the CoC is **smaller** than the size of a pixel, that pixel is in focus while a CoC **larger** than a pixel will be out of focus and cause blurring



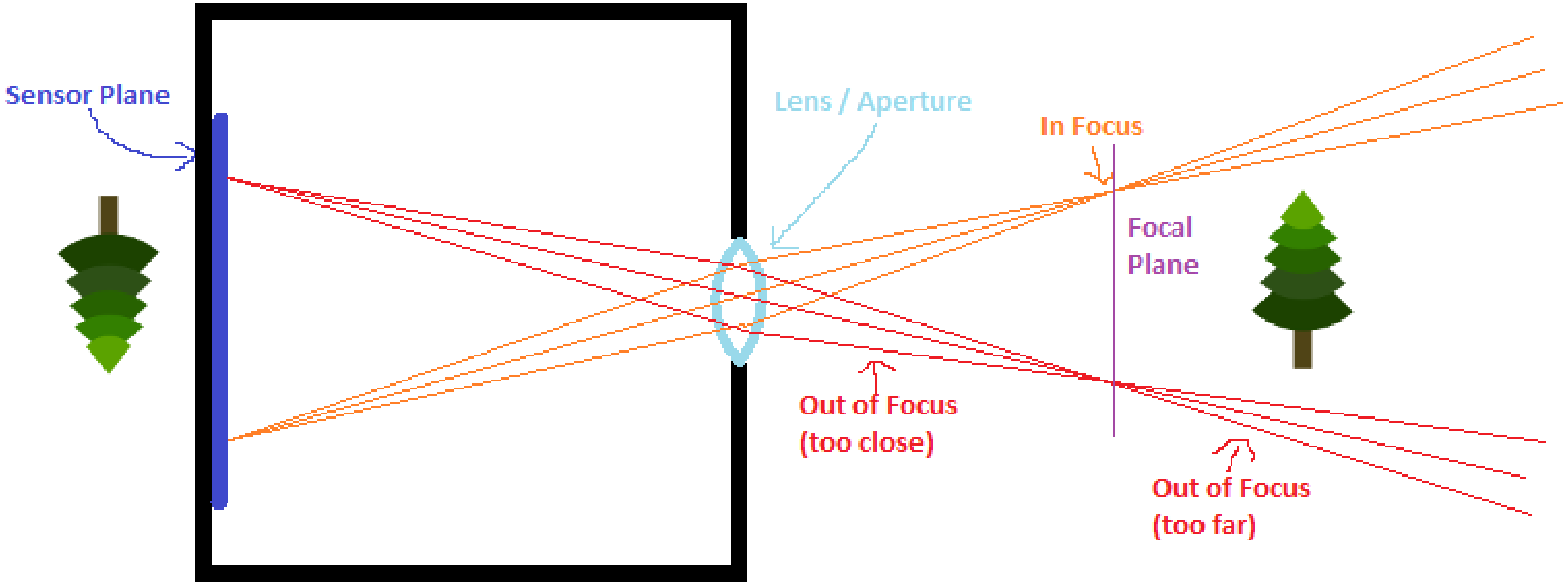
Sharp focus line is depicted by green dashed lines.

