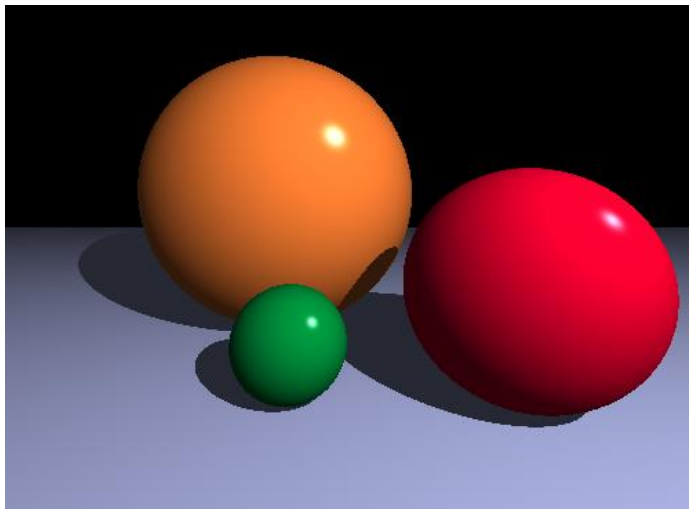
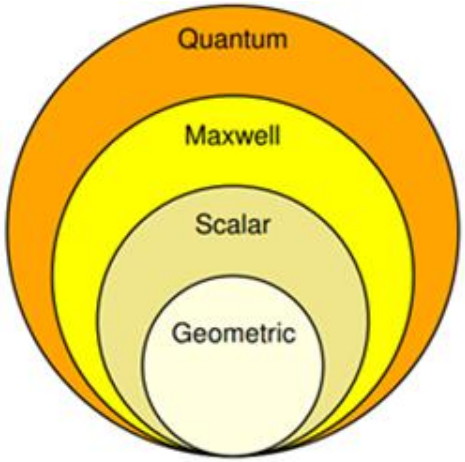
Raytracing
II – Light & Shadow

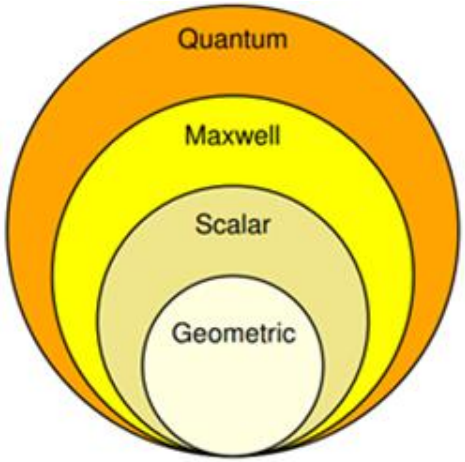


Stefan BORNHOFEN

Models of Light



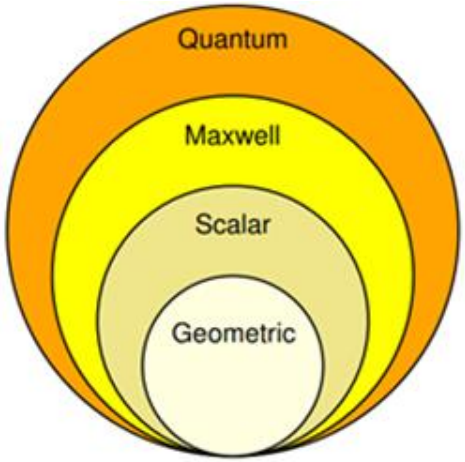
Models of Light



Classic raytracers only implement **Geometric Optics** (“Ray Optics”) and do not account for other light properties such as phase or wavelength.

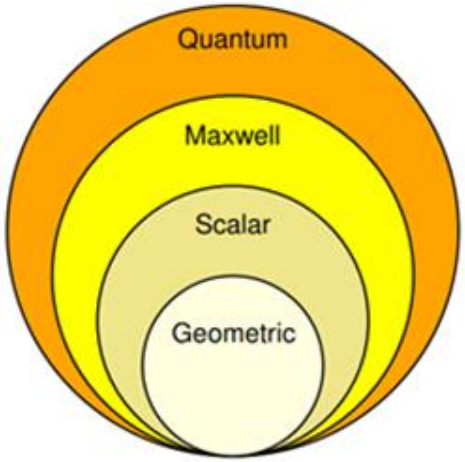
- **Geometric optics**: emission, refraction, reflection

Models of Light

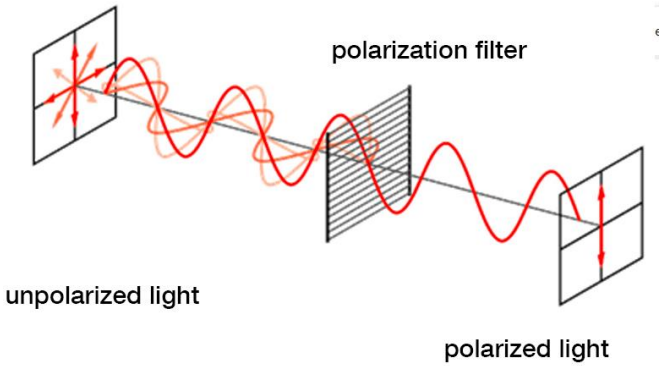


- **Scalar Wave optics:** interference, diffraction, iridescence
- Geometric optics: emission, refraction, reflection

Models of Light

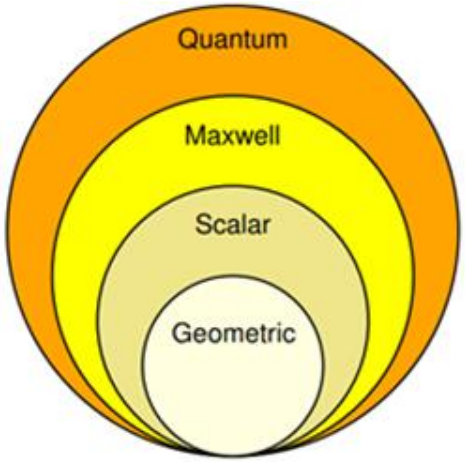


- **Maxwell Equations**: electro-magnetic field, polarization
- Scalar Wave optics: interference, diffraction, iridescence
- Geometric optics: emission, refraction, reflection



A water surface more intensely reflects light with horizontal polarization than with vertical polarization.

Models of Light



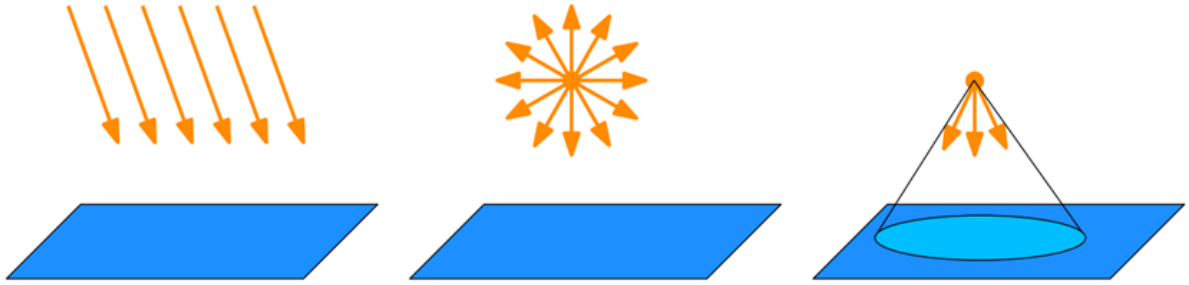
- **Quantum Optics**: fluorescence, phosphorescence
- Maxwell Equations: electro-magnetic field, polarization
- Scalar Wave optics: interference, diffraction, iridescence
- Geometric optics: emission, refraction, reflection

Geometric Optics

Rays are used to model the propagation of light through an optical system, by dividing the real light field into discrete rays that can be propagated through the scene by the techniques of ray tracing.

- Light is a flow on photons called light rays
- Light rays propagate in straight lines as they travel in a homogeneous medium
- Light rays do not interfere with each other as they cross
- Light rays obey the laws of reflection and refraction, at the interface between two dissimilar media
- Light rays travel from the light sources to the eye, but the physics is invariant under path reversal (reciprocity)

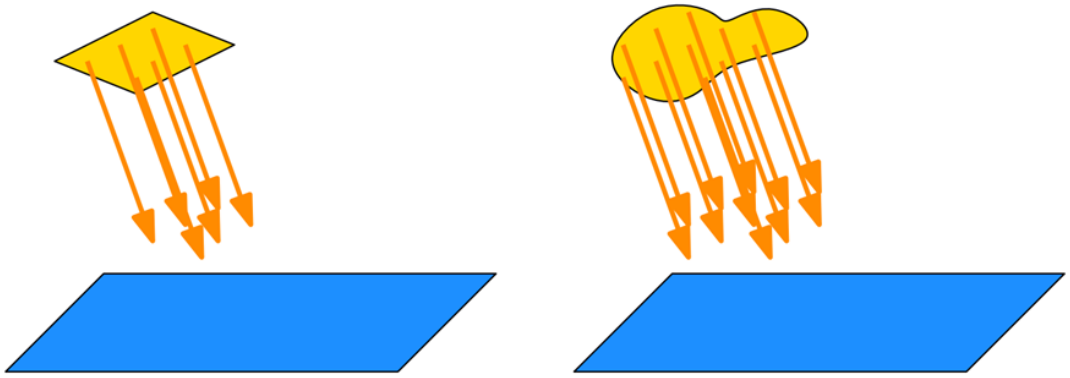
Light sources



Directional Light
- Infinite distance
- Parallel rays

Point Light
- One point
- Isotropic rays

Spot Light
- One point
- Beam direction
- Beam angle

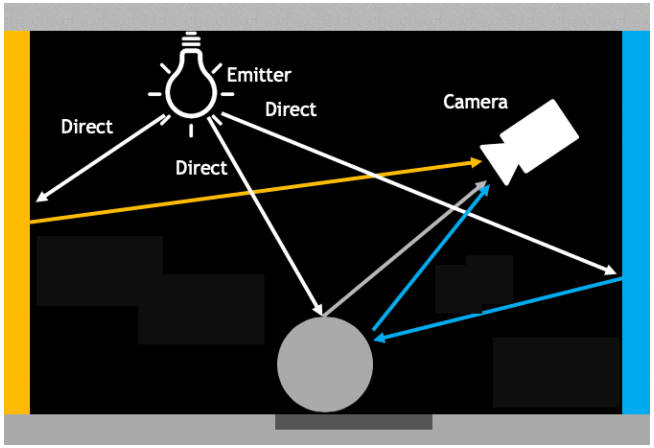


Area Lights
- Areas or objects that emit light
- Approximated with multiple point lights

Direct vs. Indirect Illumination

The light at any given point on a surface in a scene is influenced by

- **Direct illumination**: light directly arriving from the light source on the surface
- **Indirect illumination**: light arriving on the surface after having bounced off other surfaces.

