Tanaboon Tongbuasirilai
Dept. Computer Science
Kasetsart University

Week 5
Lighting and shadows

Lighting and shadows

- What happens when the background color is changed ?

- It does not affect to the material reflection (color).
- In reality, all the patches must be dark if no additional light in the scene.
- However, as of now, our implementation does not include how light should interact with objects in the scene.

Background color as a light source

- When the shot ray hits an object, the ray is scattered according to the material properties.
- The ray is recursively scattered in the scene.

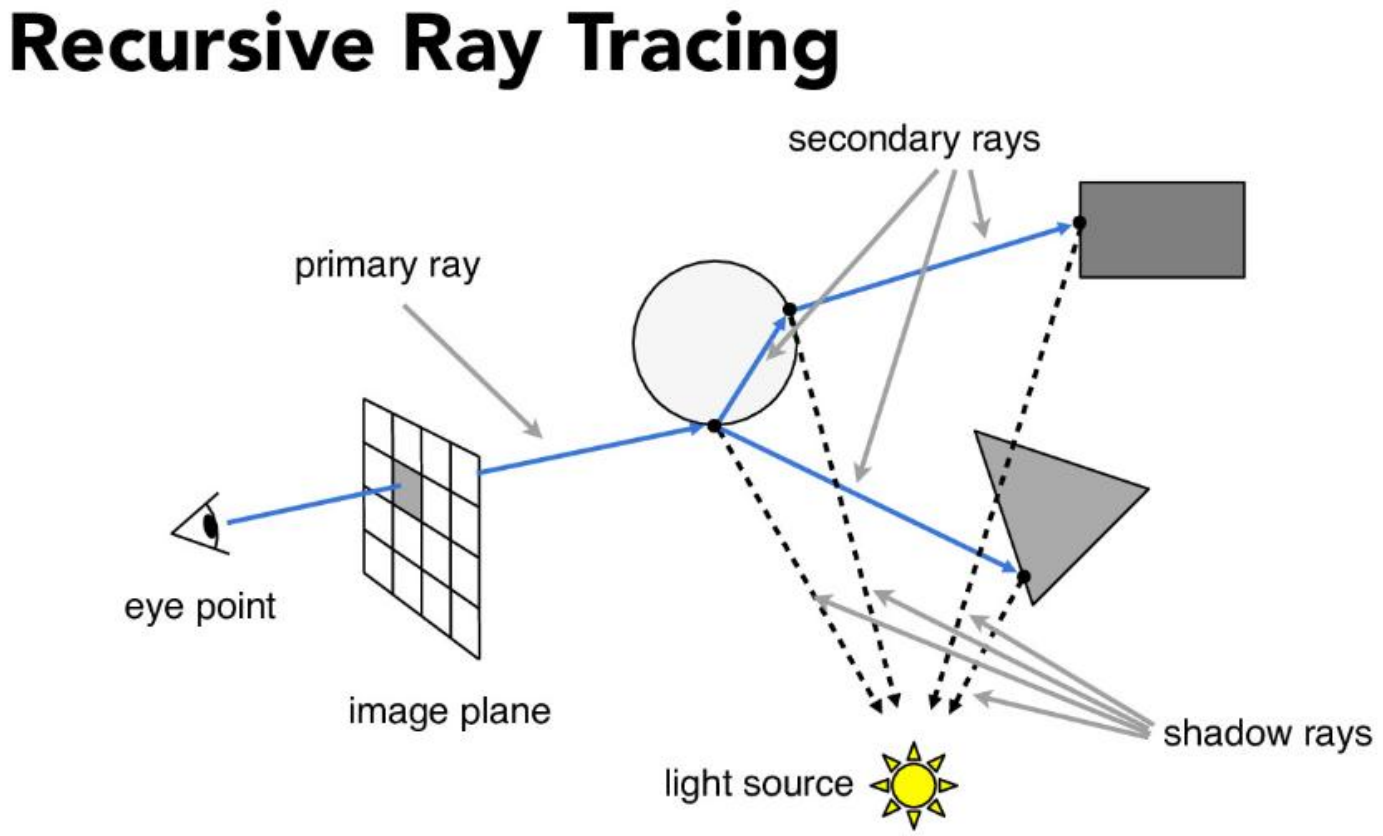


Image from cs184.eecs.berkeley.edu/sp21/lecture/9-14/ray-tracing

The current version describes a color when a ray hits a material, otherwise it returns background color.

```python
def compute_scattering(self, rGen_ray, scene):
    # if the generated ray hits an object
    found_hit = scene.find_intersection(rGen_ray, rtu.Interval(0.000001, rtu.infinity_number))
    if found_hit == True:
        # get the hit info
        hinfo = scene.getHitList()
        # get the material of the object
        hmat = hinfo.getMaterial()
        # return the color
        return hmat.color_albedo

    return self.background_color(rGen_ray)
```

Implement recursive call of compute_scattering() method

An updated version adds a recursive call of computer_scattering() terminated when maxDepth <= 0.

```python
def compute_scattering(self, rGen_ray, scene, maxDepth):
    if maxDepth <= 0:
        return rtu.Color()

    # if the generated ray hits an object
    found_hit = scene.find_intersection(rGen_ray, rtu.Interval(0.000001, rtu.infinity_number))
    if found_hit == True:
        # get the hit info
        hinfo = scene.getHitList()
        # get the material of the object
        hmat = hinfo.getMaterial()
        # compute scattering
        sinfo = hmat.scattering(rGen_ray, hinfo)
        # return the color
        return self.compute_scattering(rtr.Ray(hinfo.getP(), sinfo.scattered_ray.getDirection()), scene, maxDepth-1) * sinfo.attenuation_color

    # previous background color
    # return scene.getSkyBackgroundColor(rGen_ray)
    return scene.getBackgroundColor()
```