A tricky camera model

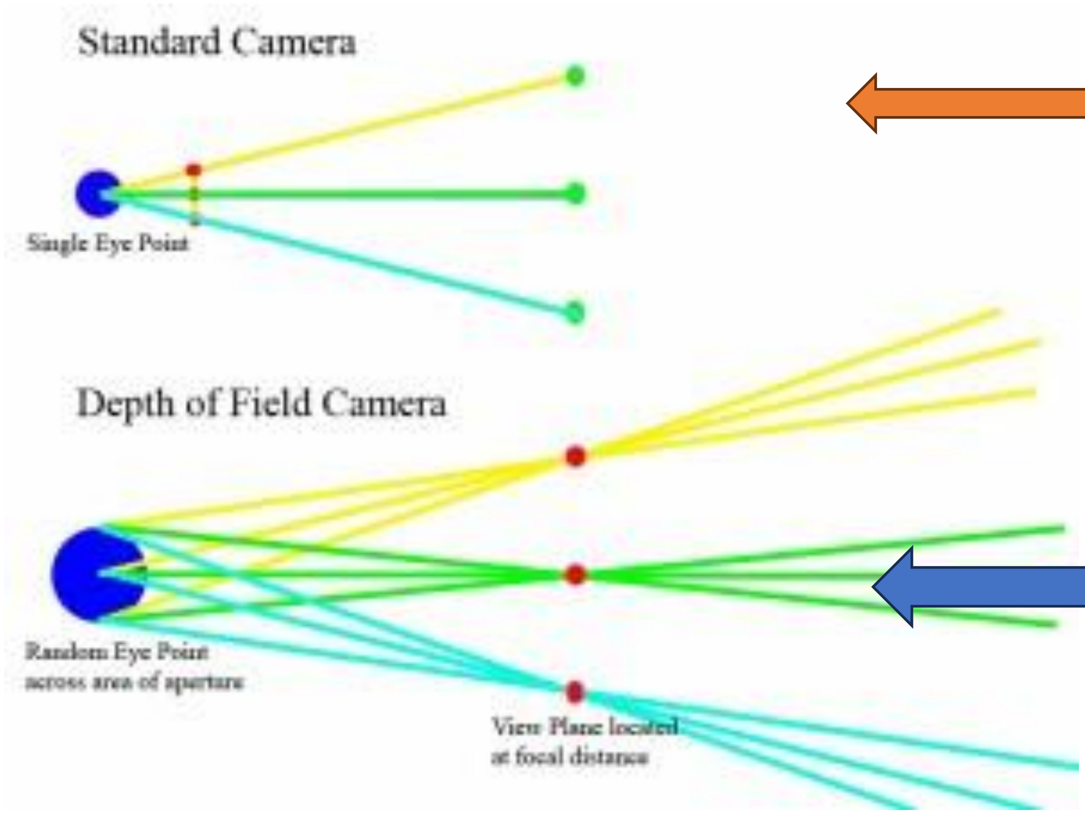


So far, we simulate this kind of pinhole camera model.

