

INFOGR – Computer Graphics

Jacco Bikker - April-July 2016 - Lecture 5: “Ray Tracing (3)”

Welcome!



Today's Agenda:

- Reflections
- Refraction
- Recursion
- Shading models
- TODO

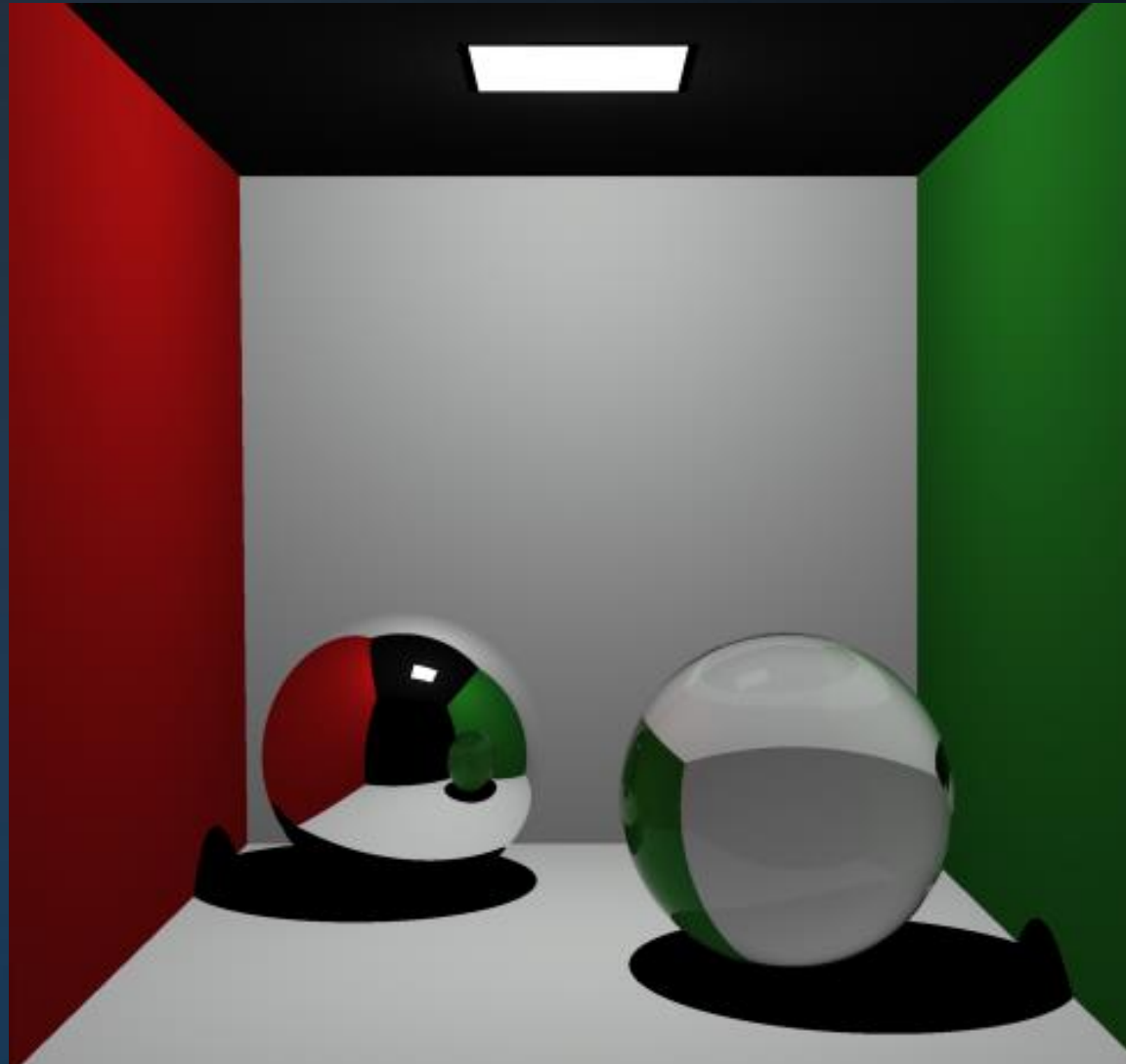


Reflections

```

    if (depth < MAXDEPTH)
    {
        // Inside the sphere
        Vec r = inside / 1.5;
        Vec nt = nt / nc, nde = nde / nc;
        Vec cos2t = 1.0f - nnt * nnt;
        Vec D, N;
        Vec a = nt - nc, b = nt + nc;
        Vec Tr = 1 - (RB + (1 - RB) * cos2t);
        Vec R = (D * nnt - N * cos2t);
        Vec E * diffuse;
        Vec refl;
        Vec refl + refr;
        Vec D, N;
        Vec refl * E * diffuse;
        Vec refl;
        Vec MAXDEPTH);
        Vec survive = SurvivalProbability( diffuse );
        Vec estimation - doing it properly, closely following wall (survive);
        Vec radiance = SampleLight( &rand, I, &L, &align );
        Vec e.x + radiance.y + radiance.z > 0) && (e.x + radiance.y + radiance.z > 0);
        Vec w = true;
        Vec brdfPdf = EvaluateDiffuse( L, N );
        Vec factor = diffuse * INVPI;
        Vec weight = Mis2( directPdf, brdfPdf );
        Vec cosThetaOut = dot( N, L );
        Vec E * ((weight * cosThetaOut) / directPdf) * (radiance);
        Vec random walk - done properly, closely following wall (survive);
        Vec survive);
        Vec brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        Vec survive;
        Vec pdf;
        Vec n = E * brdf * (dot( N, R ) / pdf);
        Vec n = true;
    }

```



Reflections

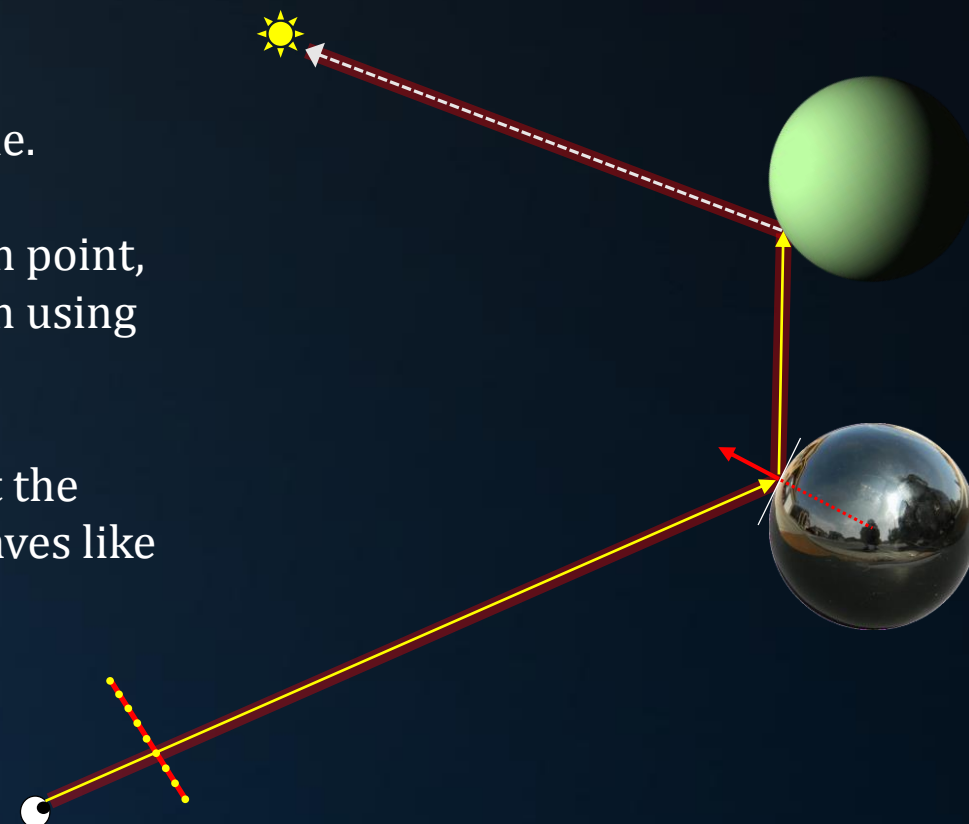
Light Transport

We introduce a *pure specular* object in the scene.

Based on the normal at the primary intersection point, we calculate a new direction. We follow the path using a *secondary ray*.

At the primary intersection point, we ‘see’ what the secondary ray ‘sees’; i.e. the secondary ray behaves like a primary ray.

We still need a shadow ray at the new intersection point to establish light transport.



