Computer Graphics (COMP0027) 2022/23

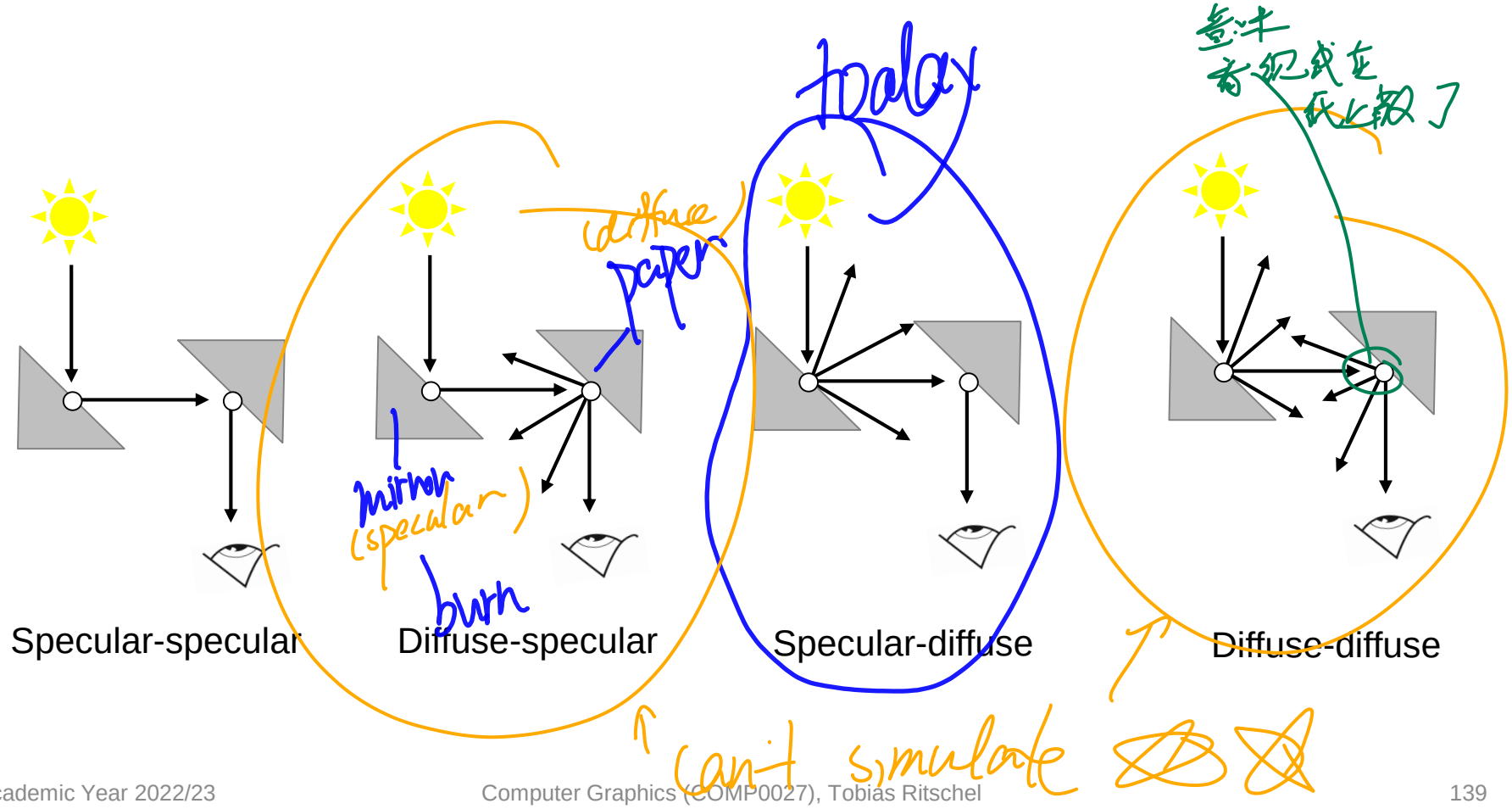
Recursive Ray-tracing

Tobias Ritschel

Overview

- Recursive ray tracing
- Shadow tests
- Snell's law for refraction
- When to stop!

