Topic 11: More ray tracing

Some slides and figures courtesy of Wolfgang Hurst, Karan Singh, Google Images
Some figures from Peter Shirley, “Fundamentals of Computer Graphics”, 3rd Ed.

Ideal specular or mirror reflection

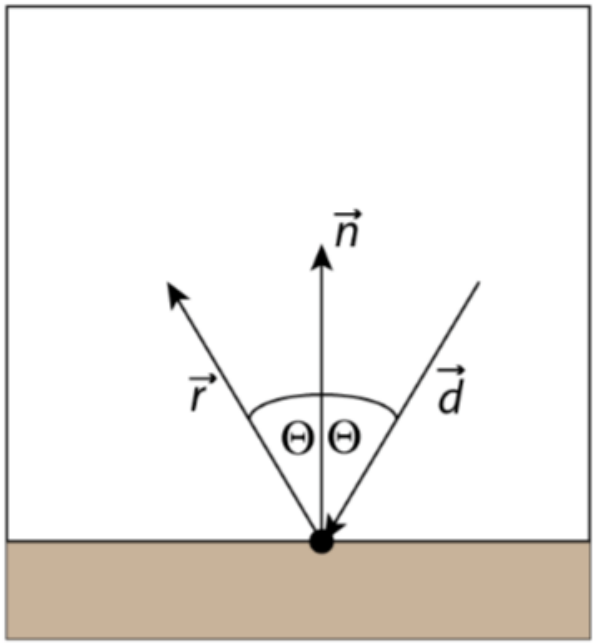
Key characteristic of a mirror:

If looking from direction $\vec{d}$ to a spot on the reflecting surface, the viewer sees the same image as if looking from the surface point in direction $\vec{r}$.

Hence, we need to calculate the reflection vector:

$$\vec{r} = \vec{d} - 2(\vec{d} \cdot \vec{n})\vec{n}$$

Then we shoot a ray in that direction.



Ideal specular or mirror reflection

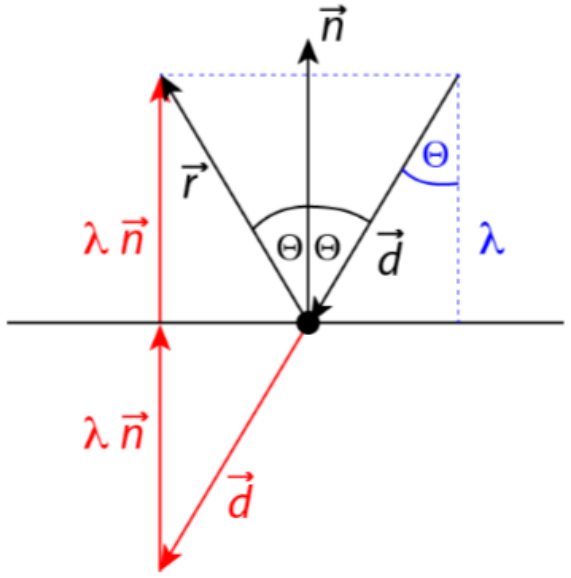
We know this from Phong reflection with light vector l:

$$\vec{r} = -\vec{l} + 2(\vec{l} \cdot \vec{n})\vec{n}$$

But be careful with the direction of $\vec{d}$ when calculating the reflection vector for mirroring:

$$\vec{r} = \vec{d} - 2(\vec{d} \cdot \vec{n})\vec{n}$$

Also: we need to include a max. recursion depth to avoid "infinite bouncing" of rays.

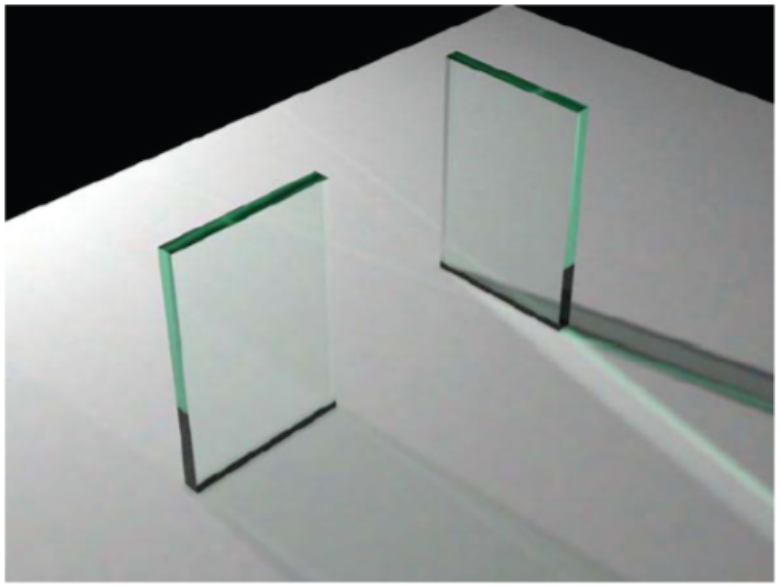


$$\lambda = \cos\theta \|\vec{n}\| \|\vec{d}\| = -\vec{d} \cdot \vec{n}$$

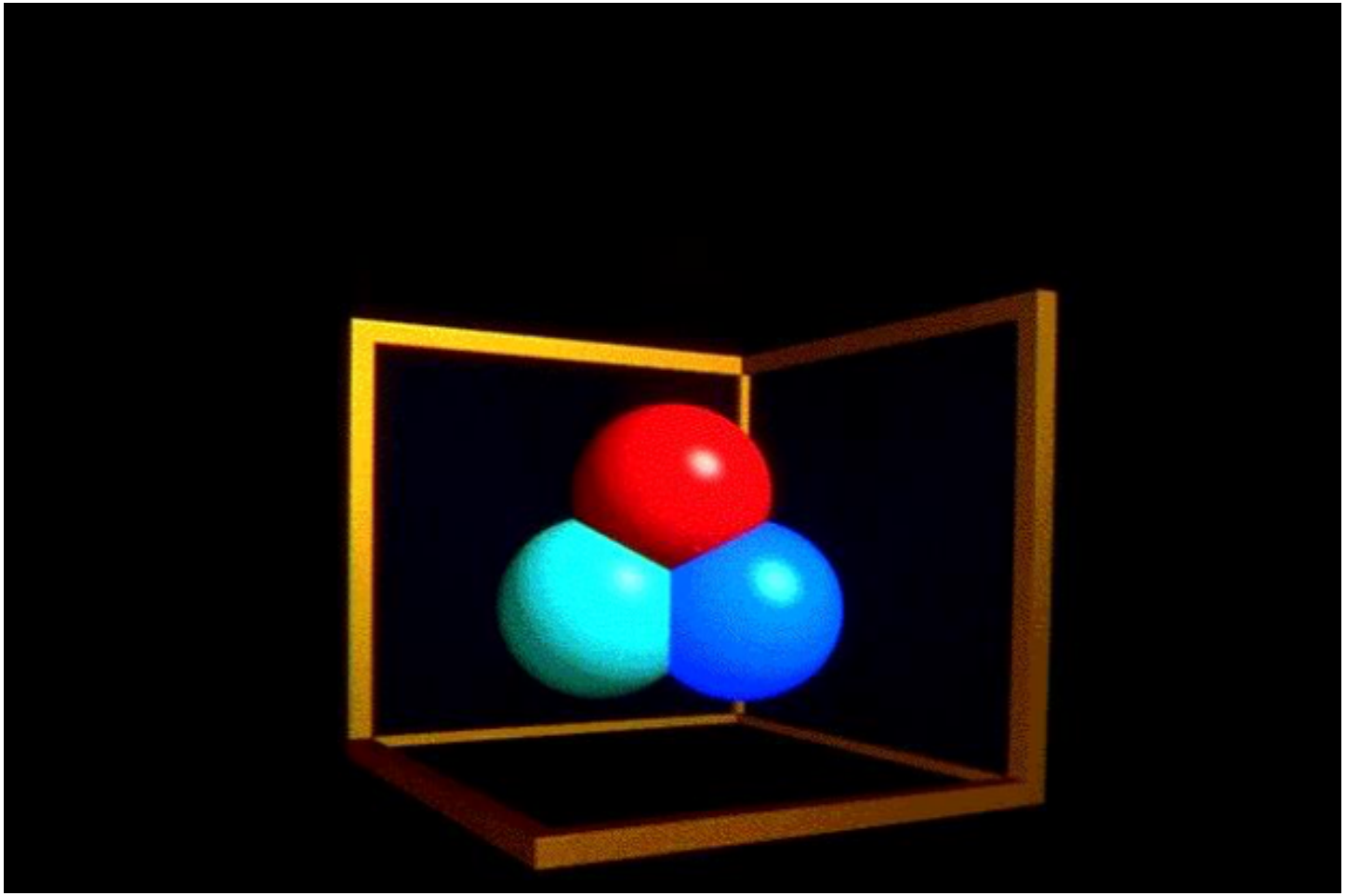
Refraction

Light traveling from one transparent medium into another one is refracted.

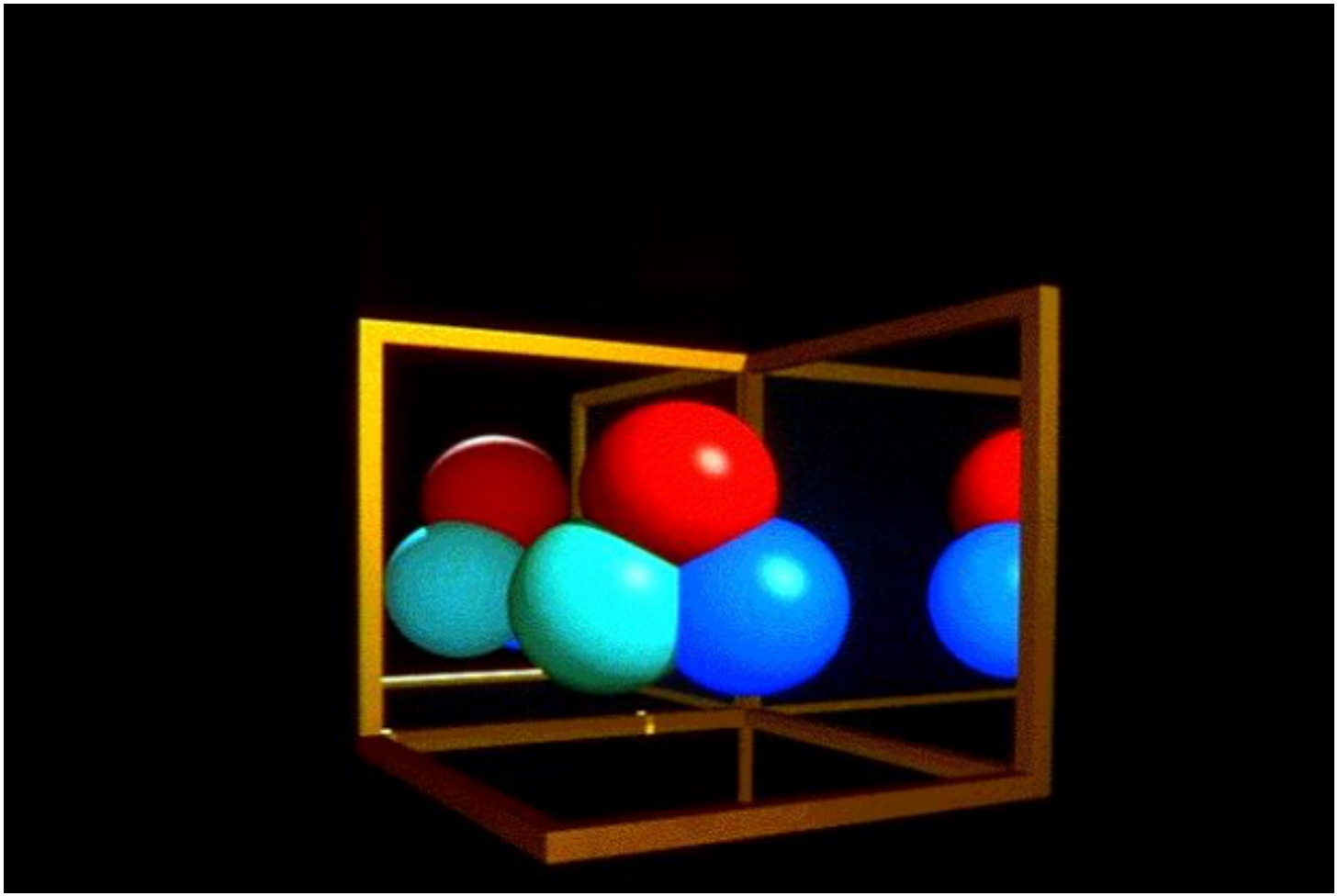
To consider this in ray tracing, we need to know the refraction angles.