Reflections

Reflected Vector

Reflection: *angle of incidence equals angle of reflection.*

Using normalized vector $\vec{D}$:

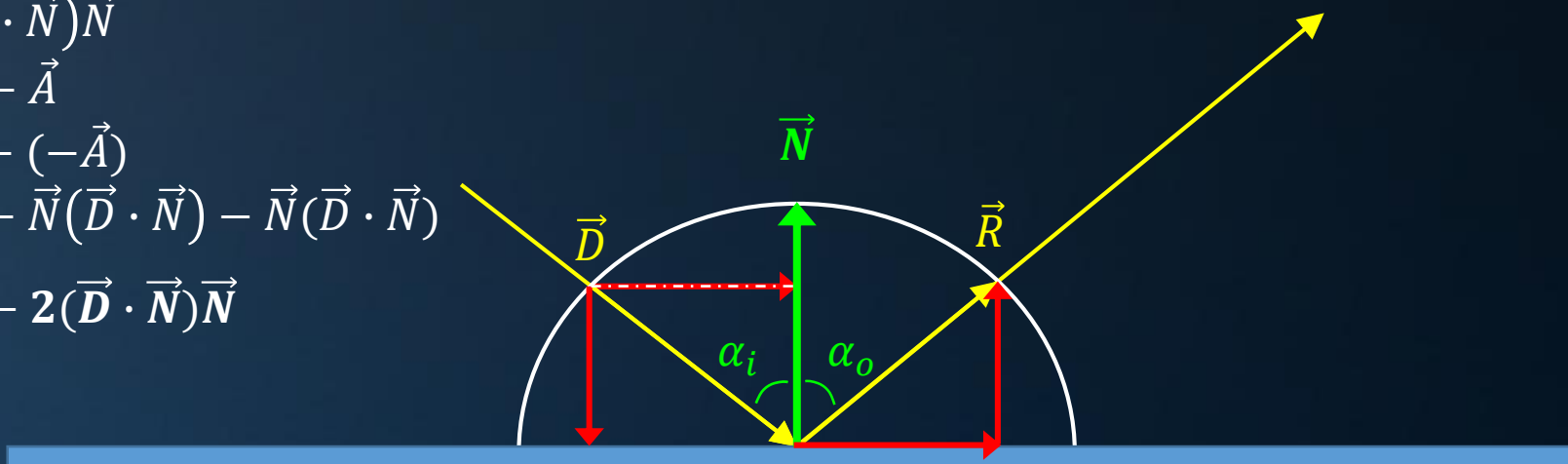
$$\vec{A} = (\vec{D} \cdot \vec{N})\vec{N}$$

$$\vec{B} = \vec{D} - \vec{A}$$

$$\vec{R} = \vec{B} + (-\vec{A})$$

$$\vec{R} = \vec{D} - \vec{N}(\vec{D} \cdot \vec{N}) - \vec{N}(\vec{D} \cdot \vec{N})$$

$$\vec{R} = \vec{D} - 2(\vec{D} \cdot \vec{N})\vec{N}$$



Reflections

Light Transport

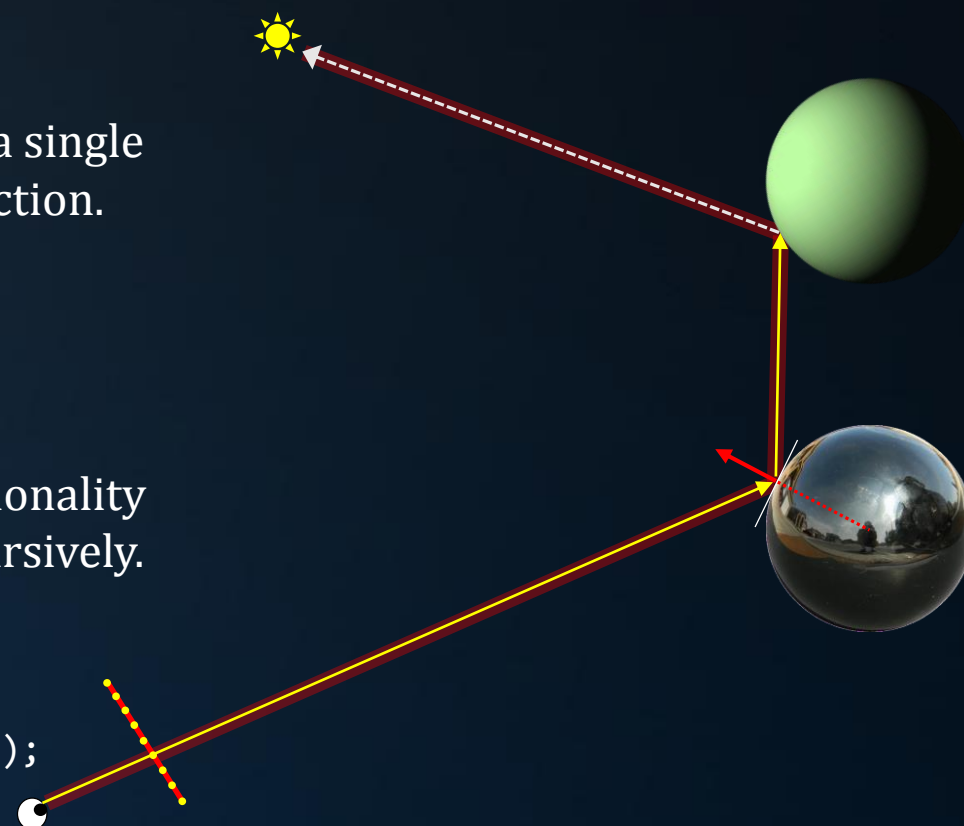
For a pure specular reflection, the energy from a single direction is reflected into a single outgoing direction.

- We do not apply $\vec{N} \cdot \vec{L}$
- We do apply absorption

Since the reflection ray requires the same functionality as a primary ray, it helps to implement this recursively.

```
vec3 Trace( Ray ray )
```

```
{
    I, N, material = scene.GetIntersection( ray );
    if (material.isMirror)
        return material.color * Trace( ... );
    return DirectIllumination() * material.color;
}
```





Light Transport

Light arriving from that direction cannot leave in the direction of the camera.




```

    if (depth < MAXDEPTH)
    {
        int inside = 1;
        int nt = nt / nc, ndd = 0;
        double s2t = 1.0f - nnt * nnt;
        double D, N;
        while (true)
        {
            int a = nt - nc, b = nt;
            double Tr = 1 - (R0 + (1 - R0) * a * a);
            double R = (D * nnt - N * a);
            if (E * diffuse)
                = true;
            if (refl + refr) && (depth < MAXDEPTH)
            {
                N;
                refl * E * diffuse;
                = true;
            }
        }
    }
    MAXDEPTH)
    survive = SurvivalProbability(
    estimation - doing it p
    ff;
    radiance = SampleLight(
    .x + radiance.y + radi
    v = true;
    brdfPdf = EvaluateDif
    t3 brdfPdf = diffuse * I
    weight = Mis2( direct
    cosThetaOut = dot( N,
    E * ((weight * cosTheta

```



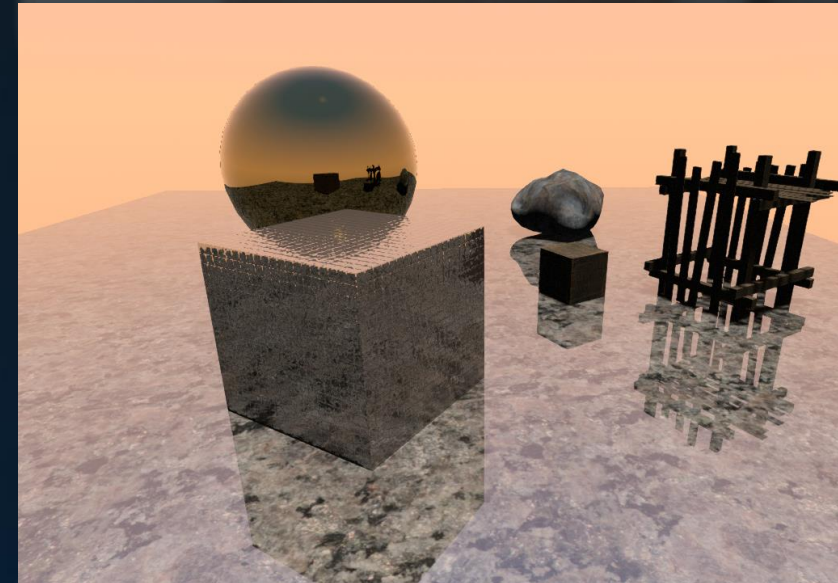
Reflections

Partially Reflective Surfaces

We can combine pure specular and diffuse properties.

Situation: our material is only 50% reflective.

In this case, we send out the reflected ray, and multiply its yield by 0.5. We also send out a shadow ray to get direct illumination, and multiply the received light by 0.5.