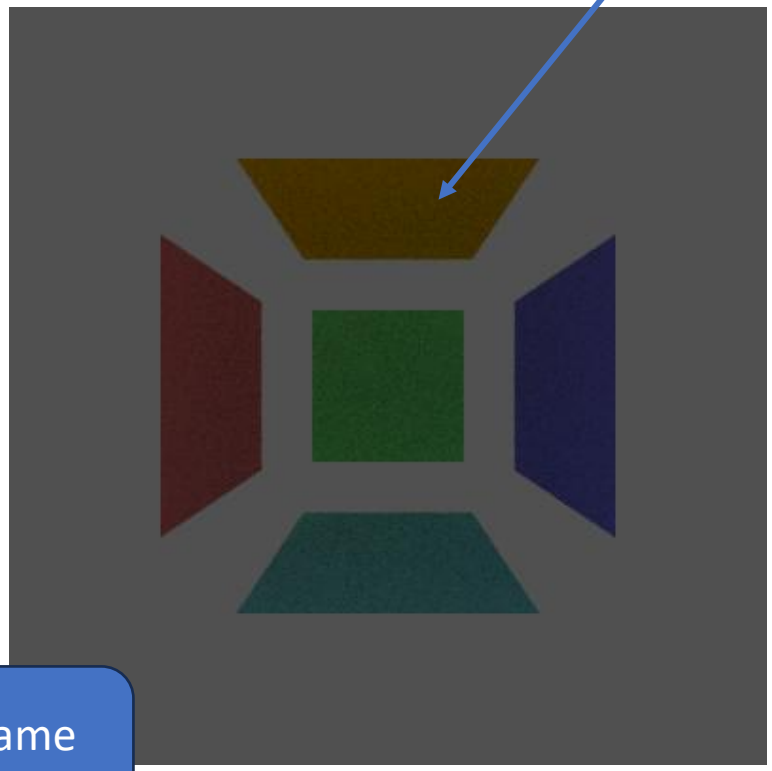
```python
def compute_scattering(self, rGen_ray, scene, maxDepth):
    if maxDepth <= 0:
        return rtu.Color()

    # if the generated ray hits an object
    found_hit = scene.find_intersection(rGen_ray, rtu.Interval(0.000001, rtu.infinity_number))
    if found_hit == True:
        # get the hit info
        hinfo = scene.getHitList()
        # get the material of the object
        hmat = hinfo.getMaterial()
        # compute scattering
        sinfo = hmat.scattering(rGen_ray, hinfo)
        # return the color
        return self.compute_scattering(rtr.Ray(hinfo.getP(), sinfo.scattered_ray.getDirection()), scene, maxDepth-1) * sinfo.attenuation_color

    # previous background color
    # return scene.getSkyBackgroundColor(rGen_ray)
    return scene.getBackgroundColor()
```

This additional call provides scattering information for later use.

The method recursively calls itself with decreasing maxDepth by one.
The color return is a weighted attenuation color.

Comparison

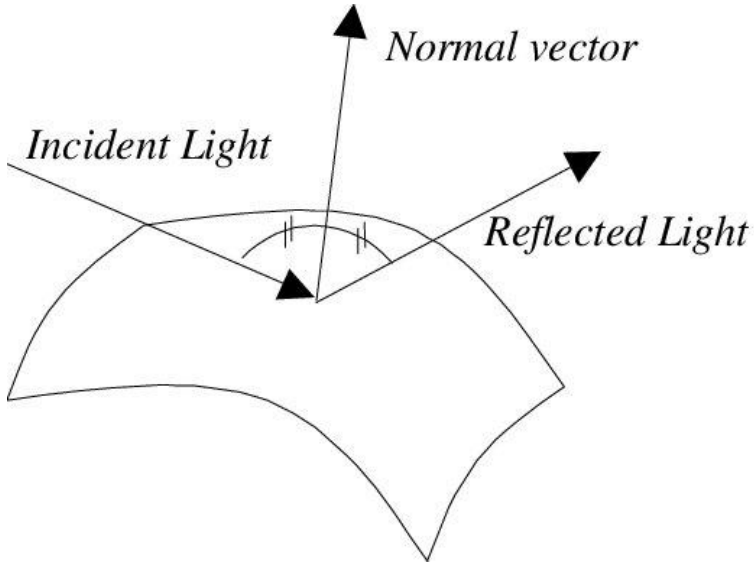
Observe the noises in the rendered image. Why the rendered image is noisy ?



Are both images using the same background color ?

How are rays scattered on (interacted with) objects (materials) ?

- Geometric optics describe how light behaves when hits a material surface. The primary directions involve in this situation are incident light direction, reflected light direction and normal vector at a hitting point.



- BRDF (Bidirectional Reflectance Distribution Function) explains reflected portion of light quantity given a pair of incident and reflected light directions.

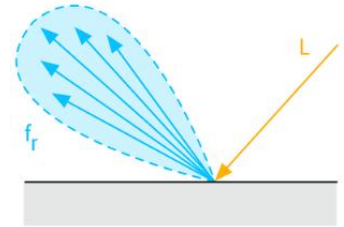


Image from https://google.github.io/filament/Filament.md.html and Jordan, Fred. (1999). View-dependent scalable coding of landscapes.

Material class (previous week)

```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):
        return None
```

Our Lambertian material has no information about scattering.

