Ray Tracing in Entertainment Industry

Tanaboon Tongbuasirilai
Dept. Computer Science
Kasetsart University

Week 8
Reflection models and indirect illuminations

Classifications of BRDF models

Empirical BRDF

- Early stage of shader development.
- Mathematically derived the light scattering behaviors.
- Model parameters are mathematical relations.

Analytical BRDF

- Physically based derivation.
- Model parameters are related to the material physical properties.

Data-driven BRDF

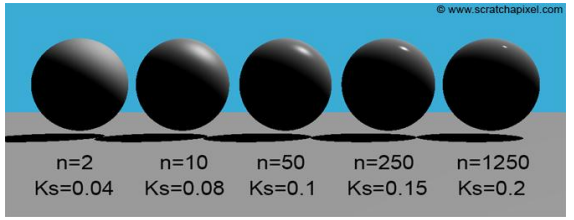
- Measured BRDFs are becoming popular.
- Model parameters are relied on the data-driven methods.

BRDF model – The empirical model

Empirical BRDF model

Phong BRDF model

$$\rho(\omega_i, \omega_o) = \underbrace{k_d \frac{1}{\pi}}_{diffuse} + \underbrace{k_s \frac{n+2}{2\pi}(R \cdot V)^n}_{specular}$$

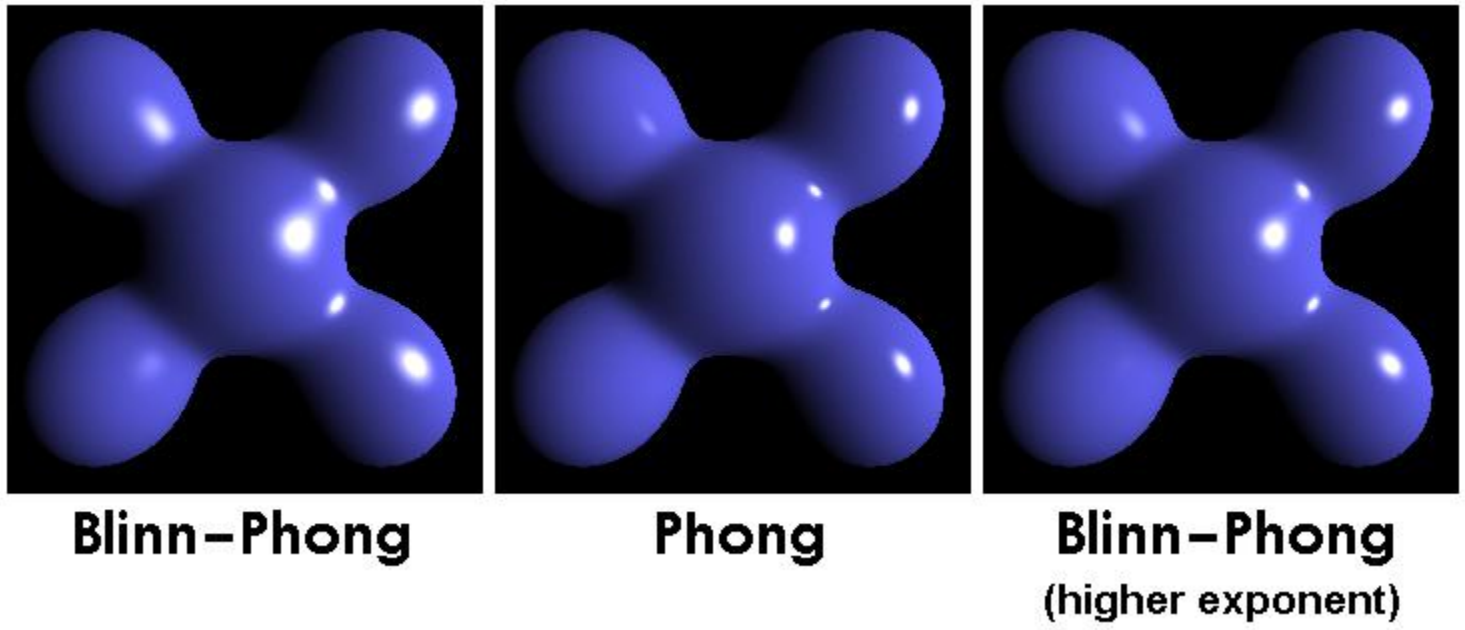


Ward BRDF model

$$\rho(\omega_i, \omega_o) = k_d \frac{1}{\pi} + k_s \frac{1}{\sqrt{cos(\theta_i) cos(\theta_o)}} \cdot \frac{exp\left(-tan^2(\theta_h)\left(\frac{cos^2\theta_h}{\alpha_x^2} + \frac{sin^2\theta_h}{\alpha_y^2}\right)\right)}{4\pi\alpha_x\alpha_y}$$

Empirical BRDFs

- Lambertian
- Phong
- Blinn-Phong
- Ward
- Oren-Nayar



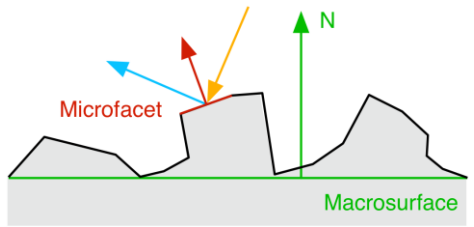
en.wikipedia.org/wiki/Blinn–Phong_reflection_model

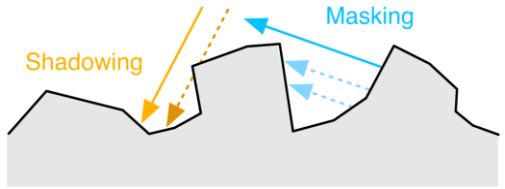
BRDF model – The analytical model

Analytical BRDF model

- The microfacet BRDF model

$$\rho_s(\omega_i, \omega_o) = \frac{D(\omega_h)G(\omega_i, \omega_o)F_r(\omega_o)}{4\ cos(\omega_i)cos(\omega_o)}$$

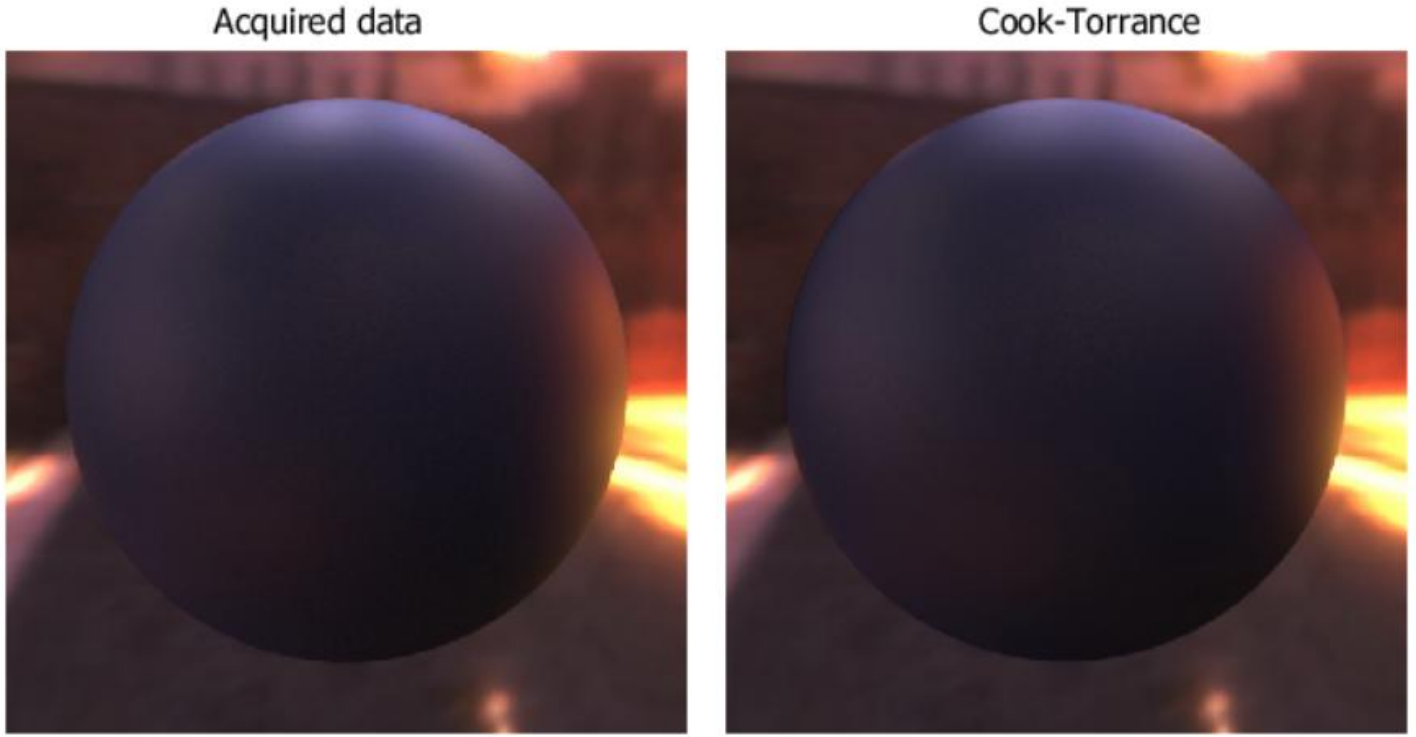




Romain Guy and Mathias Agopian, Physically Based Rendering in Filament, https://google.github.io/filament/Filament.html