Reflections

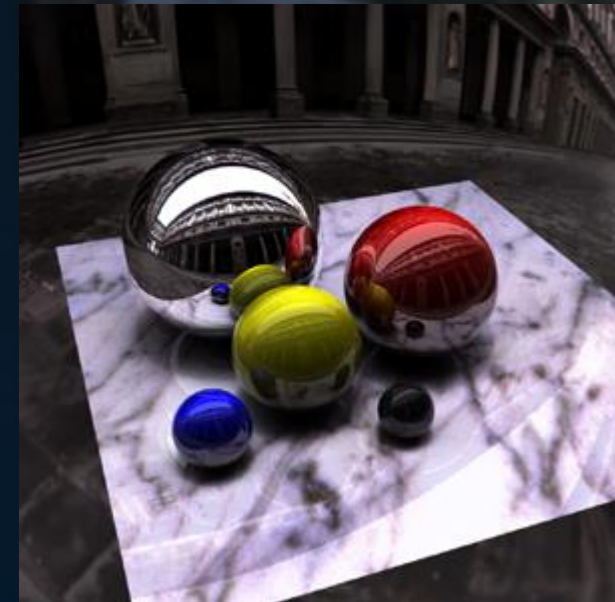
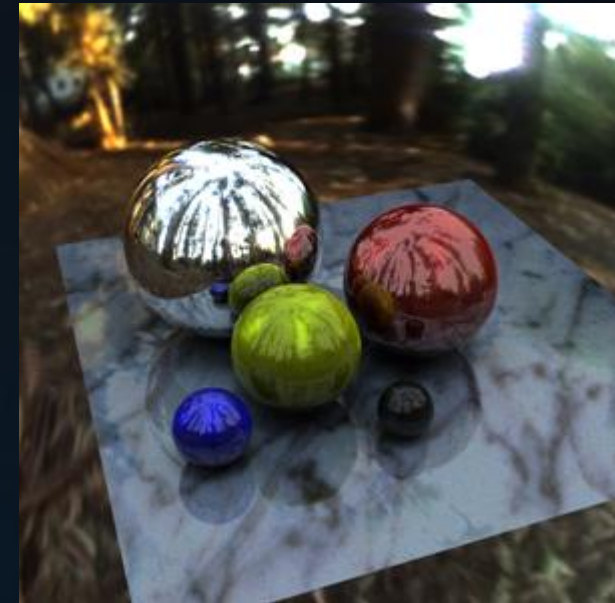
Reflecting a HDR Sky

A dark object can be quite bright when reflecting something bright.

E.g., a bowling ball, pure specular, color = (0.01, 0.01, 0.01); reflecting a ‘sun’ stored in a HDR skydome, color = (100, 100, 100).

For a collection of HDR probes, visit Paul Debevec’s page:

<http://www.pauldebevec.com/Probes>



Today's Agenda:

- Reflections
- Refraction
- Recursion
- Shading models
- TODO



Refraction

Dielectrics

Materials such as water and glass require two types of light transport:

1. Transmission
2. Reflection

Both of these happen at each *medium boundary*.



Refraction

Dielectrics

The direction of the transmitted vector $\vec{T}$ depends on the refraction indices n_1, n_2 of the media separated by the surface. According to Snell's Law:

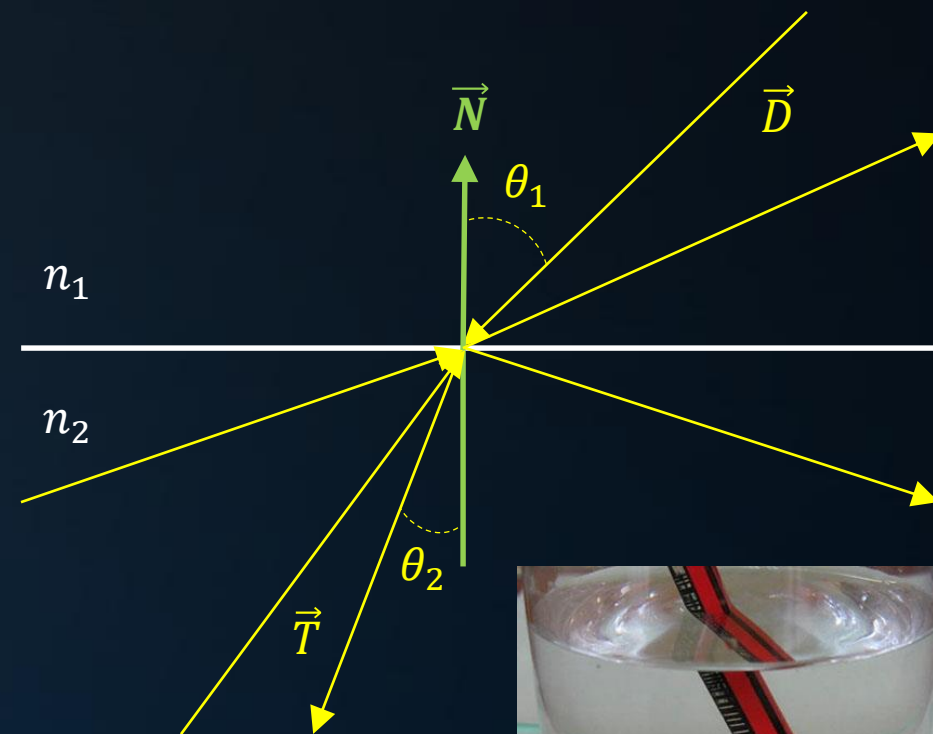
$$n_1 \sin \theta_1 = n_2 \sin \theta_2$$

or

$$\frac{n_1}{n_2} \sin \theta_1 = \sin \theta_2$$

Note: left term may exceed 1, in which case θ_2 cannot be computed. Therefore:

$$\frac{n_1}{n_2} \sin \theta_1 = \sin \theta_2 \Leftrightarrow \sin \theta_1 \leq \frac{n_2}{n_1} \Rightarrow \theta_{critical} = \arcsin \left(\frac{n_2}{n_1} \sin \theta_2 \right)$$



Refraction

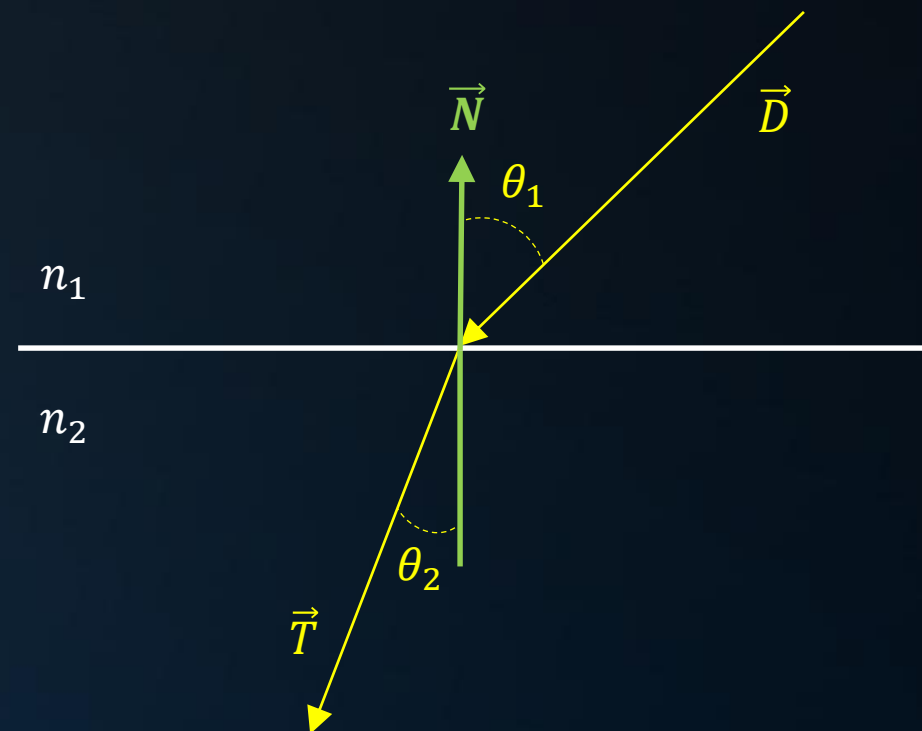
Dielectrics

$$\frac{n_1}{n_2} \sin \theta_1 = \sin \theta_2 \Leftrightarrow \sin \theta_1 \leq \frac{n_2}{n_1}$$

$$k = 1 - \left(\frac{n_1}{n_2} \right)^2 (1 - \cos \theta_1^2)$$

$$\vec{T} = \begin{cases} TIR, & \text{for } k < 0 \\ \frac{n_1}{n_2} \vec{D} + \vec{N} \left(\frac{n_1}{n_2} \cos \theta_1 - \sqrt{k} \right), & \text{for } k \geq 0 \end{cases}$$

Note: $\cos \theta_1 = \vec{N} \cdot -\vec{D}$, and $\frac{n_1}{n_2}$ should be calculated only once.



* For a full derivation, see http://www.flipcode.com/archives/reflection_transmission.pdf