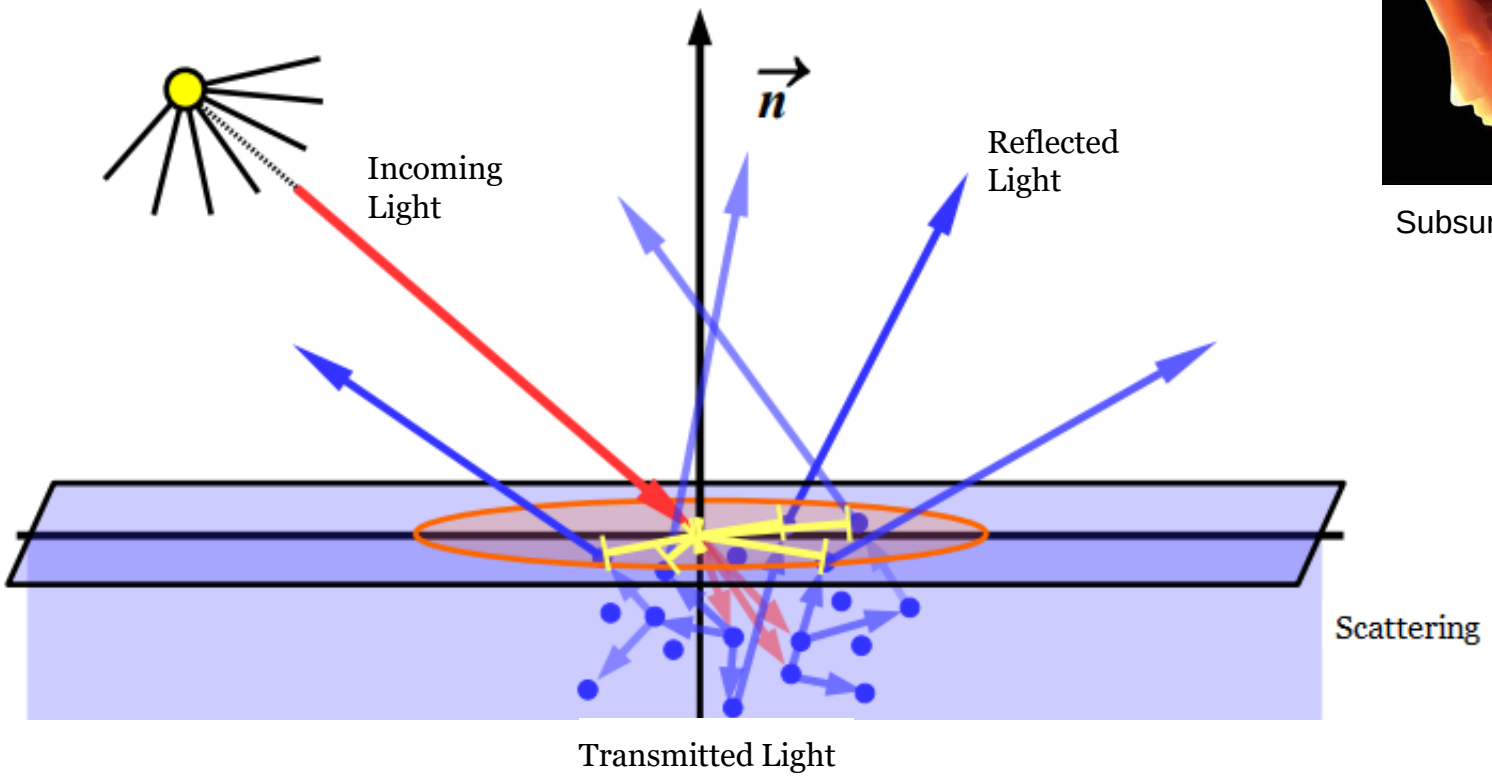


As a consequence, lighting methods can be

- **Local**: only the direct illumination is considered
- **Global**: both direct and indirect illuminations are considered

In the real world, global illumination can reflect infinitely. However, some light is absorbed at each surface. After a few reflections there is so little that it is not worth simulating any further.

Light – Material interactions

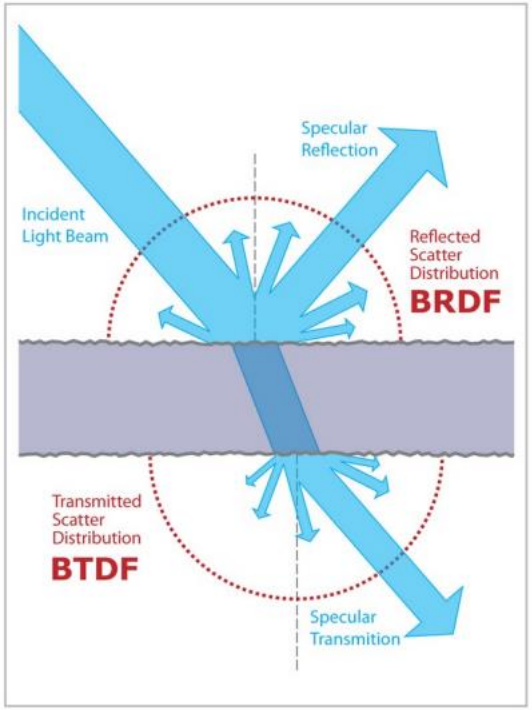


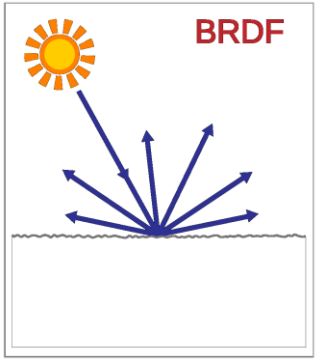
Subsurface scattering

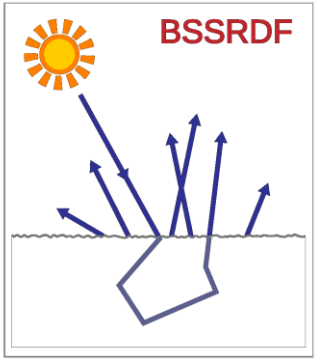
Bidirectional Distribution Functions

BxDF are local illumination modeling functions that describe how light interacts with a given surface.

- **BRDF** (Bidirectional reflectance distribution function) assumes that light enters and leaves at the same point
- **BTDF** (Bidirectional transmittance distribution function) is similar to BRDF but for the opposite side of the surface.
- **BSSRDF** (Bidirectional scattering-surface reflectance distribution function) describes the relation between outgoing radiance and the incident flux, including the phenomenon of subsurface scattering.
- **BSSTDF** (Bidirectional scattering-surface transmittance distribution function) is like BTDF but with subsurface scattering.