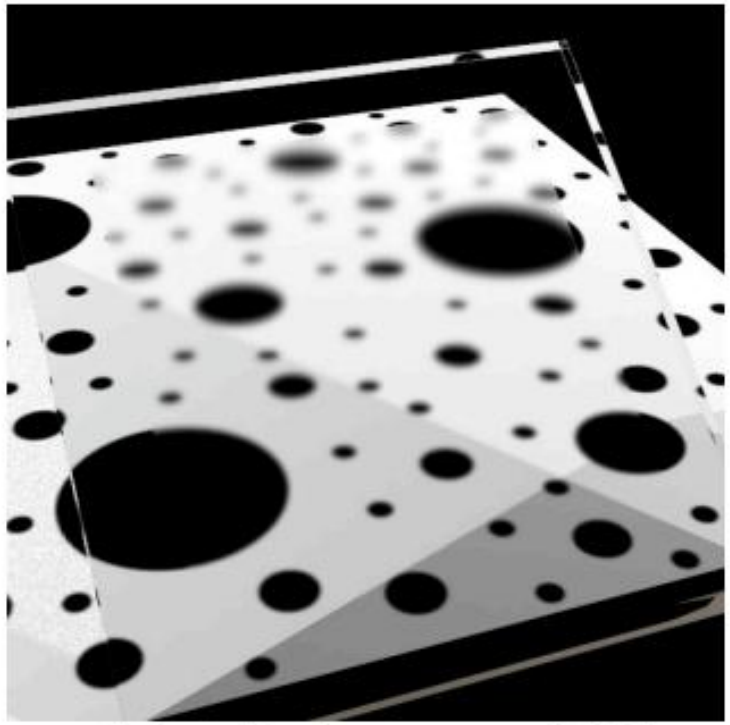
Analytical BRDFs

- Cook-Torrance
- GGX
- ABC model

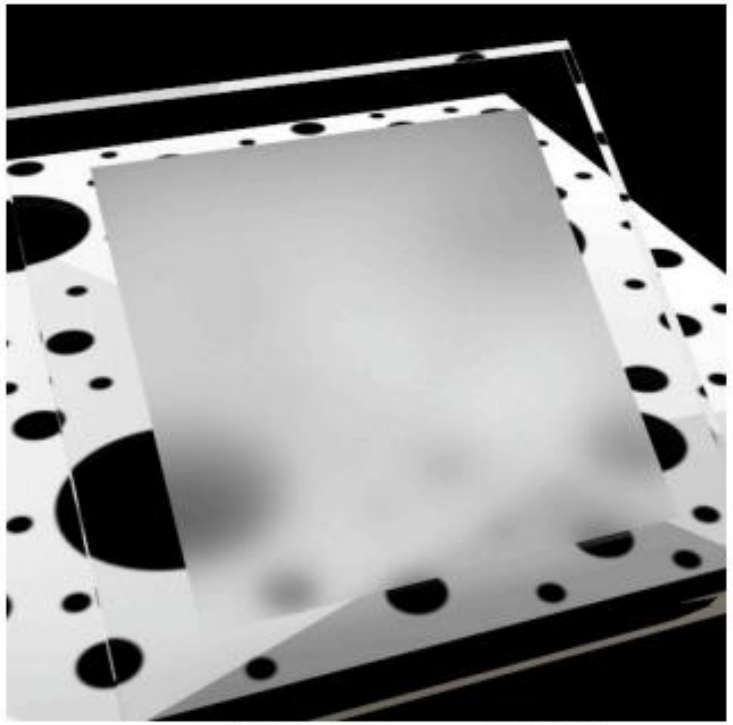


http://people.csail.mit.edu/addy/research/brdf/brdf.ps.web/slide18.html

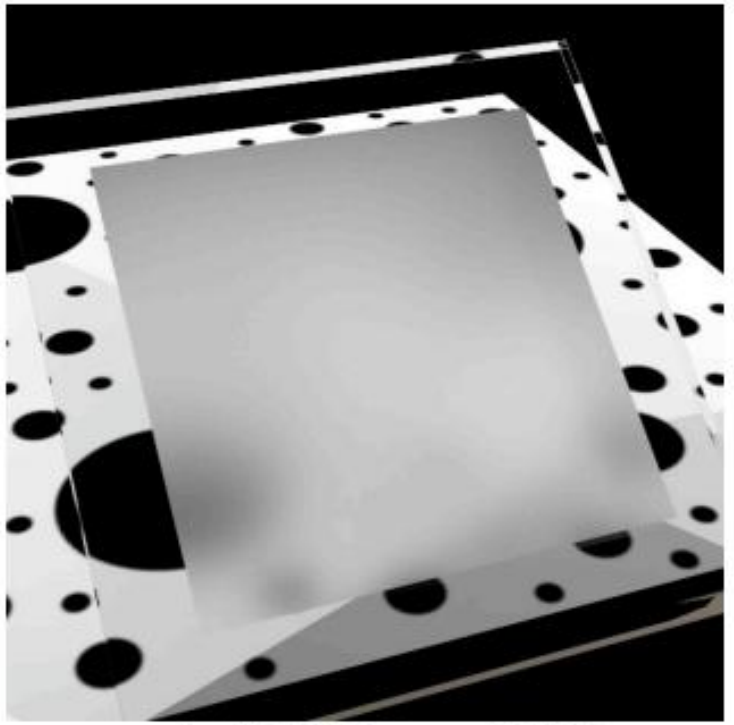
BRDF for refraction



anti-glare (Beckman, $\alpha_b = 0.023$)



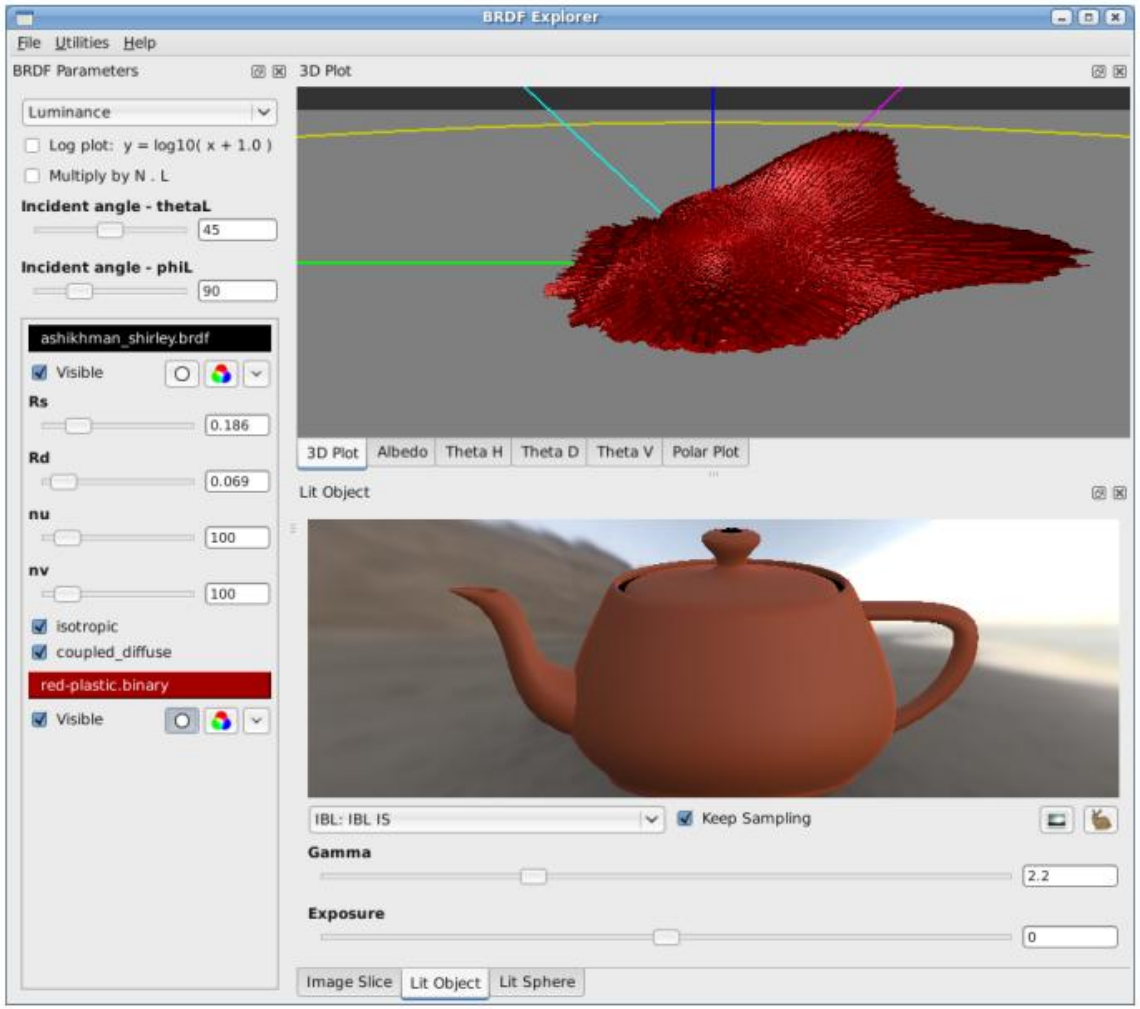
ground (GGX, $\alpha_g = 0.394$)



etched (GGX, $\alpha_g = 0.553$)

cs.cornell.edu/~srm/publications/EGSR07-btdf.pdf

Physically-based shading at Disney

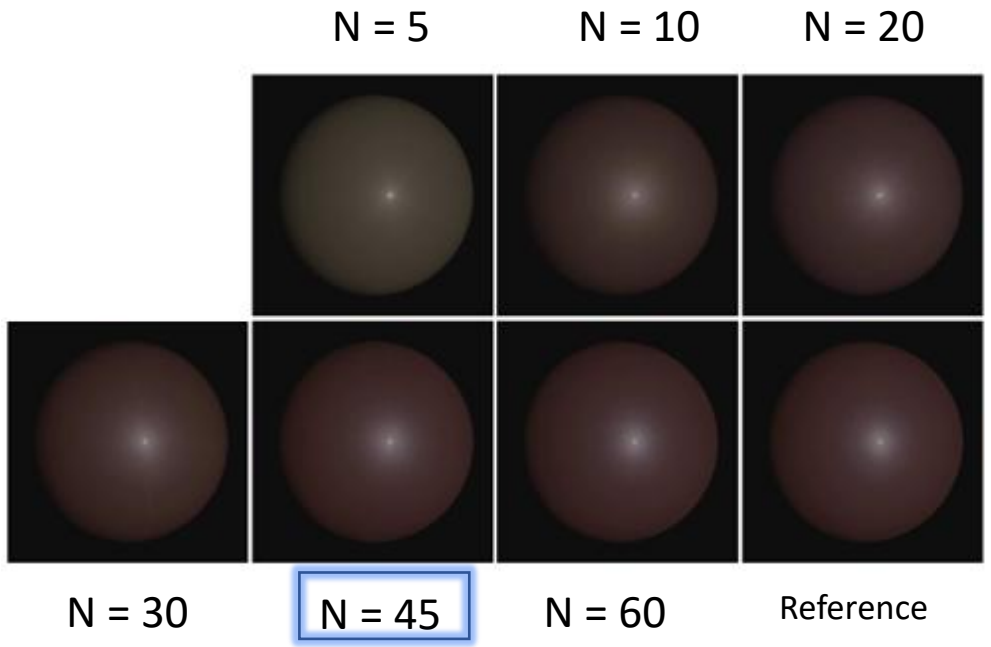


media.disneyanimation.com/uploads/production/publication_asset/48/asset/s2012_pbs_disney_brdf_notes_v3.pdf