Adding Lens to the pinhole



Comparison

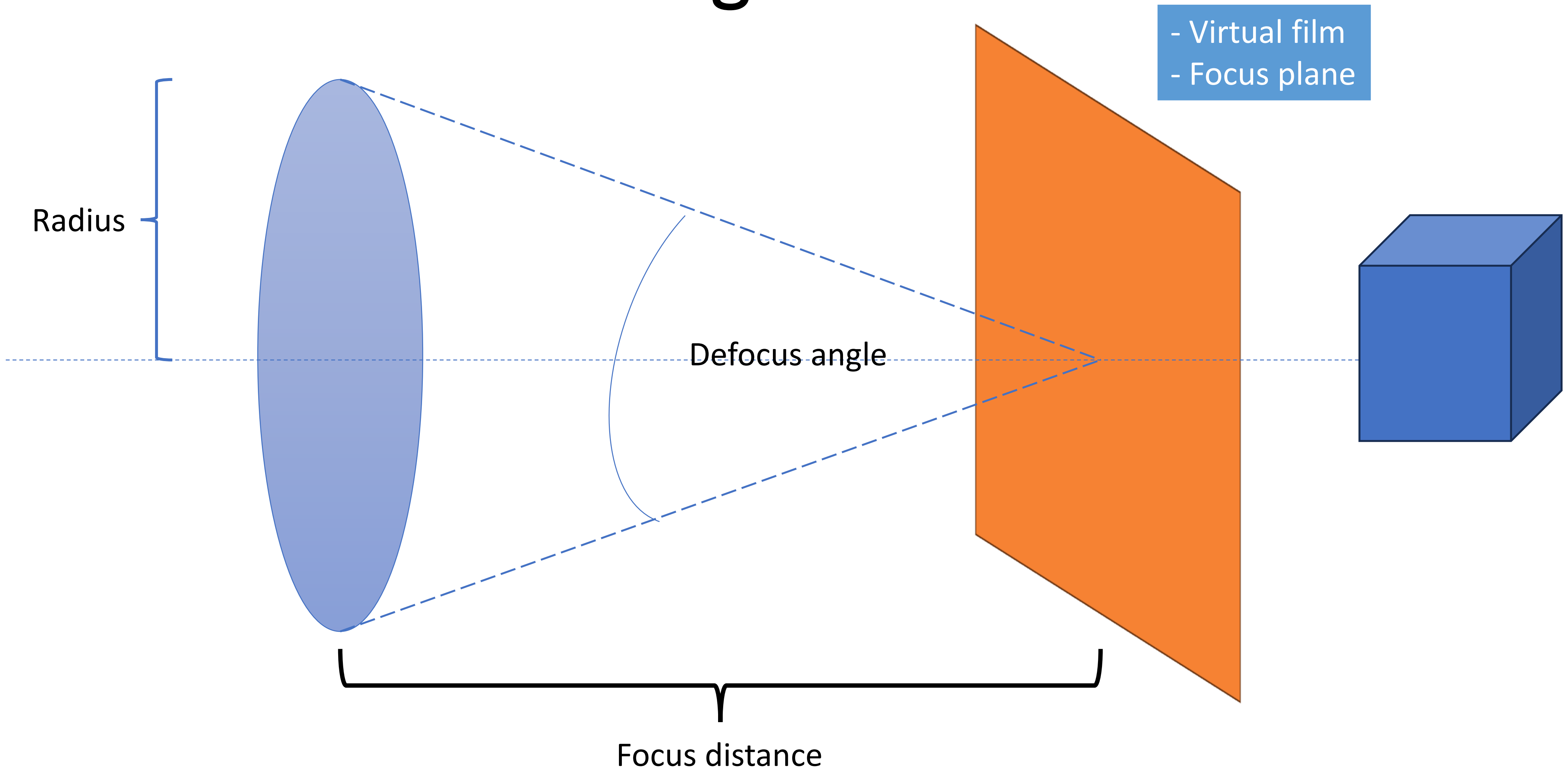


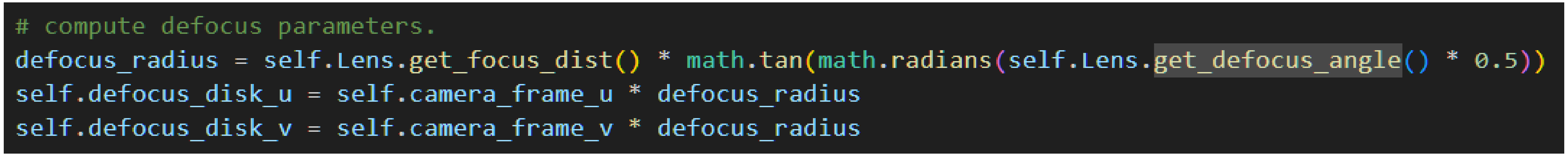
Do sampling at the center of the lens.

- Lens has its radius.
- Do sampling inside lens radius.
- Take the average of sampled rays.

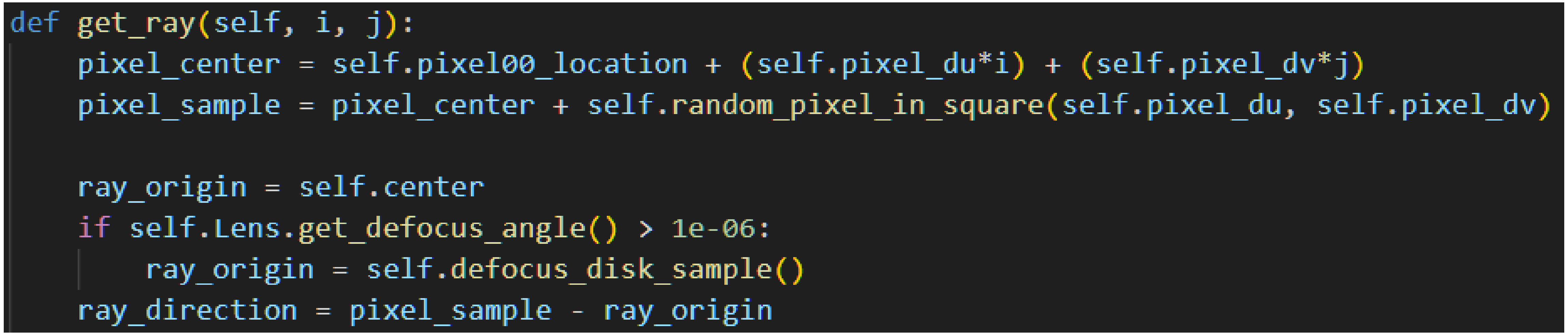
https://steveharveynz.wordpress.com/2012/12/21/ray-tracer-part-5-depth-of-field/