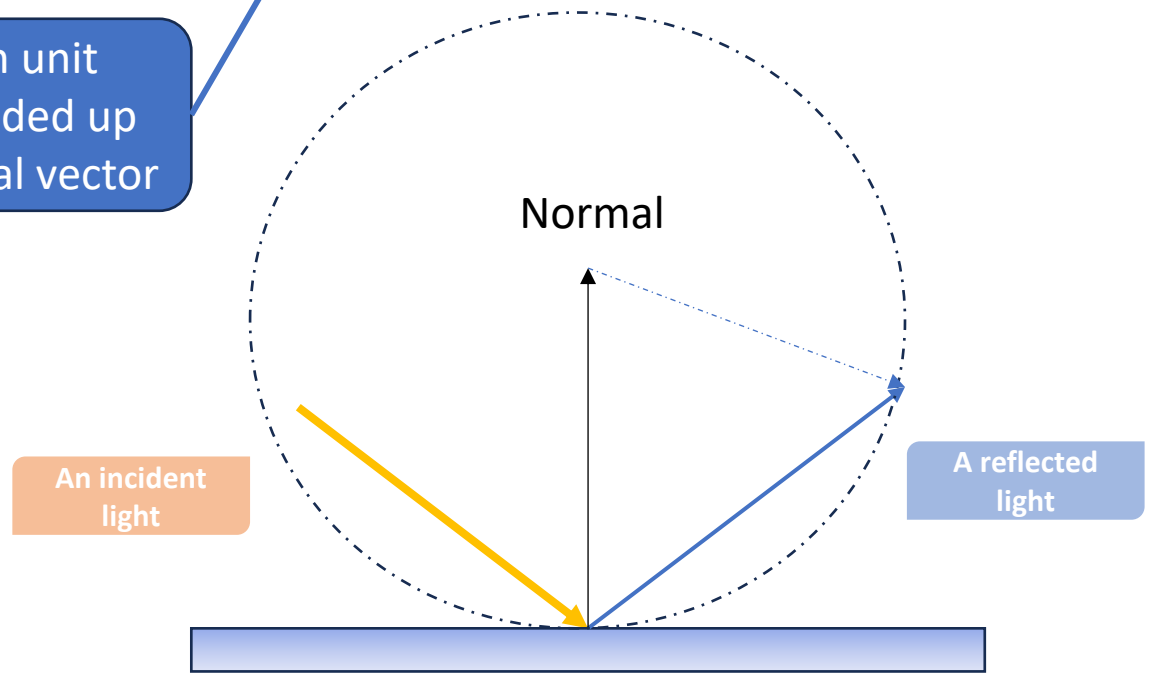
```python
class Integrator():
    def __init__(self) -> None:
        pass

    def compute_scattering(self, rGen_ray, scene):
        # if the generated ray hits an object
        found_hit = scene.find_intersection(rGen_ray, rtu.Interval(0.000001, rtu.infinity_number))
        if found_hit == True:
            # get the hit info
            hinfo = scene.getHitList()
            # get the material of the object
            hmat = hinfo.getMaterial()
            # return the color
            return hmat.color_albedo

        return self.background_color(rGen_ray)
```

Our Integrator return a color albedo of a material. That is why the rendered result looks dull.

Updated material class

```
def scattering(self, rRayIn, hHinfo):
    scattered_direction = hHinfo.getNormal() + rtu.Vec3.random_vec3_unit()
    if scattered_direction.near_zero():
        scattered_direction = hHinfo.getNormal()

    scattered_ray = rtr.Ray(hHinfo.getP(), scattered_direction)
    attenuation_color = rtu.Color(self.color_albedo.r(), self.color_albedo.g(), self.color_albedo.b())
    return rtu.Scatterinfo(scattered_ray, attenuation_color)
```

A random unit vector is added up to the normal vector



- When a Labertian material is implemented, we have a clue that the light can be scattered equally in all directions from the point of incidence.
- One way we can simulate this light behavior is to sample a reflected light around the normal vector. This ensures that the sampled reflected direction will not occur below the surface point.

Light class

- An object that can emit light (L_e) is a source of light.
- A design choice for light source is a special kind of material.
- The Light class is derived from the Material class.
- Attributes
  - Light color
- Methods
  - Is this material a light source ?
  - Light emitting

A simple light source

Light is a derived class from Material.
Diffuse_light is a derived class from Light.
Additional types of light source should be derived from Light.

```python
class Light(rtm.Material):
    def __init__(self) -> None:
        pass

    def scattering(self, rRayIn, hHinfo):
        return None

    def emitting(self):
        return rtu.Color(0,0,0)

    def is_light(self):
        return True
```

Check if this material is light.

```python
class Diffuse_light(Light):
    def __init__(self, cAlbedo) -> None:
        super().__init__()
        self.light_color = cAlbedo

    def scattering(self, rRayIn, hHinfo):
        return None

    def emitting(self):
        return self.light_color
```

It tells the integrator that this is a light source and no scattering needed.

Updating the integrator

```python
# if the generated ray hits an object
found_hit = scene.find_intersection(rGen_ray, rtu.Interval(0.000001, rtu.infinity_number))
if found_hit == True:
    # get the hit info
    hinfo = scene.getHitList()
    # get the material of the object
    hmat = hinfo.getMaterial()
    # compute scattering
    sinfo = hmat.scattering(rGen_ray, hinfo)
    # if no scattering (It is a light source)
    if sinfo is None:
        # return Le
        return hmat.emitting()
    # return the color
    return self.compute_scattering(rtr.Ray(hinfo.getP(), sinfo.scattered_ray.getDirection()), scene, maxDepth-1) * sinfo.atten
```

Add L_e when the ray hits a light source.