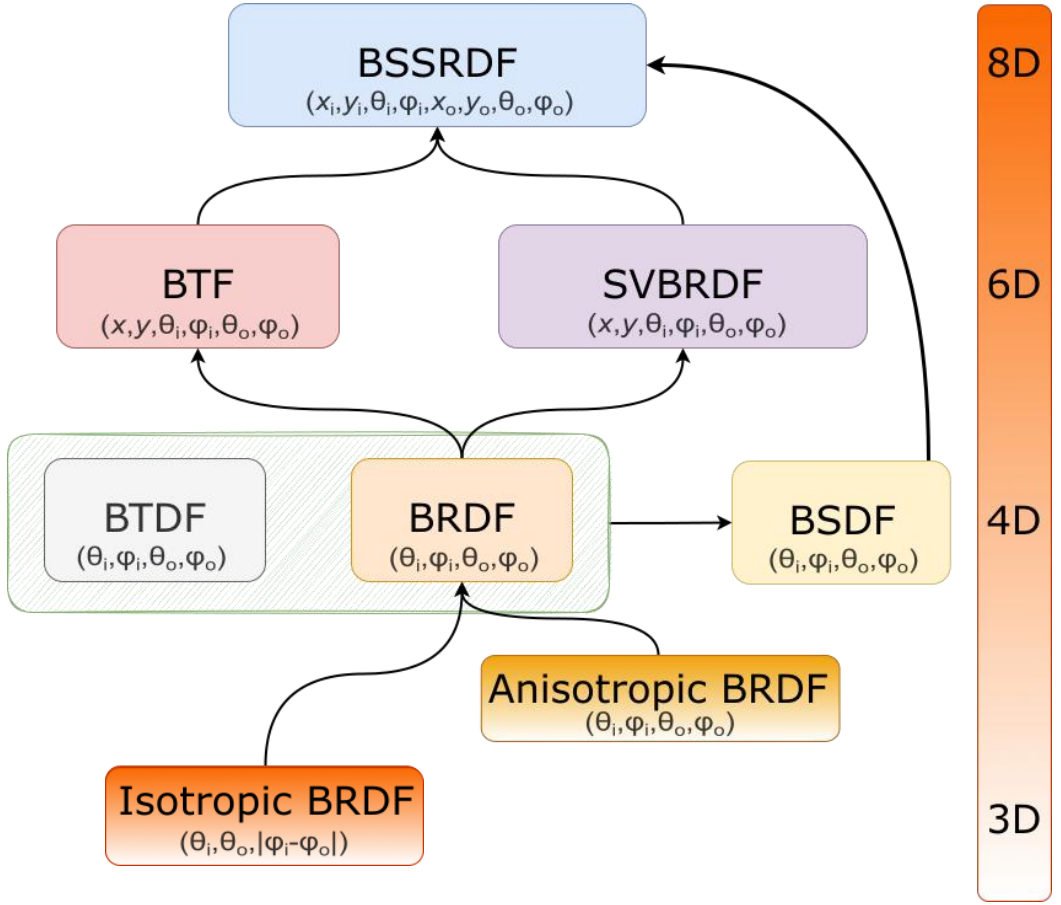
BRDF model – The data-driven model

Data-driven BRDF model

- Principal Component Analysis (PCA)

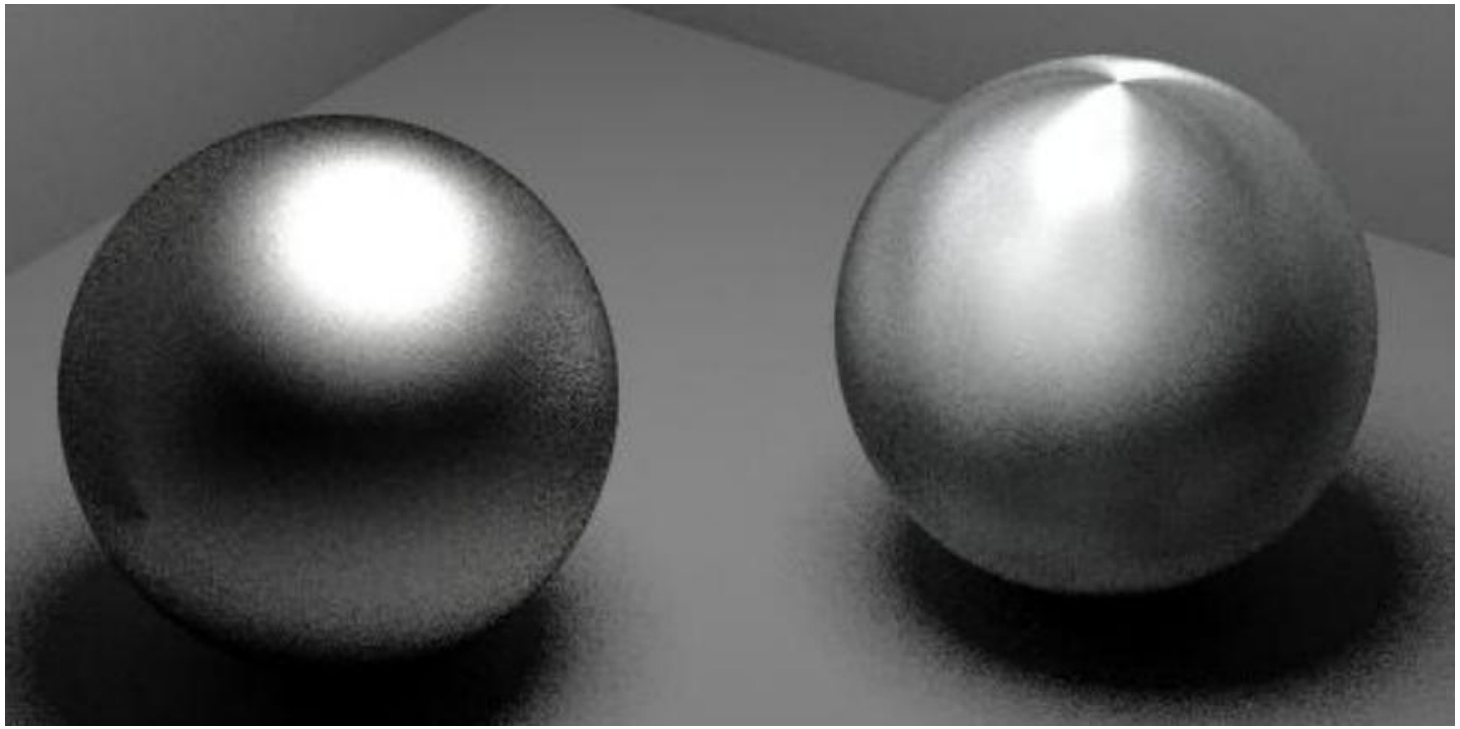


A taxonomy of reflectance functions



Isotropic vs anisotropic BRDFs

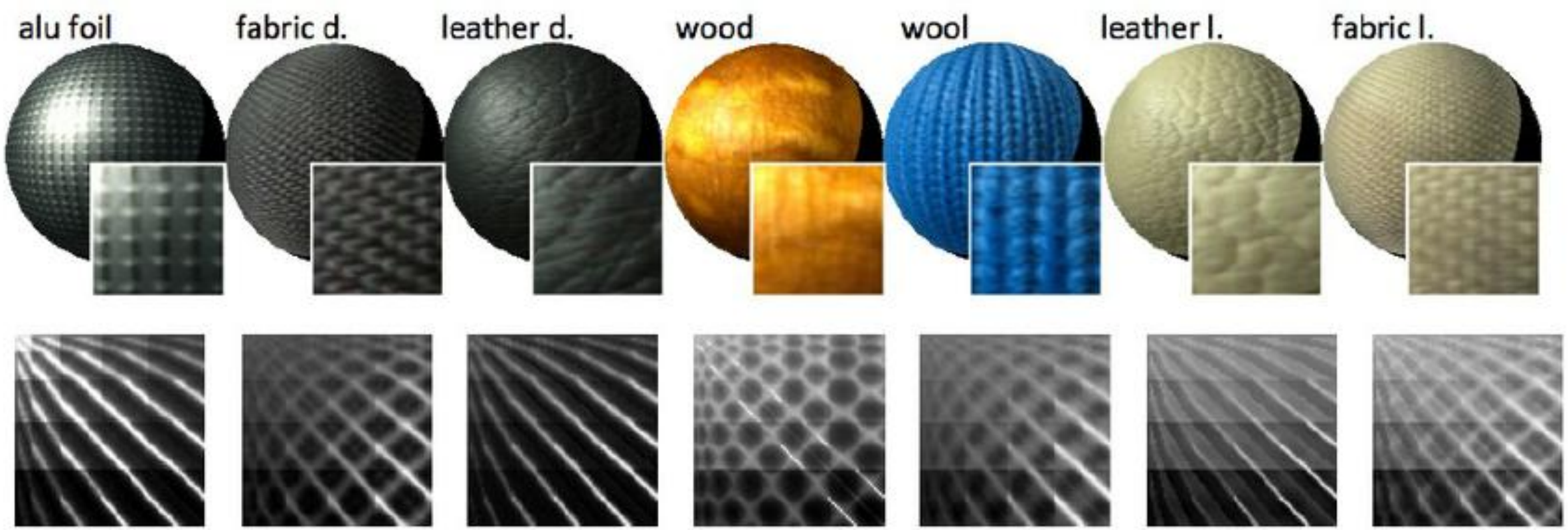
- Isotropic materials show the same properties in all directions.
- Anisotropic materials show different properties in different directions