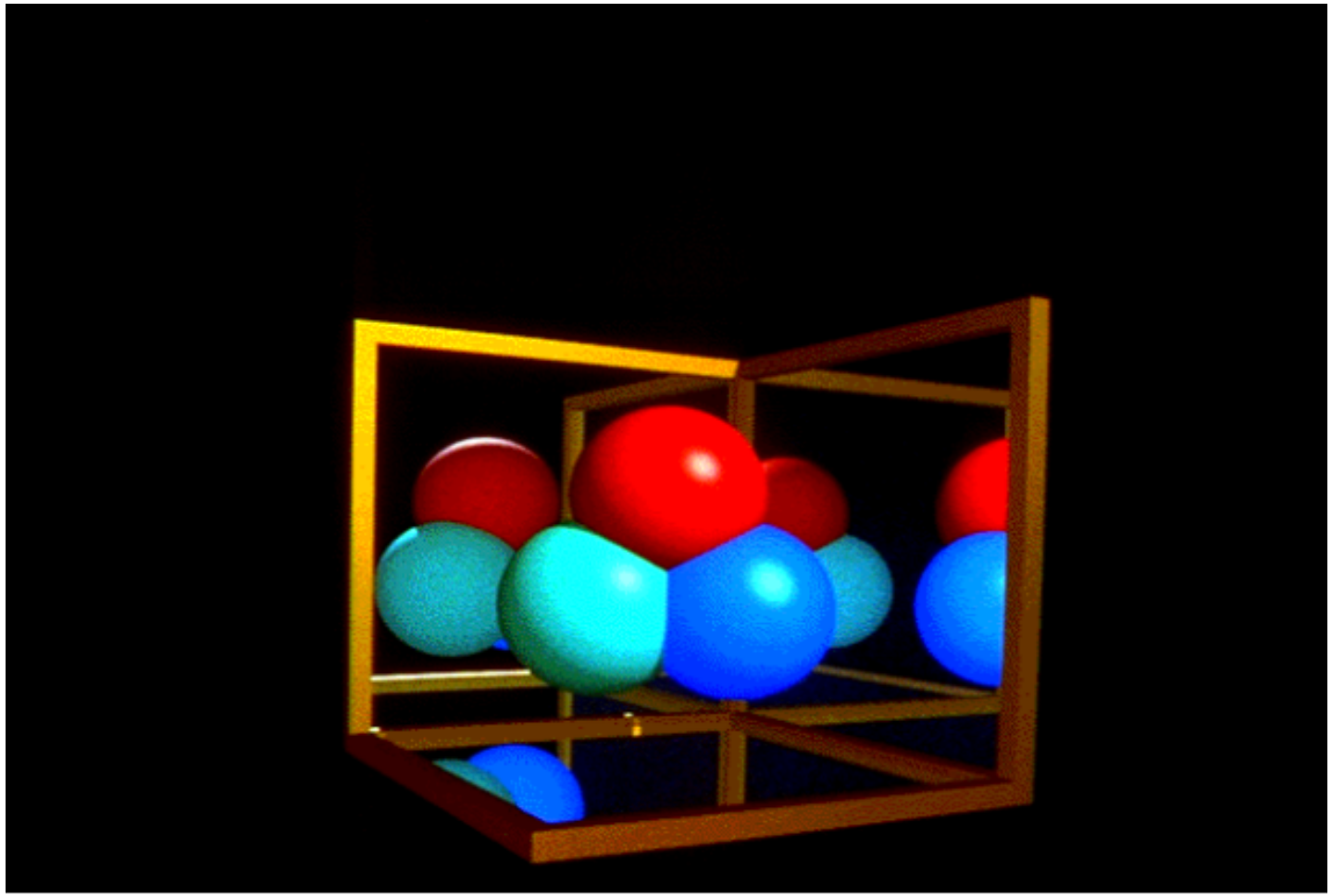
No reflection



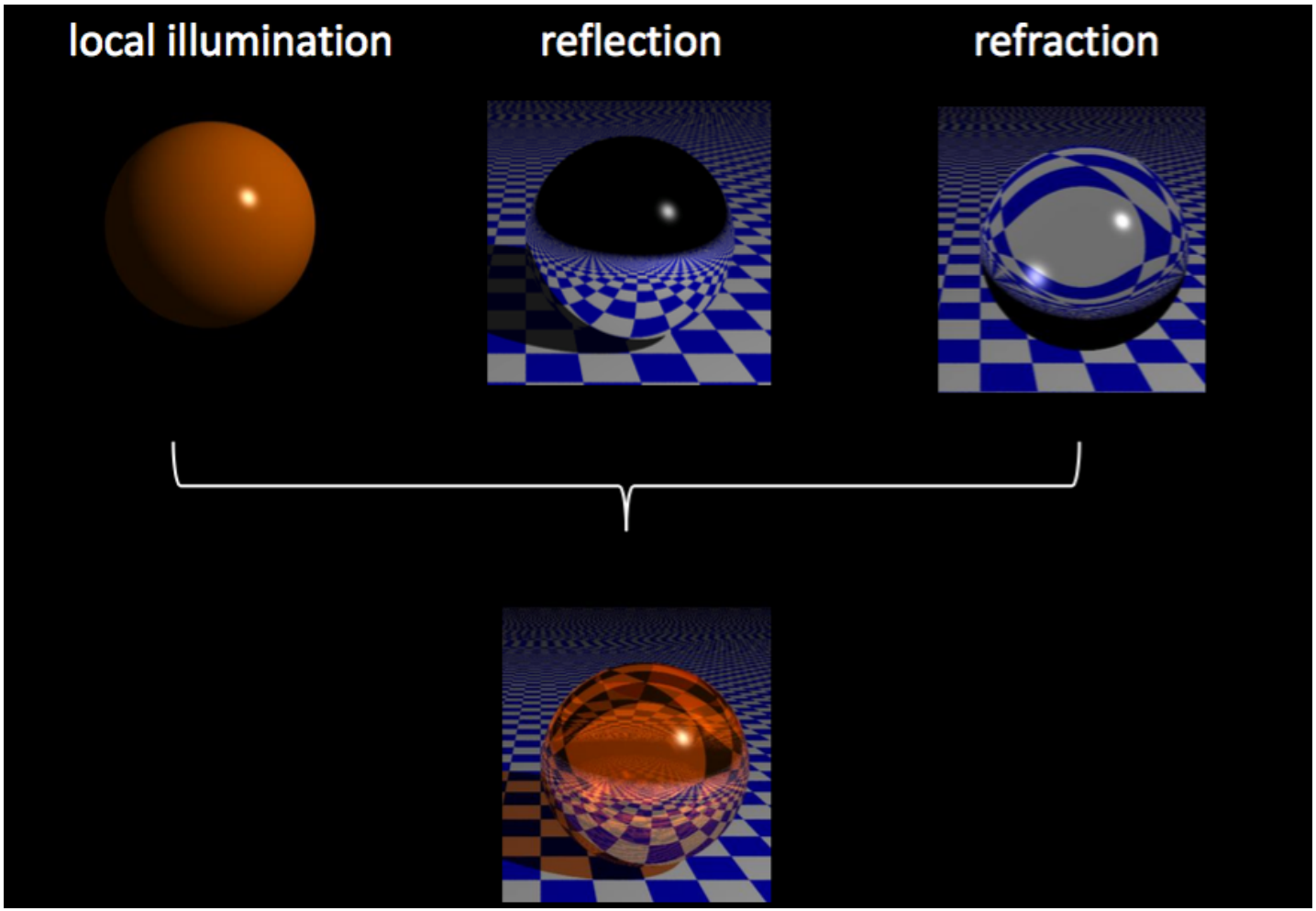
Single reflection



Double reflection



Components



Whitted ray-tracing algorithm

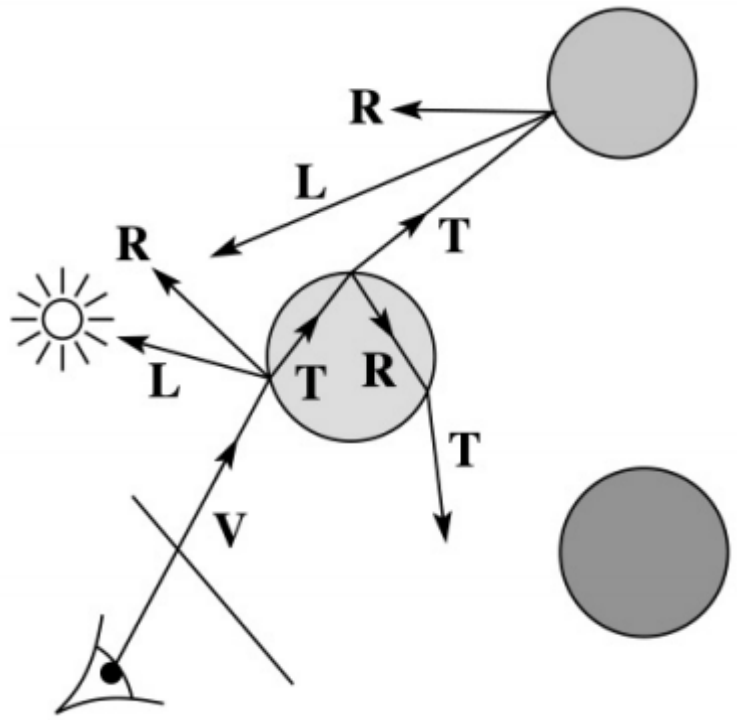
1. For each pixel, trace a **primary ray** in direction V to the first visible surface.