Lens radius can be found by using **Focus distance** and **Defocus angle**.





```
# compute defocus parameters.
defocus_radius = self.Lens.get_focus_dist() * math.tan(math.radians(self.Lens.get_defocus_angle() * 0.5))
self.defocus_disk_u = self.camera_frame_u * defocus_radius
self.defocus_disk_v = self.camera_frame_v * defocus_radius
```



```
def get_ray(self, i, j):
    pixel_center = self.pixel00_location + (self.pixel_du*i) + (self.pixel_dv*j)
    pixel_sample = pixel_center + self.random_pixel_in_square(self.pixel_du, self.pixel_dv)

    ray_origin = self.center
    if self.Lens.get_defocus_angle() > 1e-06:
        ray_origin = self.defocus_disk_sample()
    ray_direction = pixel_sample - ray_origin
```

Add DoF in our code

Motion Blur

- "Motion blur is an effect that manifests as **a visible streak** generated by **the movement of an object in front of a recording device**. It is the result of combining apparent motion in the scene and an imaging media that integrates light **during a finite exposure time**. This relative motion can be **produced from object movement and camera movement** and can be observed both in still pictures and image sequences. In general, sequences containing a moderate amount of motion blur are perceived as natural, whereas its total absence produces jerky and strobing movement"

https://graphics.unizar.es/papers/Navarro_motionblur.pdf

Origin of the phenomenon

When taking a picture, scene radiance is captured by a camera. Additionally, intrinsic parameters of the capturing device influence to natural effects such as motion blur.

Radiance information of an image

$$I(\omega) = \int_{\Delta T} f(\omega, t)\, L(\omega, t)\, \mathrm{d}t$$

Time is one of the most important parameters.

Radiance

Relevant parameters such as optics, shutter, aperture and film.

Formalized equation

$$I_{xy} = \sum_{l} \int_{\Omega} \int_{\Delta T} r(\omega, t) g_l(\omega, t) L_l(\omega, t)\, \mathrm{d}t\, \mathrm{d}\omega.$$

Spatio-temporal relationship

Geometric term (Occlusion term, Visibility term)

$$r(\omega, t) = r_s(\omega)\, r_t(t)$$

Temporal term

Spatial term

Time interval

[Time Interval Ray Tracing for Motion Blur (TVCG 2017) (youtube.com)]