SVBRDF – Spatially varying BRDF

cg.cs.uni-bonn.de/btf/bonn_svbrdf_database.html

BTF – Bidirectional Texture Function

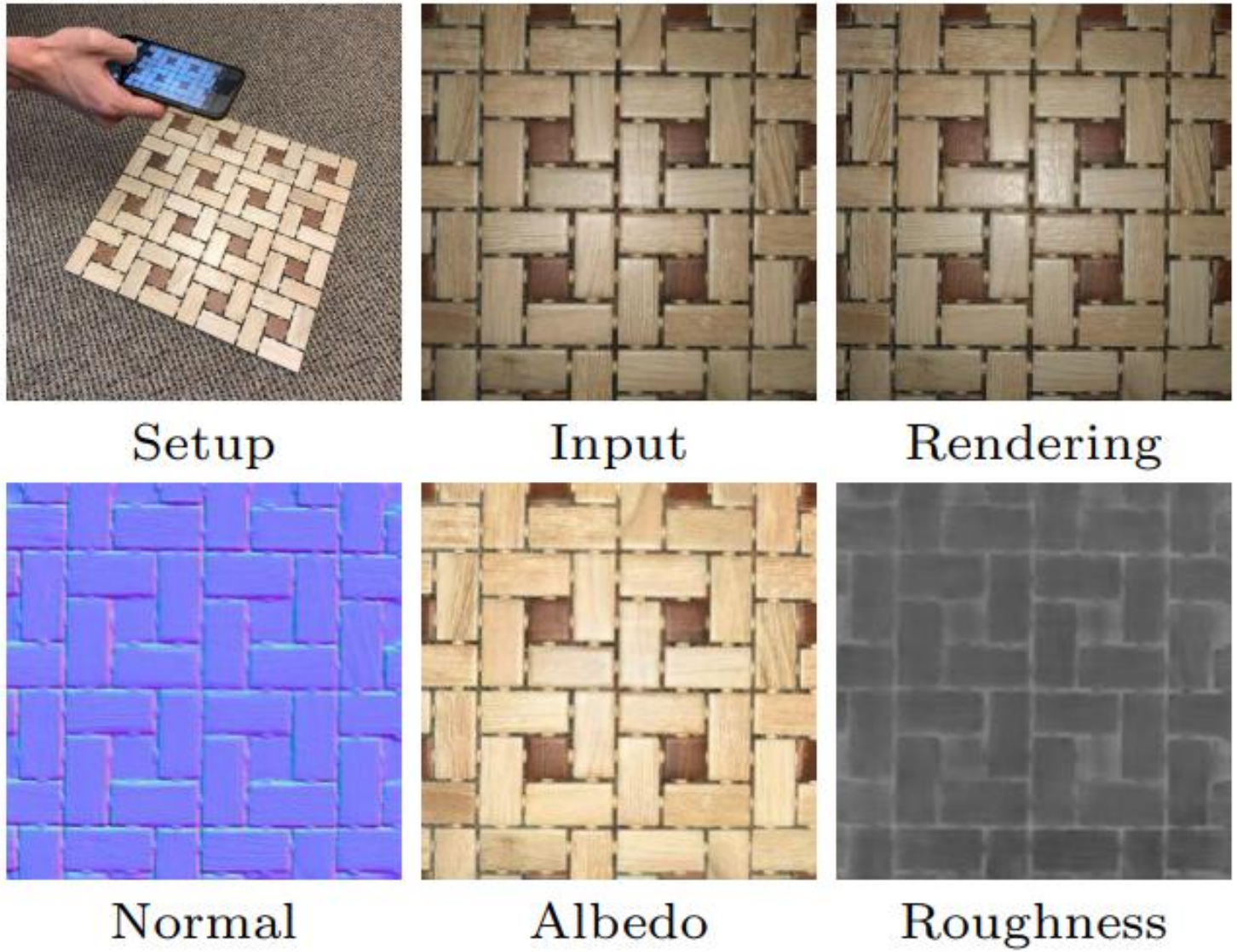


researchgate.net/figure/Material-samples-used-in-the-study-represented-as-BTF-data-Second-row-sh-ows-materials_fig1_258120939

Challenges in BRDF modeling

Accurate representation	• Accuracy for novel view synthesis, rendering.
Efficient representation	• The model is low cost both in storage needed and computation.
Intuitive for editing (optional)	

A.I. in material estimations : Materials for Masses



A.I. in material estimations : Deep inverse rendering for high-resolution SVBRDF estimation from an arbitrary number of images

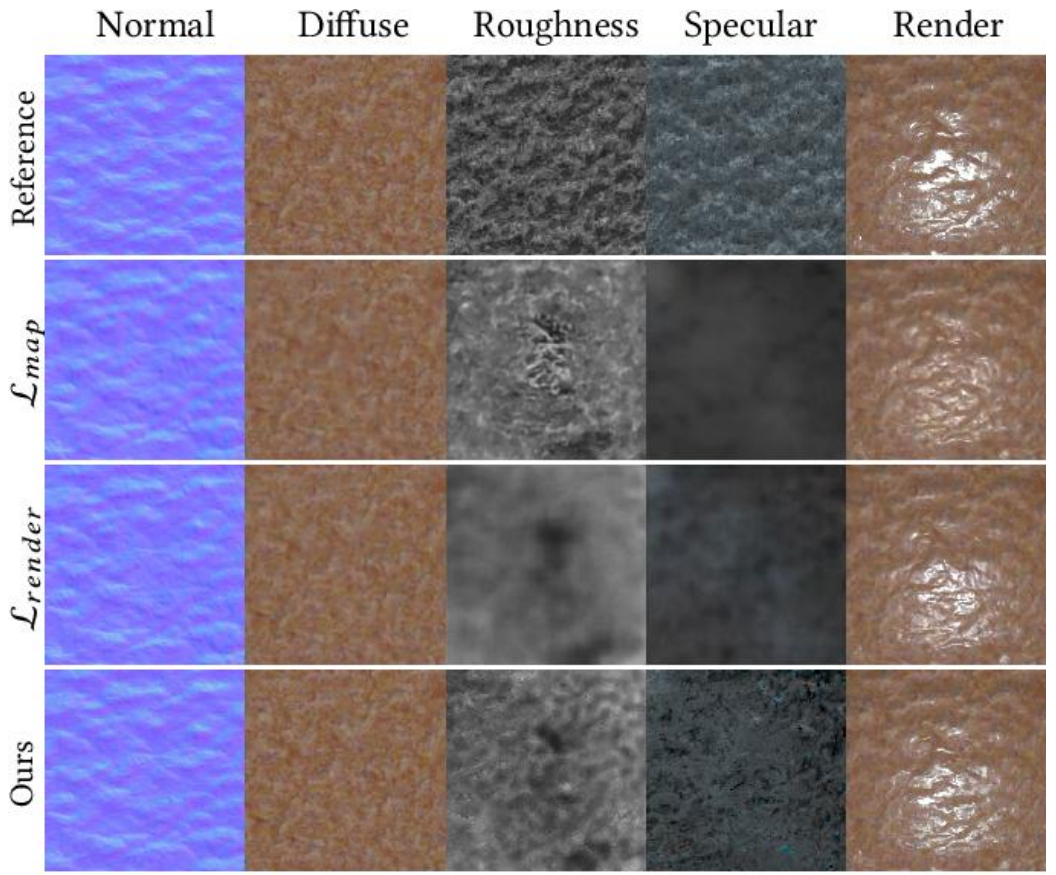
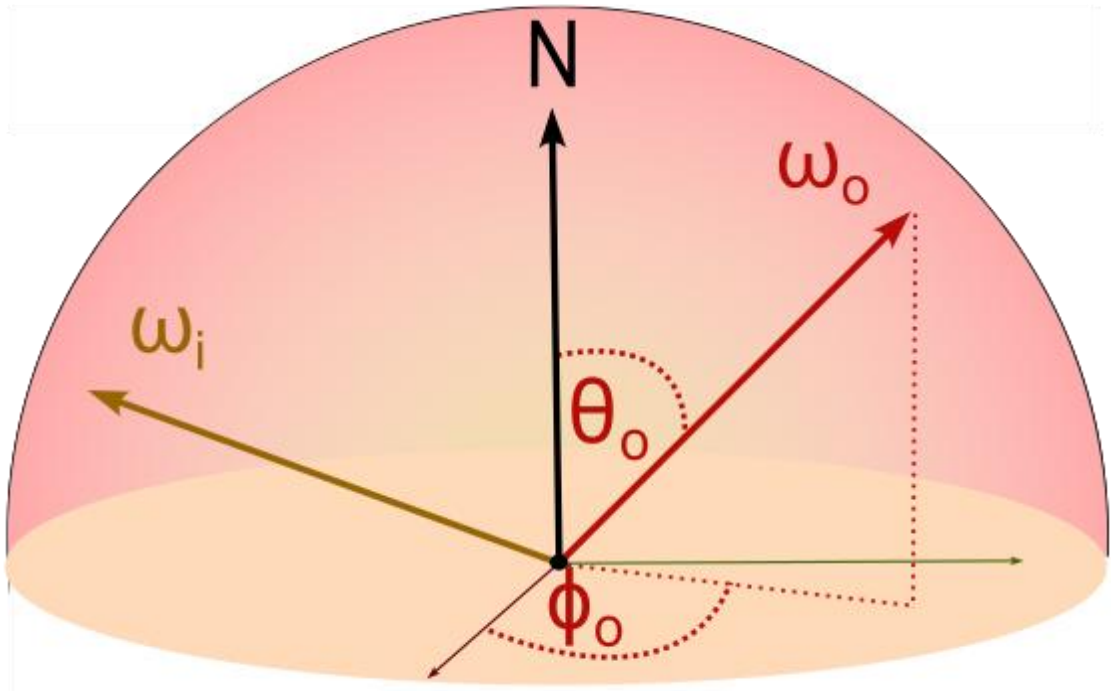
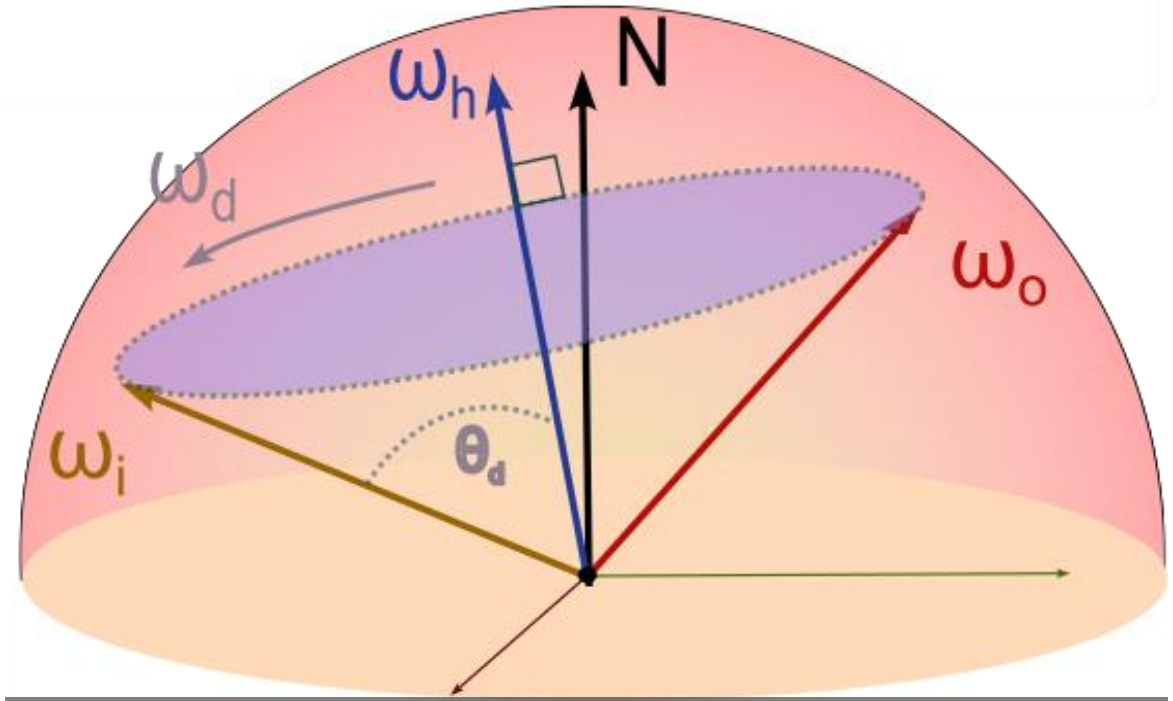