2. For each intersection, trace **secondary rays**:

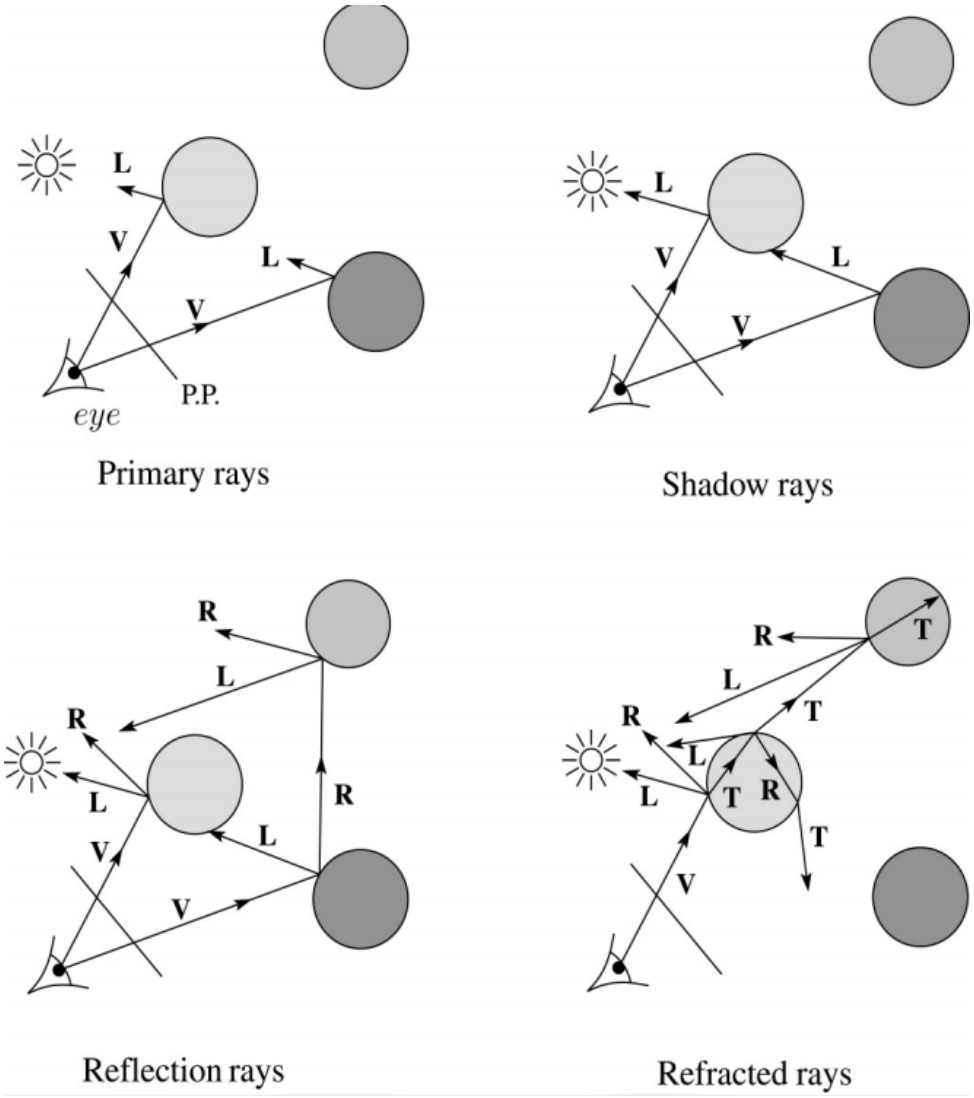
- Shadow rays in directions Li to light sources

- Reflected ray in direction R.

- Refracted ray or transmitted ray in direction T.



Whitted ray-tracing algorithm

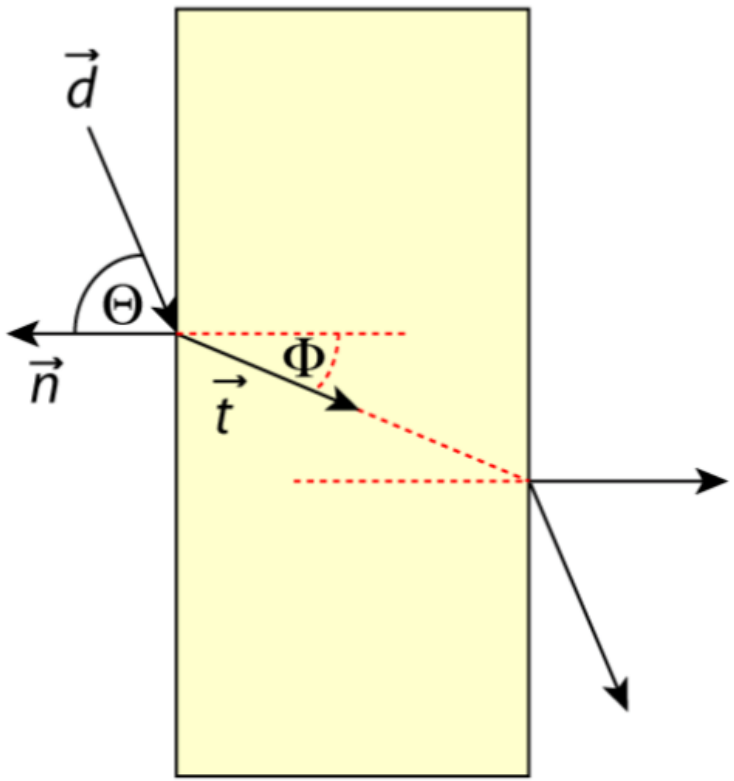


Snell's law

According to Snell's law, the angles before and after refraction are related as follows:

$$n \sin\theta = n_t \sin\phi$$

where n and n_t are the refractive indices of the source and target media, respectively, and θ and ϕ the angles indicated in the image.

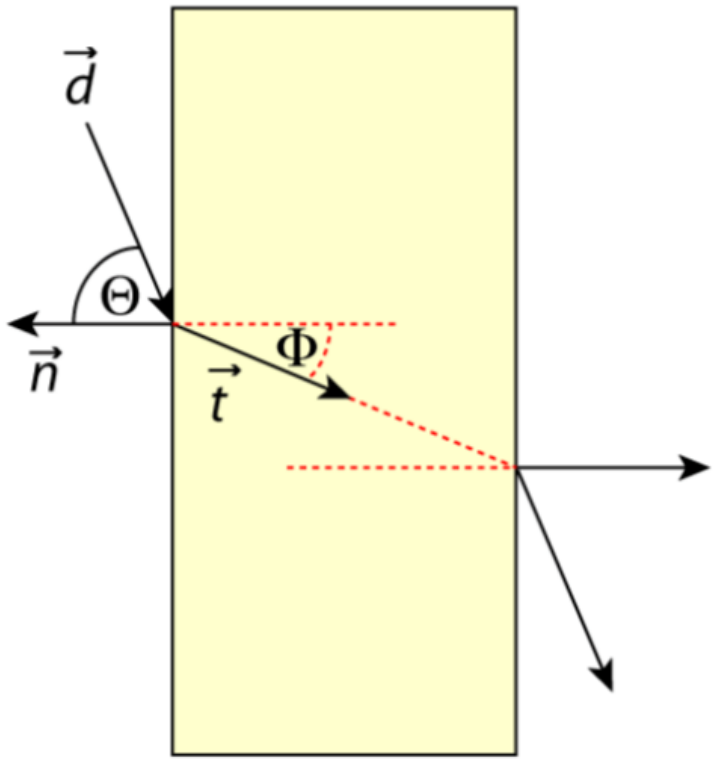


Getting rid of sines

An equation that relates sines of the angles θ and ϕ is not as convenient as an equation that relates the cosines of the angles.

Fortunately, we can eliminate the sines using the trigonometric identity:

$$\sin^2 \alpha + \cos^2 \alpha = 1$$



Getting rid of sines

Trigonometric identity: $\sin^2\alpha + \cos^2\alpha = 1$

$$\rightsquigarrow \sin^2\alpha = 1 - \cos^2\alpha \quad \textbf{(i)}$$

Snell's law: $n\sin\theta = n_t \sin\phi$

$$\rightsquigarrow \sin\phi = \frac{n}{n_t}\sin\theta$$

$$\rightsquigarrow \sin^2\phi = \frac{n}{n_t}^2 \sin^2\theta \quad \textbf{(ii)}$$

(i) with $\alpha = \phi$ in **(ii)** gives us:

$$\cos^2\phi = 1 - \frac{n}{n_t}^2 \sin^2\theta \quad \textbf{(iii)}$$

(i) with $\alpha = \theta$ in **(iii)** gives us:

$$\cos^2\phi = 1 - \frac{n}{n_t}^2 (1 - \cos^2\theta)$$

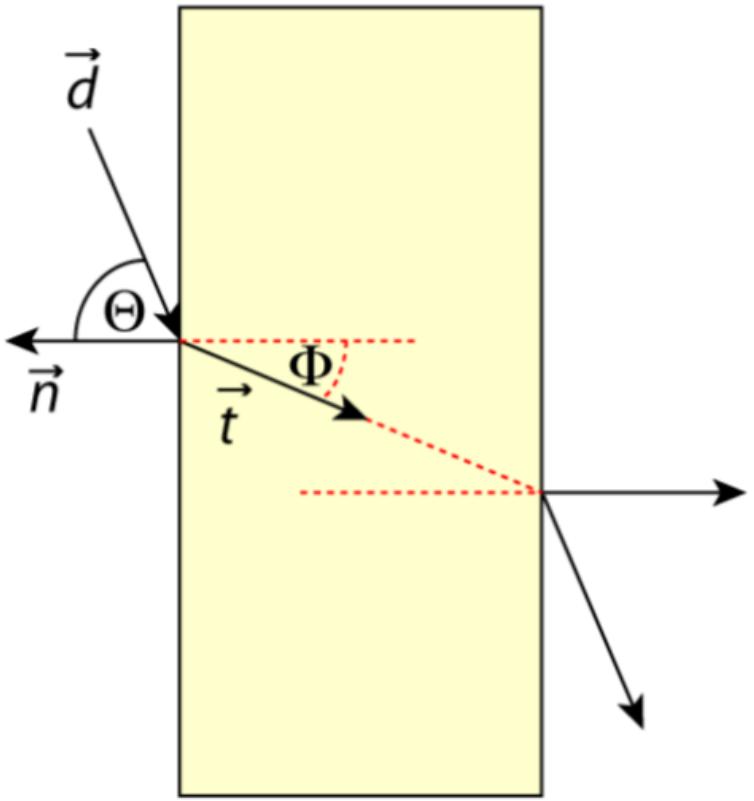
Getting rid of sines

Hence, we can rewrite Snell's law

$$n \sin\theta = n_t \sin\phi$$

with cosines instead of sines

$$\cos^2\phi = 1 - \frac{n^2(1-\cos^2\theta)}{n_t^2}$$

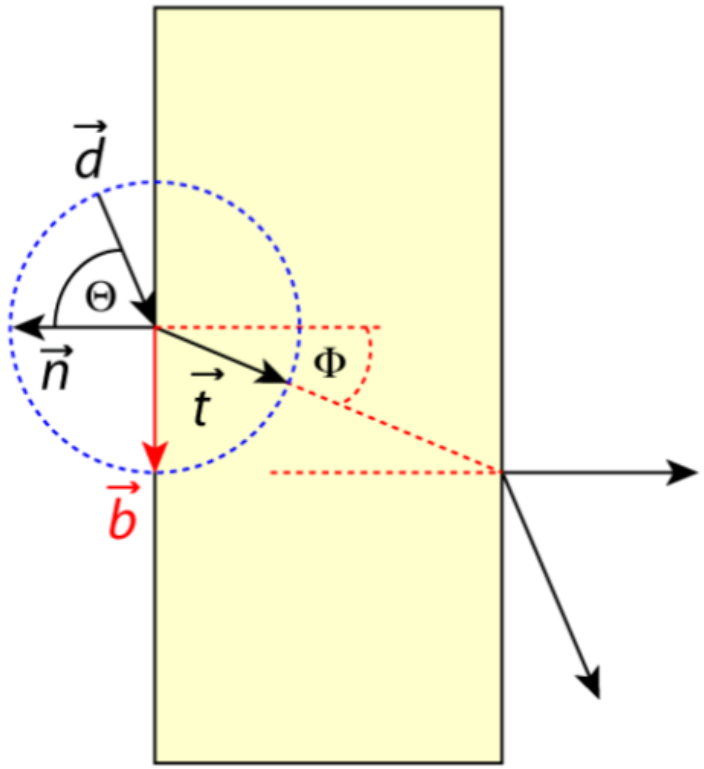


Constructing an orthonormal basis

How do we find the refracted vector $\vec{t}$?

First, notice that $\vec{t}$ lies in the plane spanned by $\vec{d}$ and $\vec{n}$.

If the incoming vector $\vec{d}$ and the normal $\vec{n}$ are normalized, we can express $\vec{t}$ in an orthonormal basis in this plane using an appropriate vector $\vec{b}$.