Refraction

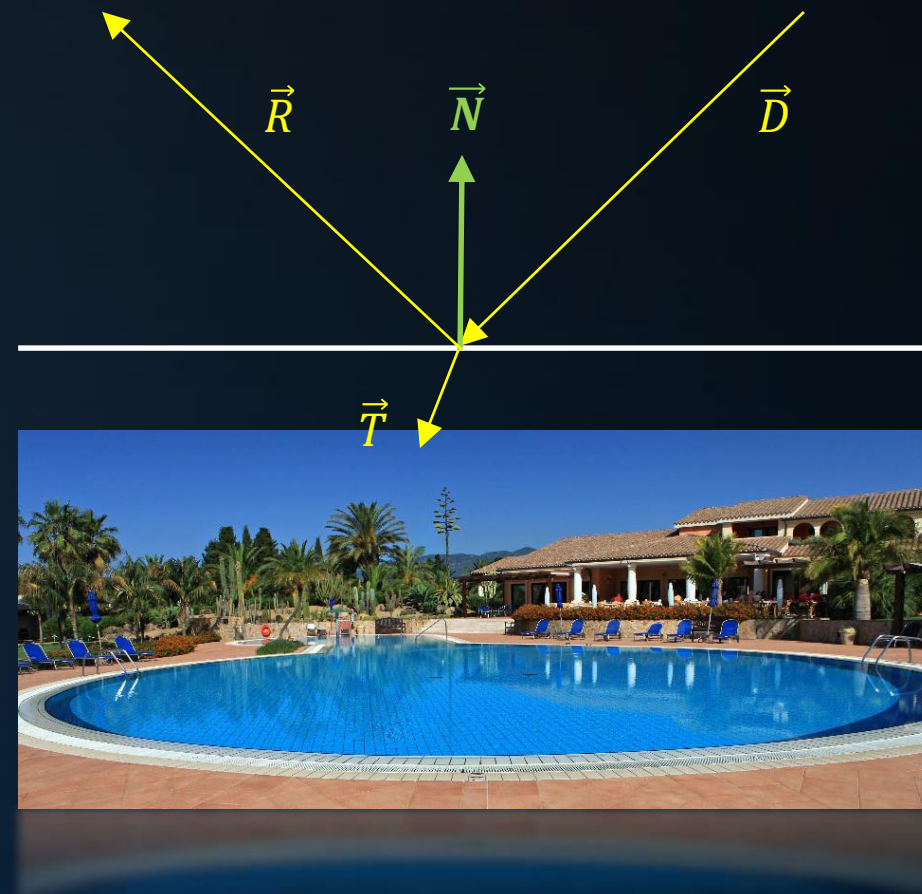
Dielectrics

A typical dielectric transmits *and* reflects light.

```

ics
& (depth < MAXDEPTH)
{
    // Inside / Outside
    int nt = nt / nc, nde = nde / nc;
    double r0s2t = 1.0f - nnt * nnt;
    double D, N;
    // ...
    at a = nt - nc, b = nt + nc;
    double Tr = 1 - (R0 + (1 - R0) * r0s2t);
    double R = (D * nnt - N * (D0 + (1 - D0) * r0s2t));
    // ...
    E * diffuse;
    // ...
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N;
        refl * E * diffuse;
        // ...
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    // estimation - doing it properly, closely following wall
    if;
    radiance = SampleLight( &rand, I, &L, &light );
    // ...
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance);
    // ...
    random walk - done properly, closely following wall
    survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    // ...
}

```



Refraction

Dielectrics

A typical dielectric transmits *and* reflects light.

Based on the *Fresnel equations*, the reflectivity of the surface for non-polarized light is formulated as:

$$F_r = \frac{1}{2} \left(\left| \frac{n_1 \cos \theta_i - n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2}}{n_1 \cos \theta_i + n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2}} \right|^2 + \left| \frac{n_1 \cos \theta_i - n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2}}{n_1 \cos \theta_i + n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2}} \right|^2 \right)$$



Refraction

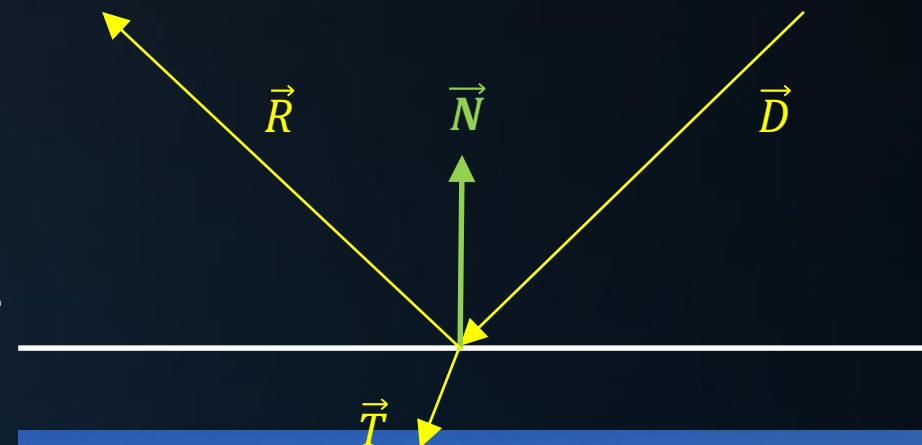
Dielectrics

In practice, we use Schlick's approximation:

$$F_r = R_0 + (1 - R_0)(1 - \cos\theta)^5, \text{ where } R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2.$$

Based on the law of conservation of energy:

$$F_t = 1 - F_r$$



Today's Agenda:

- Reflections
- Refraction
- Recursion
- Shading models
- TODO



Recursion

Whitted-style Ray Tracing, Pseudocode

```
Color Trace( vec3 O, vec3 D )
{
    I, N, mat = IntersectScene( O, D );
    if (!I) return BLACK;
    return DirectIllumination( I, N ) * mat.diffuseColor;
}
```

```
Color DirectIllumination( vec3 I, vec3 N )
{
    vec3 L = lightPos - I;
    float dist = length( L );
    L *= (1.0f / dist);
    if (!IsVisible( I, L, dist )) return BLACK;
    float attenuation = 1 / (dist * dist);
    return lightColor * dot( N, L ) * attenuation;
}
```

Todo:

- Implement IntersectScene
- Implement IsVisible



Recursion

Whitted-style Ray Tracing, Pseudocode

```

Color Trace( vec3 O, vec3 D )
{
    I, N, mat = IntersectScene( O, D );
    if (!I) return BLACK;
    if (mat.isMirror())
    {
        return Trace( I, reflect( D, N ) ) * mat.diffuseColor;
    }
    else
    {
        return DirectIllumination( I, N ) * mat.diffuseColor;
    }
}

```

Todo:

Handle partially
reflective surfaces.



Recursion

Whitted-style Ray Tracing, Pseudocode

```

Color Trace( vec3 O, vec3 D )
{
    I, N, mat = IntersectScene( O, D );
    if (!I) return BLACK;
    if (mat.isMirror())
    {
        return Trace( I, reflect( D, N ) ) * mat.diffuseColor;
    }
    else if (mat.IsDielectric())
    {
        f = Fresnel( ... );
        return (f * Trace( I, reflect( D, N ) )
            + (1-f) * Trace( I, refract( D, N, ... ) ) ) * mat.DiffuseColor;
    }
    else
    {
        return DirectIllumination( I, N ) * mat.diffuseColor;
    }
}

```

Todo:

- Implement reflect
- Implement refract
- Implement Fresnel
- Cap recursion

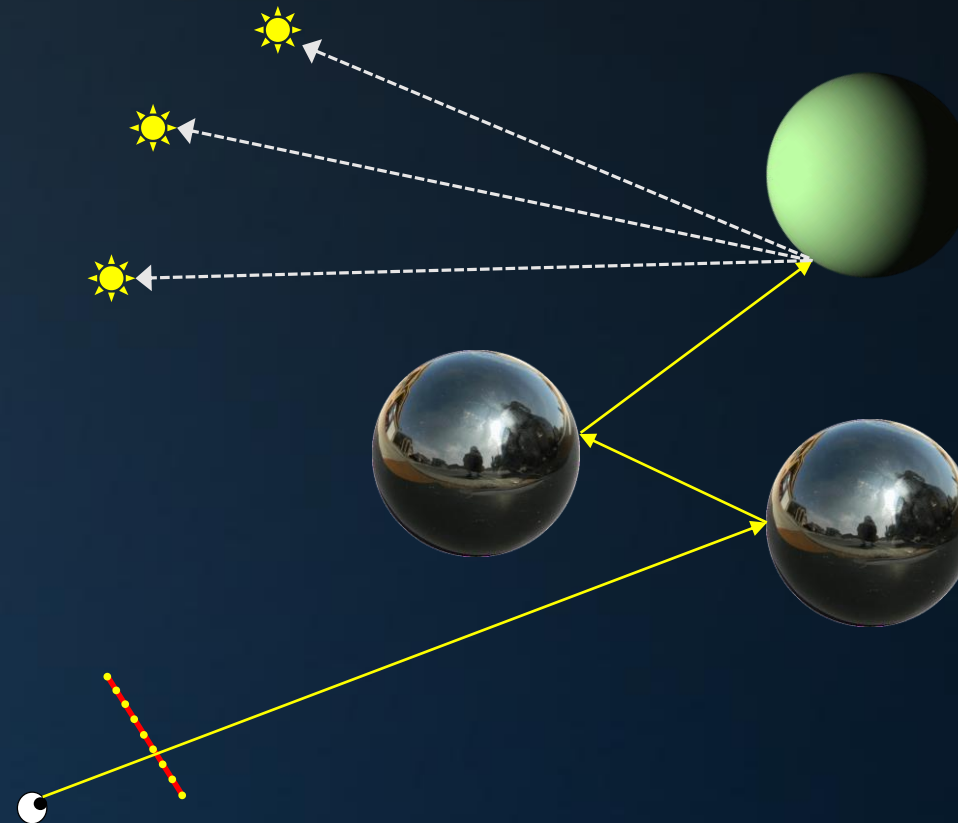


Recursion

Spheres: pure specular

```

    // Ray-sphere intersection
    float t = inside / 1.5;
    float nt = nt / nc;
    float nnt = nt * nt;
    float cos2t = 1.0f - nnt;
    float D, N;
    // ...
    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * cos2t);
    float R = (D * nnt - N * cos2t);
    // ...
    E * diffuse;
    // ...
    refl + refr)) && (depth < MAXDEPTH)
    // ...
    D, N);
    refl * E * diffuse;
    // ...
    MAXDEPTH)
    // ...
    survive = SurvivalProbability( diffuse );
    // ...
    estimation - doing it properly, closely following
    // ...
    if;
    radiance = SampleLight( &rand, I, &L, &light );
    // ...
    e.x + radiance.y + radiance.z) > 0) && (cosThetaOut > 0)
    // ...
    v = true;
    // ...
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    // ...
    at3 factor = diffuse * INVPI;
    // ...
    at weight = Mis2( directPdf, brdfPdf );
    // ...
    at cosThetaOut = dot( N, L );
    // ...
    E * ((weight * cosThetaOut) / directPdf) * (radiance
    // ...
    random walk - done properly, closely following
    // ...
    survive)
    // ...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    // ...
    survive;
    // ...
    pdf;
    // ...
    n = E * brdf * (dot( N, R ) / pdf);
    // ...
    sion = true;
  