Fig. 15. Impact of the auto-encoder training loss on deep inverse rendering



Spherical coordinates

(Standard parameterization)



Rusinkiewicz [1998]

(Half-Diff parameterization)

BRDF parameterization

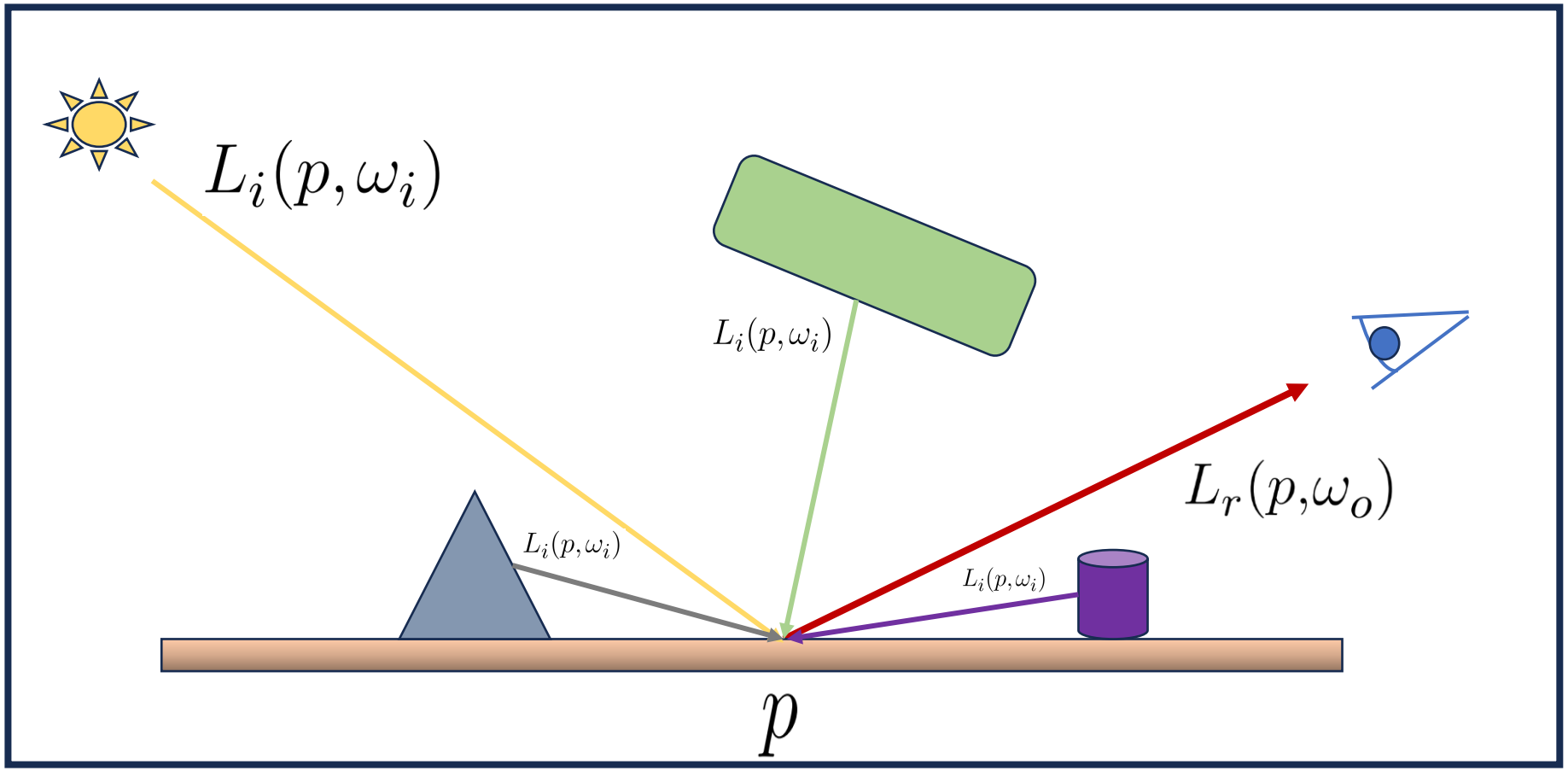
BRDF method

- We separate *scattering* method and *BRDF* method.
- *scattering* method defines how the ray is scattered, reflected.
- *BRDF* method defines how much light is reflected depending on the incoming and outgoing angles (viewing and light directions).