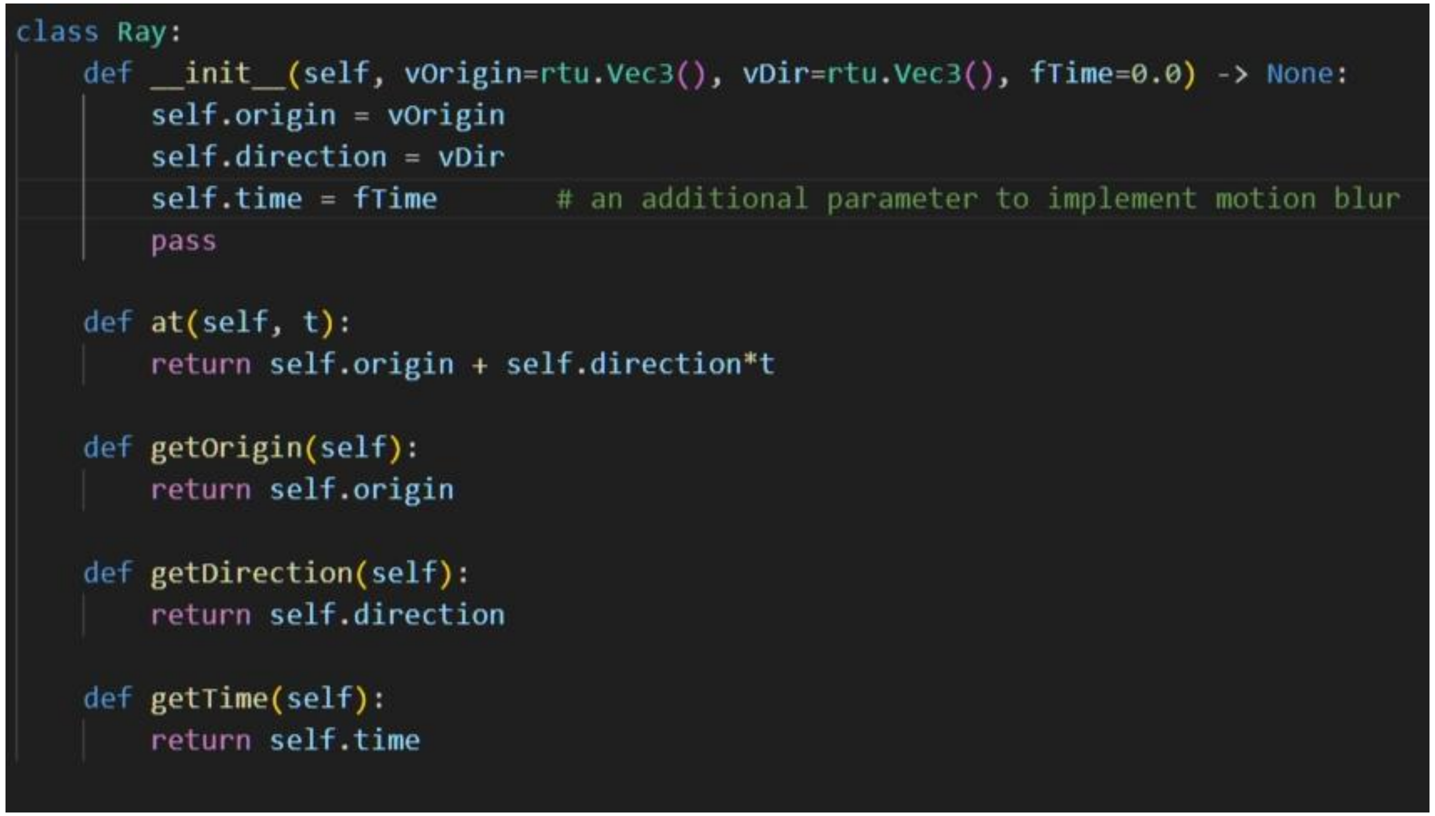
A simplified Space-Time ray tracing for motion blur

An object is allowed to be moved.

A ray has a time parameter.