Finding the refraction vector

The refraction vector $\vec{t}$ is a linear combination of $\vec{b}$ and $-\vec{n}$:

$$\vec{t} = x_t\vec{b} + y_t(-\vec{n})$$

From the image we see that

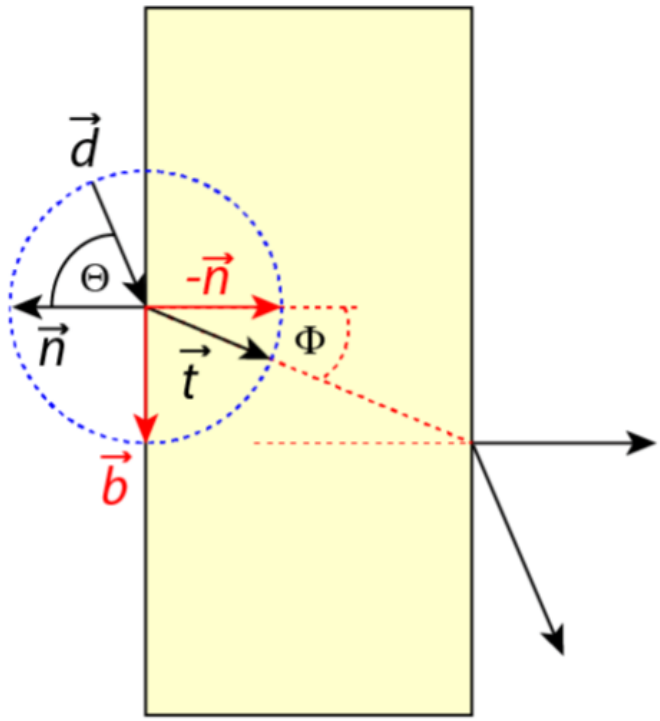
$$x_t = \sin\phi \text{ and } y_t = \cos\phi$$

Hence, we have

$$\vec{t} = \vec{b}\sin\phi - \vec{n}\cos\phi$$

Likewise, we get

$$\vec{d} = \vec{b}\sin\theta - \vec{n}\cos\theta$$



Finding the refraction vector

The refraction vector $\vec{t}$ is a linear combination of $\vec{b}$ and $-\vec{n}$:

$$\vec{t} = x_t\vec{b} + y_t(-\vec{n})$$

From the image we see that

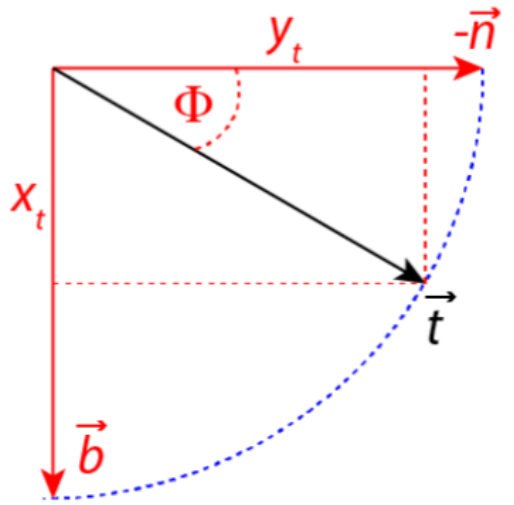
$$x_t = \sin\phi \text{ and } y_t = \cos\phi$$

Hence, we have

$$\vec{t} = \vec{b}\sin\phi - \vec{n}\cos\phi$$

Likewise, we get

$$\vec{d} = \vec{b}\sin\theta - \vec{n}\cos\theta$$



Finding the refraction vector

We have

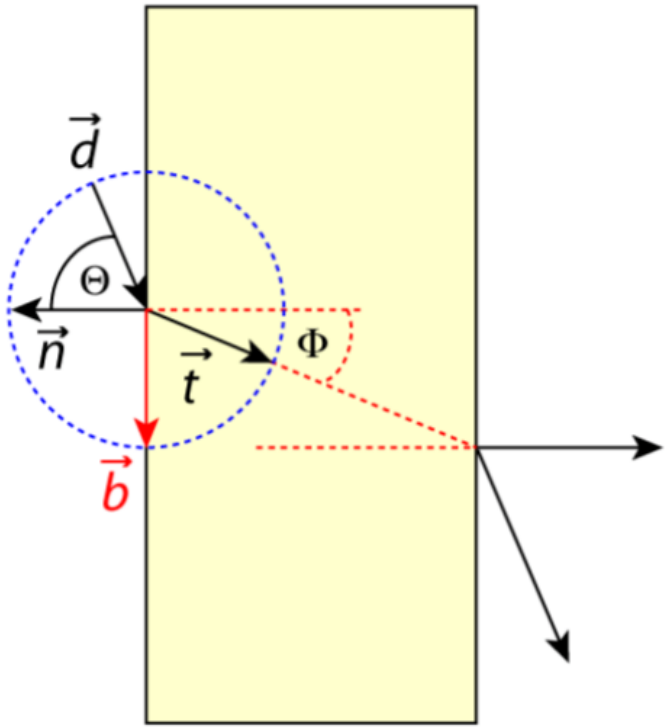
$$\begin{aligned}\vec{t} &= \vec{b}\sin\phi - \vec{n}\cos\phi \\ \vec{d} &= \vec{b}\sin\theta - \vec{n}\cos\theta\end{aligned}$$

So we can solve for $\vec{b}$:

$$\vec{b} = \frac{\vec{d}+\vec{n}\cos\theta}{\sin\theta}$$

and for $\vec{t}$:

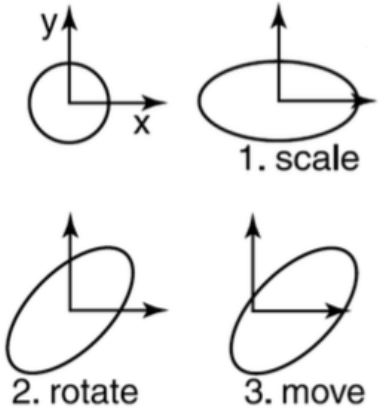
$$\begin{aligned}\vec{t} &= \frac{\sin\phi(\vec{d}+\vec{n}\cos\theta)}{\sin\theta} - \vec{n}\cos\phi \\ &= \frac{n(\vec{d}+\vec{n}\cos\theta)}{n_t} - \vec{n}\cos\phi \\ &= \frac{n(\vec{d}-\vec{n}(\vec{d}\cdot\vec{n}))}{n_t} - \vec{n}\sqrt{1-\frac{n^2(1-(\vec{d}\cdot\vec{n})^2)}{n_t^2}}\end{aligned}$$

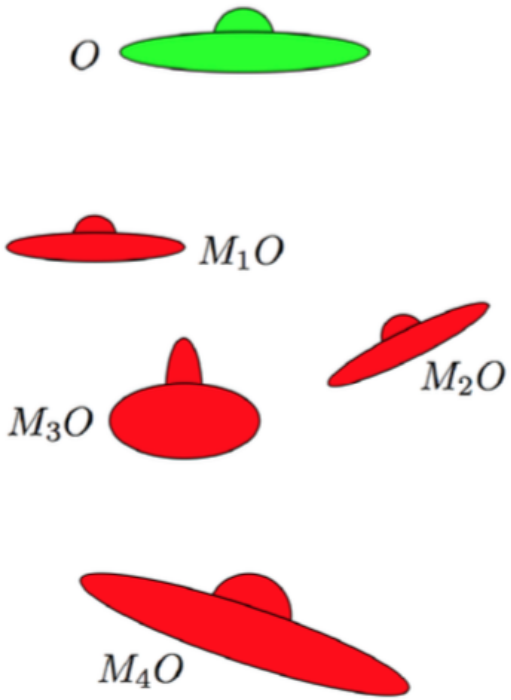


Copying and transforming objects

Instancing is an elegant technique to place various transformed copies of an object in a scene.

Expl.: circle → elipse

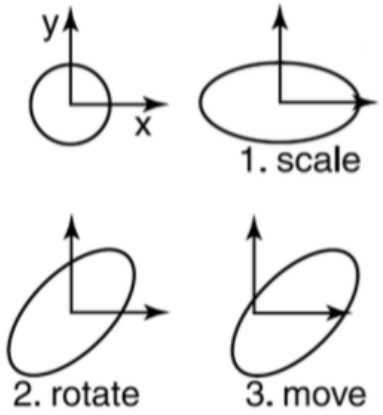


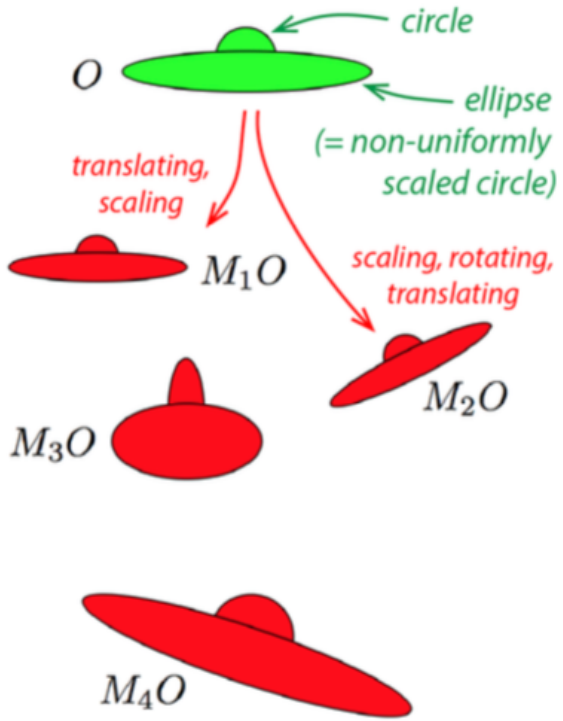


Copying and transforming objects

Instancing is an elegant technique to place various transformed copies of an object in a scene.

Expl.: circle → elipse



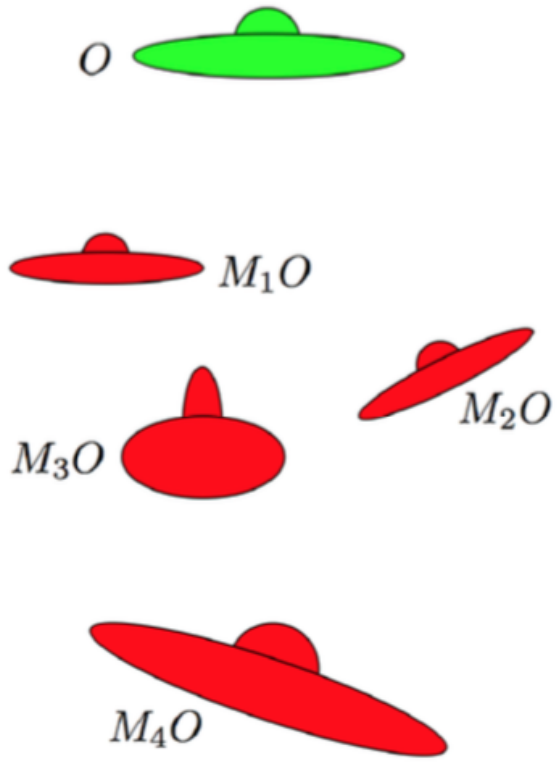


Copying and transforming objects

Instead of making actual copies, we simply store a reference to a base object, together with a transformation matrix.

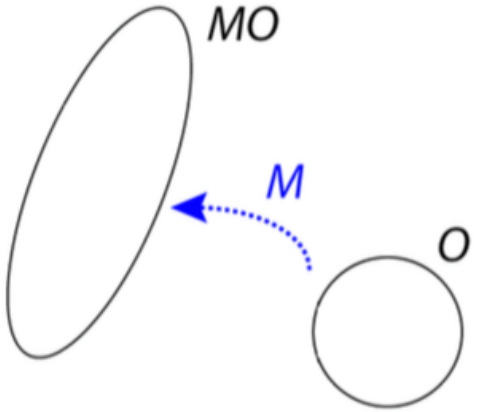
That can save us lots of storage.

Hmm, but how do we compute the intersection of a ray with a randomly rotated elipse?