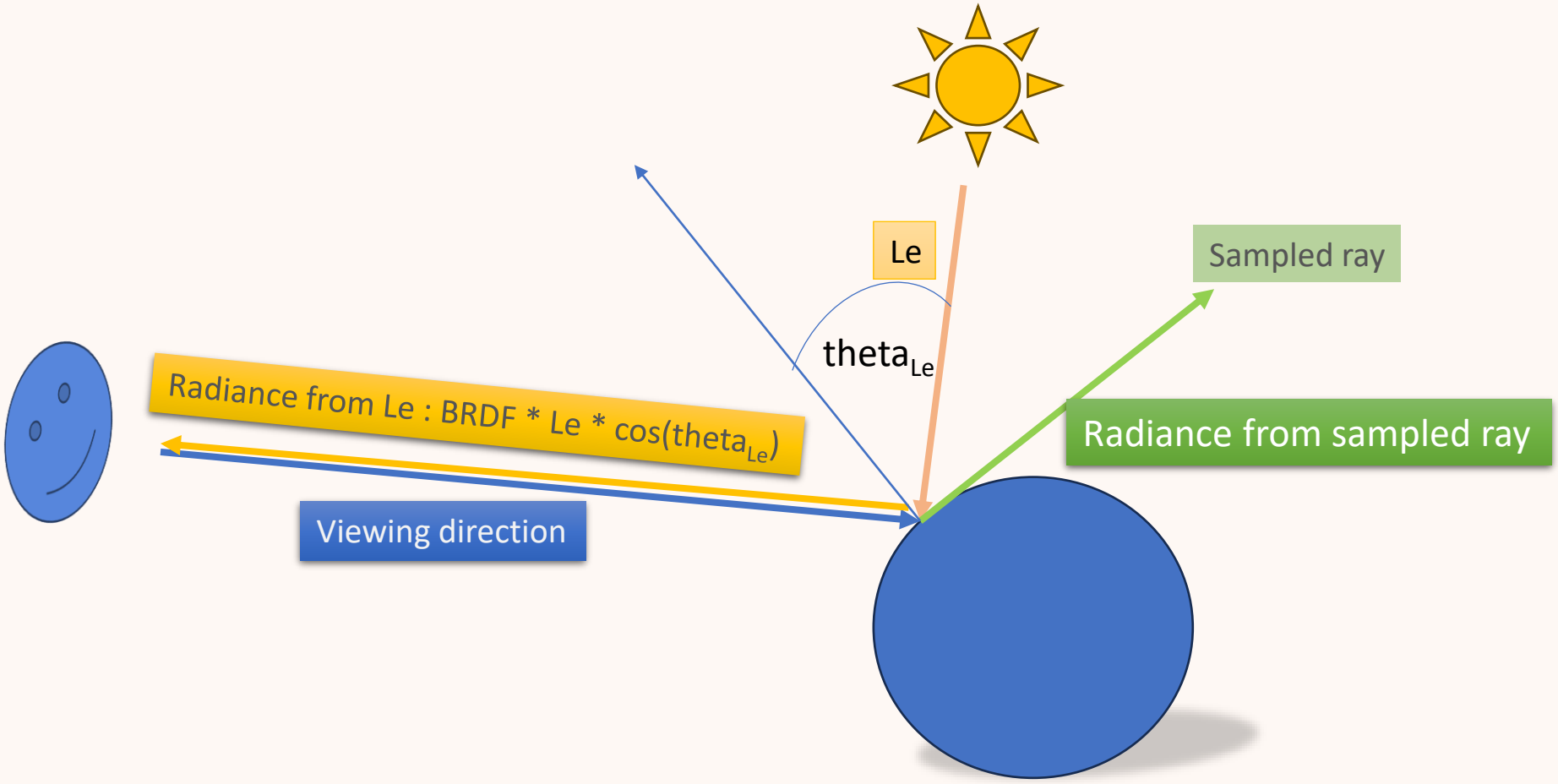
Example

```python
# A metal class with roughness parameter
class Metal(Material):
    def __init__(self, cAlbedo, fRoughness) -> None:
        super().__init__()
        self.color_albedo = rtu.Color(cAlbedo.r(), cAlbedo.g(), cAlbedo.b())
        self.roughness = fRoughness
        if self.roughness > 1.0:
            self.roughness = 1.0

    def scattering(self, rRayIn, hHinfo):
        # compute scattered ray based on the roughtness parameter
        reflected_direction = reflect(rtu.Vec3.unit_vector(rRayIn.getDirection()), hHinfo.getNormal()) + rtu.Vec3.random_vec3_unit()*self.roughness
        reflected_ray = rtr.Ray(hHinfo.getP(), reflected_direction)
        attenuation_color = self.BRDF(rRayIn, reflected_ray, hHinfo)

        # check if the reflected direction is below the surface normal
        if rtu.Vec3.dot_product(reflected_direction, hHinfo.getNormal()) <= 1e-8:
            attenuation_color = rtu.Color(0,0,0)

        return rtu.Scatterinfo(reflected_ray, attenuation_color)

    def BRDF(self, rView, rLight, hHinfo):
        attenuation_color = rtu.Color(self.color_albedo.r(), self.color_albedo.g(), self.color_albedo.b())
        return attenuation_color
```

$$\int_{\Omega} \underbrace{\rho(p, \omega_o, \omega_i)}_{reflectance} \underbrace{L_i(p, \omega_i) \mid cos(\theta_i) \mid d\omega_i}_{irradiance}$$



Direct lighting term

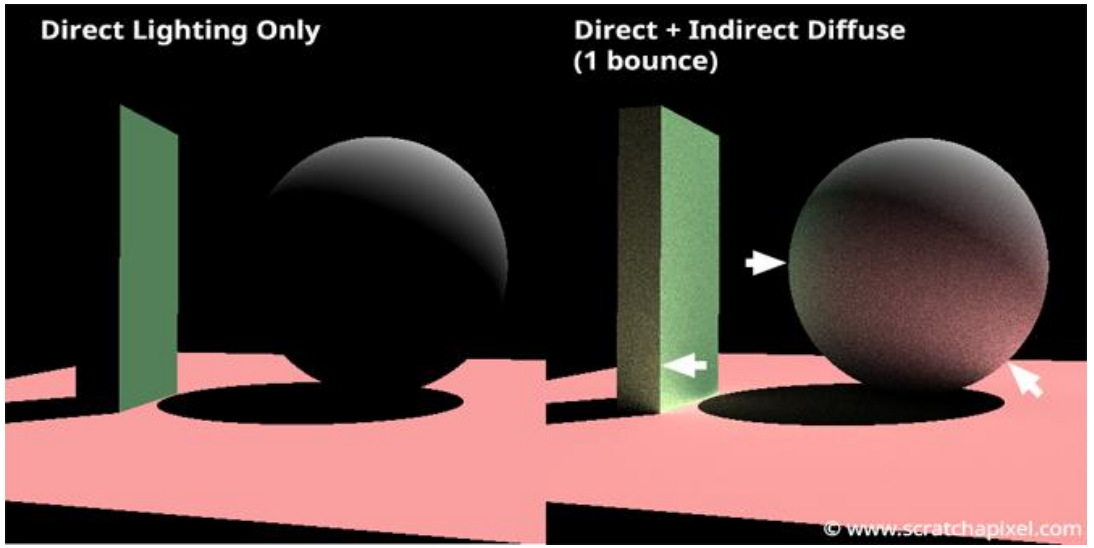


Direct lighting changed

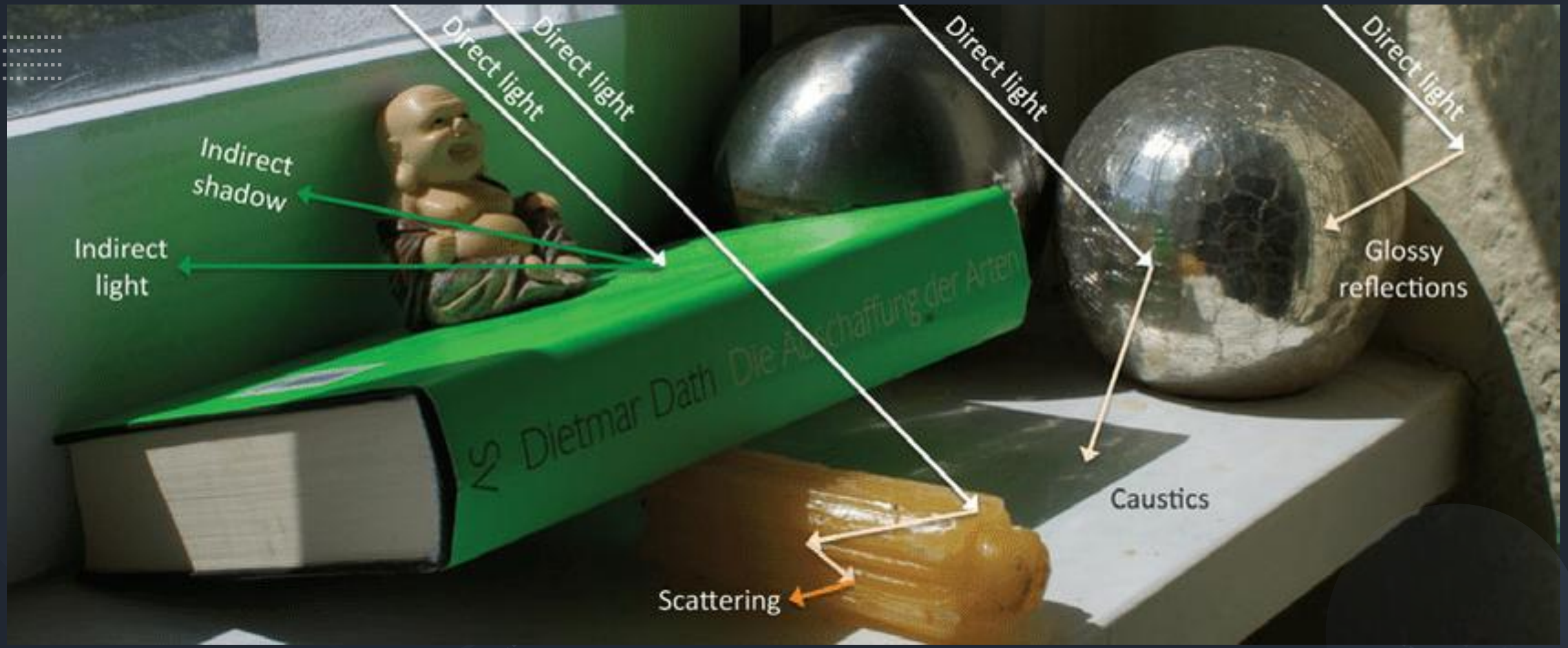
```
Le = rtu.Color()
# if direct lighting is enabled
if self.bool_direct_lighting:
    # for each point light
    for light in scene.point_light_list:
        # check if there is an occlusion between a point light and a surface point.
        tolight_dir = light.center - hinfo.getP()
        tolight_ray = rtr.Ray(hinfo.getP(), tolight_dir)
        max_distance = tolight_dir.len()
        occlusion_hit = scene.find_occlusion(tolight_ray, rtu.Interval(0.000001, max_distance))
        # if not occluded.
        if not occlusion_hit:
            # accumulate all unoccluded light
            Le_BRDF = hmat.BRDF(rGen_ray, tolight_ray, hinfo)
            NdotLe = rtu.Vec3.dot_product(hinfo.getNormal(), tolight_dir)
            direct_L_i = light.material.emitting()
            Le = Le + (Le_BRDF * direct_L_i * NdotLe)

# return the color
# Le*attennuation_color upto the point before reflection models otherwise it is not correct.
NdotL = rtu.Vec3.dot_product(hinfo.getNormal(), sinfo.scattered_ray.getDirection())
L_i = self.compute_scattering(sinfo.scattered_ray, scene, maxDepth-1)
Fr =  hmat.BRDF(rGen_ray, tolight_ray, hinfo)
return Le + ( Fr * L_i * NdotL )
```