Recap: Light Transport

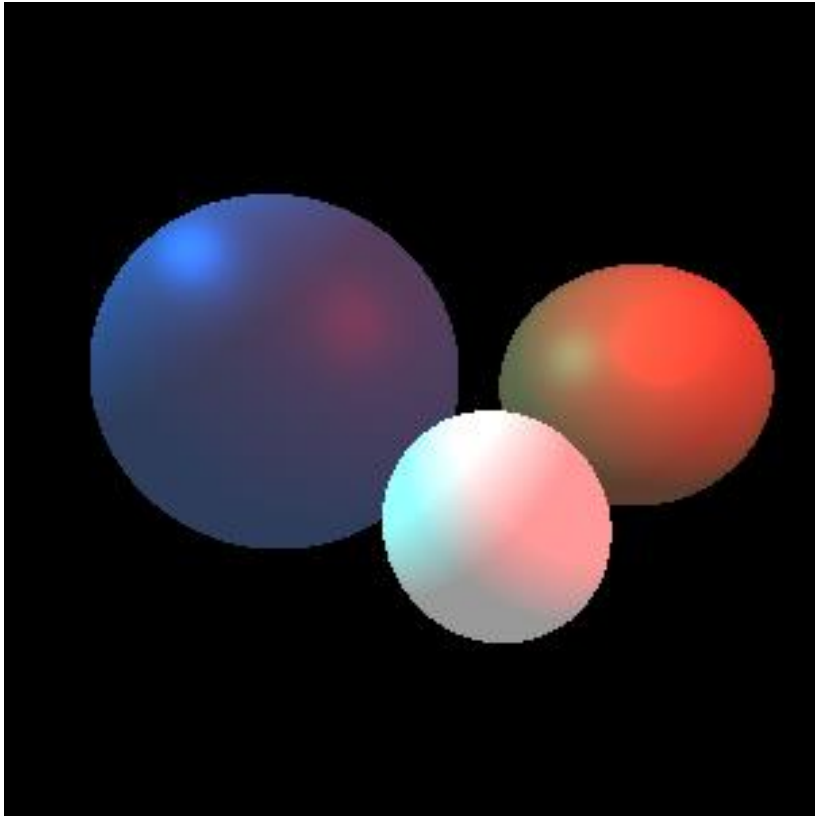


Recap: Local Illumination

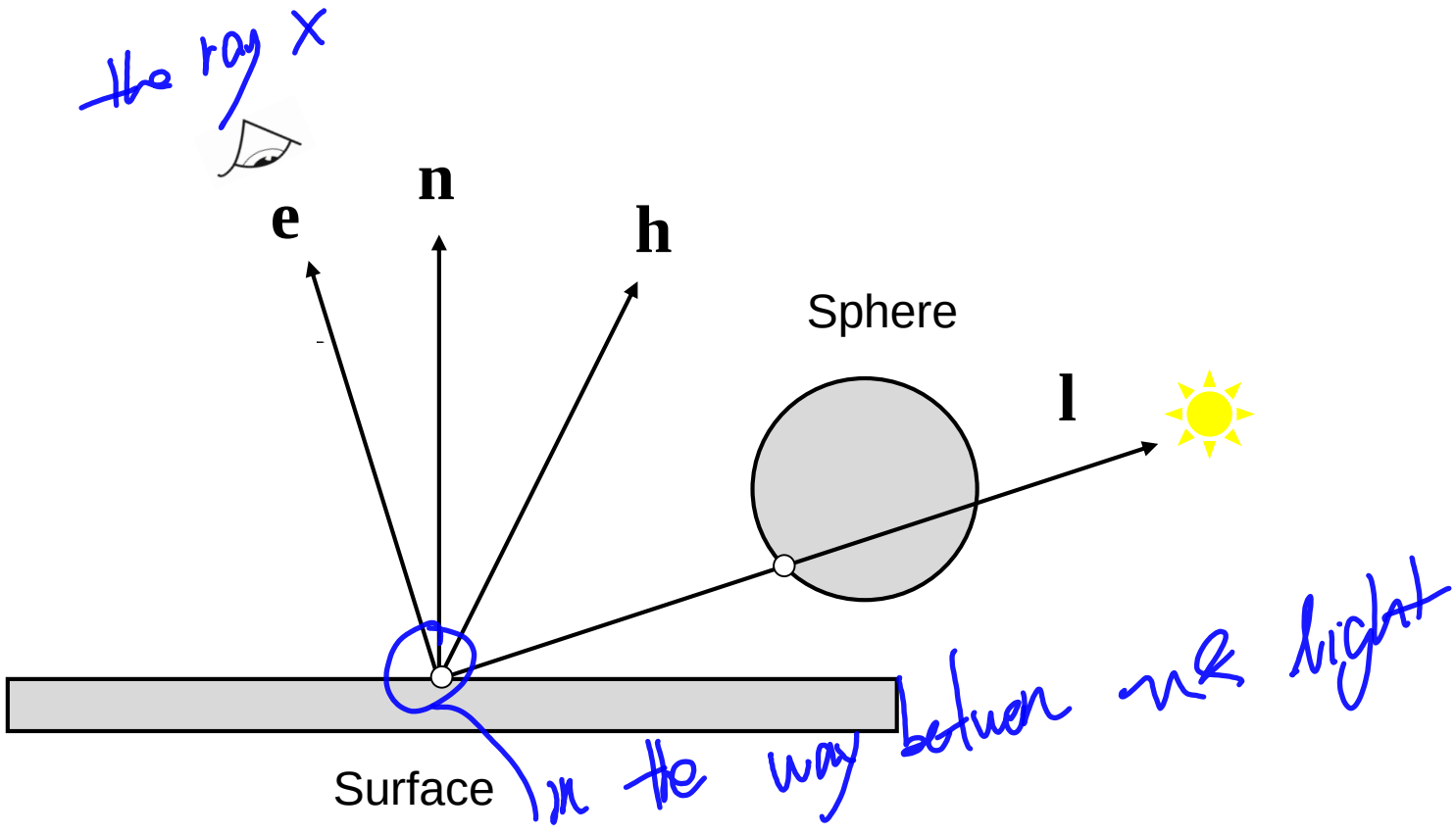
$$I_r = k_a I_a + I_i \left(k_d <\mathbf{n},\mathbf{l}>^+ + k_s (<\mathbf{h},\mathbf{n}>^+)^m \right)$$

- Ambient, diffuse & specular components
- Sum over multiple lights

Recap: Result of Ray Casting



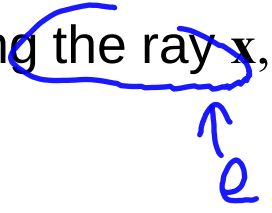
Shadows



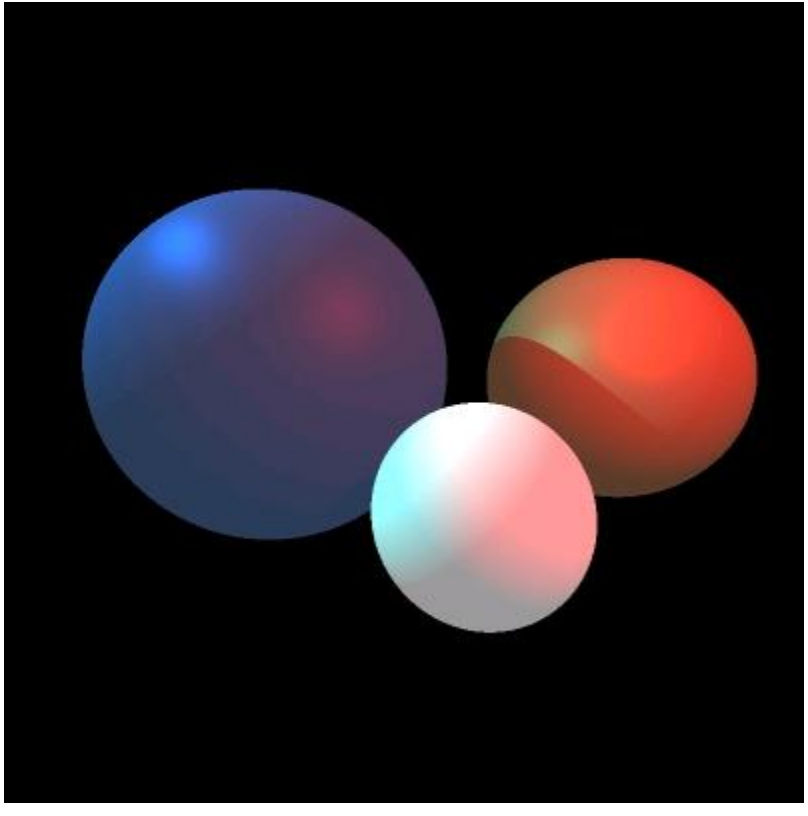
Illumination with shadows

$$I_r = k_a I_a + v_i I_i (k_d <\mathbf{n},\mathbf{l}>^+ + k_s(<\mathbf{h},\mathbf{n}>^+)^m)$$