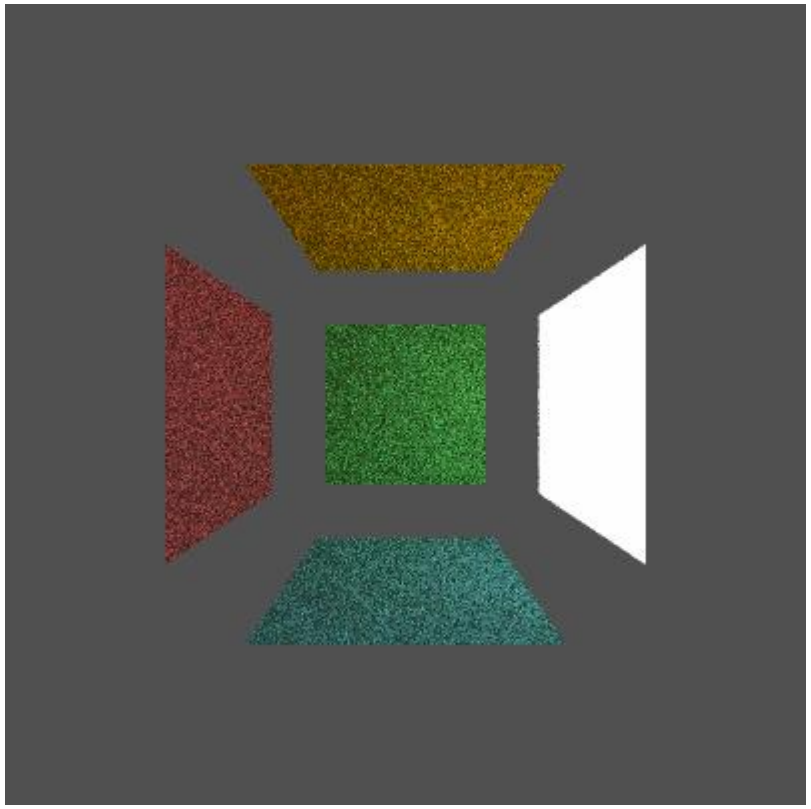
The updated integrator can now return either L_r (reflected radiance) or L_e (emitted radiance). This is useful for implementing a solver for the rendering equation.

A useful method for future

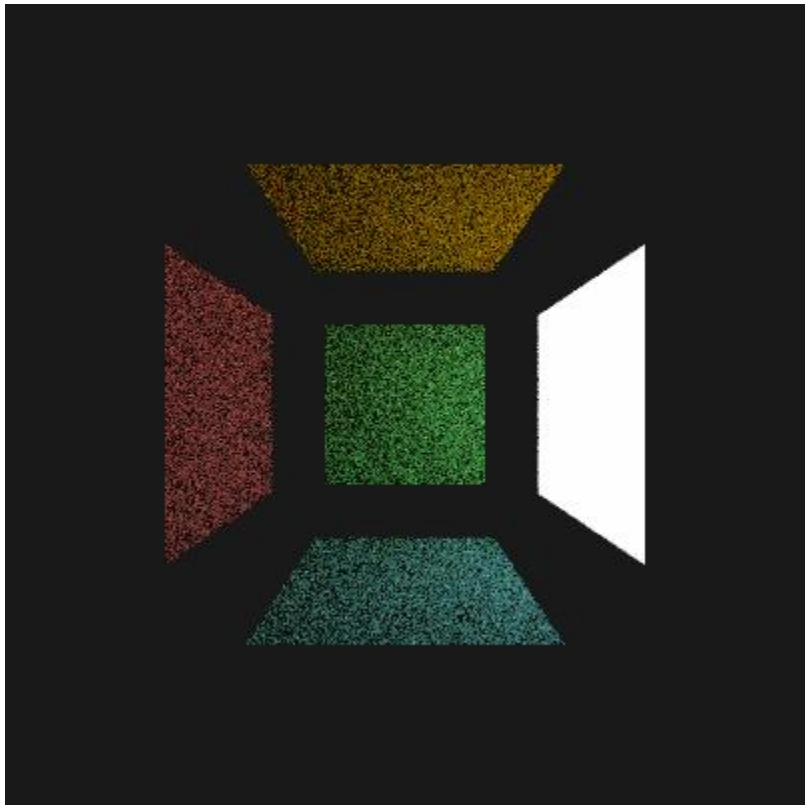
- Update the Scene class
  - Find light objects in the scene.

```
def find_lights(self):
    np_obj_list = np.array(self.obj_list)
    for obj in np_obj_list:
        if obj.material.is_light():
            self.light_list.append(obj)
```

Now we can turn a quad into a light source.