Ray-instance intersection

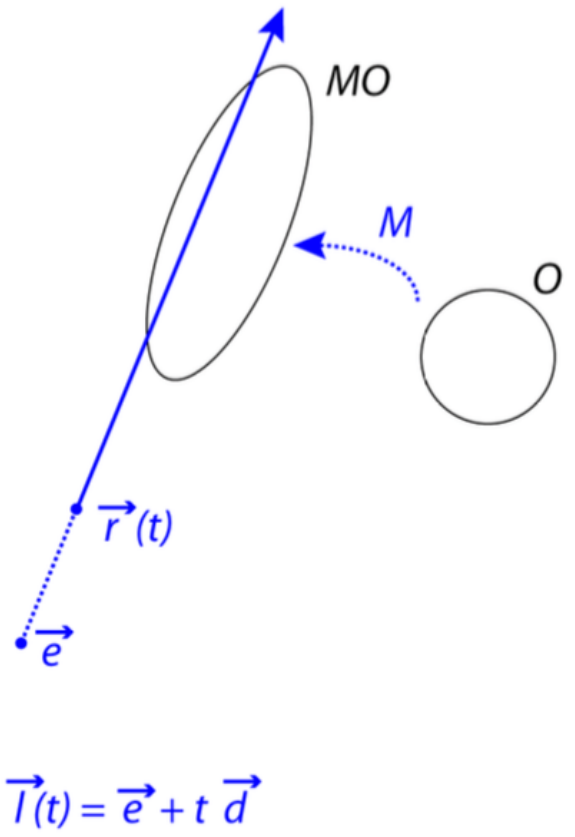
Assume an object O that is used to create an object MO via instancing.



Ray-instance intersection

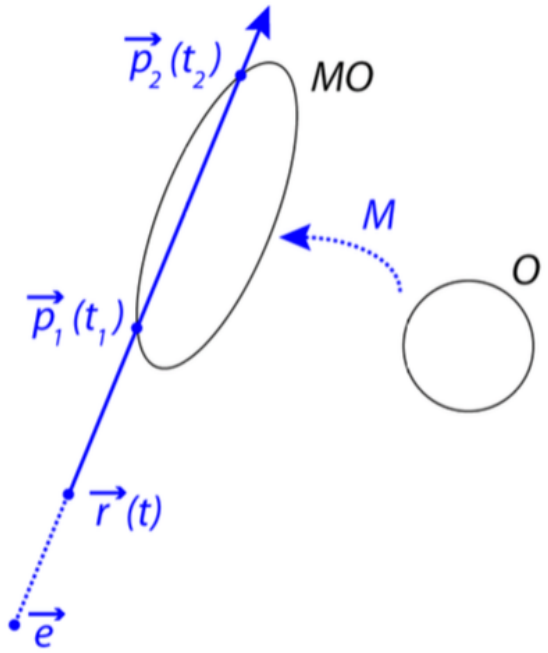
Now, we want to create the intersection of MO with the ray $\vec{r}(t)$, which in turn is defined by the line

$$\vec{l}(t) = \vec{e} + t\vec{d}.$$



Ray-instance intersection

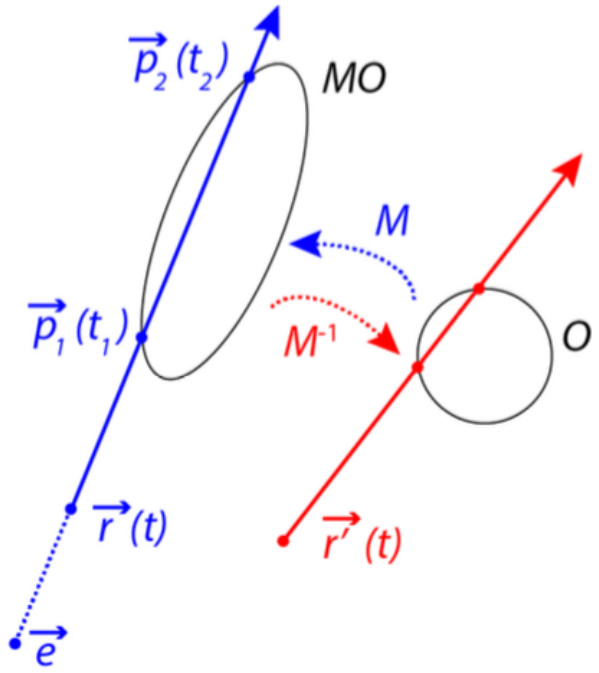
Fortunately, such
complicated intersection tests
(e.g. ray/ellipsoid) can often be
replaced by much simpler tests
(e.g. ray/sphere).



$$\vec{l}(t) = \vec{e} + t\,\vec{d}$$

Ray-instance intersection

To determine the intersections $\vec{p_i}$ of a ray $\vec{r}$ with the instance MO, we first compute the intersections $\vec{p'_i}$ of the inverse transformed ray $M^{-1}\vec{r}$ and the original object O.



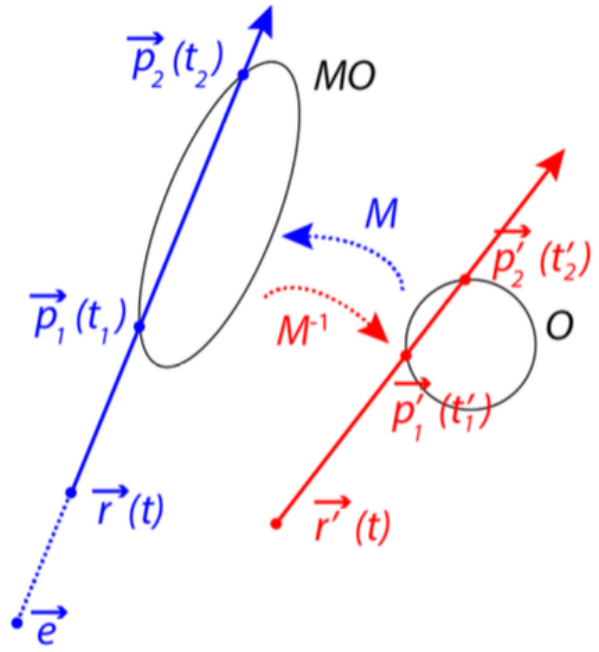
$$\vec{l}(t) = \vec{e} + t\,\vec{d}$$

Ray-instance intersection

The points $\vec{p_i}$ are then simply

$$M\vec{p_i'} \quad \text{or} \quad \vec{l}(t_i')$$

because the linear transformation preserves relative distances along the line.

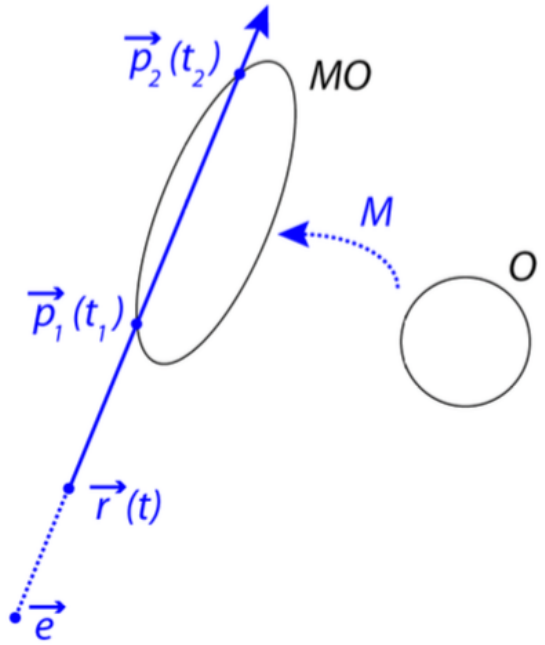


$$\vec{l}(t) = \vec{e} + t\,\vec{d}$$

Ray-instance intersection

Two pitfalls:

- The direction vector of the ray should *not* be normalized
- Surface normals transform differently!
  $\rightarrow$ use $(M^{-1})^T$ instead of M for normals



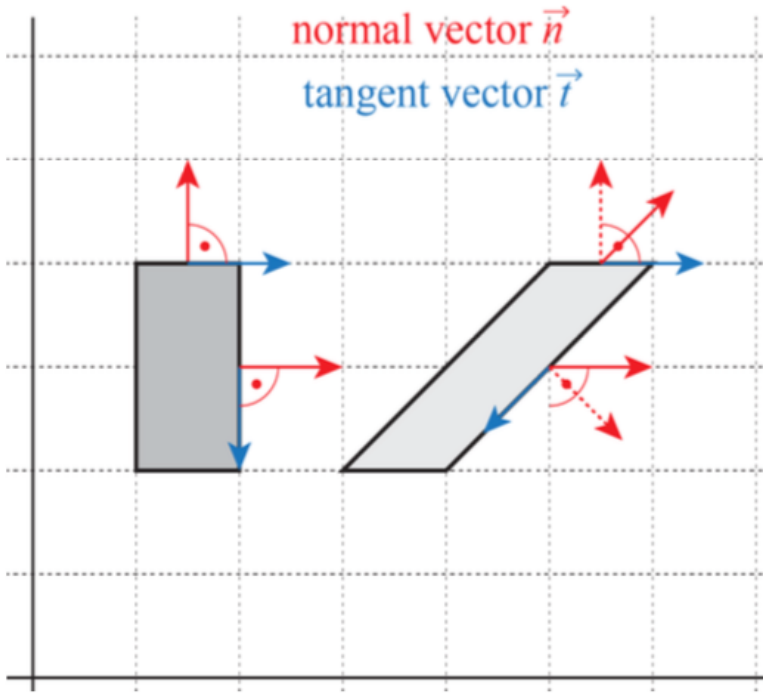
$$\vec{l}(t) = \vec{e} + t\,\vec{d}$$

Transforming normal vectors

Unfortunately, normal vectors are not always transformed properly.

E.g. look at shearing, where tangent vectors are correctly transformed but normal vectors not.

To transform a normal vector $\vec{n}$ correctly under a given linear transformation A, we have to apply the matrix $(A^{-1})^T$. Why?



Transforming normal vectors

We know that tangent vectors are tranformed correctly: $A\vec{t} = \vec{t_A}$.

But this is not necessarily true for normal vectors: $A\vec{n} \neq \vec{n_A}$.

Goal: find matrix N_A that transforms $\vec{n}$ correctly, i.e. $N_A\vec{n} = \vec{n_N}$ where $\vec{n_N}$ is the correct normal vector of the transformed surface.