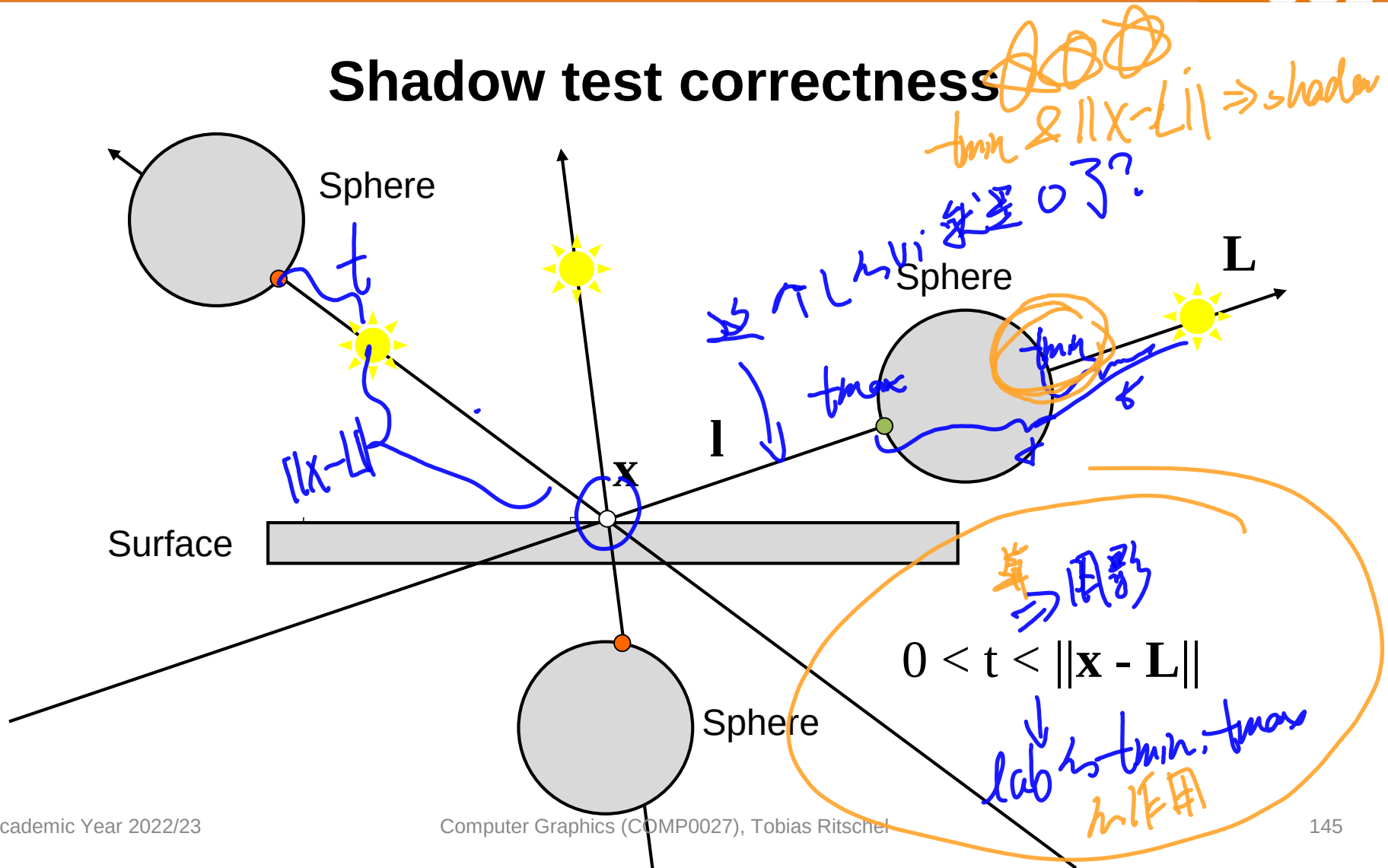
- Where v_i is the result of intersecting the ray $\mathbf{x}, \mathbf{l}$ with the scene
- v_i is the visibility of light i



Result of visibility tests



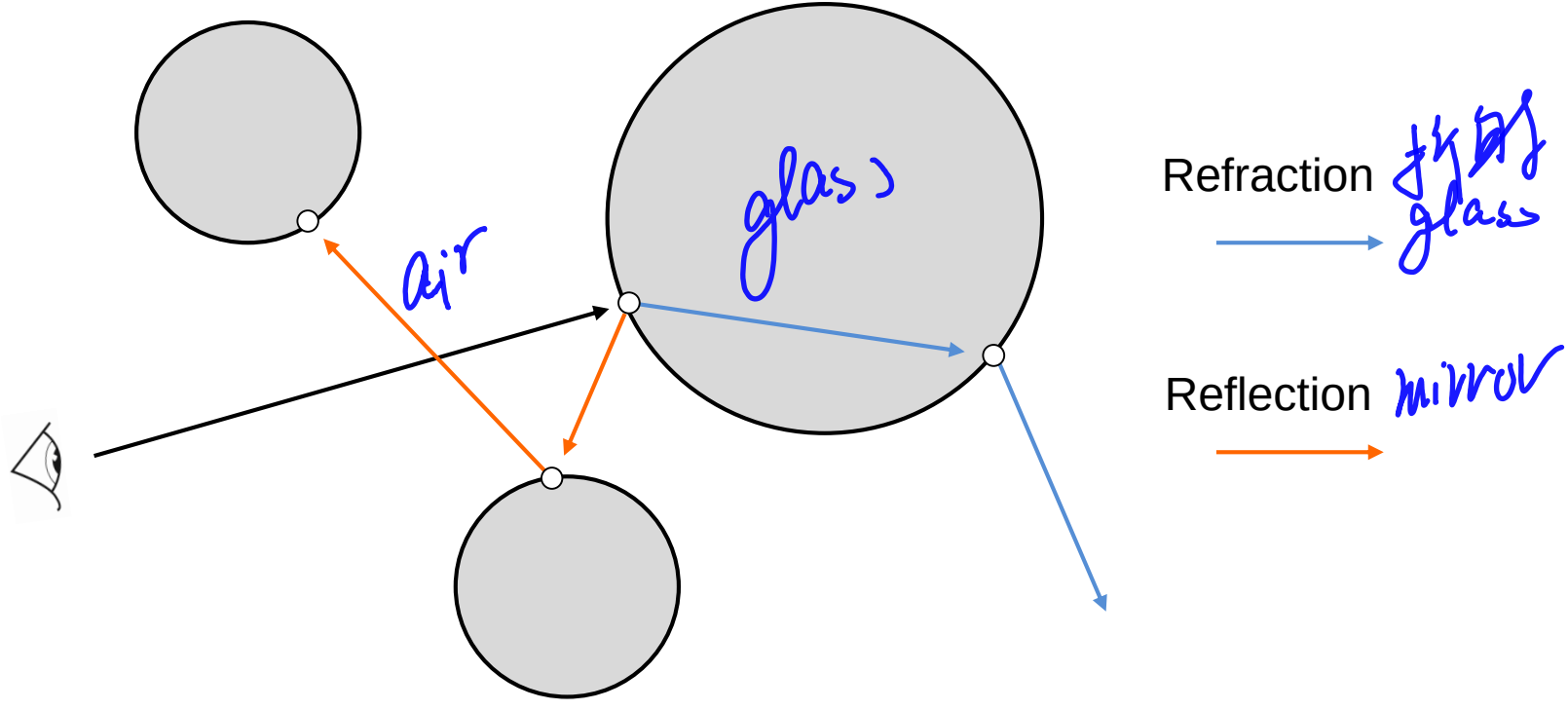
Shadow test correctness



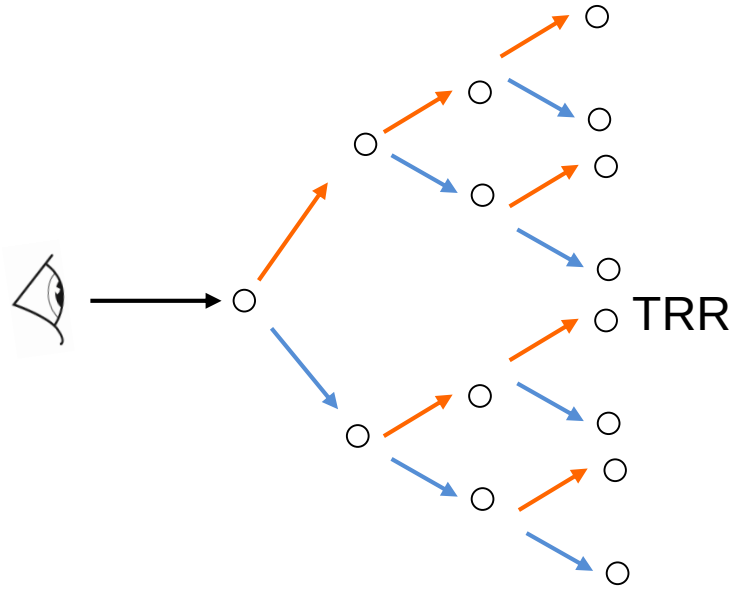
Recursive ray-tracing

- We can simulate specular-specular transmission elegantly by recursing and casting secondary rays from the intersection points
- We must obviously choose a termination depth to cope with multiple reflections