Indirect illumination

- What effects do we obtain when our renderer can account for indirect lighting ?
- How do we simulate the indirect lighting ?



www.scratchapixel.com/lessons/3d-basic-rendering/global-illumination-path-tracing/global-illumination-path-tracing-practical-implementation.html

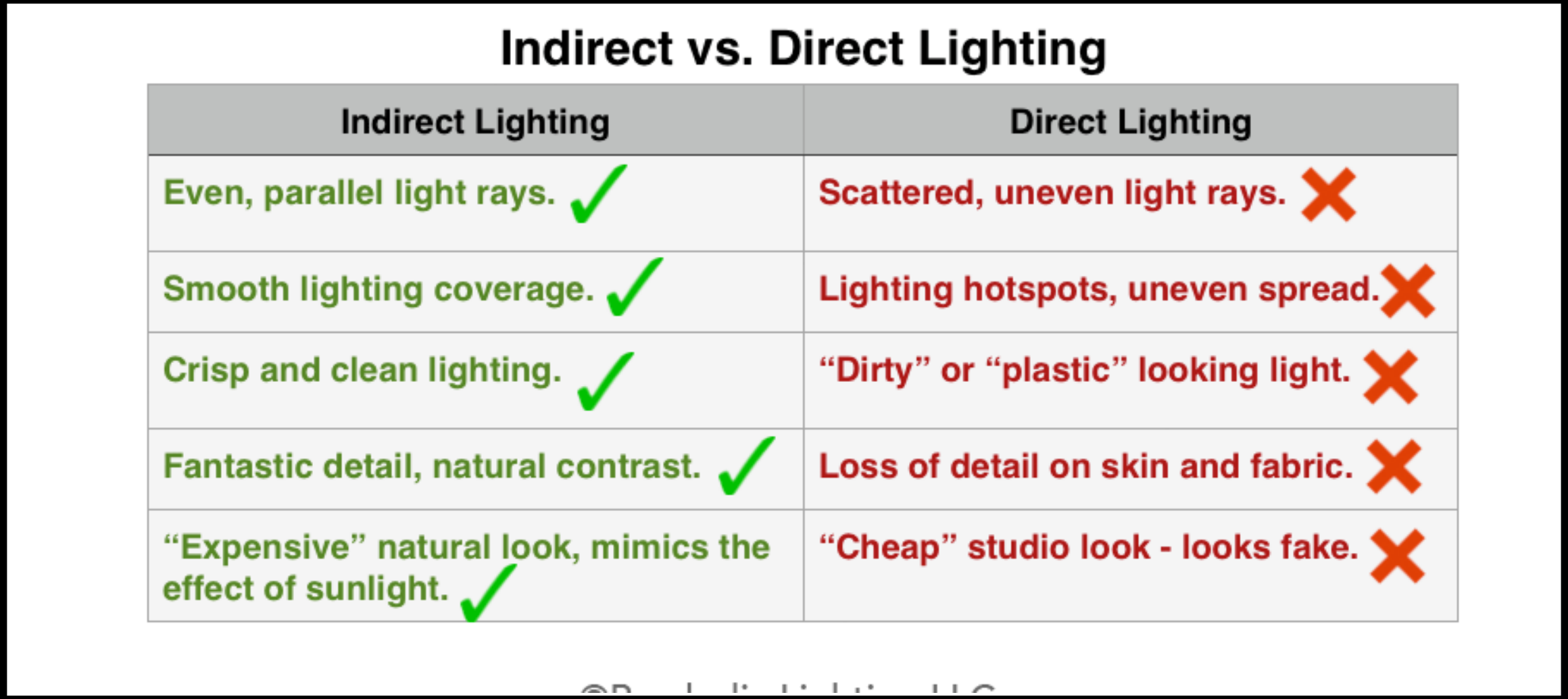


Lighting effects

- Rauwendaal, Randall. (2013). Voxel Based Indirect Illumination using Spherical Harmonics. 10.13140/2.1.1300.6403.

Indirect lighting effects

- https://blogs.nvidia.com/blog/2022/08/04/direct-indirect-lighting/



Photography perspectives

https://www.parabolixlight.com/indirect-vs-directlighting

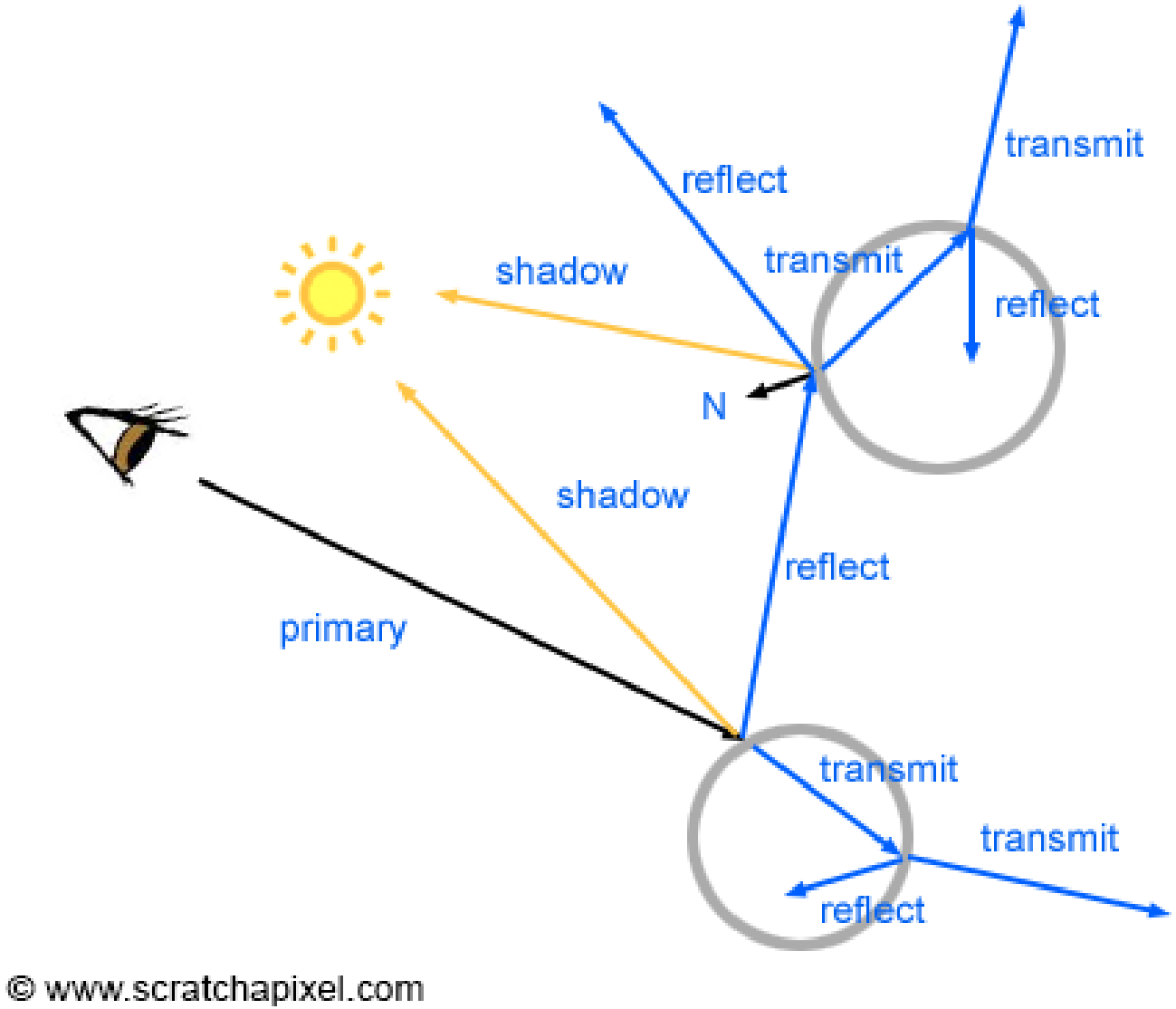
Solving for indirect illumination

- It means we have to find an integrator such that it accounts for the indirect illumination.
- The first week we covered types of global illumination solver.
  - Ray tracing
  - Radiosity
  - Photon mapping

- In this course, we focus on ray tracing.
- The integrator is an important key to render photo-realistic images.