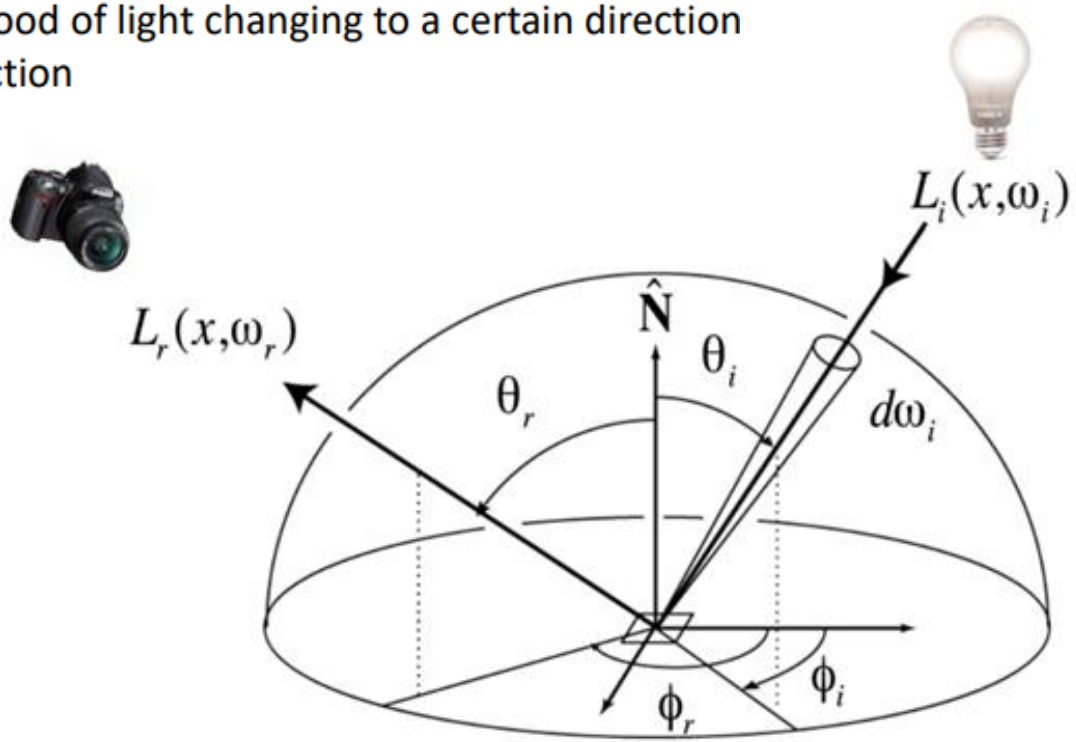




BRDF

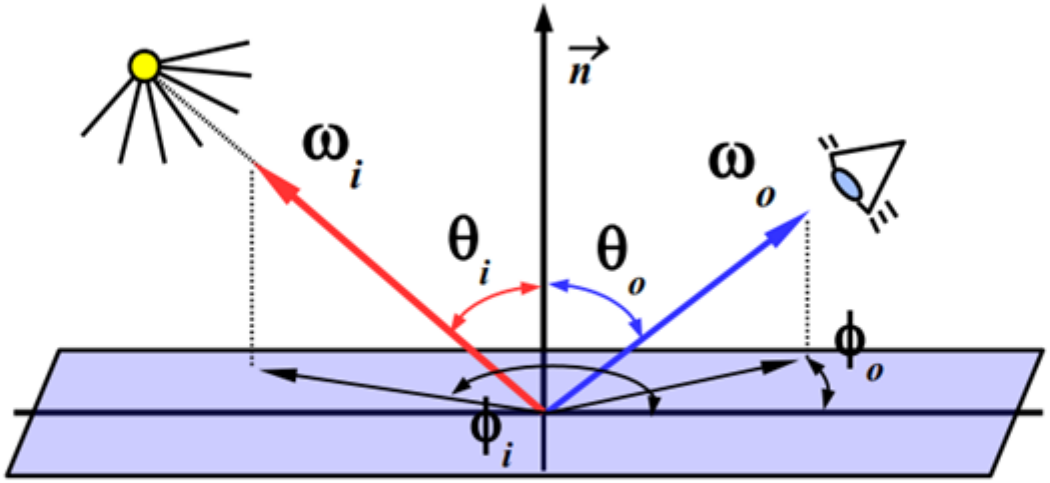
The **Bidirectional Reflectance Distribution Function** is a function that defines the ratio of the quantity of reflected light in direction w_r, to the amount of light that reaches the surface from direction w_i.

- *Bidirectional* – a function of two directions ω_i and ω_o
- *Reflectance* – light changing directions
- *Distribution* – likelihood of light changing to a certain direction
- *Function* – it's a function



The Rendering Equation

BRDF function: $\mathbf{R}^4 \rightarrow \mathbf{R}$ $\quad f(\omega_i, \omega_o)$



$$L_o(\omega_o) = \int_{\Omega} f(\omega_i, \omega_o) \cdot L_i(\omega_i) \, (n \cdot \omega_i) \, d\omega_i$$

$\cos\theta_i$

BRDF Properties

Physically plausible BRDFs are

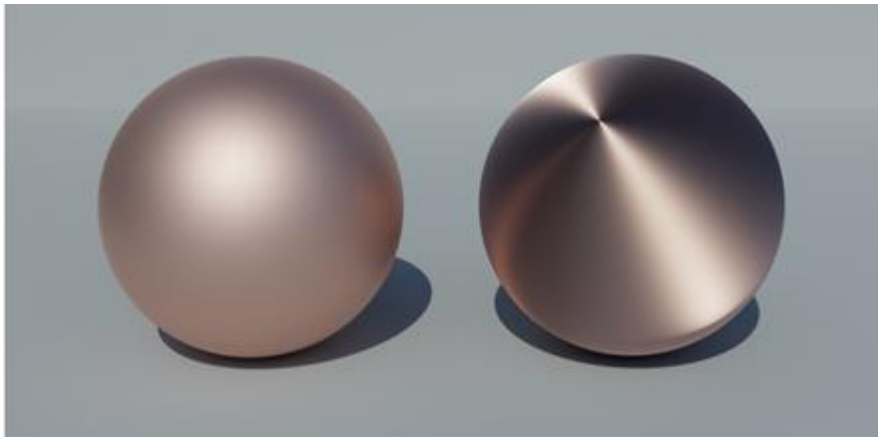
- **positive** everywhere over the range of valid incoming and outgoing directions.
- **Helmholtz reciprocal,** i.e if you swap the incoming and outgoing directions in the function, the function returns the same result.
- **energy conserving,** i.e. the BRDF can not reflect more light than it receives.

$$f(\omega_i, \omega_o, \lambda) \geq 0$$

$$f(\omega_i, \omega_o, \lambda) = f(\omega_o, \omega_i, \lambda)$$

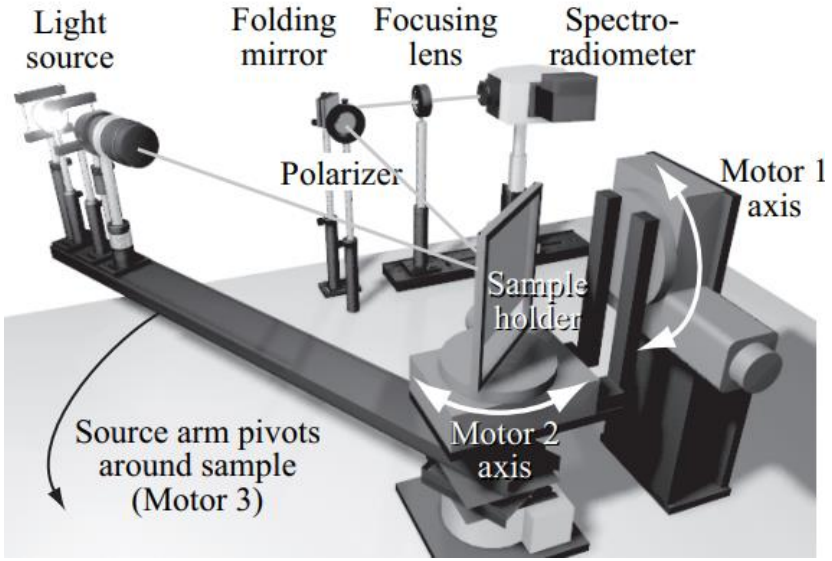
$$\int_{\Omega} f(\omega_i, \omega_o)(n \cdot \omega_i)\, d\omega_i \leq 1$$

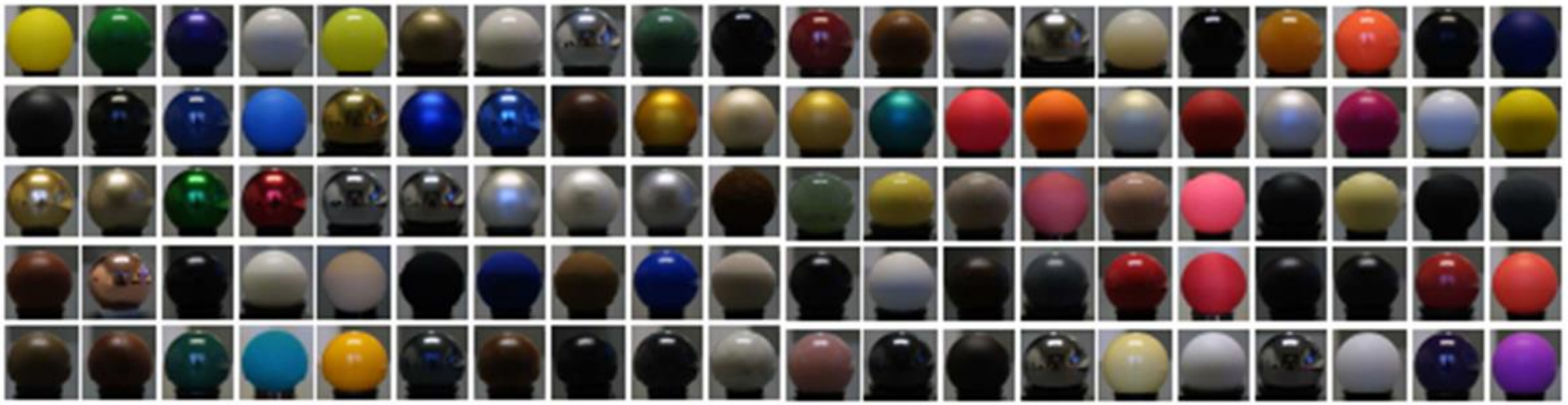
BRDFs can be **isotropic** or **anisotropic**, i.e. they change with respect to rotation of the surface around the surface normal vector.



BRDF Measurement

- BRDFS of physical materials can be acquired with a **gonio-reflectometer**, by moving a sensor and a light source, taking measurements every few degrees.
- The MERL BRDF database contains reflectance functions of 100 different materials.





BRDF Models

Different models are useful for modeling the reflectance characteristics of different types of materials.