Recursive ray-tracing



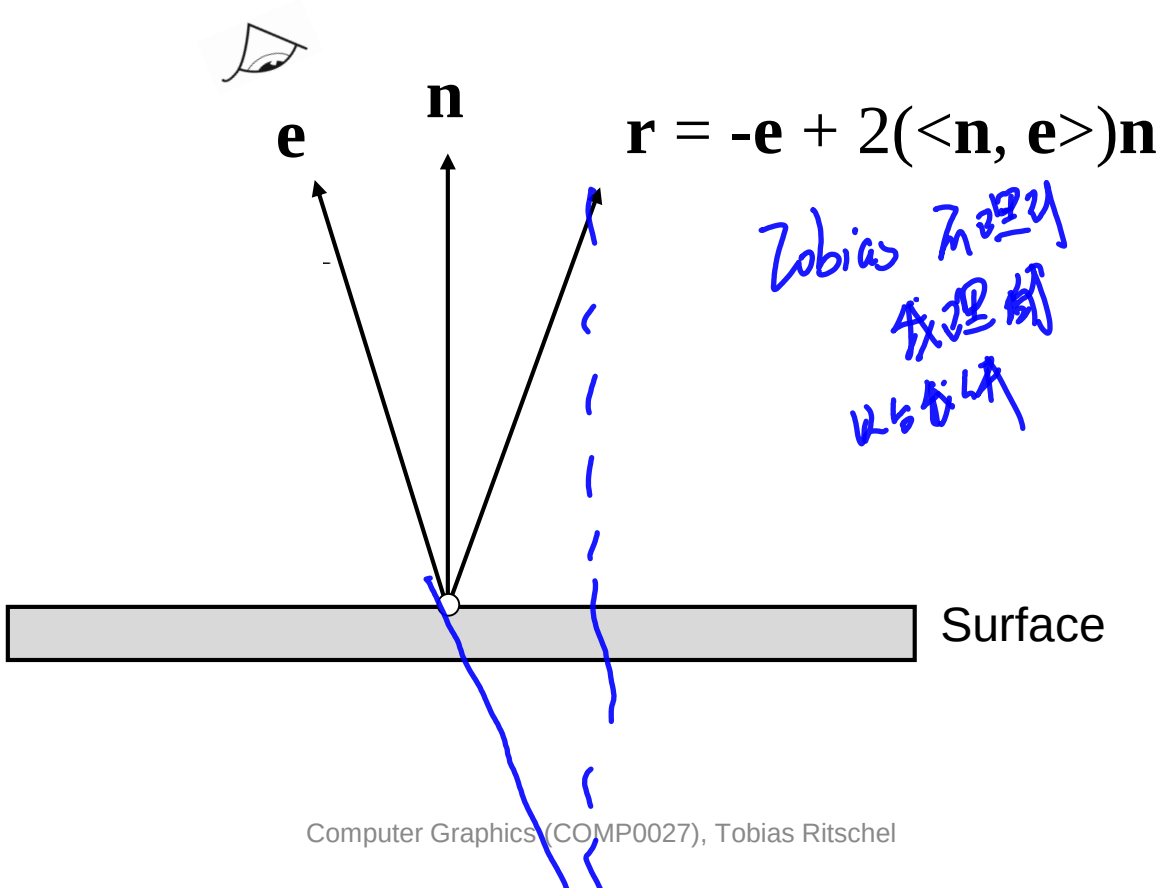
Ray tree



Refraction (T)

Reflection (R)

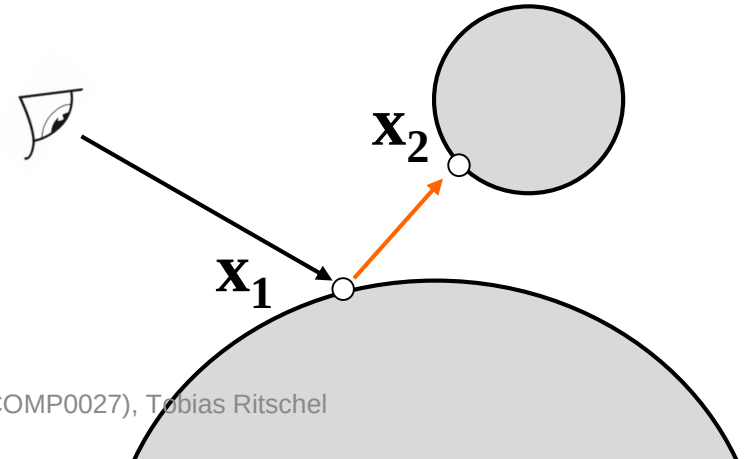
Introducing reflection



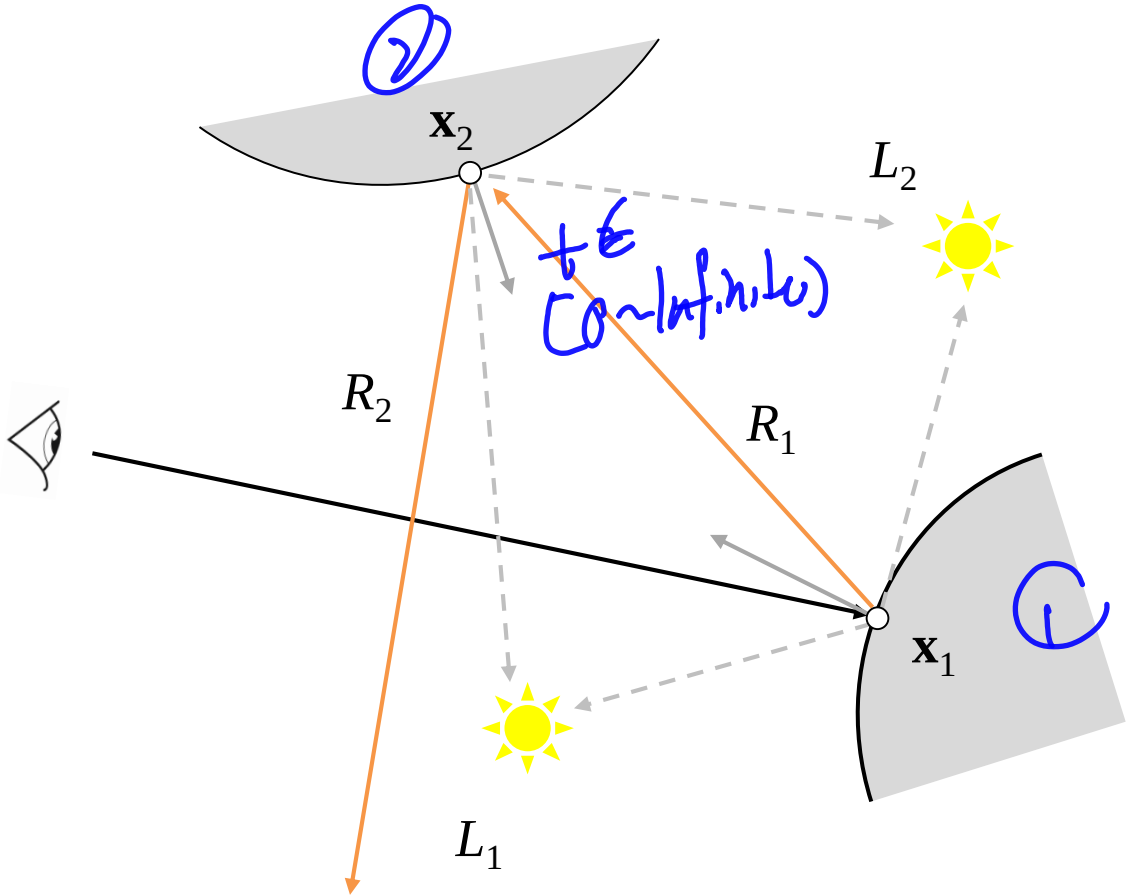
Computing mirror reflections

$$I_r = I_{Local} + k_r I_{Mirror}$$

- I_{Local} is computed at $\mathbf{x}_1$ as before before
- I_{Mirror} is computed at $\mathbf{x}_2$, the location visibble in the mirror