```



```
survive = SurvivalProbability( diffuse, I, &t, alignment,
estimation - doing it properly, classically);
if(
radiancex = SampleLight( &rand, I, &t, alignment,
e.x + radiancex.y + radiancex.z) > 0) && !env[0].
```

```

mat3 brdf = SampleDiffuse( diffuse, N, r1, r2, BR, Apdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    refraction = true;

```



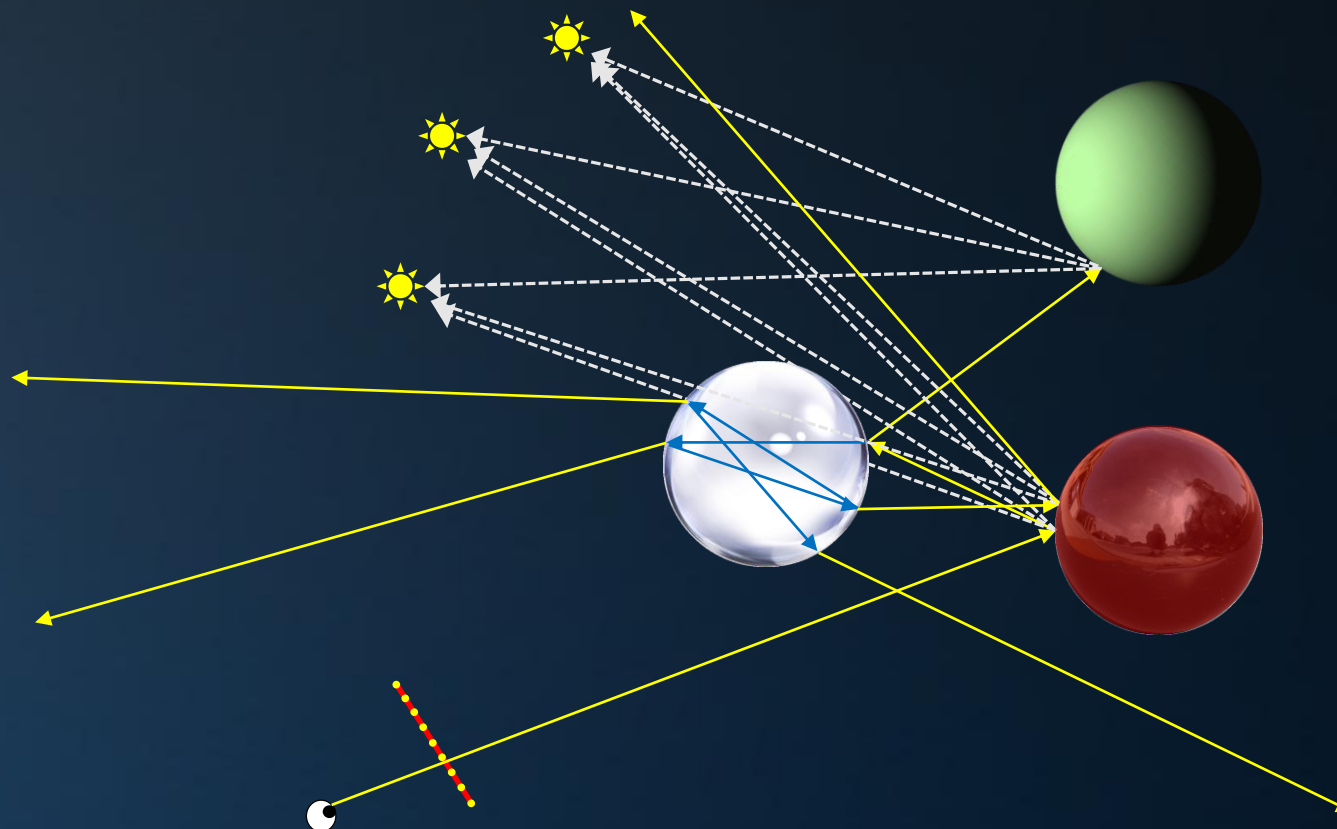
Recursion

Spheres: one 50% specular, one glass sphere

```

ices
& (depth < MAXDEPTH)
{
    // Inside / Outside
    nt = inside / nc;
    nt = nt / nc;
    dds2t = 1.0f - nnt;
    D, N );
    )
    at a = nt - nc;
    at Tr = 1 - (R0 + (1 - R0) * dds2t);
    Tr) R = (D * nnt - N * (dds2t
    E * diffuse;
    = true;
    efl + refr)) && (depth < MAXDEPTH)
    D, N );
    efl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &light );
    e.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
    v = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
    random walk - done properly, closely following
    survive)
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

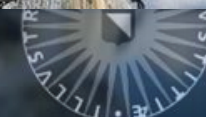
```




```

    if (depth < MAXDEPTH)
    {
        nc = inside / (1 - R);
        nt = nt / nc; ddx = (1 - R) * ddx;
        ds2t = 1.0f - nnt * ddx * ddx;
        D, N );
    }
    nt = nt - nc; b = nt * R;
    at Tr = 1 - (Rb + (1 - Rb) * b);
    Tr) R = (D * nnt - N * (ddx *
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;

```



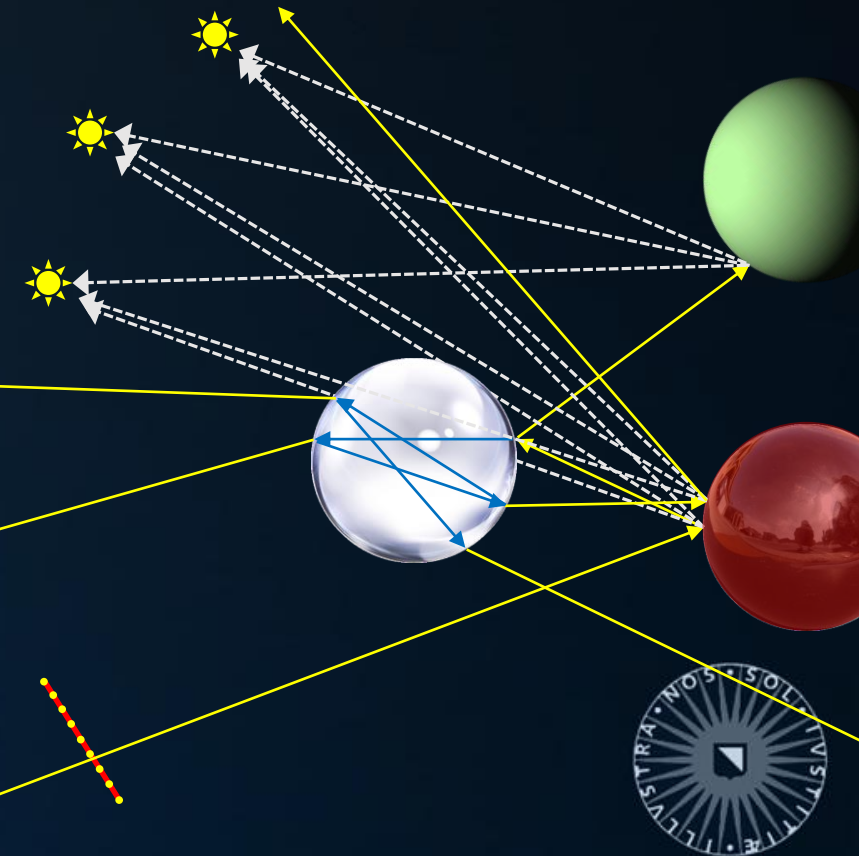
Recursion

Ray Tree

Recursion, multiple light sampling and path splitting in a Whitted-style ray tracer leads to a structure that we refer to as the *ray tree*.

All energy is ultimately transported by a single primary ray.

Since the energy does not increase deeper in the tree (on the contrary), the average amount of energy transported by rays decreases with depth.



Today's Agenda:

- Reflections
- Refraction
- Recursion
- Shading models
- TODO



Shading

Diffuse Material

A diffuse material scatters incoming light in all directions.