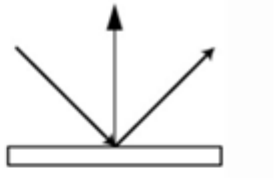
Empirical

- Minnaert
- Phong
- Blinn
- Strauss
- …

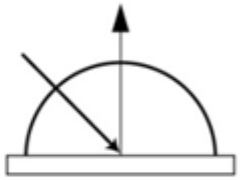
Physical

- Lambert
- Torrance-Sparrow
- Cook-Torrance
- Oren-Nayar
- He et al.
- …

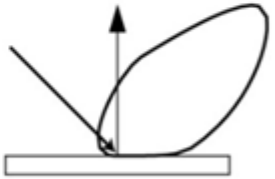
- **Ideal specular**
  Perfect mirror



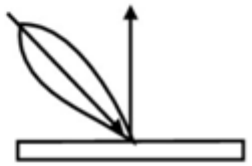
- **Ideal diffuse**
  Uniform reflection in all directions



- **Glossy specular**
  Majority of light distributed in reflection direction



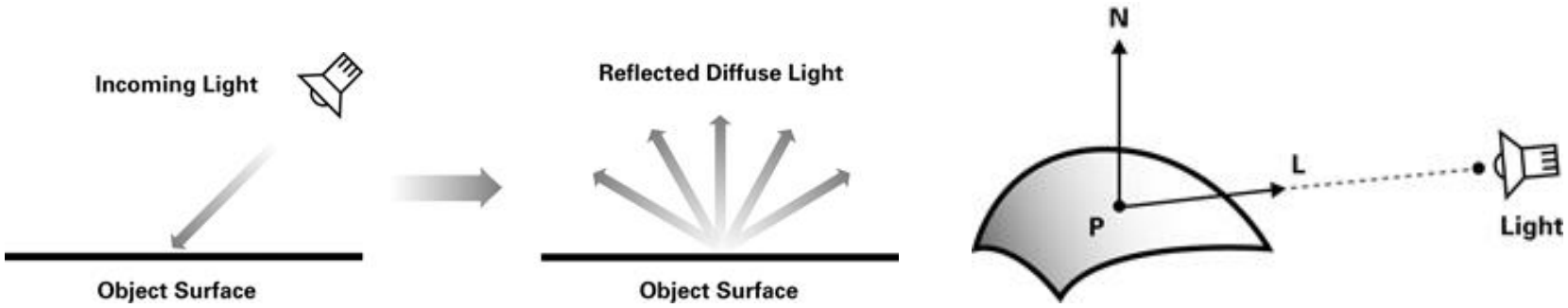
- **Retro-reflective**
  Reflects light back toward source



Lambert BRDF

- Models direct diffuse reflection coming from a light source
- A Lambertian surface by definition reflects radiance equally into all directions.
- The intensity depends on the angle of incoming light

$$I = K_d * I_d * (N.L) = K_d * I_d * \cos(N,L)$$



A popular real-time BRDF

$$I = K_a * I_a + K_d * I_d * (N.L) + \ldots$$

ambient

diffuse

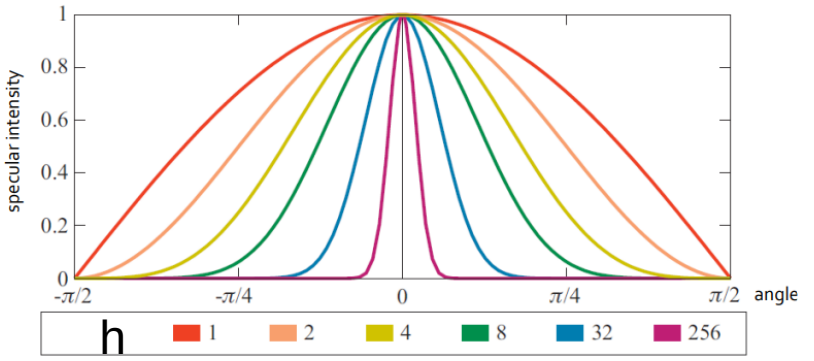
specular

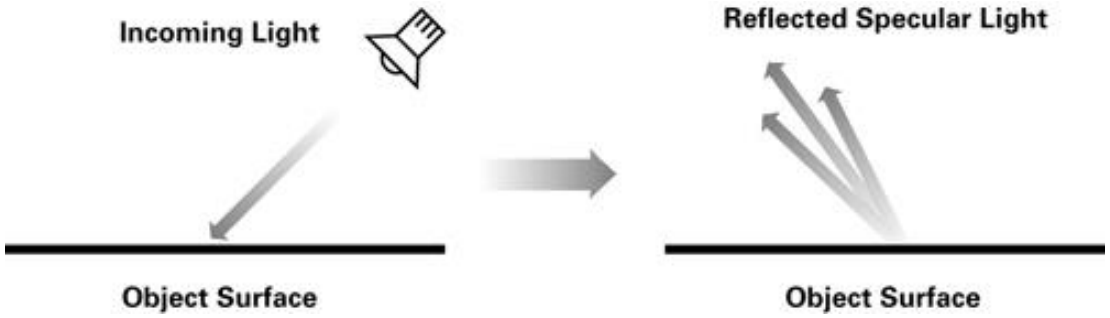
- The **ambient** term models indirect diffuse light with a constant
- The **diffuse** term is a classical Lambert BRDF
- The **specular** term has many variants (Phong, Blinn, Lafortune, etc.)
- I's are function of light (often $I_d = I_s$) , $I \in vec3(R,G,B)$
- K's are function of material, $K \in vec3(R,G,B)$
- Not energy conserving, not reciprocal. This is not a major issue for direct local illumination, but for global lighting simulations the lack of physical validity is problematic.

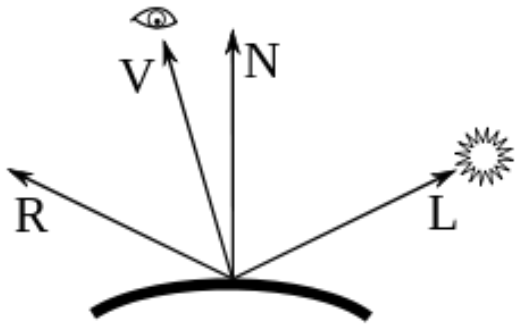
Phong BRDF: Specular term

$$I = K_s * I_s * (V.R)^h$$

- Phong, 1975
- Represents light scattered predominantly around the mirror direction.
- Intensity depends on the angle between viewpoint and perfect reflection
- h allows modulating the specular highlights





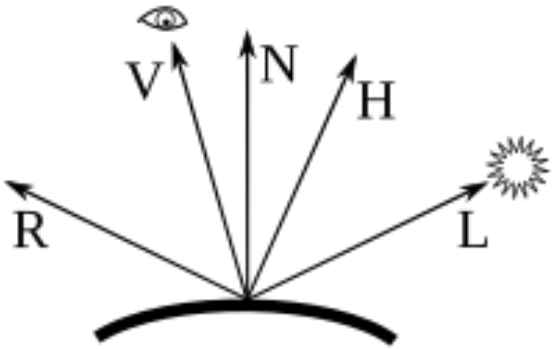


Blinn BRDF: Specular term

- Blinn, 1977
- As a modification to the Phong model, Blinn used the so **halfway vector** that is a unit vector exactly halfway between the view direction and the light direction.
- The Blinn BRDF, also known as the **Blinn-Phong reflection model**, is the standard lighting model used in both the DirectX and OpenGL rendering pipelines.

$$H = \text{normalize}(L+V)$$

$$I = K_s * I_s * (H.N)^h$$



For front-lit surfaces, h′=4h will result in specular highlights that very closely match the corresponding Phong reflections. However, while the Phong reflections are always round for a flat surface, the Blinn reflections become elliptical when the surface is viewed from a steep angle.

$$\cos^h \beta \approx \cos^{4h} \beta/2$$

$$(R \cdot V)^h \approx (H \cdot N)^{4h}$$

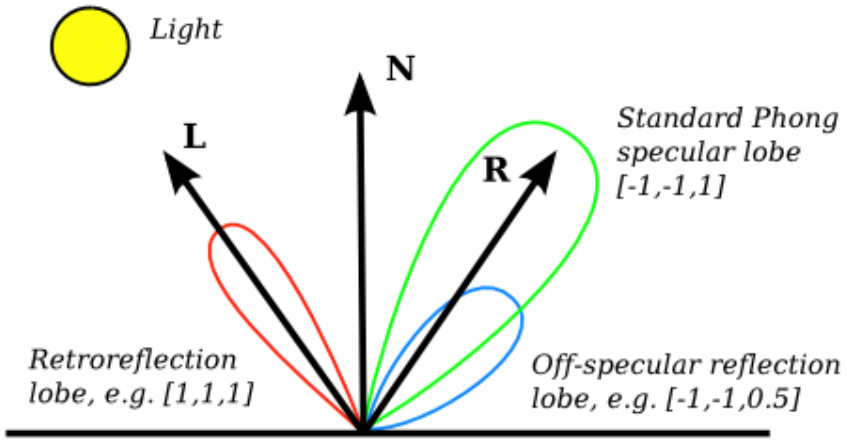
Lafortune BRDF: Specular term

- Lafortune, 1997
- Generalization of the Phong BRDF with multiple configurable lobes
- Replaces dot product with weighted dot product
- Can be used to fit measurements from real surfaces

$$I = I_s * \sum_{i=1}^{nbLobes} (c_{xi} V_x L_x + c_{yi} V_y L_y + c_{zi} V_z L_z)^h$$

Note that the vectors must be put in a local reference system of the surface.

- cx = cy = -1, cz = 1 => Phong model
- |cx| = |cy| > |cz| => Off-specular reflection
- cx = cy > 0 => Retroreflective
- cx = cy = 0 => Generalized diffuse
- cx = cy => Isotropic
- cx ≠ cy => Anisotropic



Your work

Create a point light source and implement direct illumination using Phong and Blinn specular BRDFs. The user can switch between theses two solutions.

Assign different materials to the objects.

Move the position of the light source with the mouse.