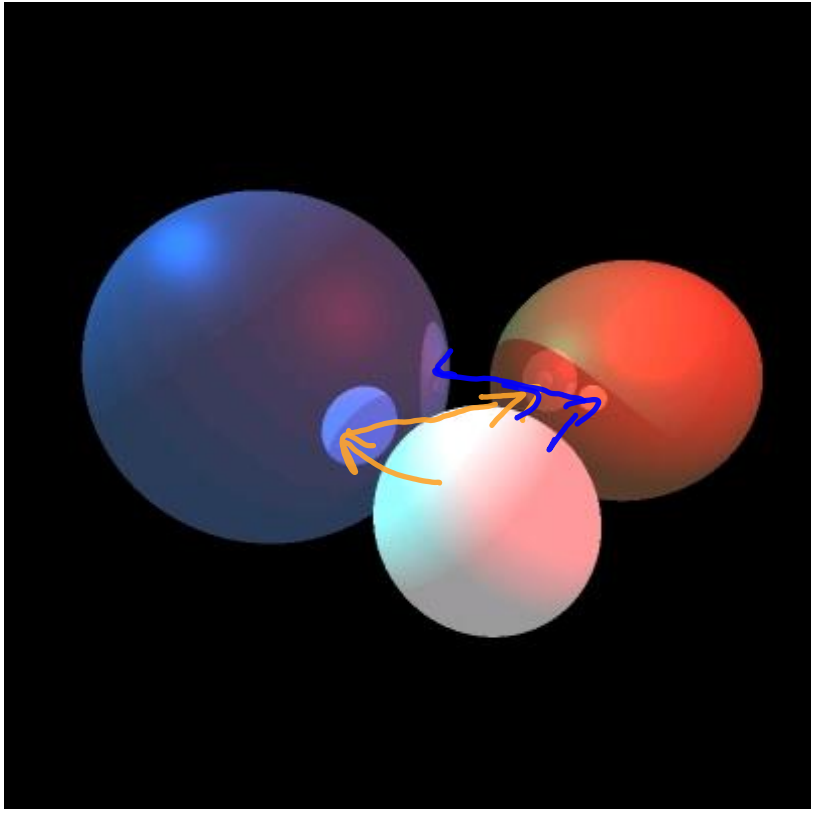
Recursive ray-tracing

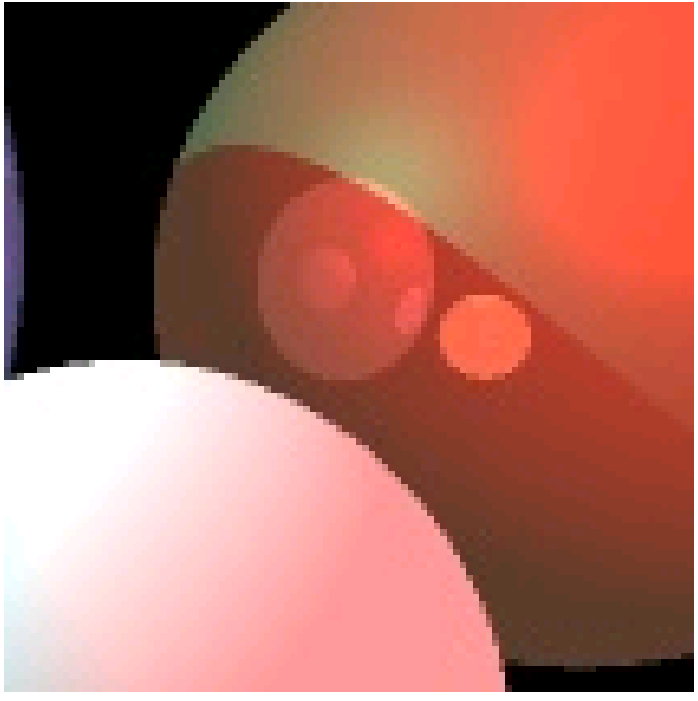


Pseudo code

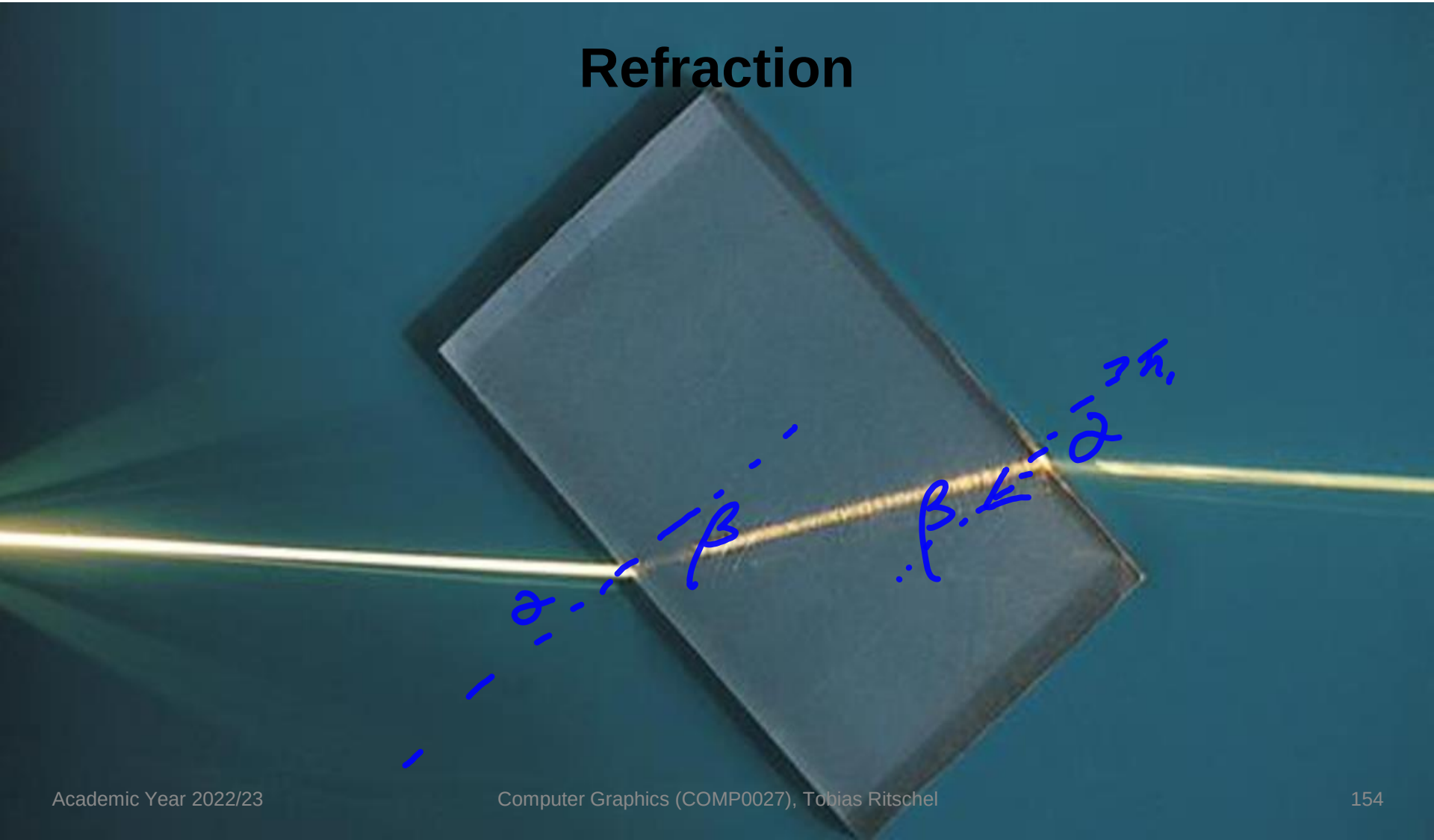
```
vec3 trace(Ray ray, Scene scene, int depth) {
   if (depth > MAX) return Black;
   HitInfo hitInfo = intersect(ray, scene);
   if (!hitInfo.isHit) return Background

   return
    shade(ray, hitInfo, scene) +
        hitInfo.k_r* trace(reflect(ray, hitInfo), scene, depth + 1);
}
```