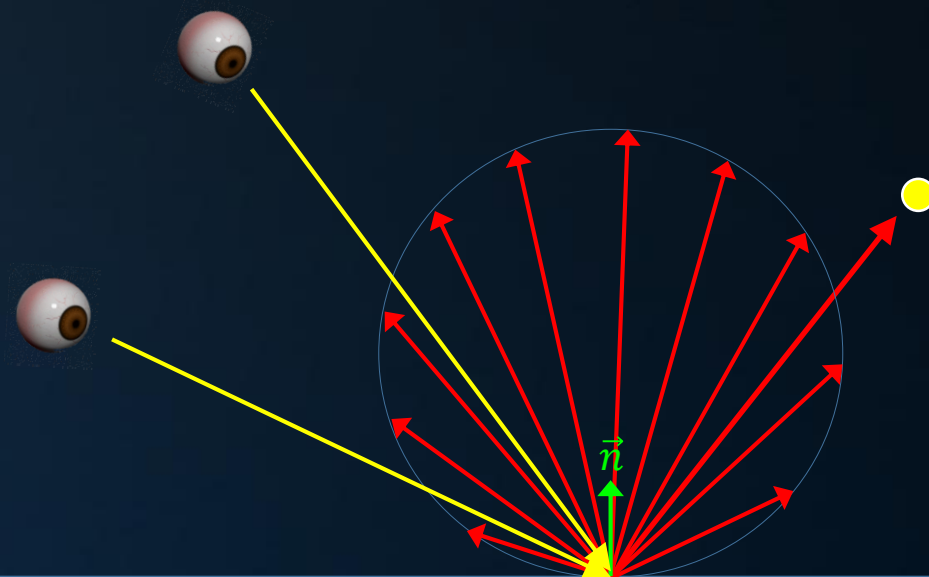
A diffuse material appears the same regardless of eye position.

Incoming: $E_{light} * \frac{1}{dist^2} * \vec{N} \cdot \vec{L}$

Absorption: $C_{material}$

Reflection: $(\vec{V} \cdot \vec{N})$ | *terms cancel out.*

Eye sees: $\frac{1}{(\vec{V} \cdot \vec{N})}$



Shading

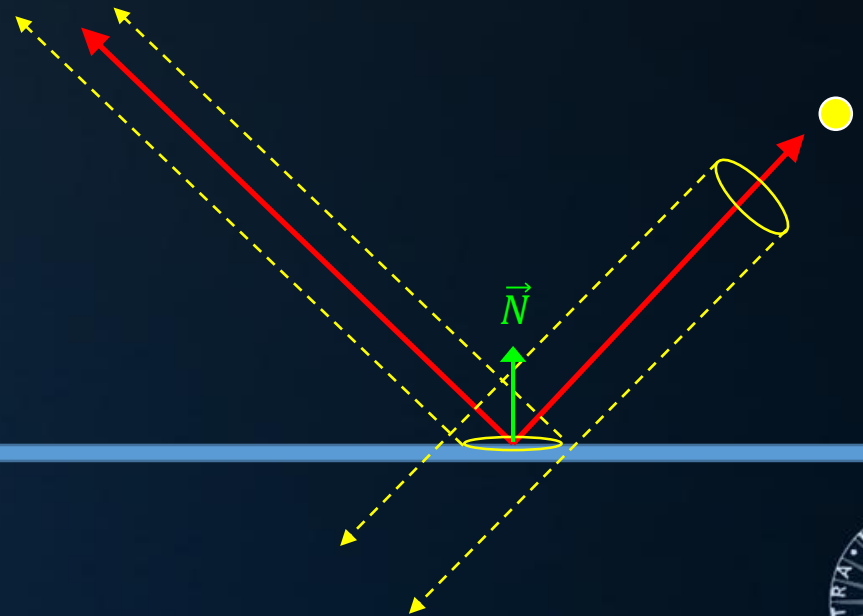
Specular Material

A specular material reflects light from a particular direction in a single outgoing direction.

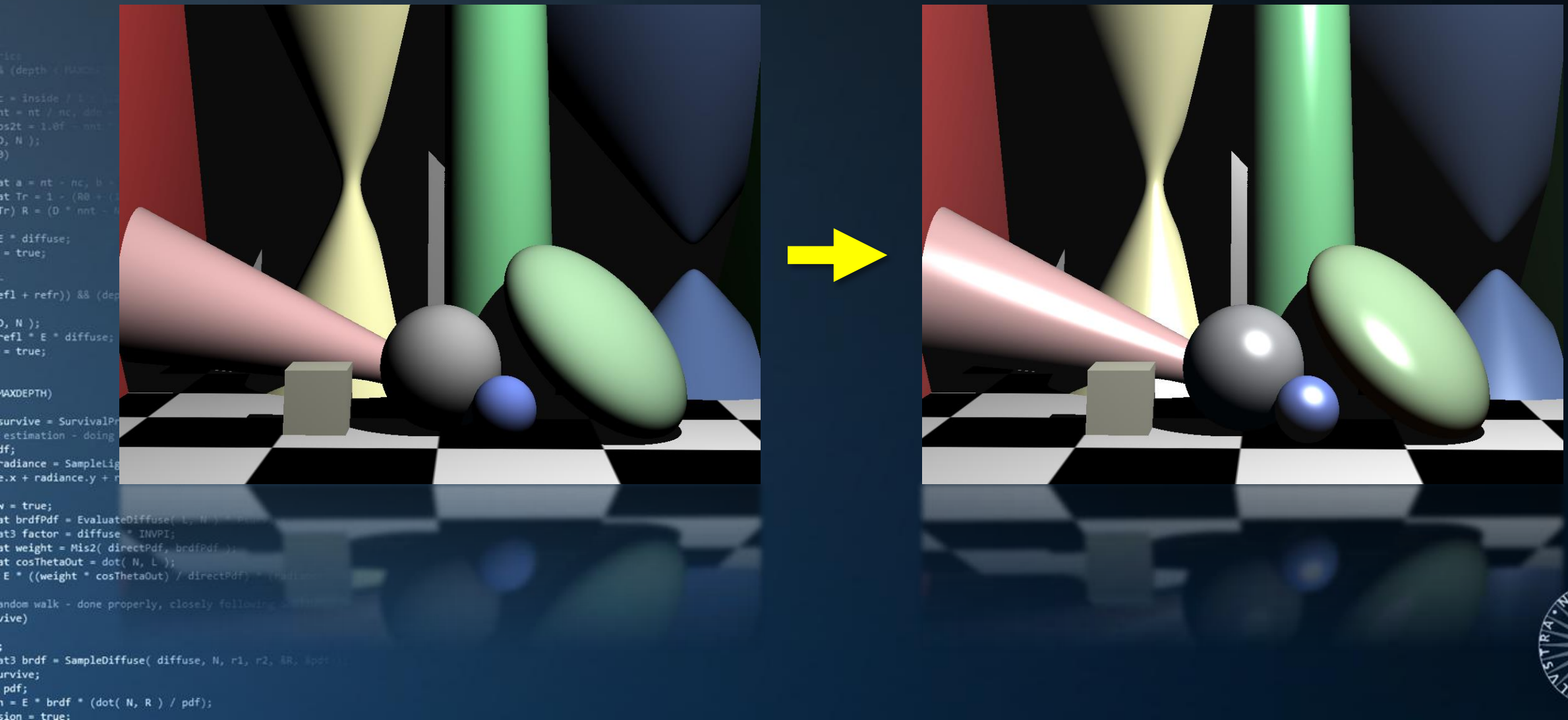
```

    if (depth < MAXDEPTH)
    {
        // Inside the sphere
        Vec r = inside / 1.5;
        Vec nt = nt / nc;
        double r2 = r.x*r.x + r.y*r.y + r.z*r.z;
        double r2t = 1.0f - nnt * r2;
        if (r2t < 0) r2t = 0;
        Vec d = N * (D < 0 ? -1 : 1);
        Vec R = (D * nnt - N * (D < 0 ? 1 : -1));
        Vec refl = E * diffuse;
        refl = true;
        Vec refl = refl + refr;
        if (depth < MAXDEPTH)
        {
            Vec D, N;
            Vec refl = E * diffuse;
            refl = true;
            Vec refl = refl + refr;
            if (depth < MAXDEPTH)
            {
                Vec survive = SurvivalProbability( diffuse );
                // estimation - doing it properly, closely following well-behaved
                if (survive)
                {
                    Vec radiance = SampleLight( &rand, I, &lt;, &light );
                    Vec e.x + radiance.y + radiance.z > 0) && (e.x + radiance.y + radiance.z > 0) && (e.x + radiance.y + radiance.z > 0)
                }
                Vec v = true;
                Vec brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
                Vec t3 factor = diffuse * INVPI;
                Vec weight = Mis2( directPdf, brdf );
                Vec cosThetaOut = dot( N, L );
                Vec E = ((weight * cosThetaOut) / directPdf) * (radiance);
                Vec random walk - done properly, closely following well-behaved
                Vec survive);
                Vec t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
                Vec survive;
                Vec pdf;
                Vec n = E * brdf * (dot( N, R ) / pdf);
                Vec n = true;
            }
        }
    }