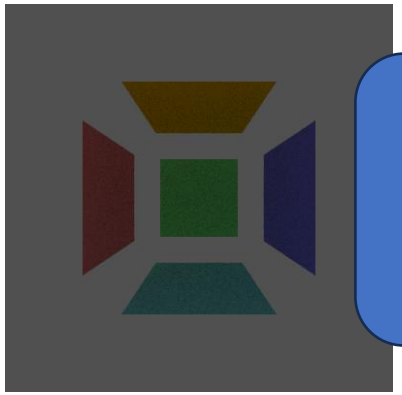


10 spp, 8 max depth



Background color = grey

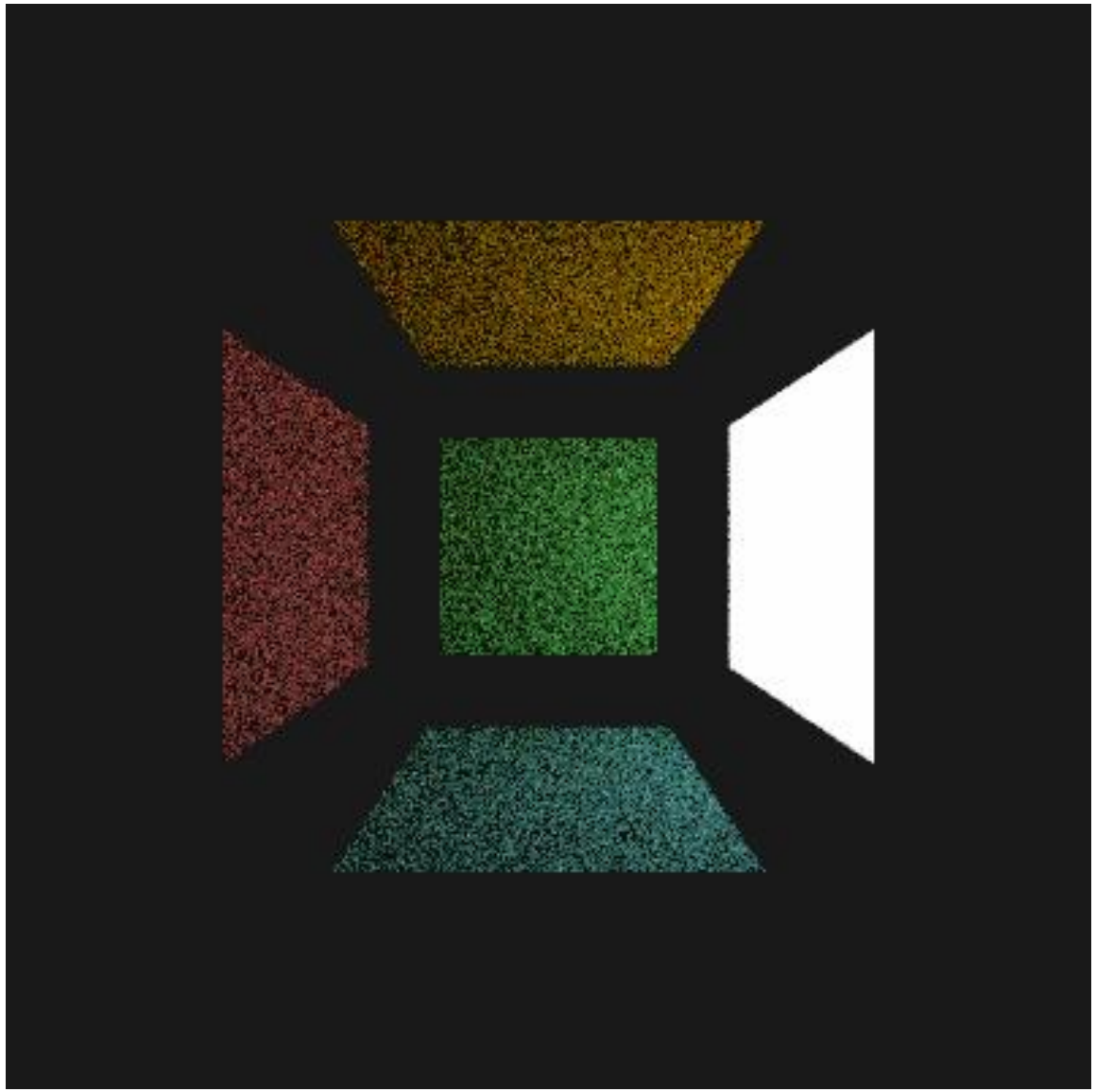
Background color = black



This image demonstrates indirect illumination when background is a kind of light source.

Why we see this Lighting effect.

We are approaching to solving the light transport problem. Solving the problem requires an indirect illumination solver.



Current technology (RTXDI)

- NVIDIA RTXDI | NVIDIA Developer
- https://developer.nvidia.com/blog/render-millions-of-direct-lights-in-real-time-with-rtx-direct-illumination-rtxdi/
- https://developer.nvidia.com/blog/turning-up-the-lights-interactive-path-tracing-scenes-from-a-short-film/
- RTX Direct Illumination (RTXDI)
  - Imagine adding millions of dynamic lights to your game environments without worrying about performance or resource constraints. NVIDIA RTX™ Direct Illumination (RTXDI) figures out the most important light samples in a scene and renders them physically accurate. Geometry of any shape can emit light, cast appropriate shadows, and move freely and dynamically.

Direct lighting

- In ray tracing, direct lighting describes the radiance accumulation at the surface point when light can be traced directly without occlusion.
- This can enhance realism by adding L_e to the integrator.
- So far we don't have any L_e term yet. By adding this term, it add more complications to the codes.
- Hence, for simplicity, we implement a point light instead.