Result of recursion

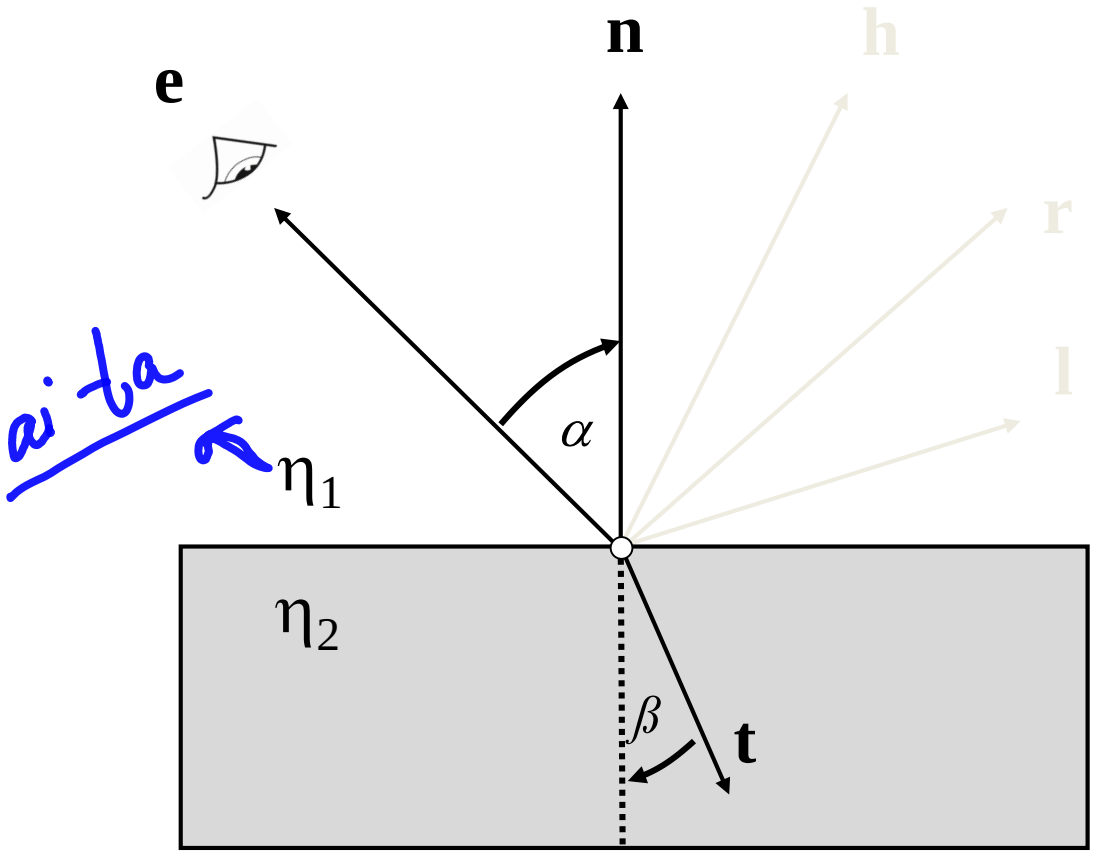




Refraction



Refraction

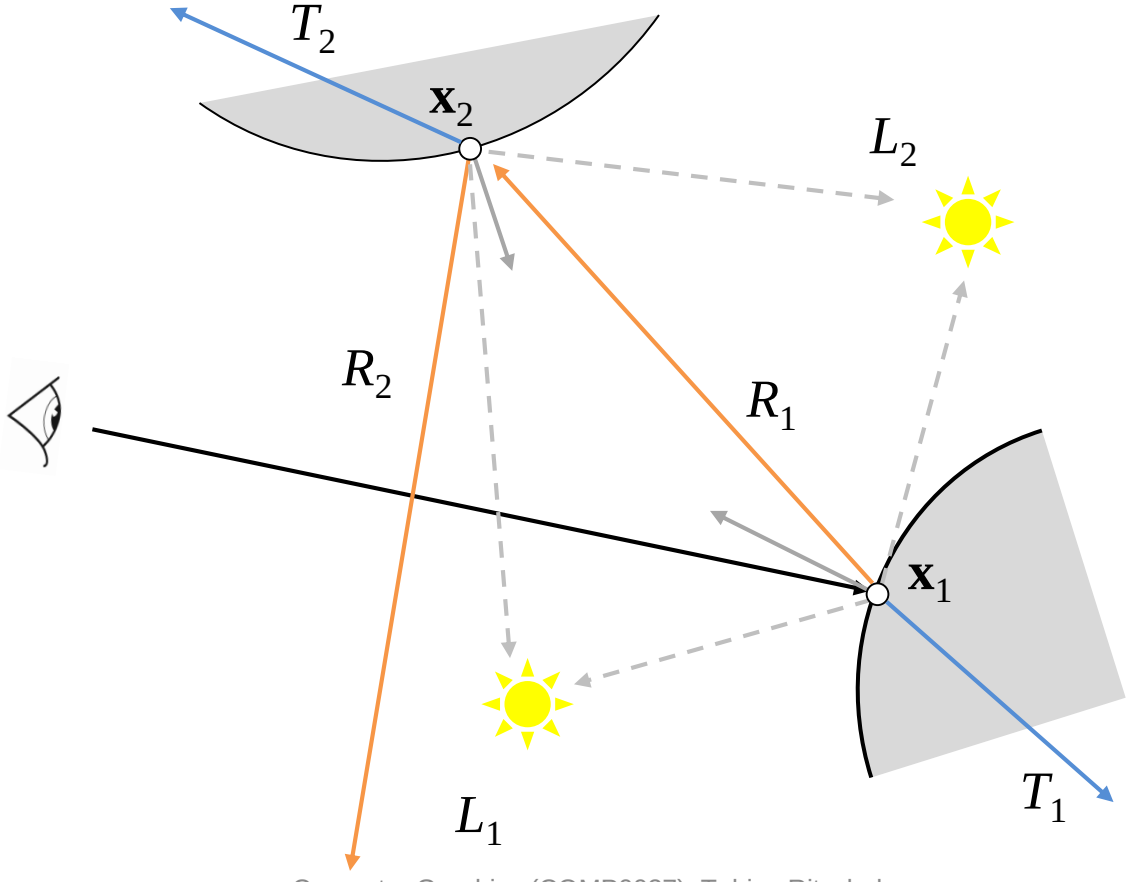


Snell's law

$$\frac{\sin a}{\sin b} = \frac{h_2}{h_1}$$

η is index of refraction

Recursive ray-tracing



Using Snell's law

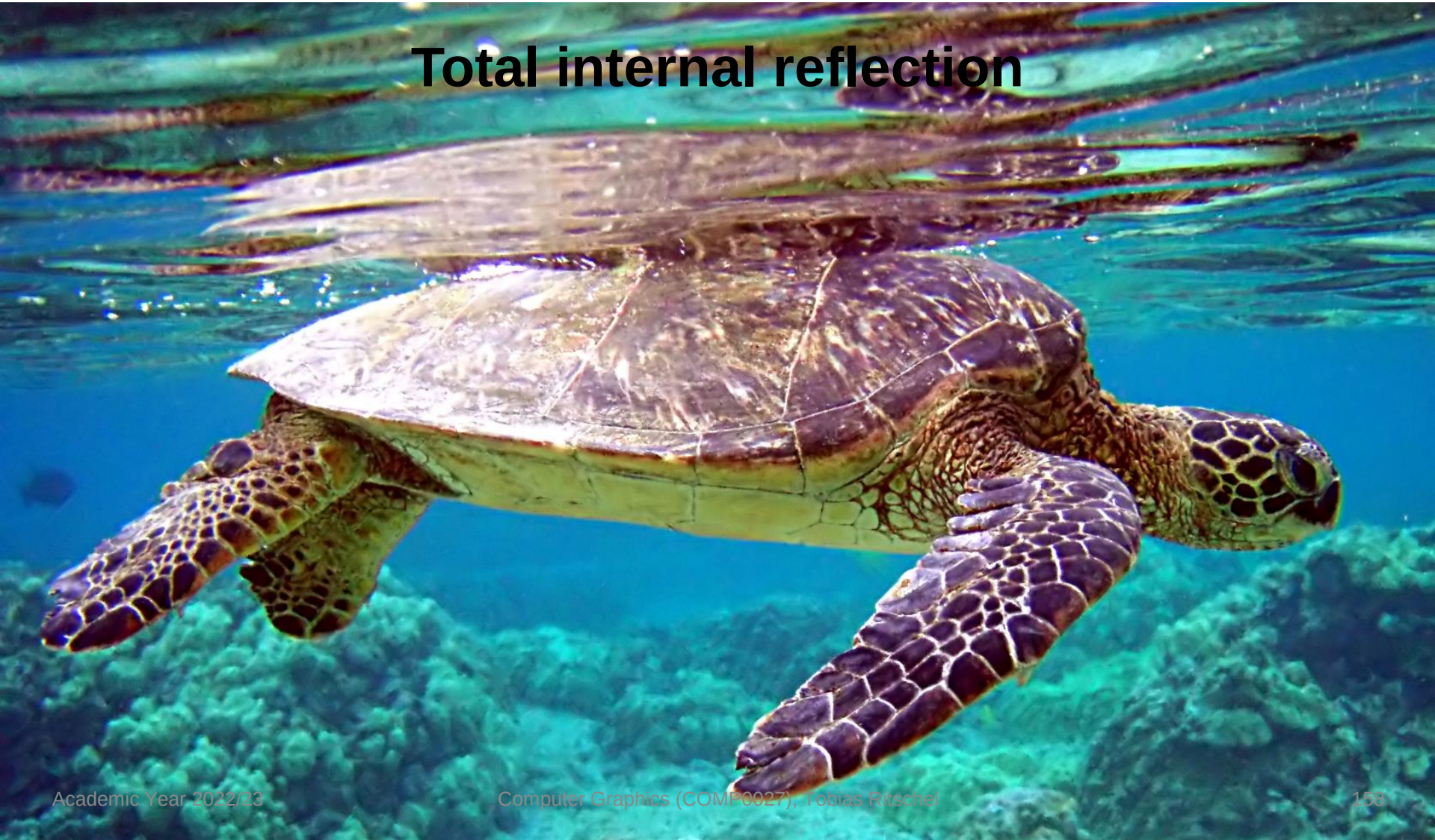
Refraction index $\frac{\sin\alpha}{\sin\beta} = \frac{\eta_2}{\eta_1} = \eta_{21}$

$$\mathbf{t} = -h_{12}\mathbf{e} + \mathbf{n}\left(h_{12} \times \cos a - \sqrt{1 + h_{12}^2 \times \left(\cos^2 a - 1\right)}\right)$$

↑ don't have to remember!