Because our original normal vector $\vec{n}^T$ is perpendicular to the original tangent vector $\vec{t}$, we know that:

$$\vec{n}^T\vec{t} = 0.$$

This is the same as

$$\vec{n}^T I\vec{t} = 0$$

which is is the same as

$$\vec{n}^T A^{-1}A\vec{t} = 0$$

Transforming normal vectors

Because $A\vec{t} = \vec{t_A}$ is our correctly transformed tangent vector,
we have

$$\vec{n}^T A^{-1} \vec{t_A} = 0$$

Because their scaler product is 0, $\vec{n}^T A^{-1}$ must be orthogonal to it.
So, the vector we are looking for must be

$$\vec{n}_N^T = \vec{n}^T A^{-1}.$$

Because of how matrix multiplication is defined,
this is a transposed vector. But we can rewrite this to

$$\vec{n}_N = (\vec{n}^T A^{-1})^T.$$

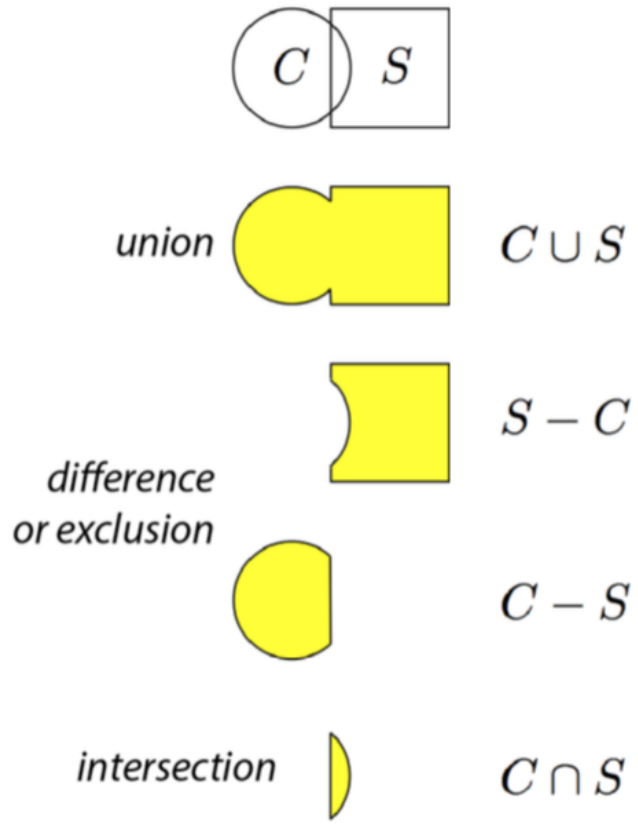
And if you remember that $(AB)^T = B^T A^T$, we get

$$\vec{n}_N = (A^{-1})^T \vec{n}$$

Constructive solid geometry

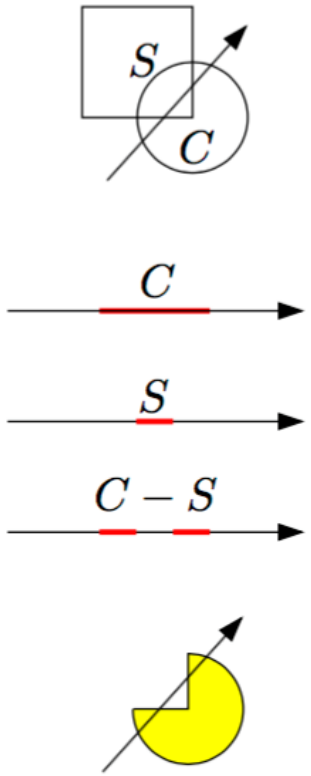
To model our scenes in ray tracing, we can basically use any object that allows us to calculate its intersection with a 3D line.

Using Constructive Solid Geometry (CSG), we can build complex objects from simple ones with set operations.



Intersection and CSG

Instead of actually constructing the objects, we can calculate ray-object intersections with the original objects and perform set operations on the resulting intervals.

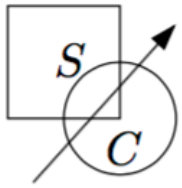


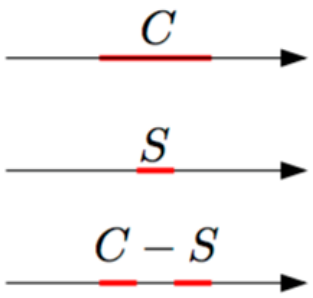
Intersection and CSG

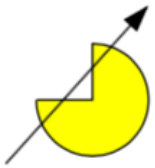
For every base object, we maintain an interval (or set of intervals) representing the part of the ray inside the object.

The intervals for combined objects are computed with the same set operations that are applied to the base objects.

The borders of the resulting intervals are our intersection points.





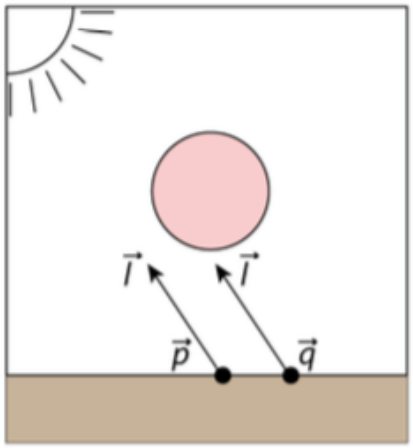


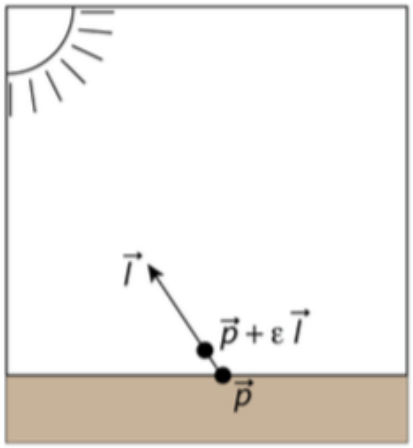
Shadow feelers

Shadows can be implemented fairly easy by so called shadow feelers / rays.

- Shoot a shadow ray $\vec{p} + t\vec{l}$ from a point $\vec{p}$ towards a light source $\vec{l}$.
- If ray hits an object, $\vec{p}$ is in the shadow. Otherwise it's not.

Because of potential precision issues, we often look at point $\vec{p} + \epsilon\vec{l}$ instead of $\vec{p}$.





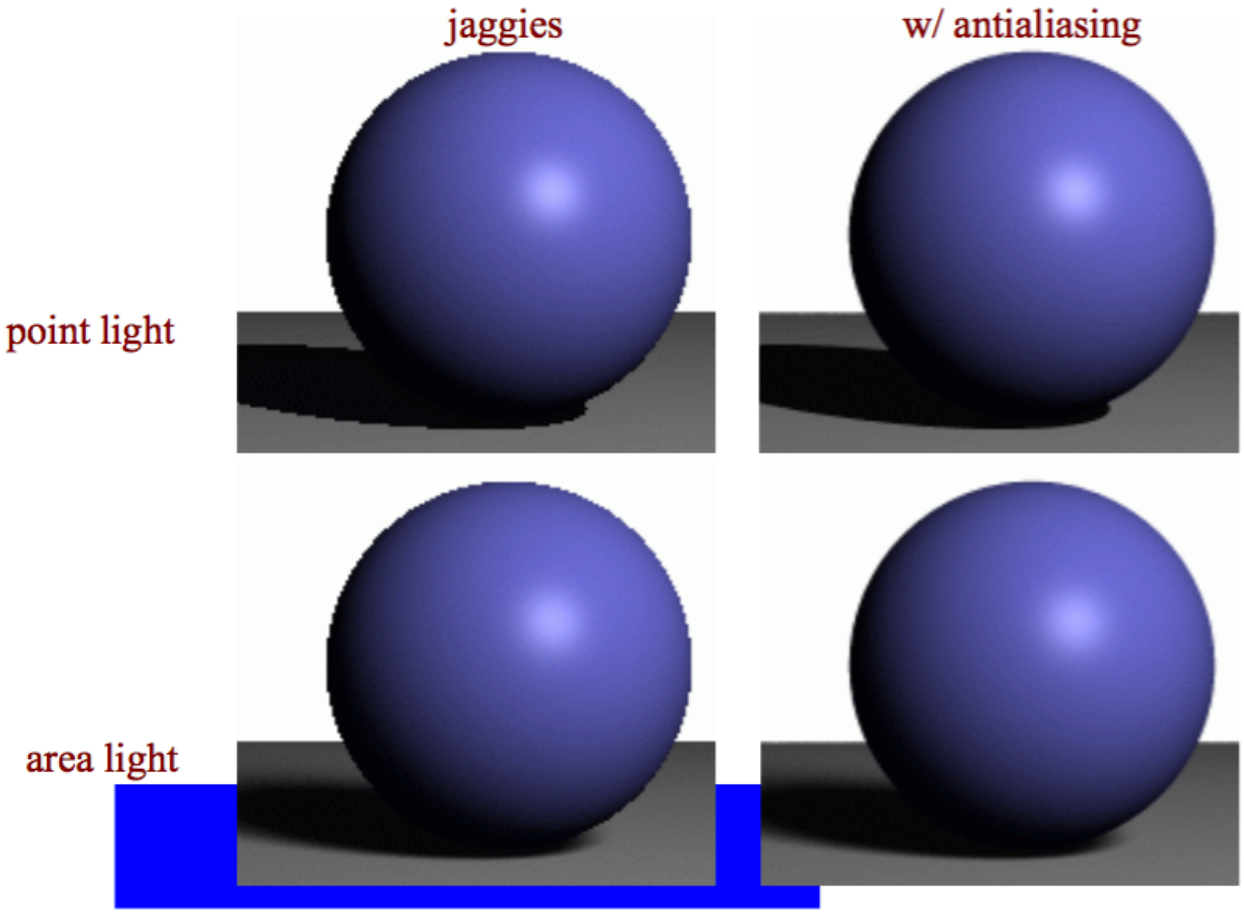
Distributed Ray Tracing

- Not ray tracing on a distributed system…
- Based on randomly distributed oversampling
- Reduces “aliasing artifacts” in renderings
- What is aliasing?
- Issue of having limited resolution on a screen.

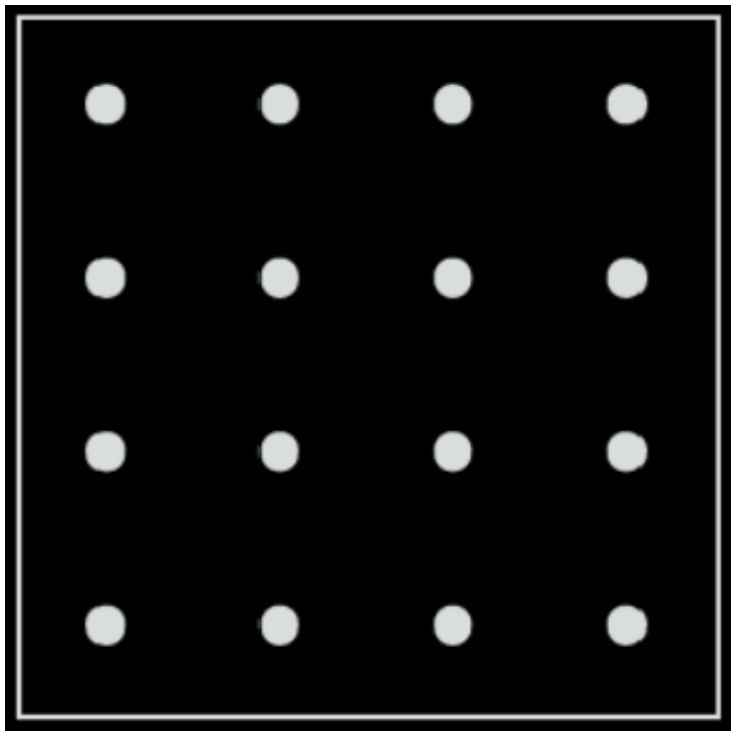
Distributed Ray Tracing

- Real world has infinite resolution
- Each pixel is a small window through which we can see part of the scene.
- Single ray per pixel is an inaccurate representation