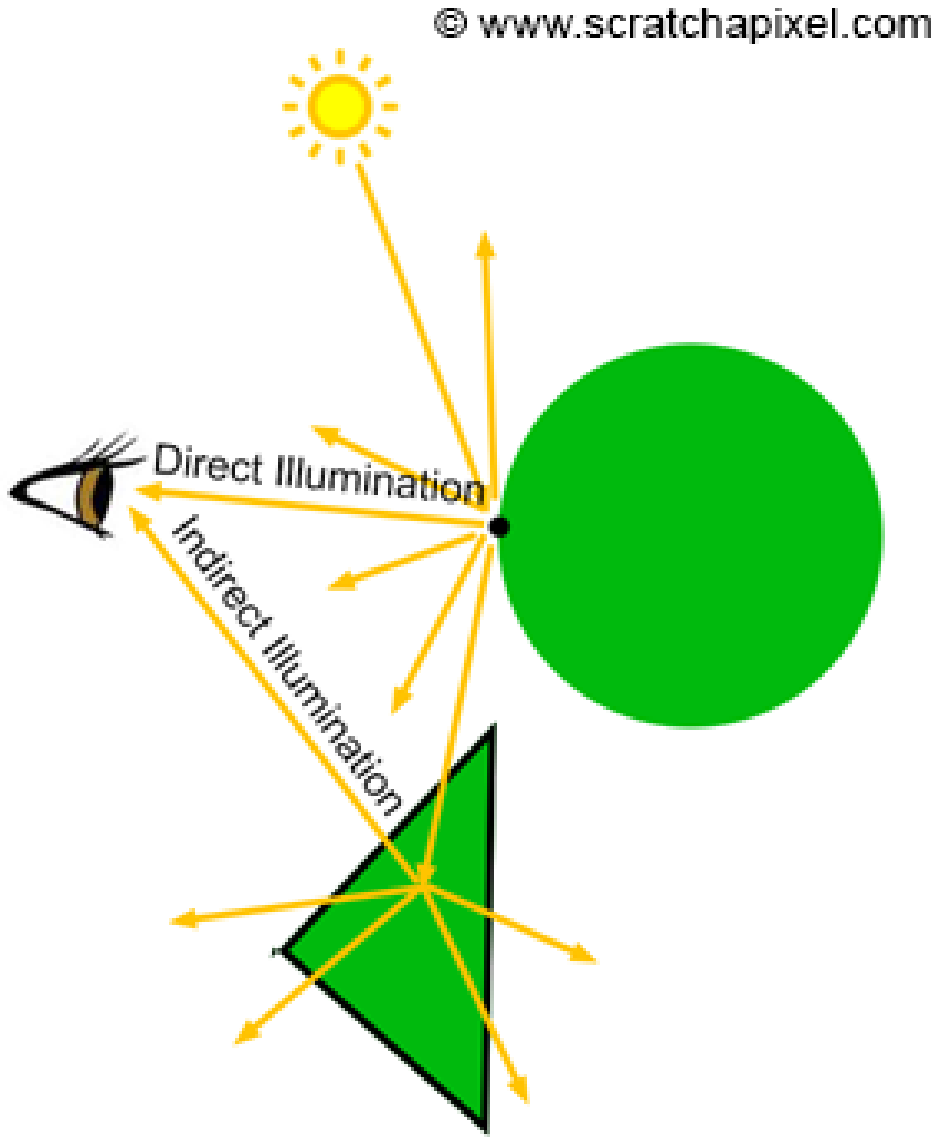
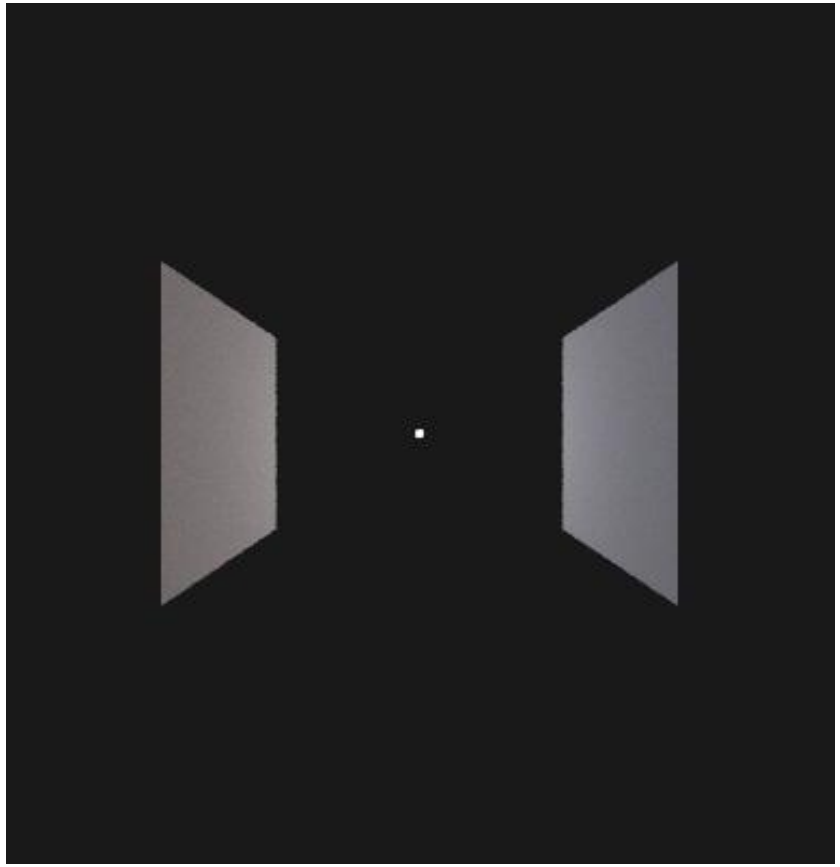