Note that if the root is negative then total internal reflection has occurred and you just reflect the vector as normal

Total internal reflection



New pseudo code

```
vec3 trace(Ray ray, Scene scene, int depth) {
    if (depth > MAX) return Black;
    HitInfo hitInfo = intersect(ray, scene);
    if (!hitInfo.isHit) return Background

    return
     shade(ray, hitInfo, scene) +
          hitInfo.k_r * trace(reflect(ray, hitInfo), scene, depth + 1) +
          hitInfo.k_t * trace(refract(ray, hitInfo), scene, depth + 1);
}
```

和上面没引进 reflect 时为 pseudo 准 [illegible]

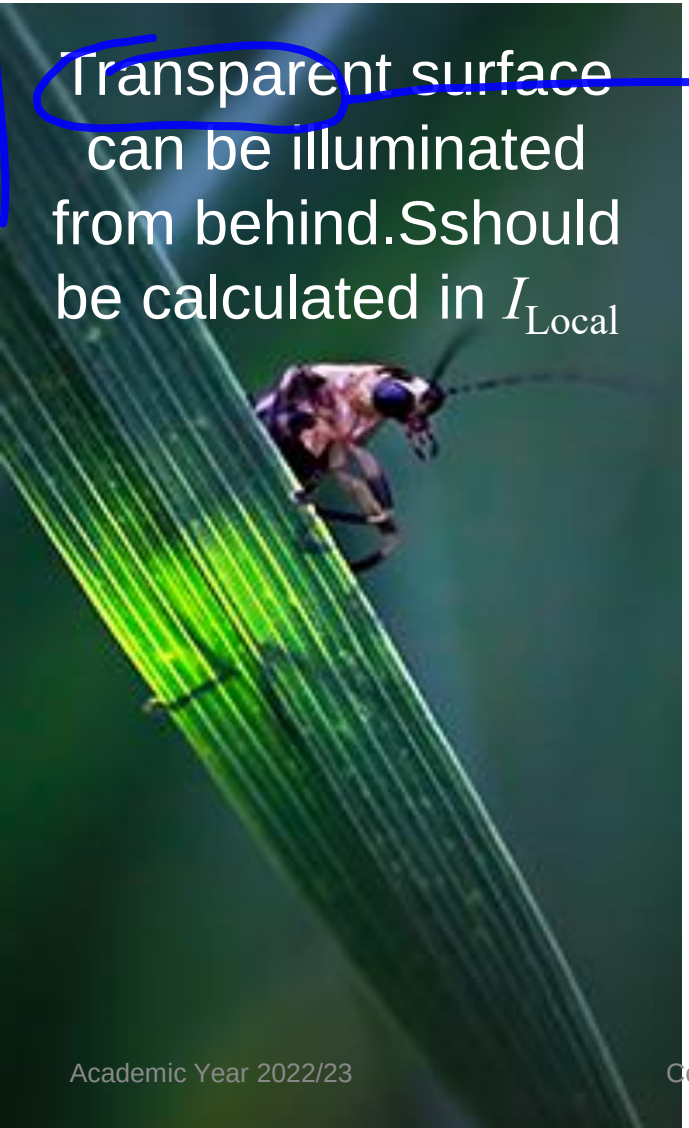
Stackless raytracing

Implementation with stack:

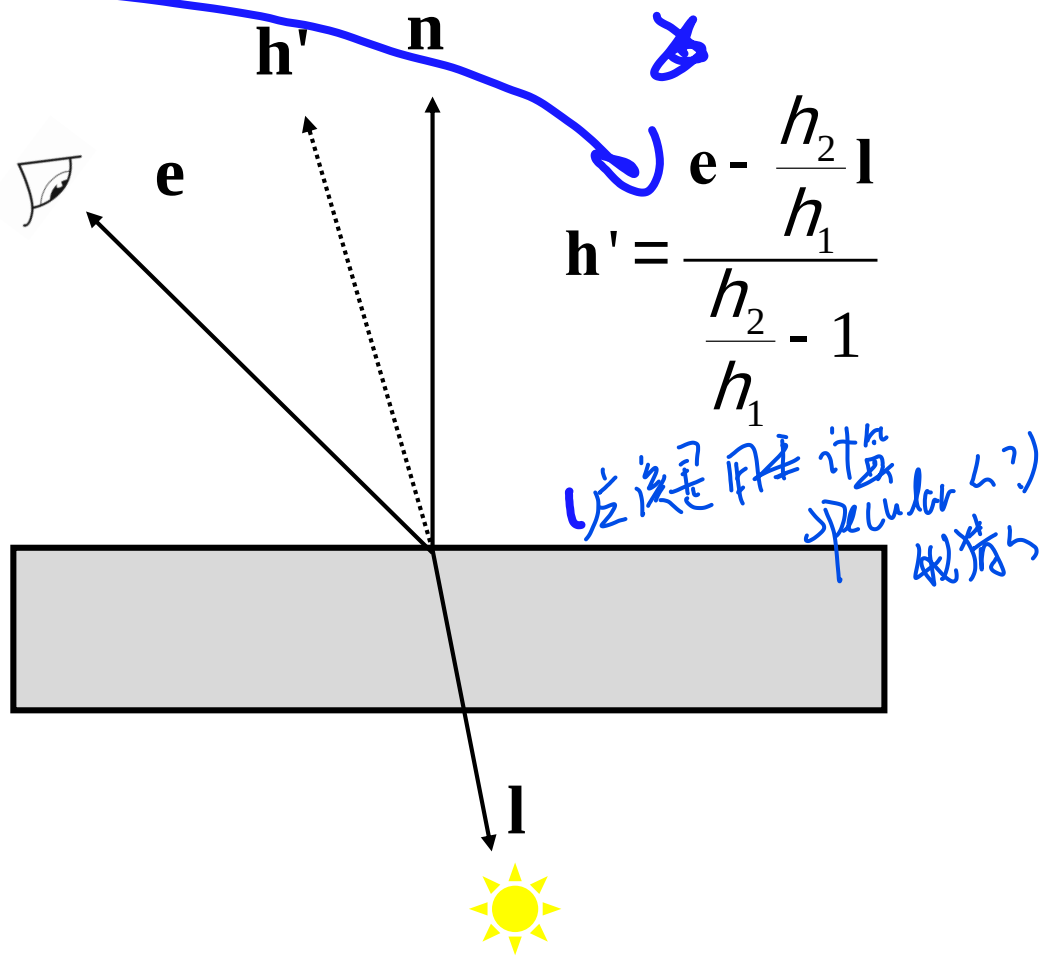
```
void a(int depth) {
 echo depth;
 if (depth > 10) return;
 a(depth + 1);
}

a(0);
```

Implementation w/o stack:

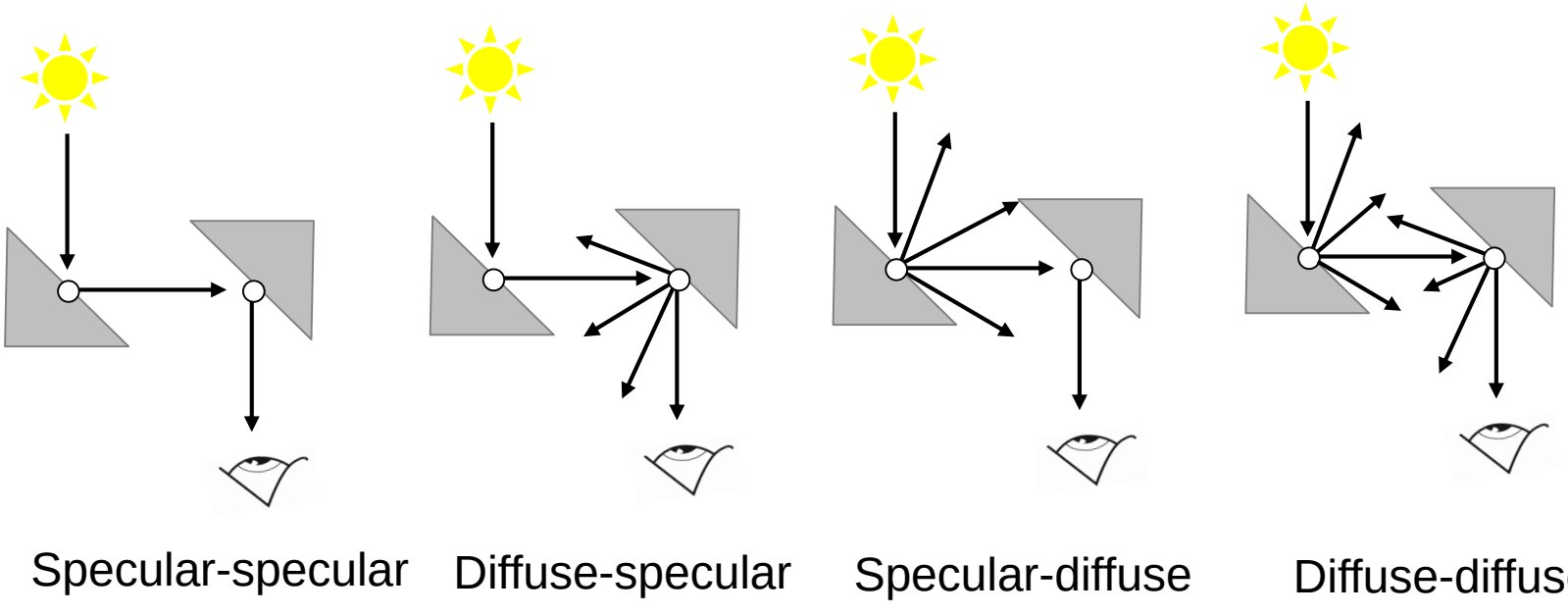
```
depth = 0
for(int i from 0 to 10) {
 echo depth;
 depth = depth + 1;
}
```



Transparent surface can be illuminated from behind.Sshould be calculated in I_{Local}



Discussion – What Can't We Simulate?



Remark

- Reflection and refraction-only
  - What should be added to consider diffuse reflection?
- Why it's expensive
  - Intersection of rays with polygons (90%)
- How to reduce the cost?
  - Reduce the number of rays
  - Reduce the cost on each ray
    - First check with bounding box of the object
    - Methods to sort the scene and make it faster

Summary

- Recursive ray tracing is a good simulation of specular reflections
- We've seen how the ray-casting can be extended to include shadows, reflections and transparent surfaces
- However this is a very slow process and still misses some types of effect!