Antialiasing - Supersampling

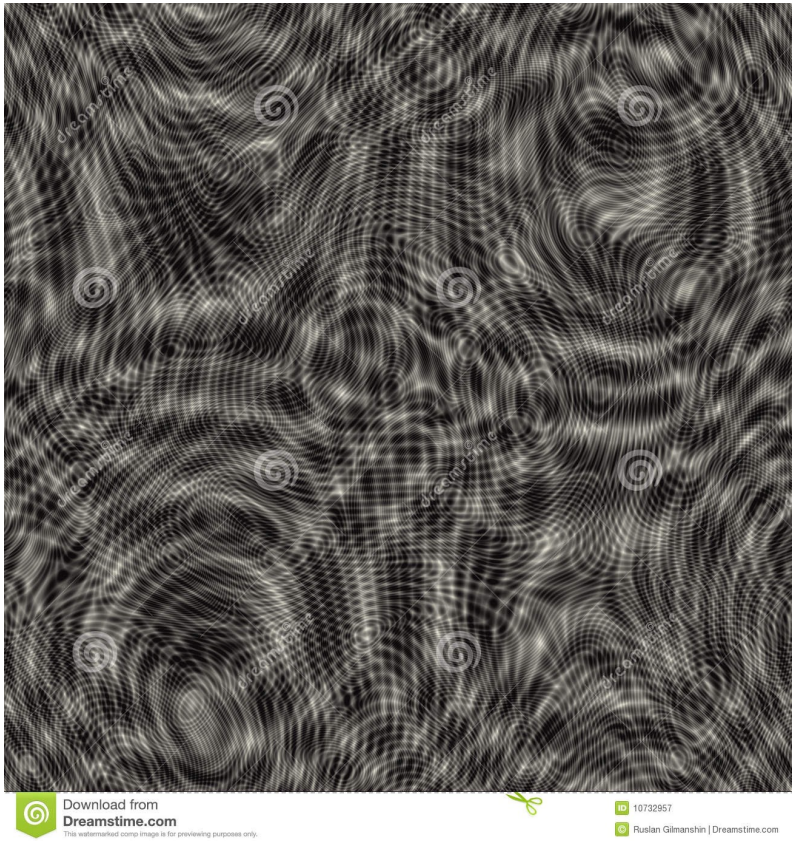


Antialiasing - Supersampling

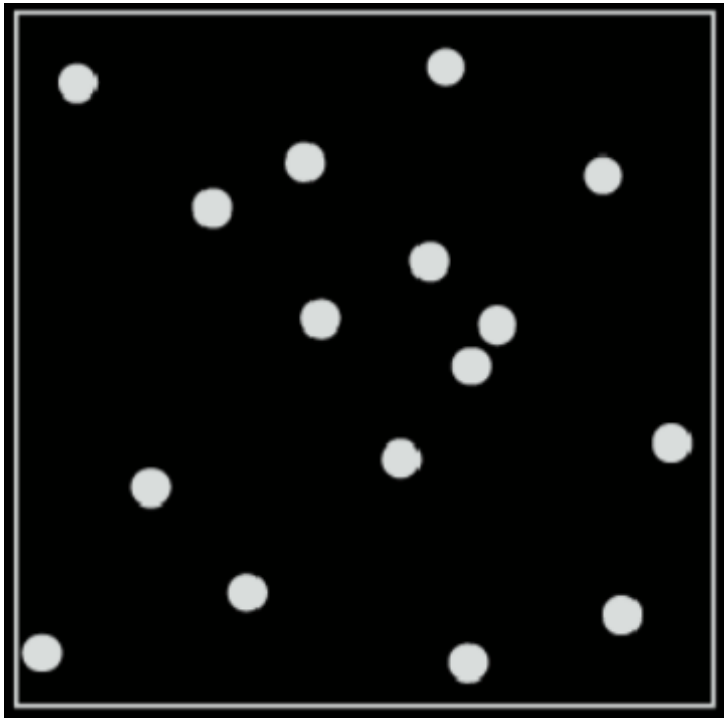


- Regularities can create moiré pattern (high frequency masquerading for low frequency)

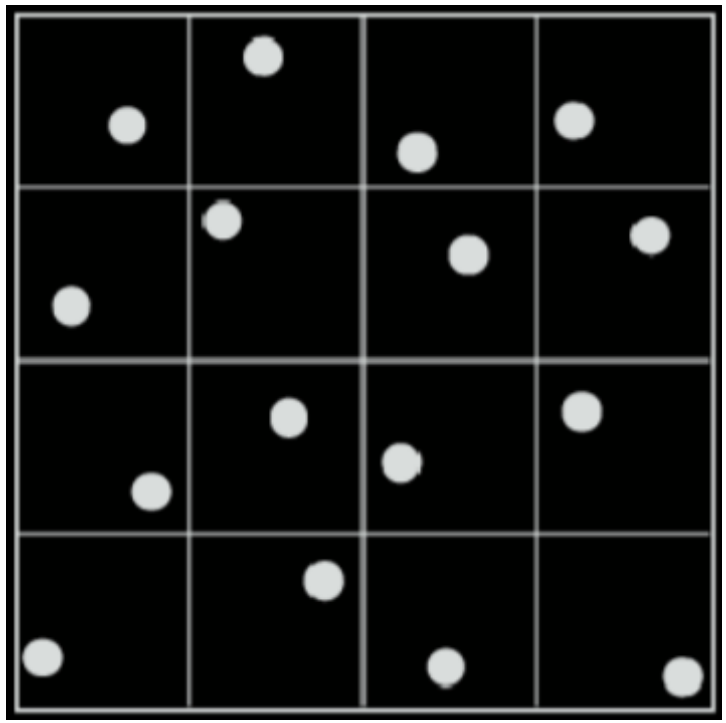
Antialiasing - Moiré Patterns



Antialiasing - Supersampling



Antialiasing - Supersampling



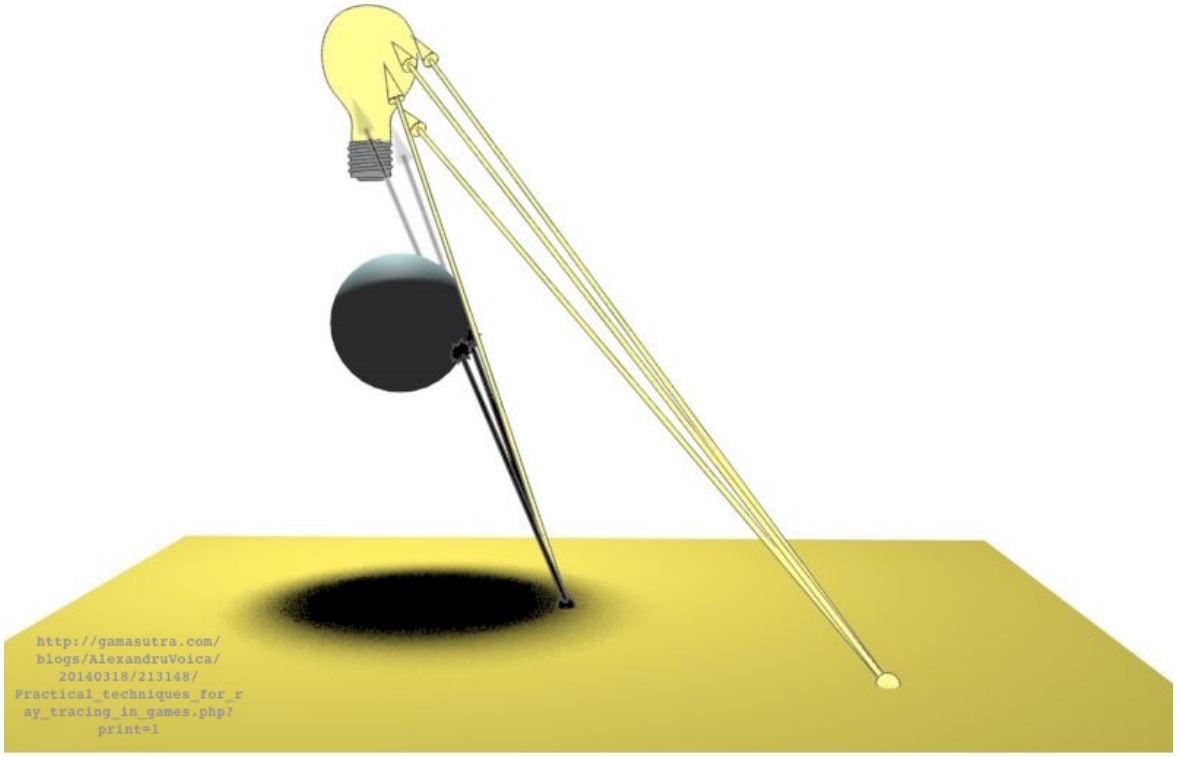
Soft Shadows

- Simple ray tracing is discrete
- Single point light sources inaccurate for close or large light sources
- Result: edges are very sharp with binary decisions
- In reality, there is a gradient due to partial lighting
- Lights are finite in area
- Scattering effects affect shadows

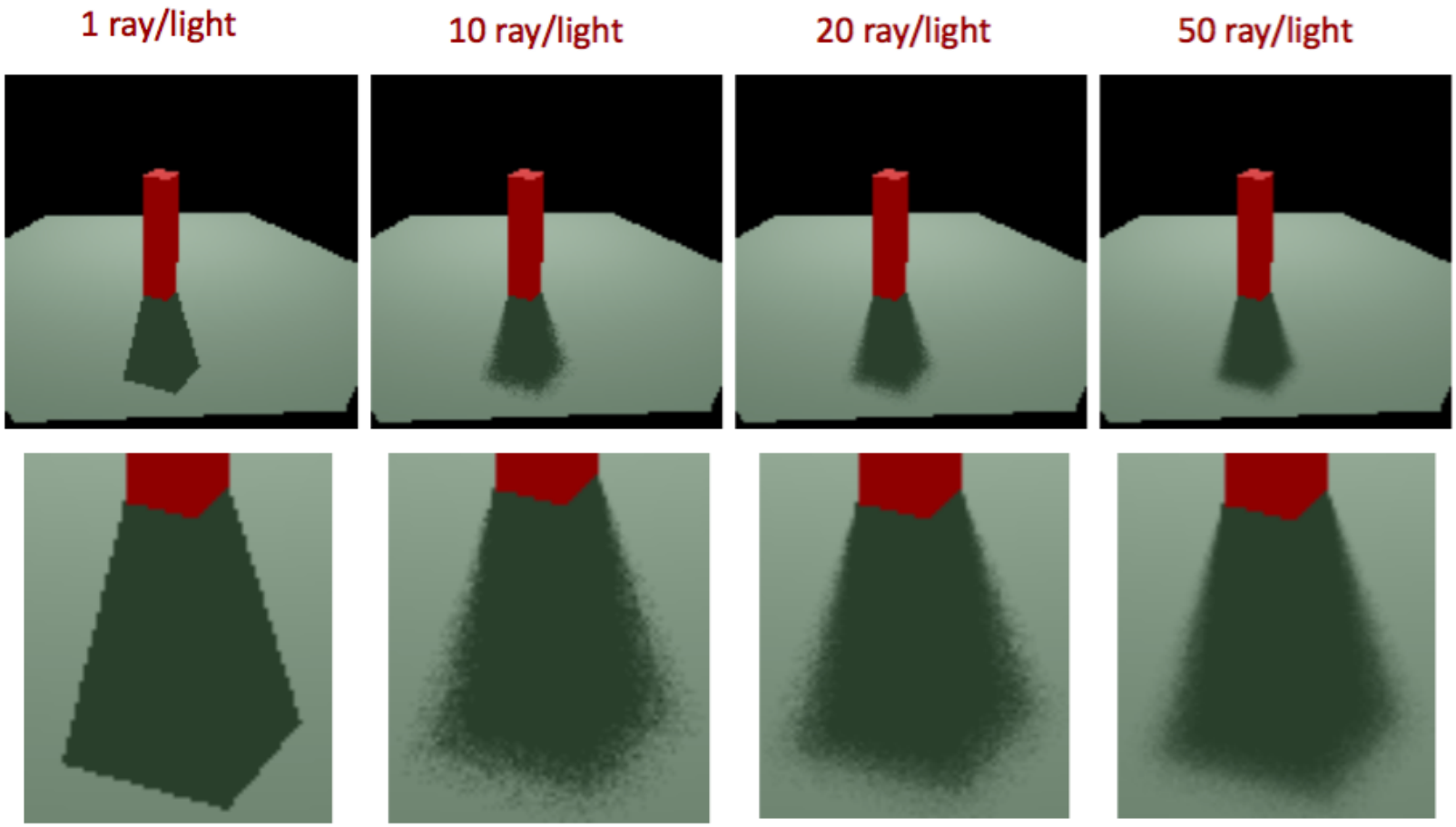
Soft Shadows

- Solution: Oversampling by using an area source light
- Can also model the light source using a volume
- Cast a set of shadow rays about the area of light source
- Amount of light contribution at a point: ration of #rays hit the source to the # of cast rays

Soft Shadows



How many rays do you need?