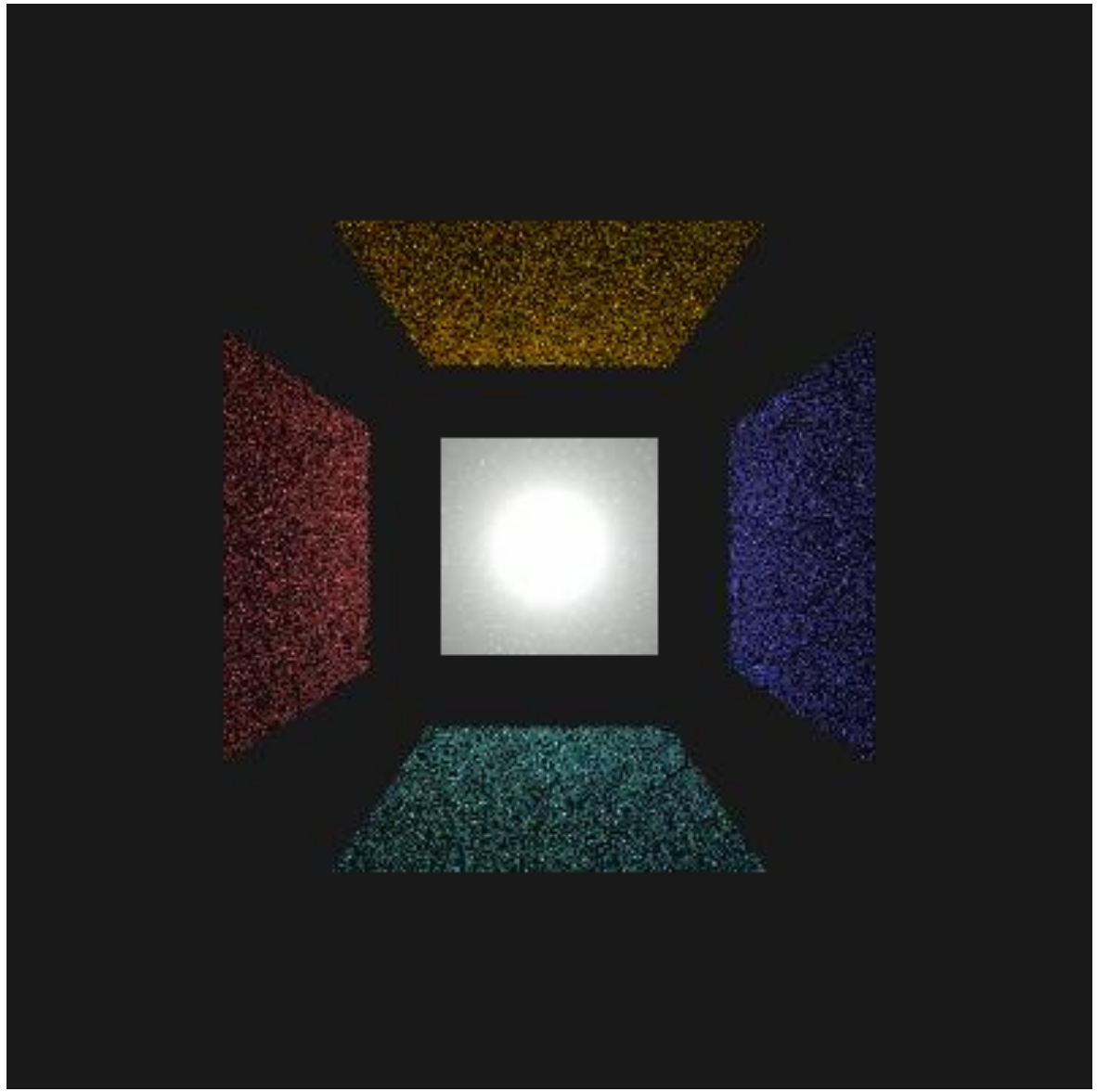
Image from www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-overview/light-transport-ray-tracing-whitted.html

Point light

- An object that emit diffuse light.
- A point light can be defined by a tiny sphere (very small radius).
- Generating a sphere instance and add ‘diffuse_light’ as its material.

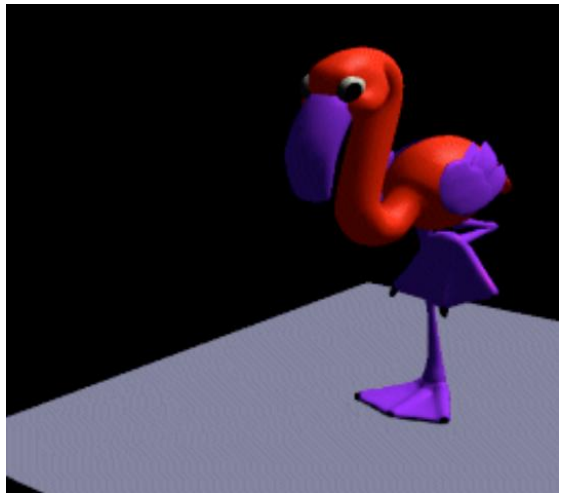


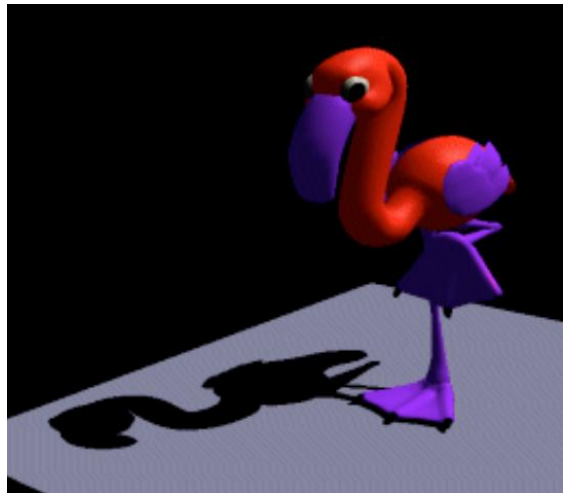
Implement direct lighting by point light