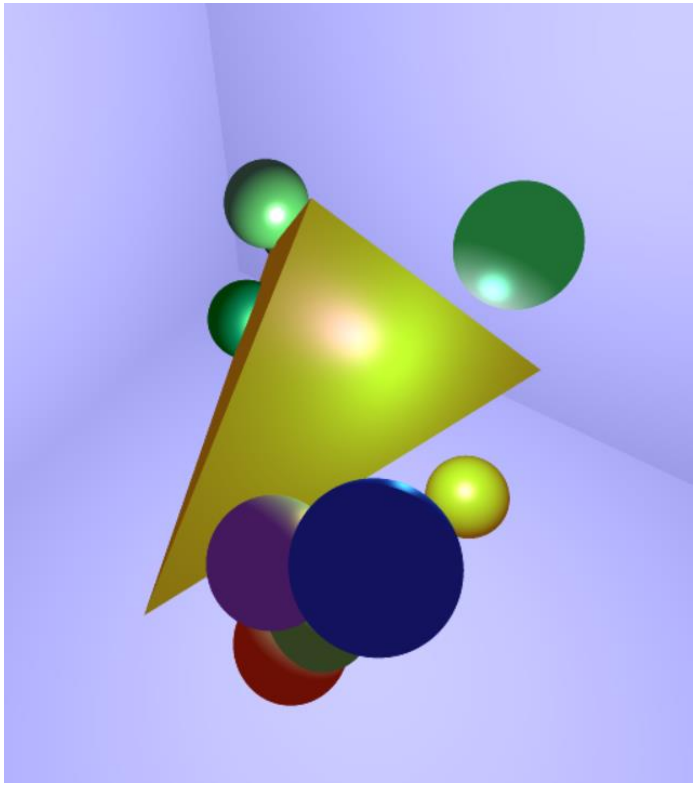
<u>New shader uniform parameters:</u>

Point Light:

- `Position`
- `Color`

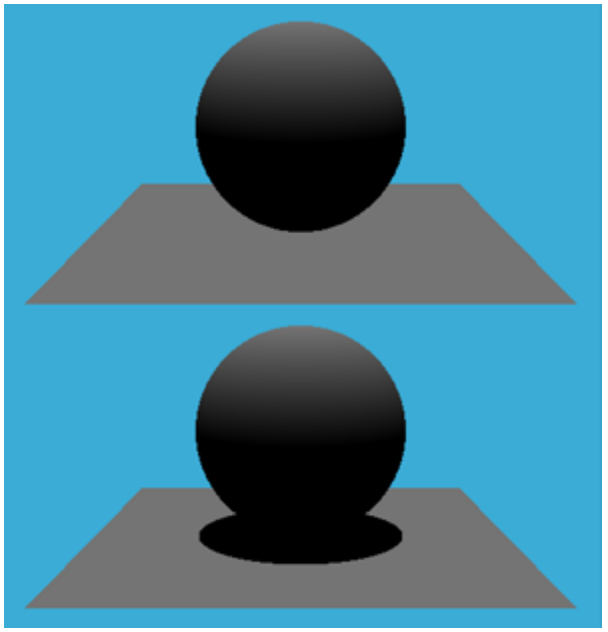
(Array of) Material

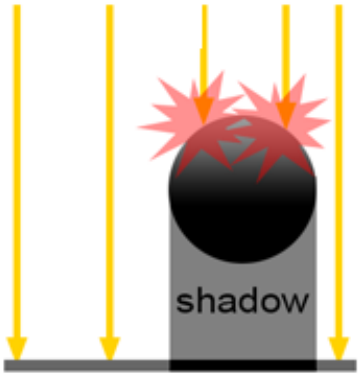
- `Ambient color ka`
- `Diffuse color kd`
- `Specular color ks`
- `Shininess h`



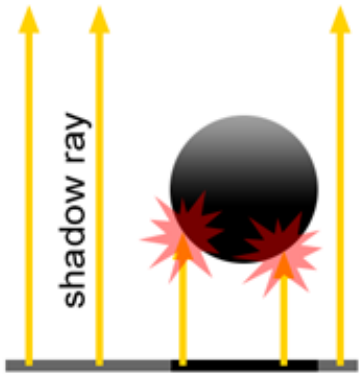
Shadows

- Shadows are an important visual clue to evaluate the position of objects relative to each other.
- If the primary ray intersects an object, we cast a ray from the point of intersection to the light source (**shadow ray**).
- If this shadow ray intersects another object on its way to the light, then the point that we are shading is in the shadow.





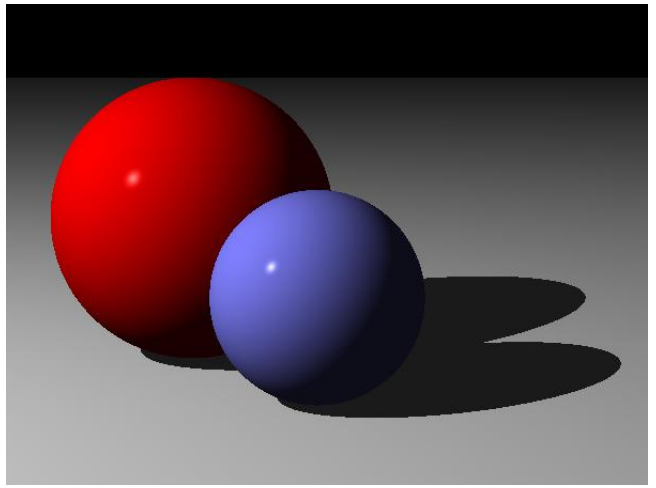
Real world

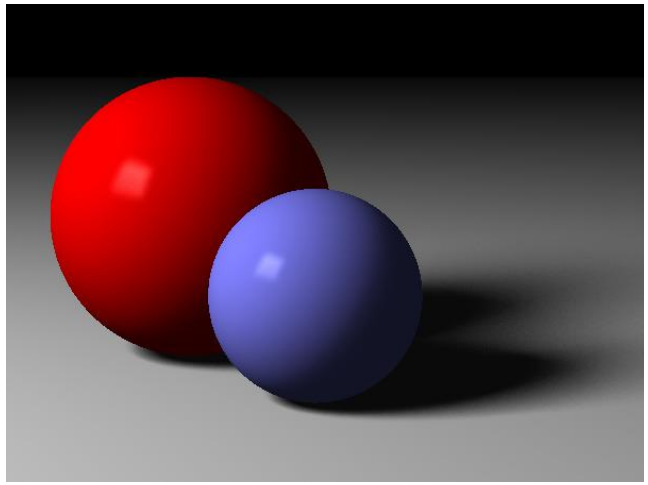


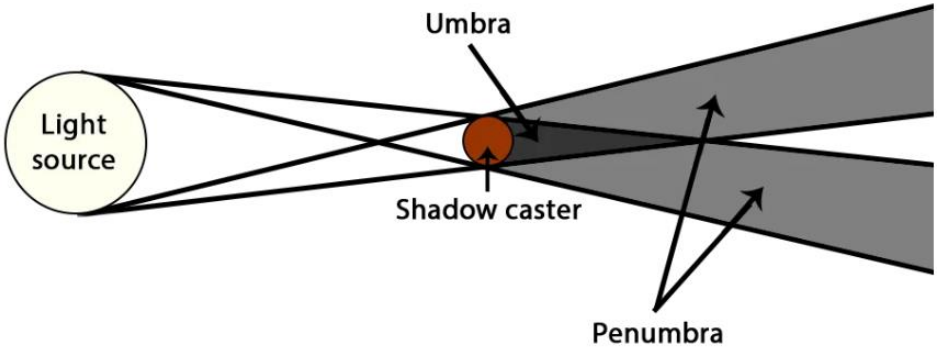
Raytracer

Hard vs. Soft Shadows

- Without any physical dimension, light sources cast **hard shadows** with well-defined edges
- Real light sources have physical dimension, which means that they may be only partially visible from any given point. This partial occlusion creates a **penumbra**: an area where the surface is lit by only a part of the light.
- **Soft shadows** can be modeled through multiple shadow rays cast towards the surface of the light source to sample the visibility of the light surface.

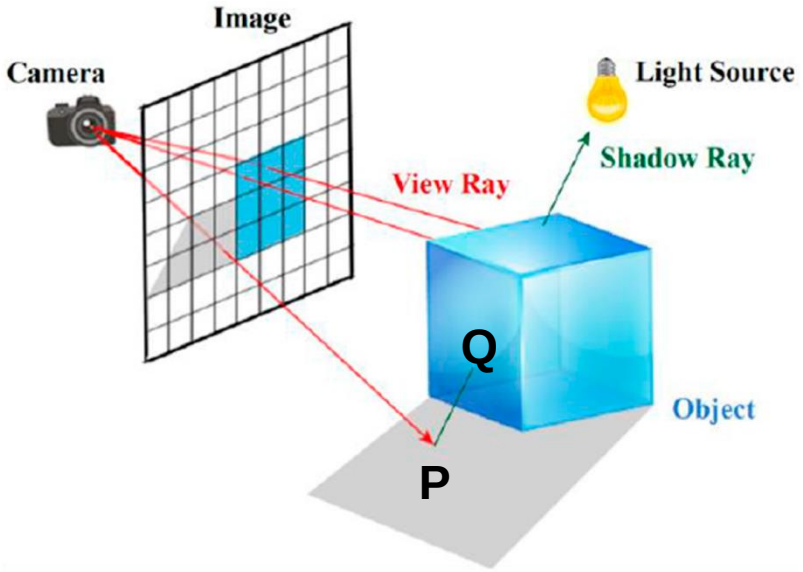






Shadows: how-to

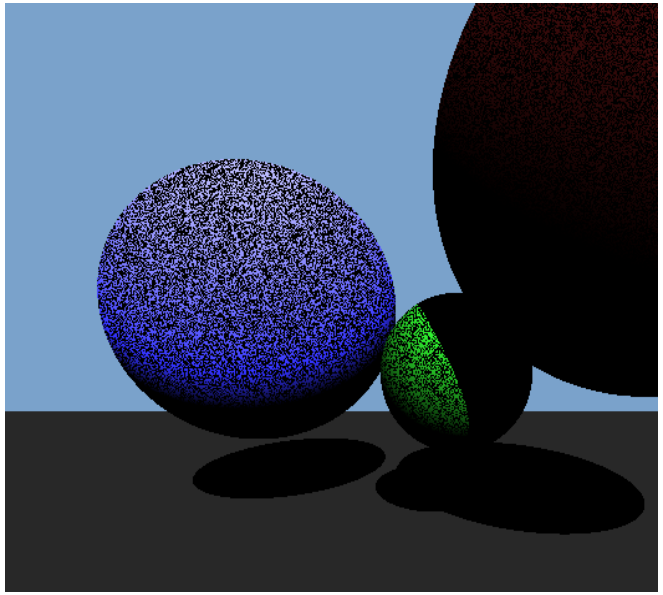
- Cast a **shadow ray** from the intersection point P towards the light source
- Compute the shadow factor S_L : if ray is blocked => S_L=0, otherwise S_L=1
- Instead of S_L=0, we dilute the shadow using S_L = dist(P,Q) / dist(P,lightsource)
- Apply S_L to the diffuse and specular terms of the lighting formula