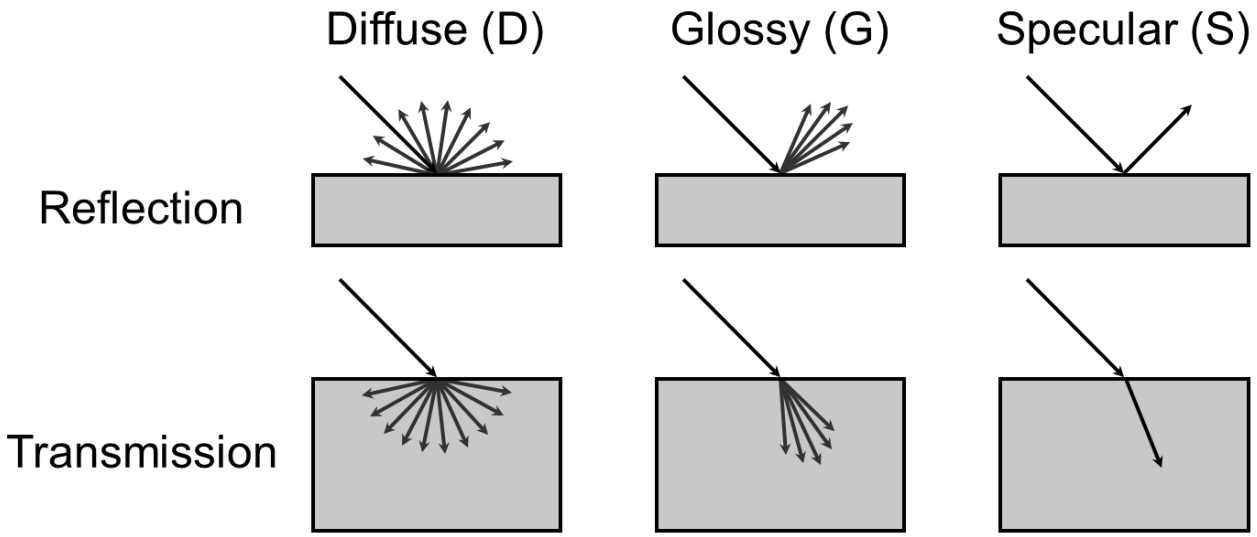


Images taken from http://web.cs.wpi.edu/~matt/courses/cs563/talks/dist_ray/dist.html

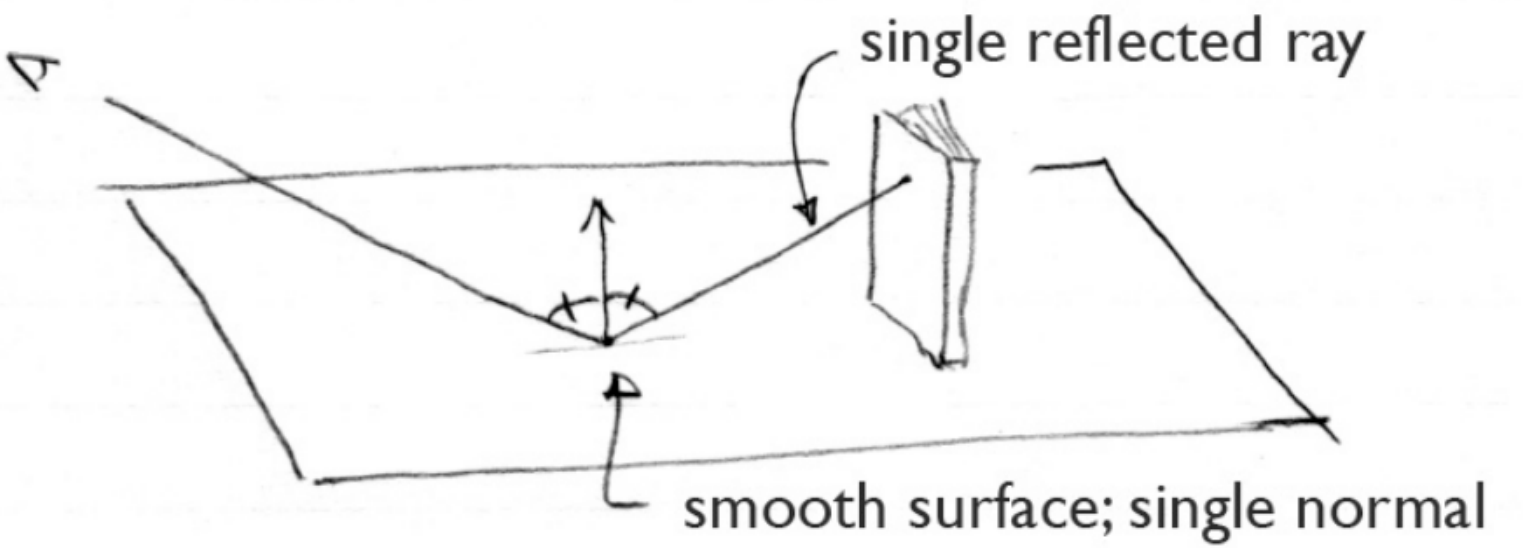
Glossy Reflections & Translucent Materials

- Simple ray tracing good for reflective surfaces
- Poor for glossy surfaces
- In reality, light scatters on the uneven surface
- Reflections are not always sharp, unless mirror-like
- Idea: Jitter the rays about the mirror direction with a falloff
- Value of reflectance found by averaging result of packet of rays

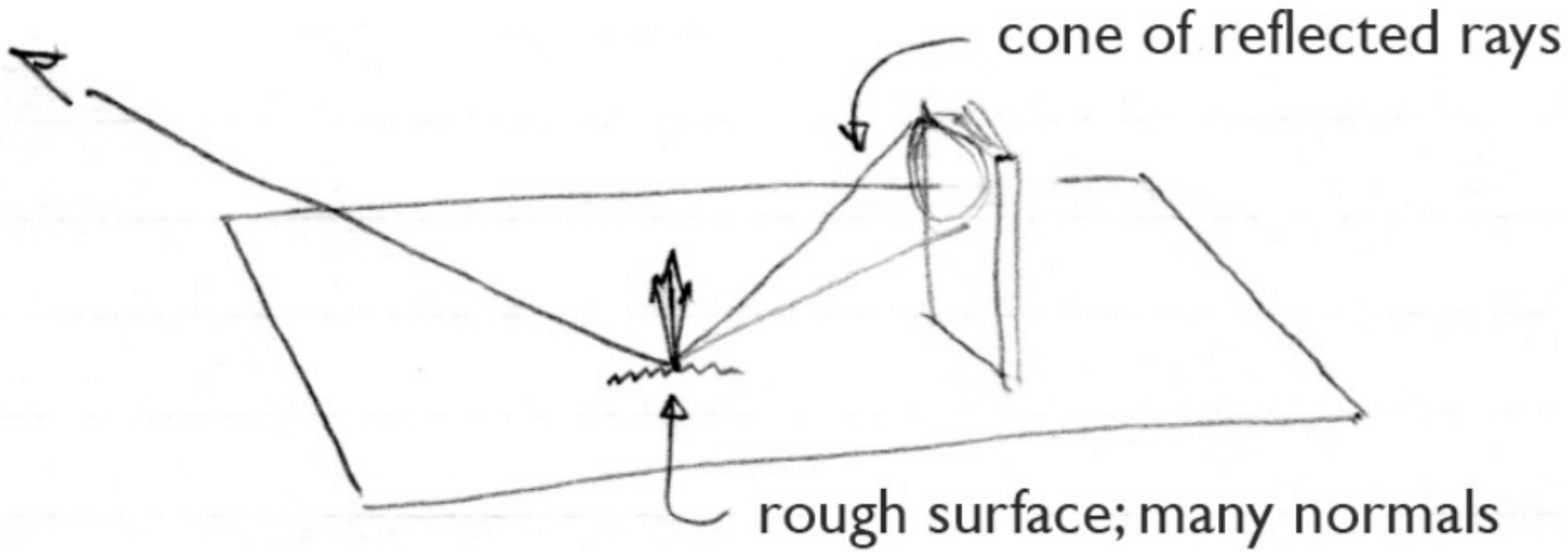
Glossy Reflections & Translucent Materials



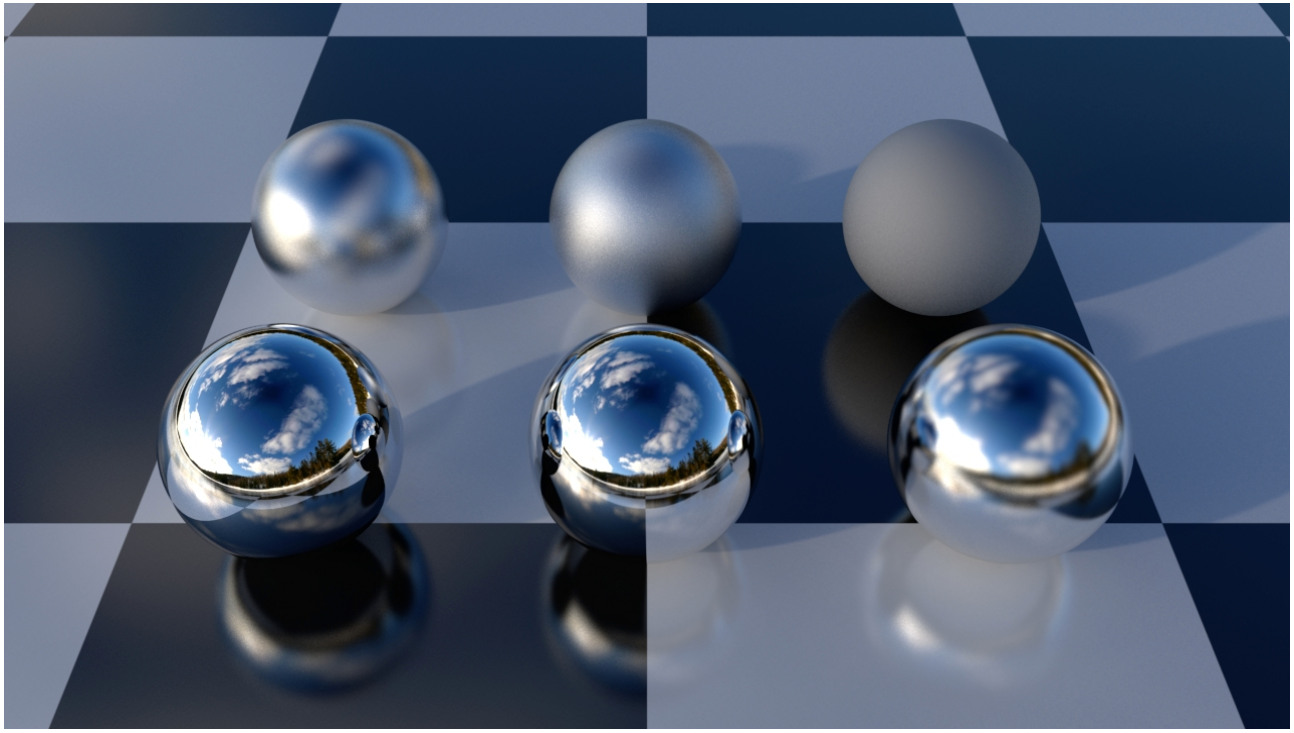
Glossy Reflections & Translucent Materials



Glossy Reflections & Translucent Materials



Glossy Reflections & Translucent Materials - examples



Motion Blur

- Scene is still and sampled over time (temporal sampling)
- When image is taken and shutter is open for fixed amount of time, the final image is an average of the light rays over time period.
- When objects are in motion, they will appear blurred
- Introduce time variable throughout the system.
  - i.e. objects are hit by rays given their position at a particular time
- Generate rays over shutter interval $T = T_0 + \xi(T_1 - T_0)$

Motion Blur