Shadows

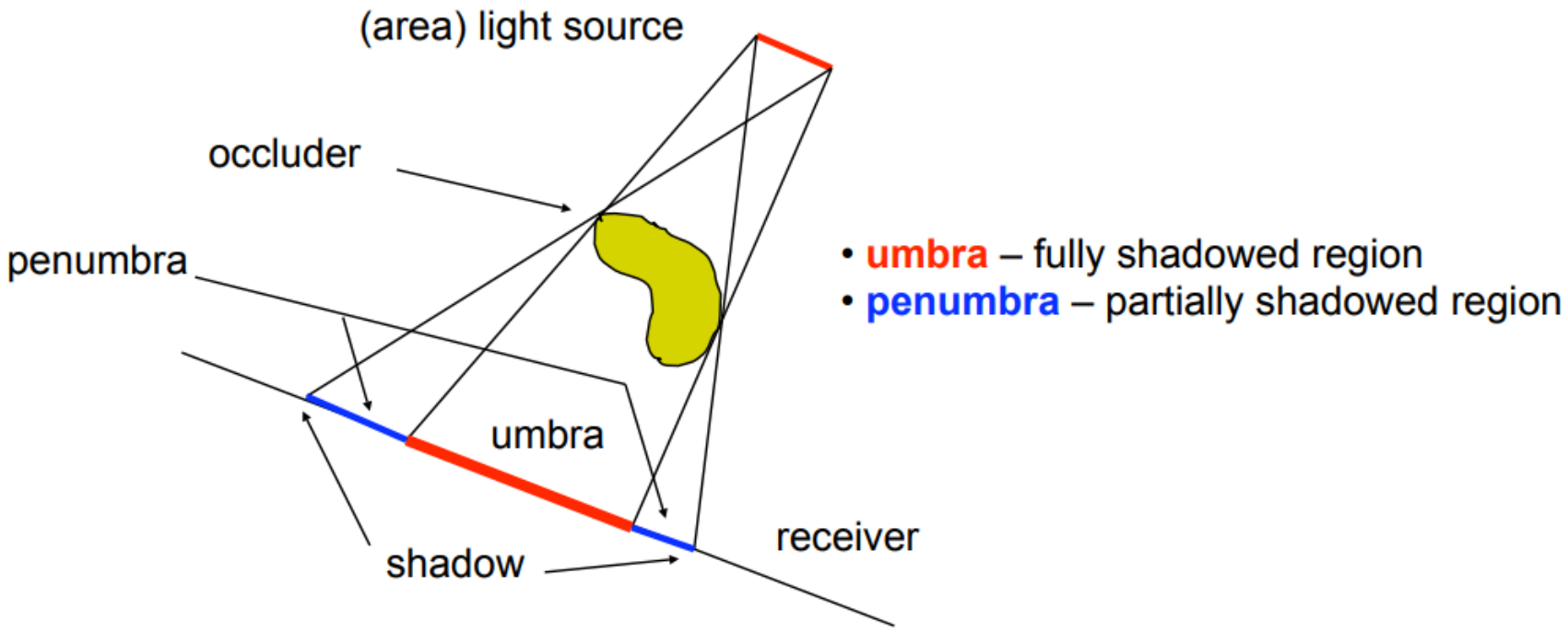
- Why do we need a shadow ?
  - Enhance realism
  - Additional clues for shape and positions of 3D objects





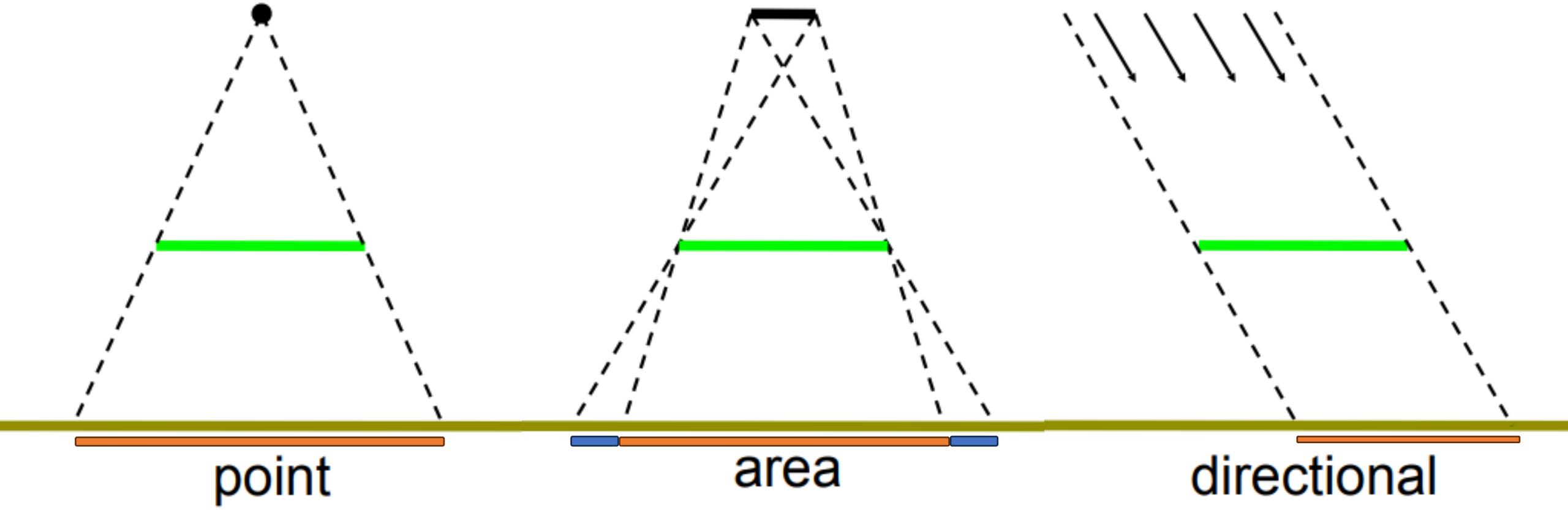
Images from web.cse.ohio-state.edu/~shen.94/681/Site/Slides_files/shadow.pdf

Terminology



- **umbra** – fully shadowed region
- **penumbra** – partially shadowed region

web.cse.ohio-state.edu/~shen.94/681/Site/Slides_files/shadow.pdf



Hard and Soft shadows

From left to right, the illustration demonstrates how each type of light casts a shadow on a surface.

A point light source and directional light can, in most cases, present hard shadows, while an area light source gives hard and soft shadows.
However, this depends on situations.

web.cse.ohio-state.edu/~shen.94/681/Site/Slides_files/shadow.pdf

Call of Duty – Ray traced shadows vs Shadow maps

Shadow maps

Ray traced shadows

Ray traced shadows –

sphere light + denoising + blur

When using shadow maps, there are some noticeable artifacts (the soft shadows at the location between a pole and floor).
Ray traced shadows provide more physically plausible shadows with the trade of performances.
Baking multiple techniques to create realistic shadows is a still a challenging problem for interactive rendering.

www.activision.com/cdn/research/Raytraced_Shadows_in_Call_of_Duty_Modern_Warfare.pdf

Codes and class assignment !

- Github : RT-python-week05
  - https://github.com/KUGA-01418283-Raytracing/RT-python-week05