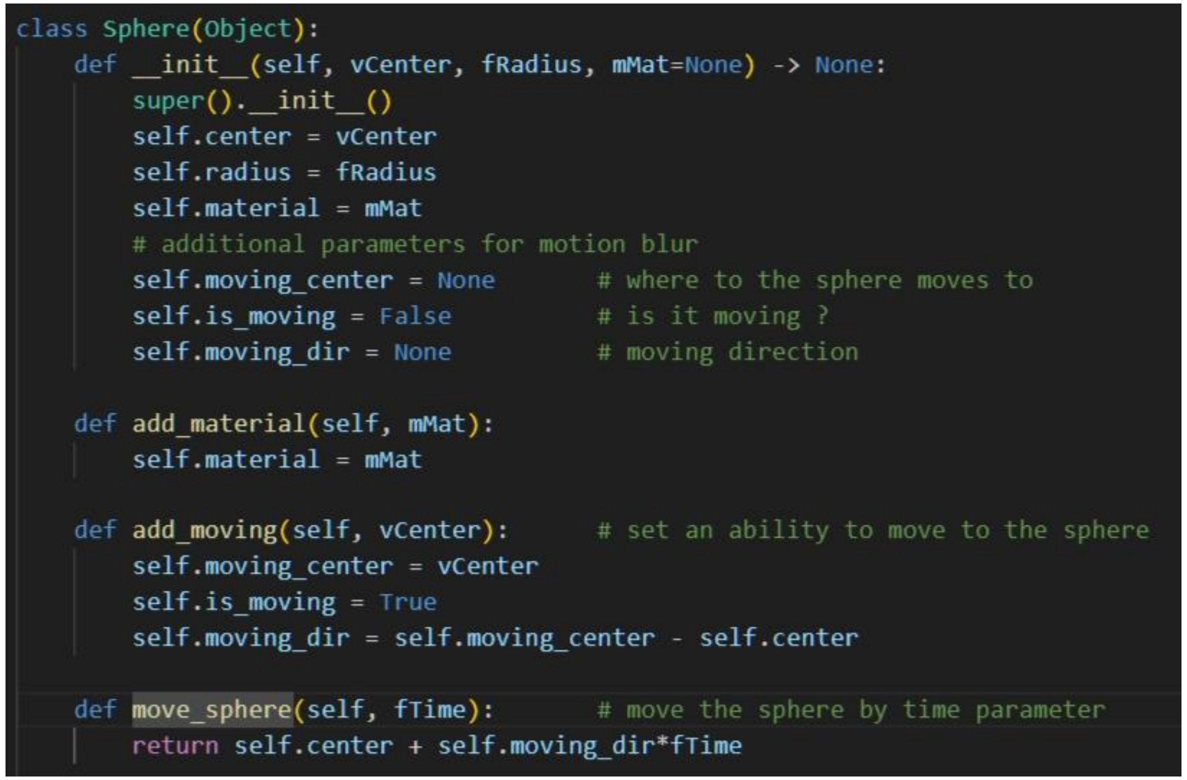
Key idea : A random time sample captures a moving object independently.

Updating Ray class

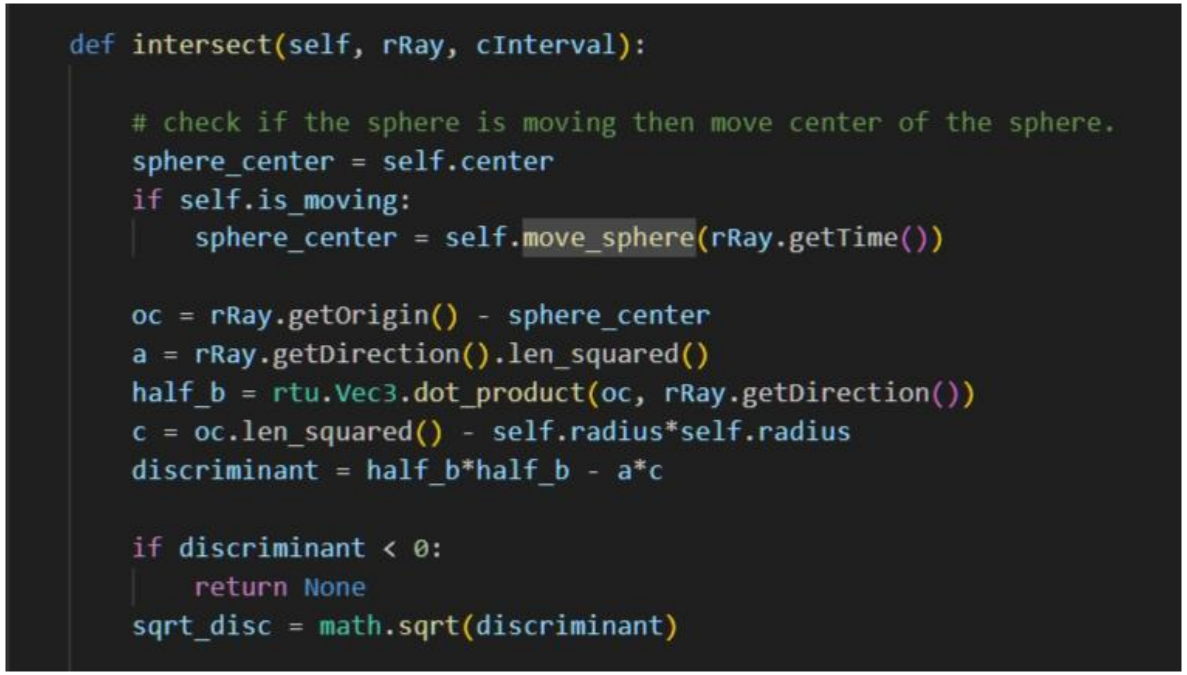


```python
class Ray:
    def __init__(self, vOrigin=rtu.Vec3(), vDir=rtu.Vec3(), fTime=0.0) -> None:
        self.origin = vOrigin
        self.direction = vDir
        self.time = fTime       # an additional parameter to implement motion blur
        pass

    def at(self, t):
        return self.origin + self.direction*t

    def getOrigin(self):
        return self.origin

    def getDirection(self):
        return self.direction

    def getTime(self):
        return self.time
```