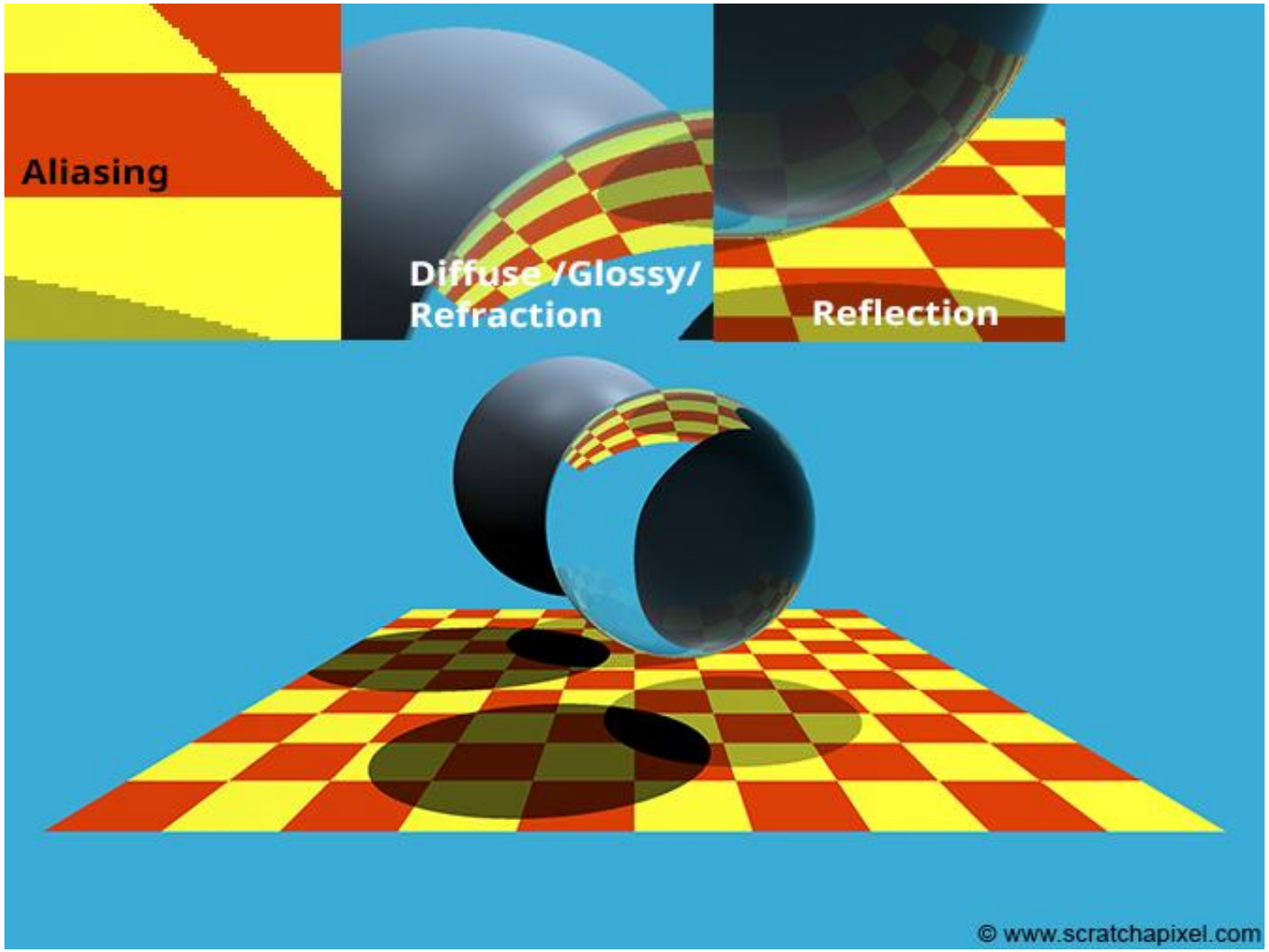
Whitted Ray Tracing

Only 1 ray per pixel !



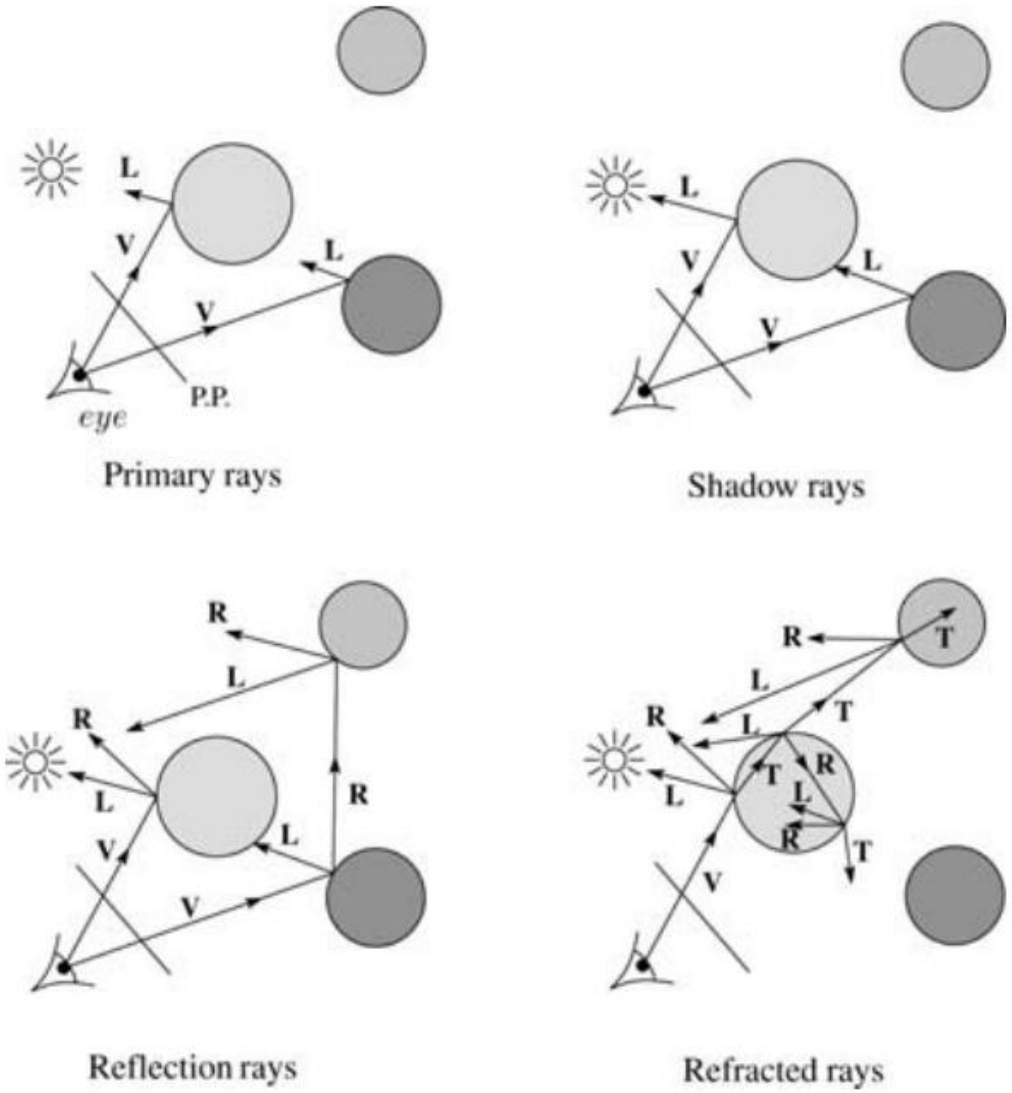
www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-overview/light-transport-ray-tracing-whitted.html

Result


Aliasing
Diffuse /Glossy/
Refraction
Reflection
© www.scratchapixel.com

Algorithm

- For each pixel, shoot a primary ray in direction **V** to the first visible surface.
- For each intersection, trace secondary rays
  - Shadow rays in direction **L**, to light sources
  - Reflected ray in direction **R**.
  - Refracted ray or transmitted ray in direction **T**.

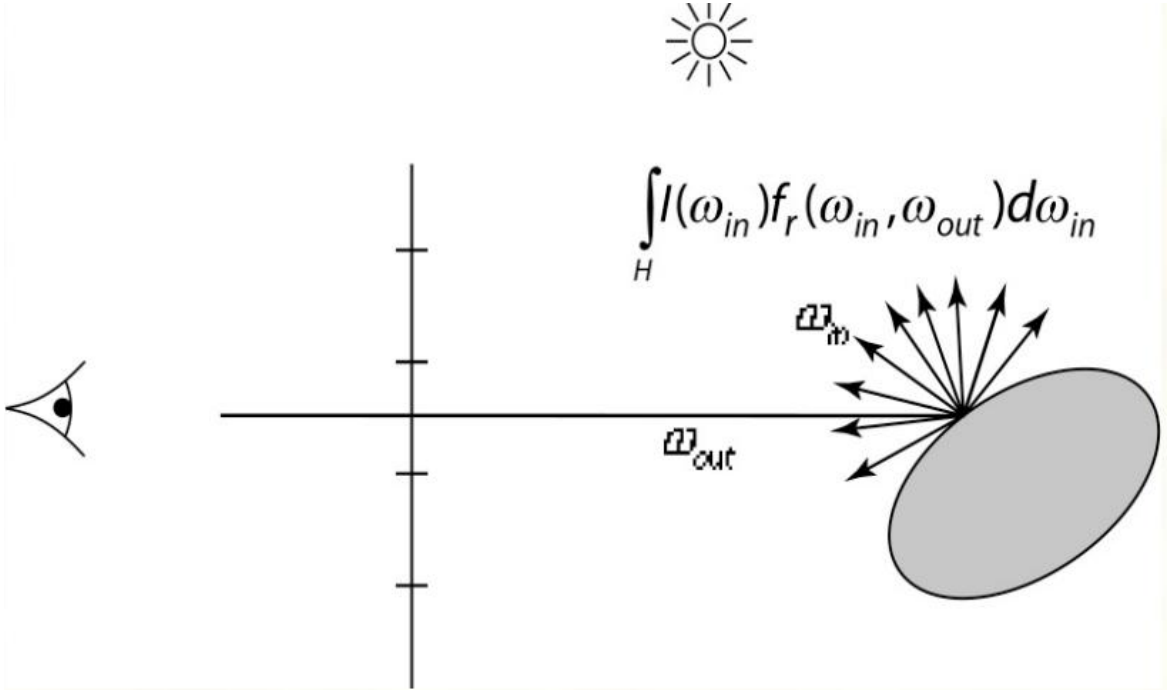


courses.cs.washington.edu/courses/cse457/11au/lectures/markup/ray-tracing-markup.pdf

Indirect illumination with the Whitted ray tracing ?

- It’s very limited to account for indirect illumination.
  - One ray per pixel.
  - Only perfect reflection.
- The following idea came after the Whitted ray tracing to account for more integration domains.
  - Distributed ray tracing, Distribution ray tracing, Stochastic ray tracing
  - Unbiased path tracing, Path tracing

Distributed ray tracing



www.cs.utexas.edu/users/fussell/courses/cs384g-fall2011/lectures/lecture11-Drt.pdf

What kind of integrators do we have ?

- Week03 assignment
  - 1 ray at the pixel center
  - 1 ray at random location of the pixel
  - Multiple rays at random locations
- Week04 assignment
  - Multiple rays at random locations + no scattering
- Week05 assignment
  - Multiple rays at random locations + random scattering

Codes and class assignment !

- Github : RT-python-week08
  - https://github.com/KUGA-01418283-Raytracing/RT-python-week08