```



Shading



Shading

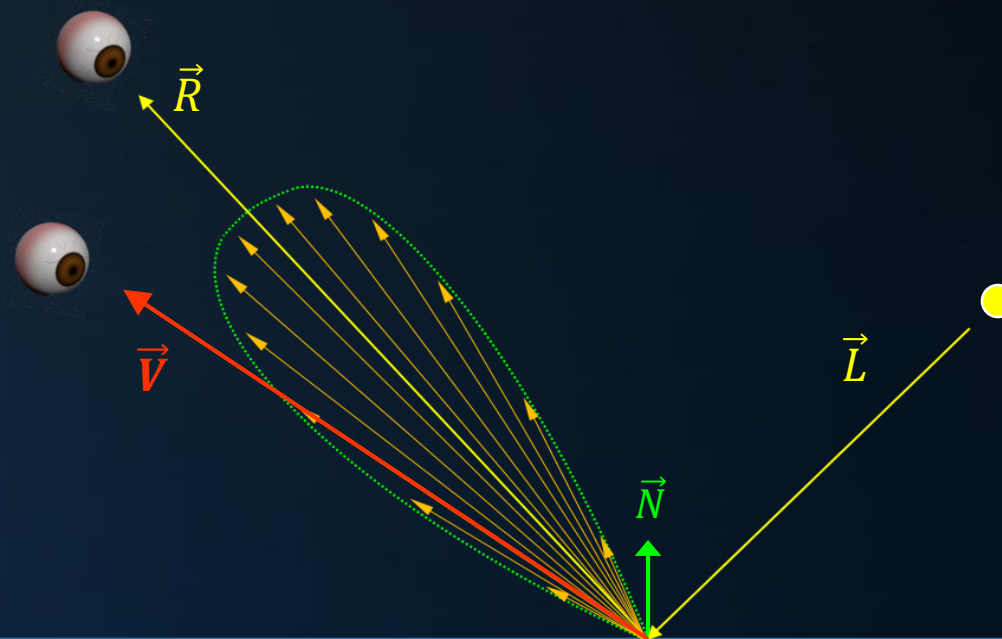
Glossy Material

A glossy material reflects *most* light along the reflected vector.

$$\vec{R} = \vec{L} - 2(\vec{L} \cdot \vec{N})\vec{N}$$

For other directions, the amount of energy is:

$(\vec{V} \cdot \vec{R})^\alpha$, where exponent α determines the specularity of the surface.



Shading

Phong Shading

Complex materials can be obtained by blending diffuse and glossy.

$$I = C_{material} * \left((1 - f_{spec}) (\vec{N} \cdot \vec{L}) + f_{spec} (\vec{V} \cdot \vec{R})^\alpha \right)$$

where

α is the specularity of the glossy reflection;

f_{spec} is the glossy part of the reflection;

$1 - f_{spec}$ is the diffuse part of the reflection.

Note that the glossy reflection *only* reflects light sources.



Today's Agenda:

- Reflections
- Refraction
- Recursion
- Shading models
- TODO



TODO

Limitations of Whitted-style

```

    if (depth < MAXDEPTH)
    {
        t = inside / 1.5;
        nt = nt / nc; nct = nc / nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }

    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (RB + (1 - RB) * cos2t);
    R = (D * nnt - N * (cos2t > 0 ? 1 : -1));

    E * diffuse;
    = true;

    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }

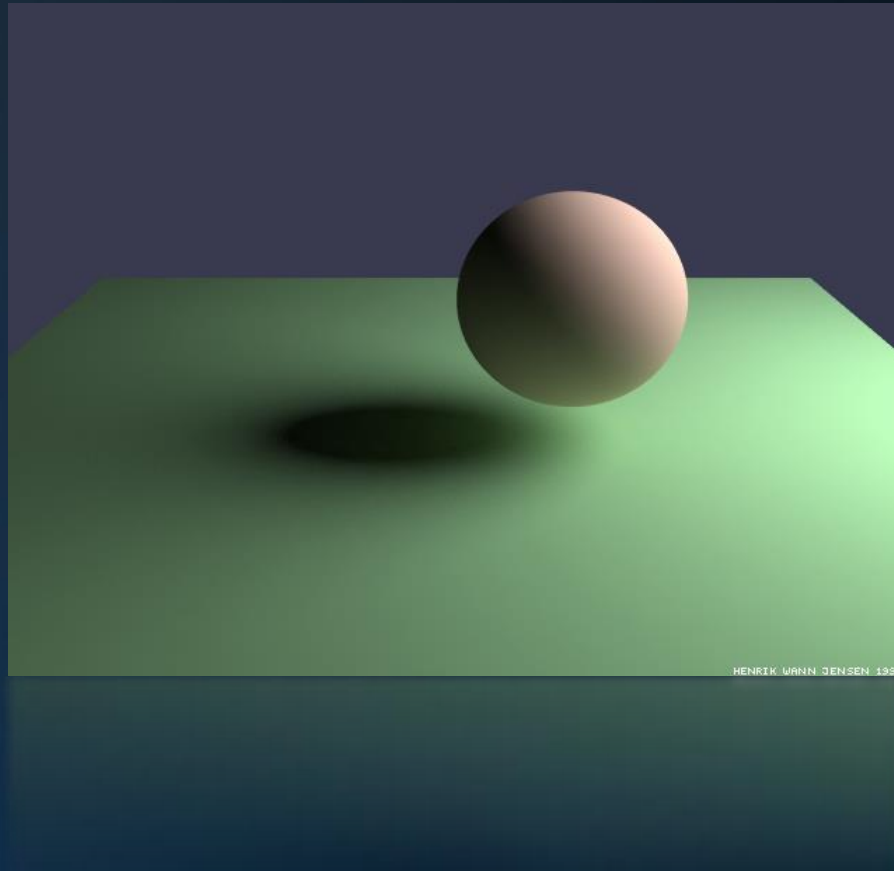
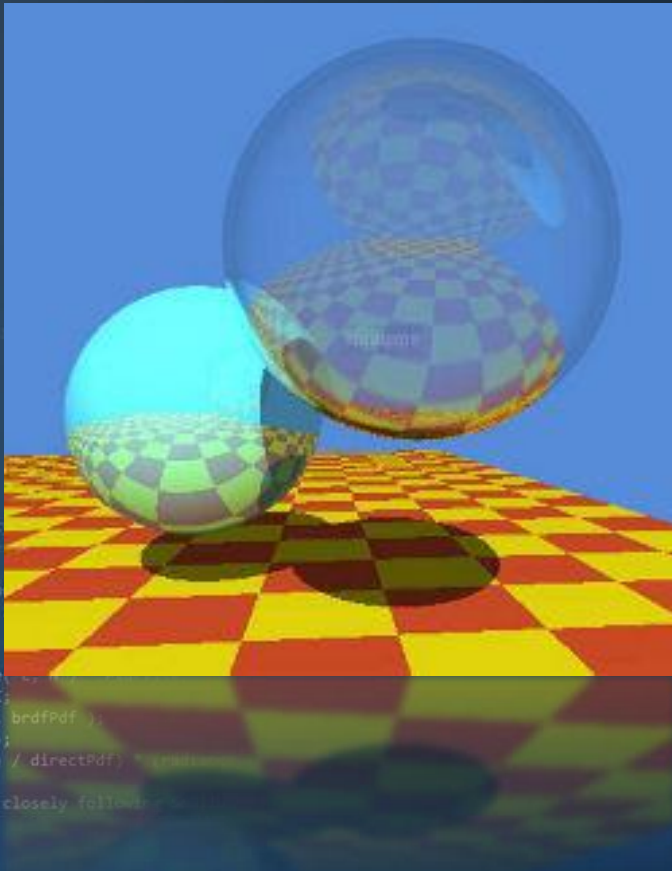
    MAXDEPTH)

    survive = SurvivalProbability(
    estimation - doing it properly
    if;
    radiance = SampleLight( &rand,
    e.x + radiance.y + radiance.z);

    v = true;
    at brdfPdf = EvaluateDiffuse( E, N, r1, r2, &R, &pdf );
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * radiance;

    random walk - done properly, closely following a random walk
    survive)

    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
  
```



TODO

Limitations of Whitted-style



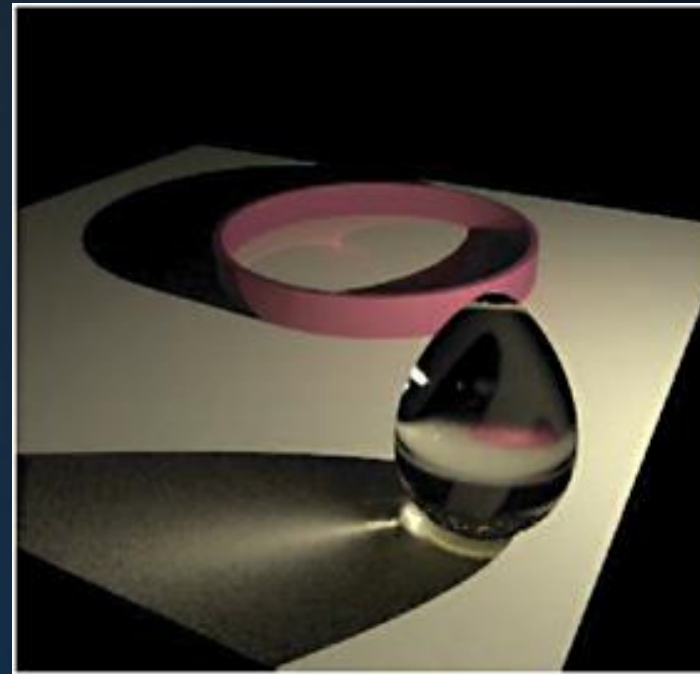
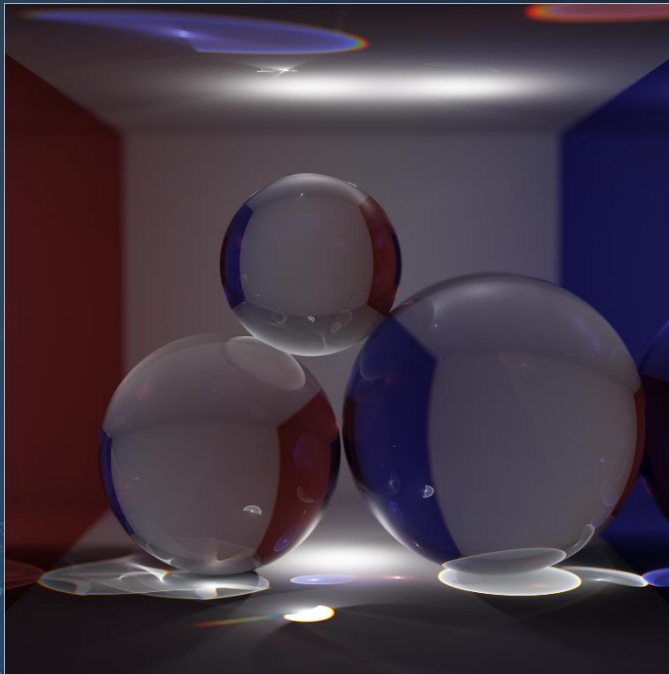
TODO