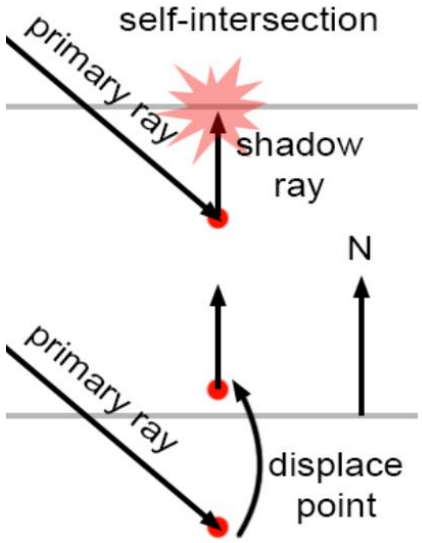


$$I = ambient + S_L*(diffuse + specular)$$

Shadow Acne

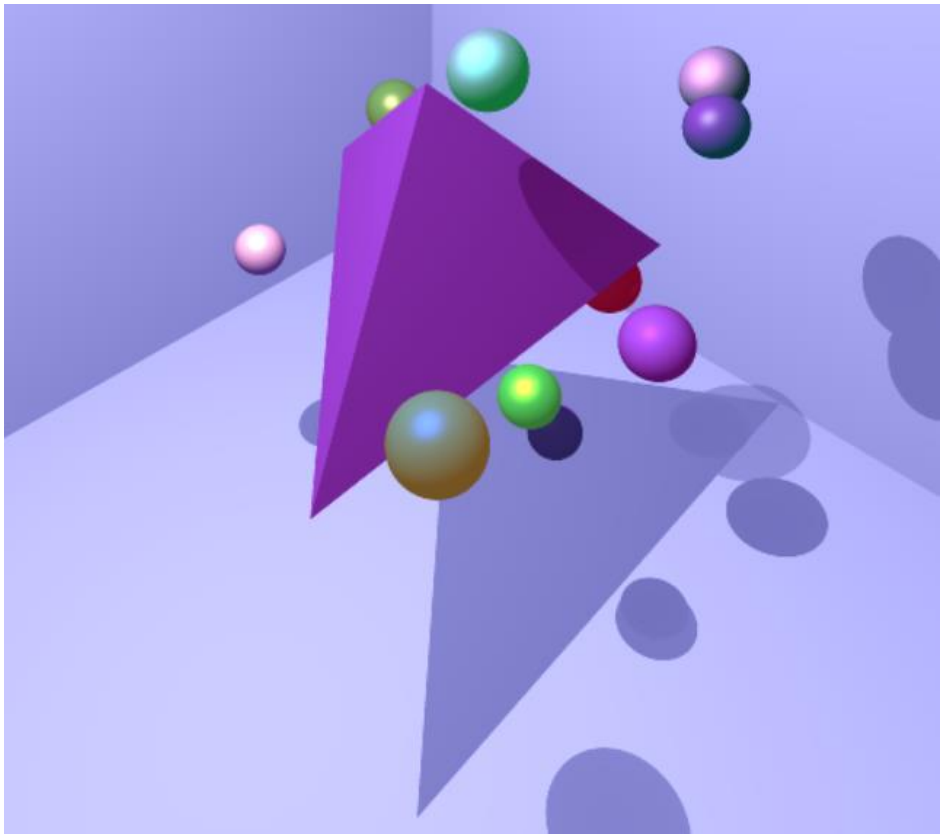
- Due to numerical errors, the intersection point may not be exactly on the surface, but slightly below.
- When this happens and a shadow ray is cast in the light direction, it intersects the surface from which it is cast. We have a case of **self-intersection**.
- A simple and robust solution consists in slightly displacing the origin of the shadow rays in the direction of the surface normal to move it above the surface.
- The amount by which you displace the origin can be tweaked. This value is referred to as **shadow bias**.





Your work

Add (hard) shadows on/off to your raytracer.

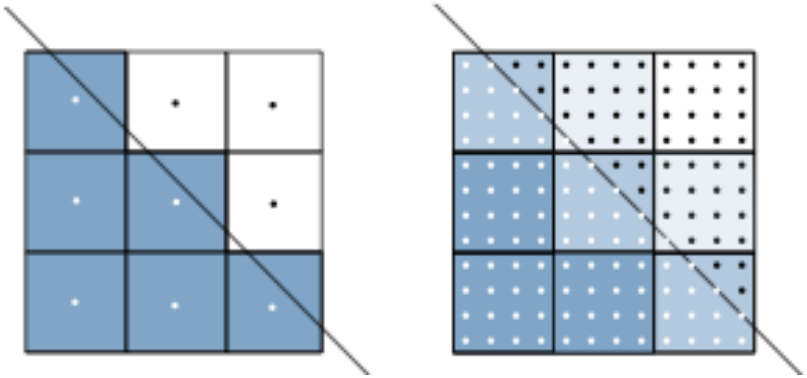


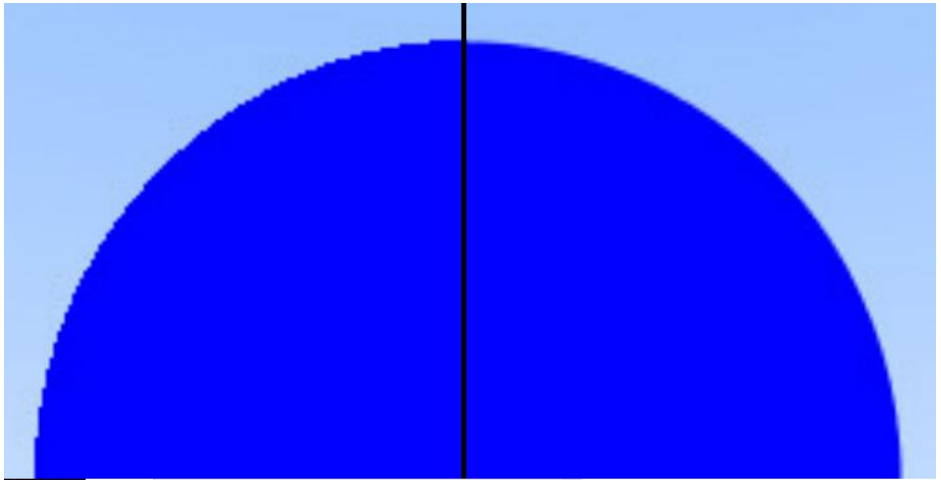
Antialiasing

Antialiasing is a technique used to reduce the appearance of jagged edges.

- Cast multiple rays through each image pixel (**supersampling**)
- The amount of rays per pixel is called **sample rate.**
- Color the pixel with the average ray contribution

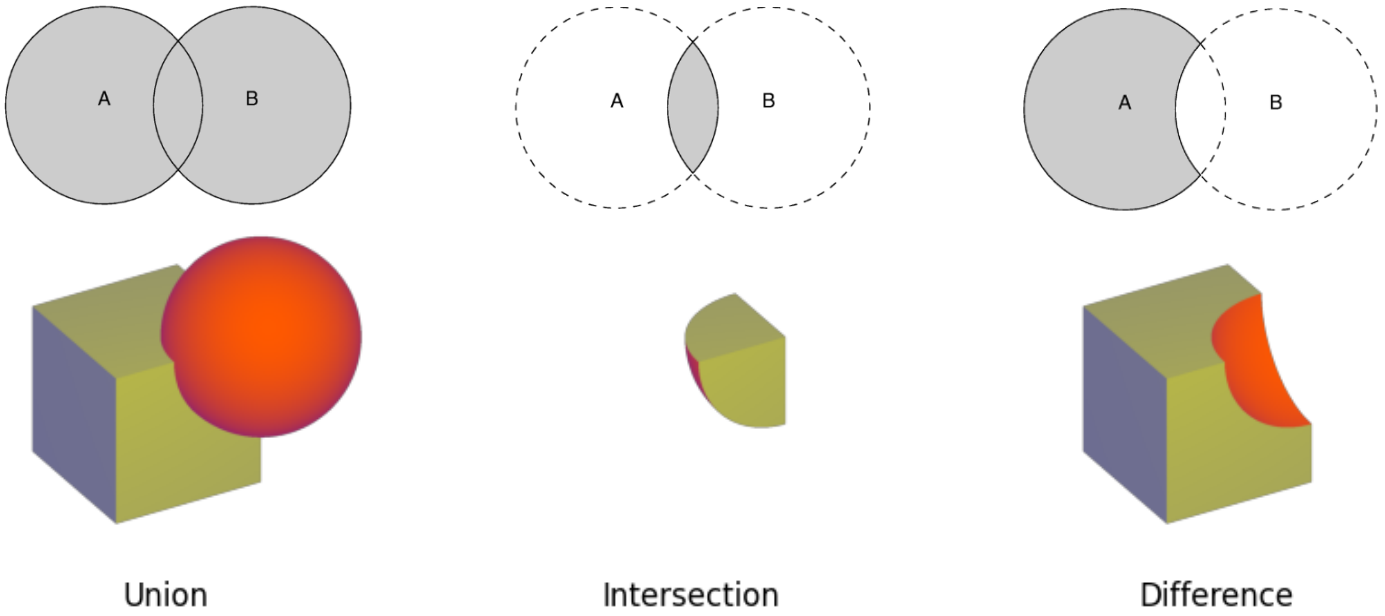
A simple solution, but it considerably multiplies the number of rays and hence computation time.