Updating an object class



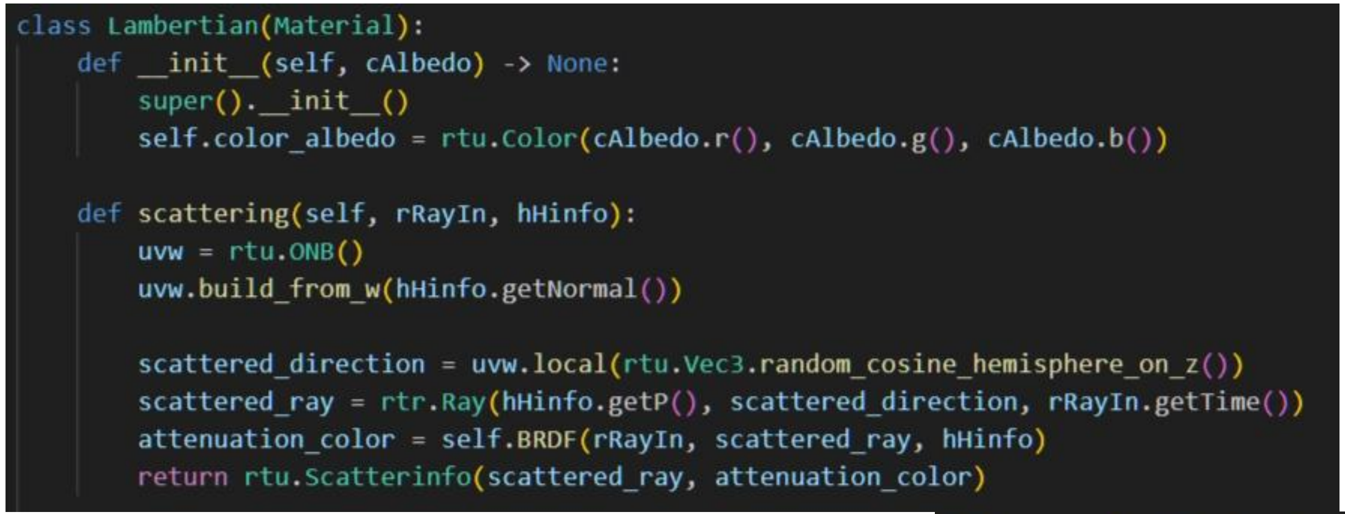
```python
class Sphere(Object):
    def __init__(self, vCenter, fRadius, mMat=None) -> None:
        super().__init__()
        self.center = vCenter
        self.radius = fRadius
        self.material = mMat
        # additional parameters for motion blur
        self.moving_center = None       # where to the sphere moves to
        self.is_moving = False          # is it moving ?
        self.moving_dir = None          # moving direction

    def add_material(self, mMat):
        self.material = mMat

    def add_moving(self, vCenter):      # set an ability to move to the sphere
        self.moving_center = vCenter
        self.is_moving = True
        self.moving_dir = self.moving_center - self.center

    def move_sphere(self, fTime):       # move the sphere by time parameter
        return self.center + self.moving_dir*fTime
```



```python
    def intersect(self, rRay, cInterval):

        # check if the sphere is moving then move center of the sphere.
        sphere_center = self.center
        if self.is_moving:
            sphere_center = self.move_sphere(rRay.getTime())

        oc = rRay.getOrigin() - sphere_center
        a = rRay.getDirection().len_squared()
        half_b = rtu.Vec3.dot_product(oc, rRay.getDirection())
        c = oc.len_squared() - self.radius*self.radius
        discriminant = half_b*half_b - a*c

        if discriminant < 0:
            return None
        sqrt_disc = math.sqrt(discriminant)
```