Limitations of Whitted-style

```

    if (depth < MAXDEPTH)
    {
        // Inside / Outside
        int nt = nt / nc, nde = nde / nc;
        float r2t = 1.0f - nnt * nnt;
        float D, N;
        // ...
        float a = nt - nc, b = nt + nc;
        float Tr = 1 - (R0 + (1 - R0) * a * b);
        float R = (D * nnt - N * (1 - D));
        // ...
        E * diffuse;
        // ...
        refl + refr)) && (depth < MAXDEPTH)
        {
            D, N;
            refl * E * diffuse;
            // ...
            MAXDEPTH)
            survive = SurvivalProbability(
                estimation - doing it properly
            );
            radiance = SampleLight( &rand,
                e.x + radiance.y + radiance.z,
                // ...
            );
            // ...
            w = true;
            at brdfPdf = EvaluateDiffuse( L, N, r1, r2, &R, &pdf );
            at3 factor = diffuse * INVPI;
            at weight = Mis2( directPdf, brdfPdf );
            at cosThetaOut = dot( N, L );
            E * ((weight * cosThetaOut) / directPdf) * (radiance);
            // ...
            random walk - done properly, closely following walk
            (survive)
            // ...
            at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
            survive;
            pdf;
            n = E * brdf * (dot( N, R ) / pdf);
            // ...
            // ...
        }
    }

```



Limitations of Whitted-style



TODO

Limitations of Whitted-style

```

    if (depth > MAXDEPTH)
        return Color(0, 0, 0);

    if (inside) {
        nt = inside / 1.5;
        nt = nt / nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }

    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 - R0) * cos2t);
    R = (D * nnt - N * Tr);

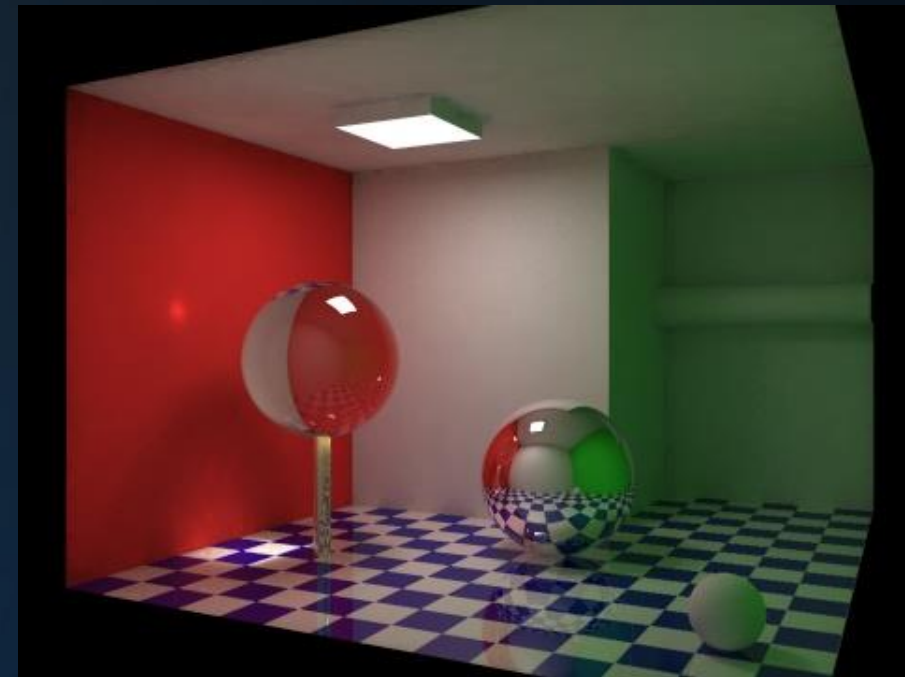
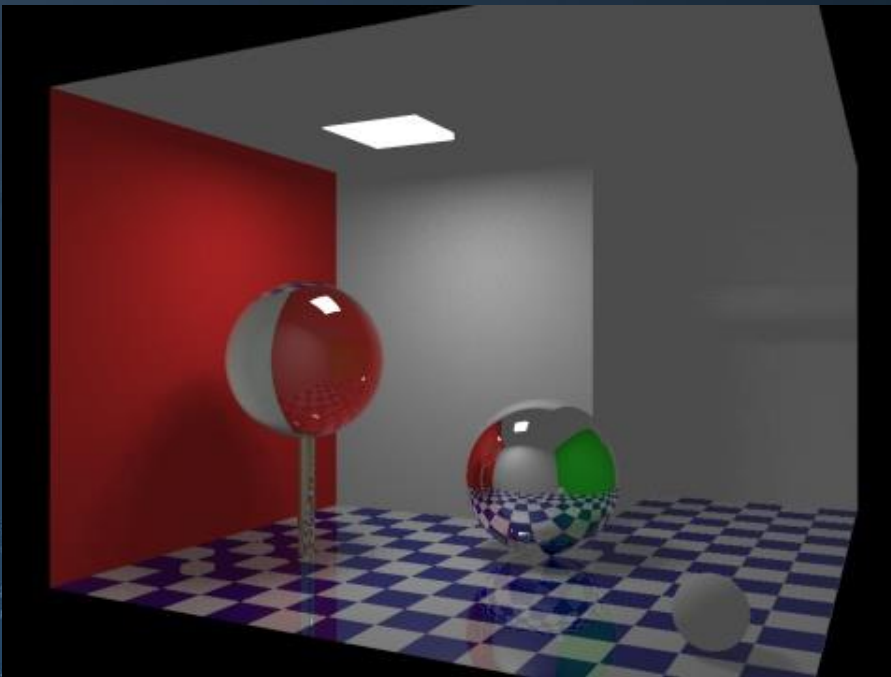
    E * diffuse;
    = true;

    refl + refr)) && (depth <
    D, N );
    refl * E * diffuse;
    = true;

    MAXDEPTH)

    survive = SurvivalProbab
    estimation - doing it p
    df;
    radiance = SampleLight(
    e.x + radiance.y + radia

```



Tutorial 2 - Geometry

Basic geometric entities

Exercise 1.

Given: a line in $\mathbb{R}^2$ through two points $(-3,0)$ and $(2,2)$.

- What is the slope-intersect form of this line?
- Verify your answer for a) for the two given points.
- What is the implicit form of the line?
- Determine two normals for the line, with different directions.
- What is the relation between the distance of the line to the origin, 'C' in the general representation, and the magnitude of the normal?

Exercise 2.

Let l be a line in $\mathbb{R}^2$ through the origin, and let $\vec{n}$ be a normal vector for l .

- Show geometrically that all points p that lie on the line satisfy $\vec{n} \cdot \vec{p} = 0$.

Now, let l be a line in $\mathbb{R}^2$ that does not go through the origin, and let $\vec{n}$ be a normal vector for l .

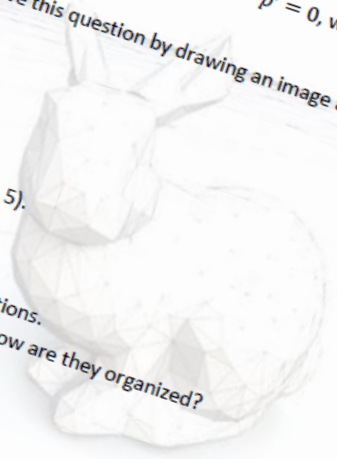
- Show geometrically that all points p that lie on the line satisfy $\vec{n} \cdot \vec{p} - \vec{n} \cdot \vec{p}' = 0$, where p' is an arbitrary point on the line.

Note: "geometrically" here means that you can solve this question by drawing an image and using it to explain the solution.)

Exercise 3.

Given: a line in $\mathbb{R}^3$ through two points, $(-4,1,1)$ and $(2, 2, 5)$.

- What is the length of this line segment?
- What is the parametric representation of this line?
- Determine two normals for the line, with different directions.
- How many normals of unit length exist for this line, and how are they organized?



INFOGR – Computer Graphics

Jacco Bikker - April-July 2016 - Lecture 5: “Ray Tracing (3)”

END of “Ray Tracing (3)”

next lecture: “Boxes”