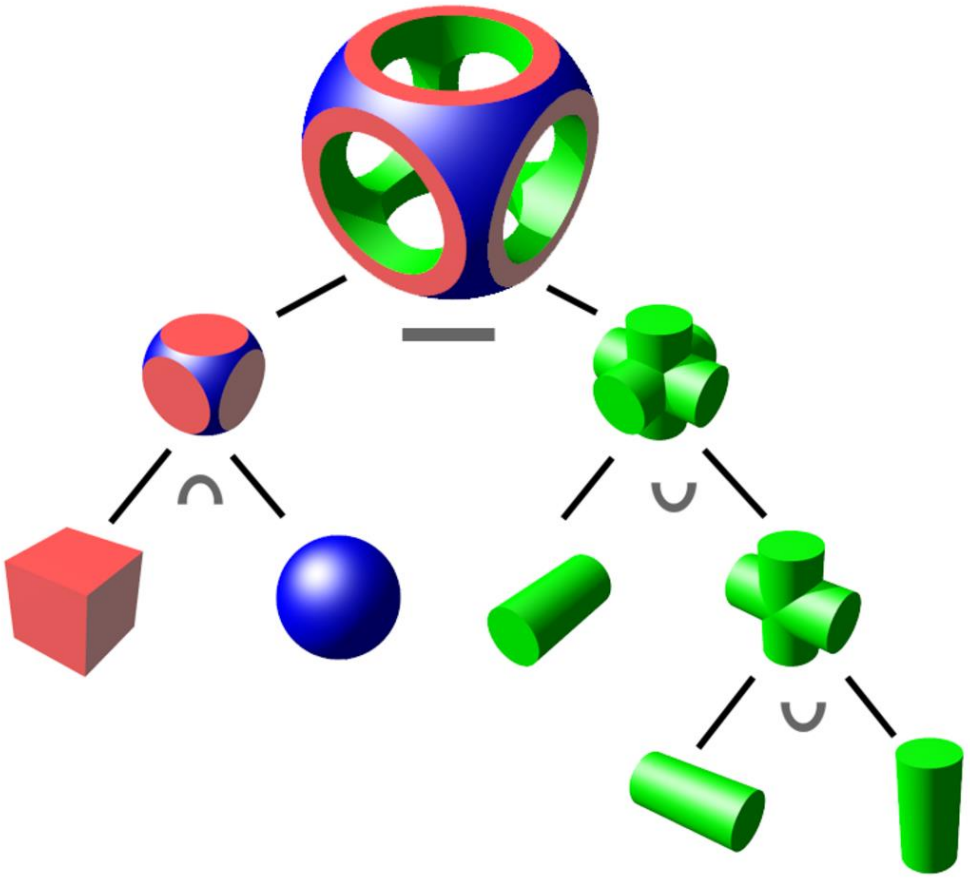
Constructive Solid Geometry

- CSG allows creating new shapes from basic primitives by using boolean set operations:
  - Union (+): combine both shapes
  - Intersection (∩): return the common volume of both shapes
  - Difference (–): subtract the second shape from the first.
- Rendering of CSG is straightforward in a raytracer.



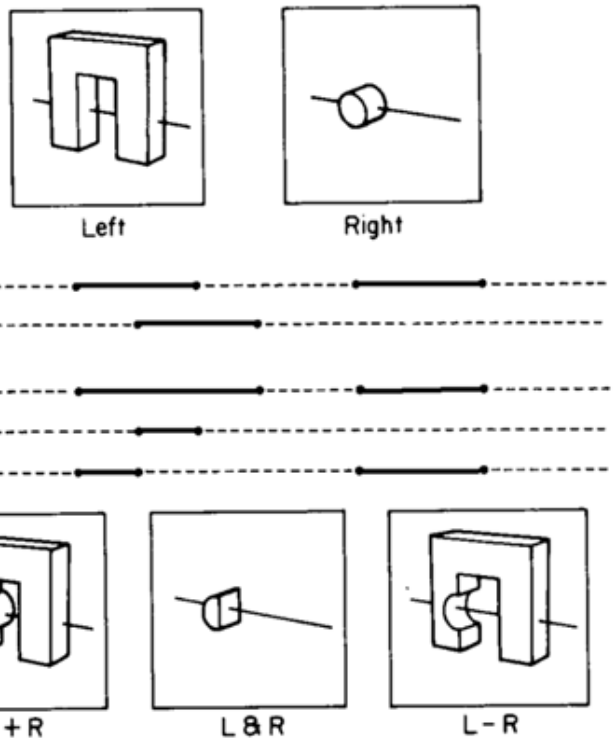
CSG Tree

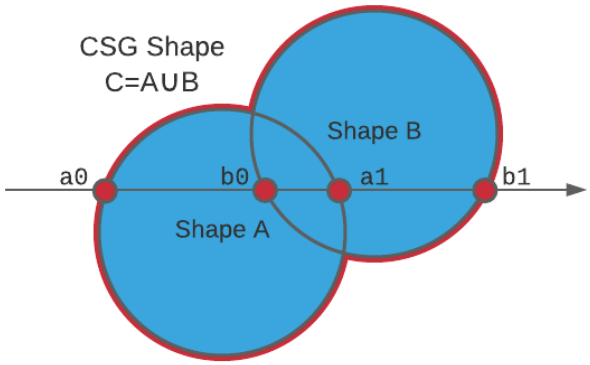
CSG operations can be chained together to form increasingly complex shapes.



Roth Diagrams

- We intersect the ray with a shape that is being operated on and record all hits
- The hit points define intervals that determine the ranges where the ray runs inside or outside of a shape. The intervals are grouped in a **Roth diagram**
- A **CSG filter** takes the two sets of hit points and filters them so that only valid ones remain.
- We can then determine the closest surface that the ray interacts with.





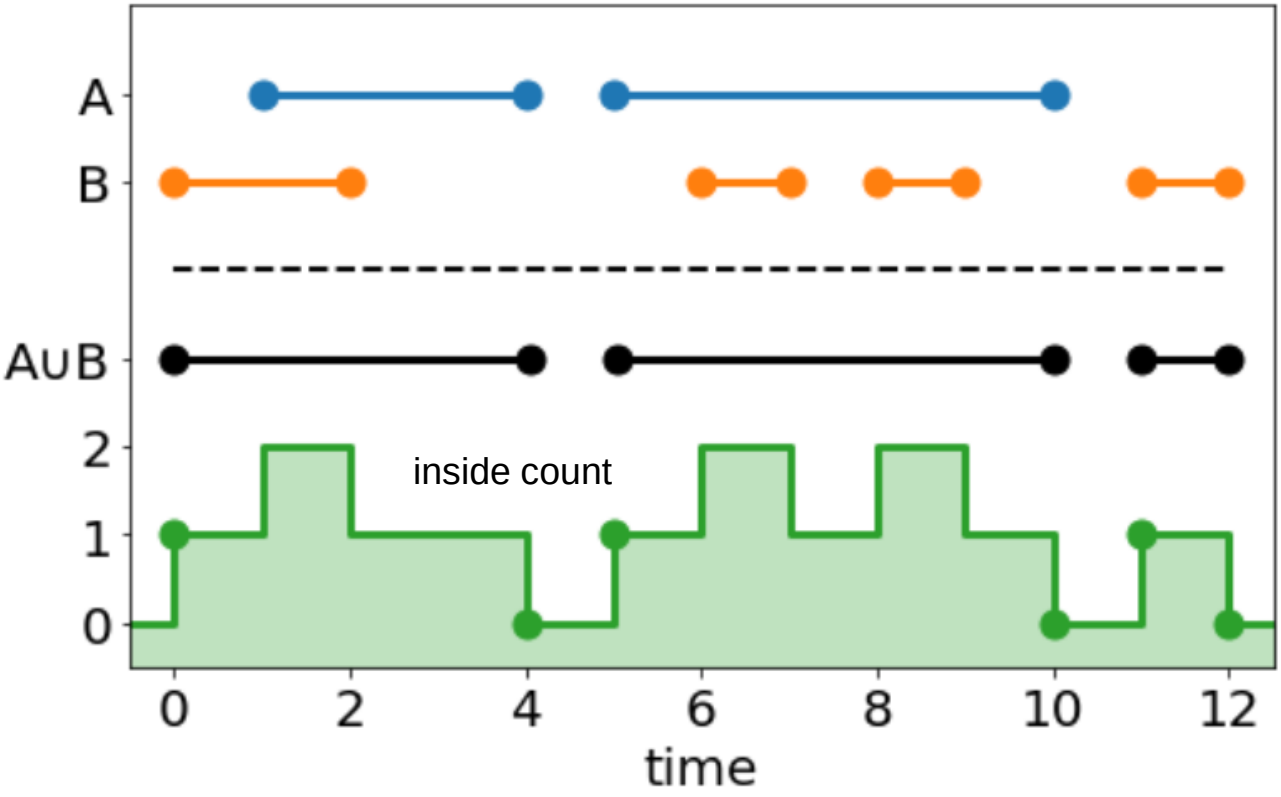

```
A_hits = [a0,a1]
B_hits = [b0,b1]
```

csg_filter → `C_hits = [a0,b1]`

For simplification purposes, we only consider closed surfaces, and we do not consider the special case where the ray is tangent to the surface (one hit instead of two).