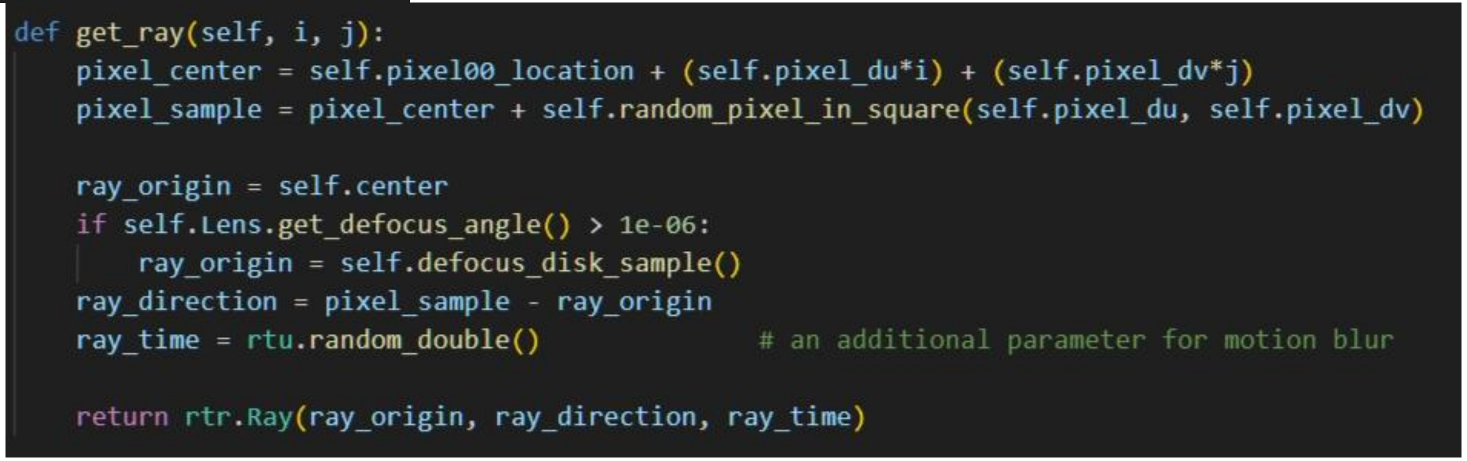
Updating scattering behavior !

A material class



```python
class Lambertian(Material):
    def __init__(self, cAlbedo) -> None:
        super().__init__()
        self.color_albedo = rtu.Color(cAlbedo.r(), cAlbedo.g(), cAlbedo.b())

    def scattering(self, rRayIn, hHinfo):
        uvw = rtu.ONB()
        uvw.build_from_w(hHinfo.getNormal())

        scattered_direction = uvw.local(rtu.Vec3.random_cosine_hemisphere_on_z())
        scattered_ray = rtr.Ray(hHinfo.getP(), scattered_direction, rRayIn.getTime())
        attenuation_color = self.BRDF(rRayIn, scattered_ray, hHinfo)
        return rtu.Scatterinfo(scattered_ray, attenuation_color)
```

The camera class



```python
def get_ray(self, i, j):
    pixel_center = self.pixel00_location + (self.pixel_du*i) + (self.pixel_dv*j)
    pixel_sample = pixel_center + self.random_pixel_in_square(self.pixel_du, self.pixel_dv)

    ray_origin = self.center
    if self.Lens.get_defocus_angle() > 1e-06:
        ray_origin = self.defocus_disk_sample()
    ray_direction = pixel_sample - ray_origin
    ray_time = rtu.random_double()          # an additional parameter for motion blur

    return rtr.Ray(ray_origin, ray_direction, ray_time)
```

Codes and class assignment !

- Github : RT-python-week10
  - https://github.com/KUGA-01418283-Raytracing/RT-python-week10