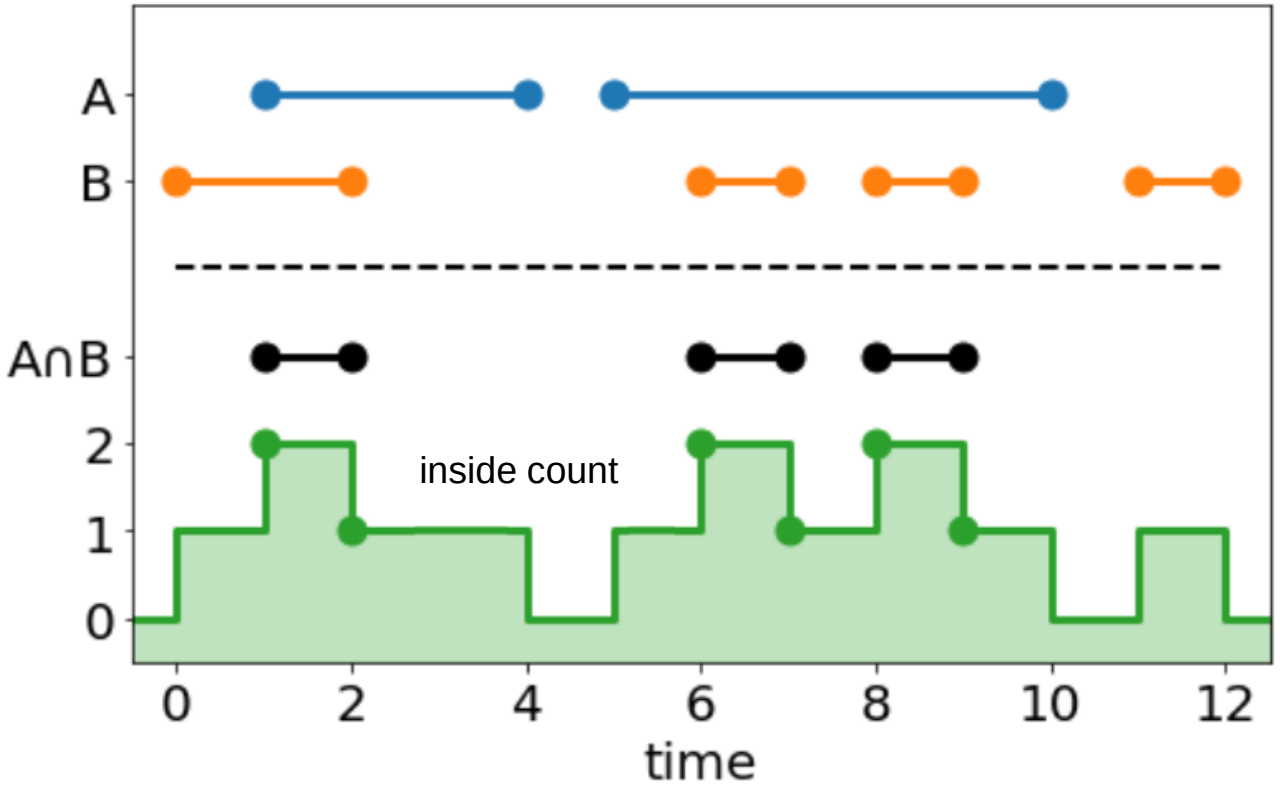
CSG Filter: Union

- On the 1st, 3rd, 5th, … hit of A/B the ray enters A/B => inside count ++
- On the 2nd, 4th, 6th, … hit of A/B the ray leaves A/B => inside count --
- The ray enters A+B when the inside count switches from 0 to 1
- The ray leaves A+B when the inside count switches from 1 to 0



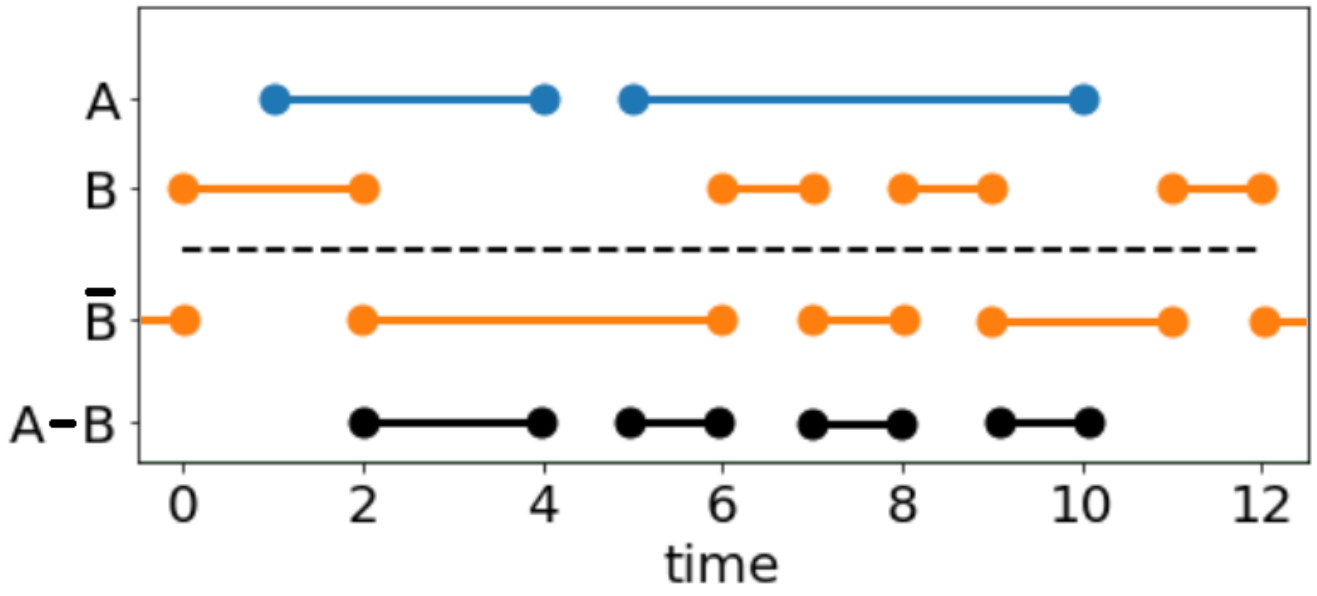
CSG Filter: Intersection

- On the 1st, 3rd, 5th, … hit of A/B the ray enters A/B => inside count ++
- On the 2nd, 4th, 6th, … hit of A/B the ray leaves A/B => inside count --
- The ray enters A ∩ B when the inside count switches from 1 to 2
- The ray leaves A ∩ B when the inside count switches from 2 to 1



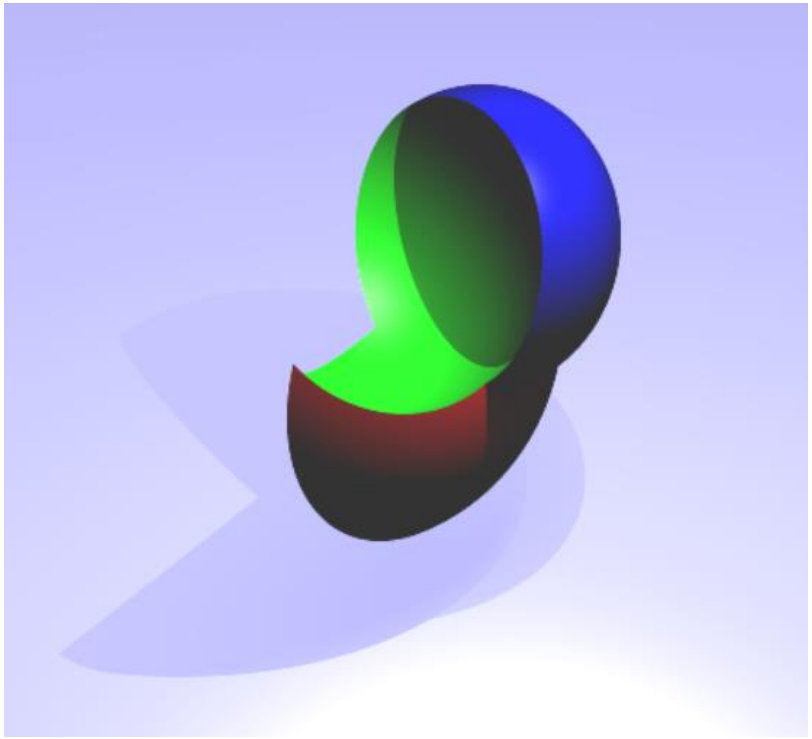
CSG Filter: Difference

- The difference operation can be conveniently computed by a complement and an intersection operation: **A - B = A ∩ compl(B)**



Your work

- Add antialiasing on/off to your raytracer.
- Render the following CSG:
  - Sphere1: center = (-1, 2, 0), radius = 1.5 , color = red
  - Sphere2: center = (1, 2, 0), radius = 1.5 , color = red
  - Sphere3: center = (0, 2.7, -0.3), radius = 0.8 , color = blue
  - Sphere4: center = (0, 2.8, 0.3), radius = 0.8 , color = green
  - CSG = ((Sphere1 ∩ Sphere2) + Sphere3) – Sphere4



```
struct Hit {
    float t;     // distance
    vec3 normal; // surface normal
    int mat;     // surface material index
};

struct Roth {
    int n;
    Hit hit[8]; // max 8 hit points
};

// CSG filters
Roth unionCSG (Roth r1, Roth r2) { ... }
Roth intersectionCSG (Roth r1, Roth r2) { ... }
Roth complementCSG (Roth r) { ... }
Roth differenceCSG (Roth r1, Roth r2) { ... }

Roth raySphere(vec3 rayPos, vec3 rayDir, vec3 sphPos, float sphRad, int mat)
{...} // return the intersection points of a ray with a sphere as hits

float rayCSG(vec3 rayPos, vec3 rayDir, out vec3 intersecPt, out vec3 normal, out
int materialIndex)
{...} // compute Roth of each sphere, assemble CSG, return ray-CSG intersection
```