

Solo Race

Race against Ghost

View Ghost

More Ghosts



Upload Ghost Data



MattC



1:42.953



1 0:33.9 16

2 0:34.303

3 0:34.734



Blue Falcon



Cyber Slick



MKTV Parafoil

Mii



Mario Circuit



A OK

0:48.121

10 2/3



INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2019

Lecture 10: “Shaders”

Welcome!



INFOGR – Computer Graphics

P2 - 2019



```

ics
& (depth < MAXDEPTH)
{
    if (nt < inside ? 1.0 : 0.2)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
}

```

```

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

```

```

E * diffuse;
= true;

```

```

efl + refr)) && (depth < MAX

```

```

D, N );
refl * E * diffuse;
= true;

```

```

MAXDEPTH)

```

```

survive = SurvivalProbability
estimation - doing it proper
df;
radiance = SampleLight( &rand
e.x + radiance.y + radiance.z

```

```

w = true;
at brdfPdf = EvaluateDiffuse(
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf,
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut)

```

```

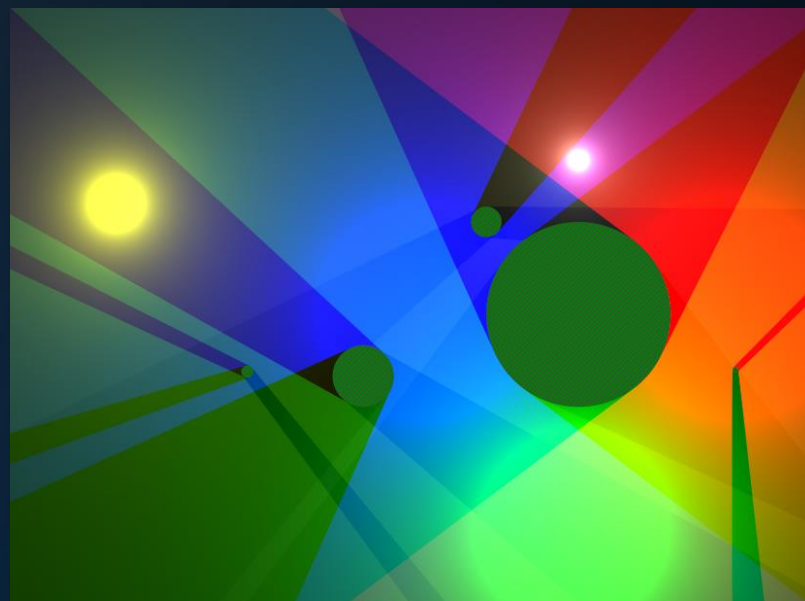
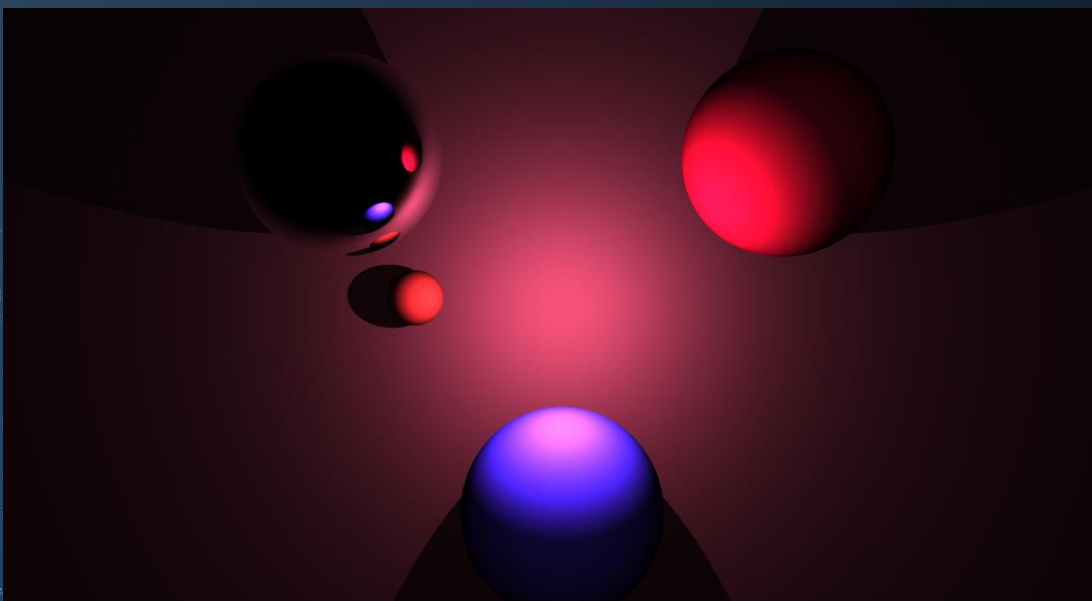
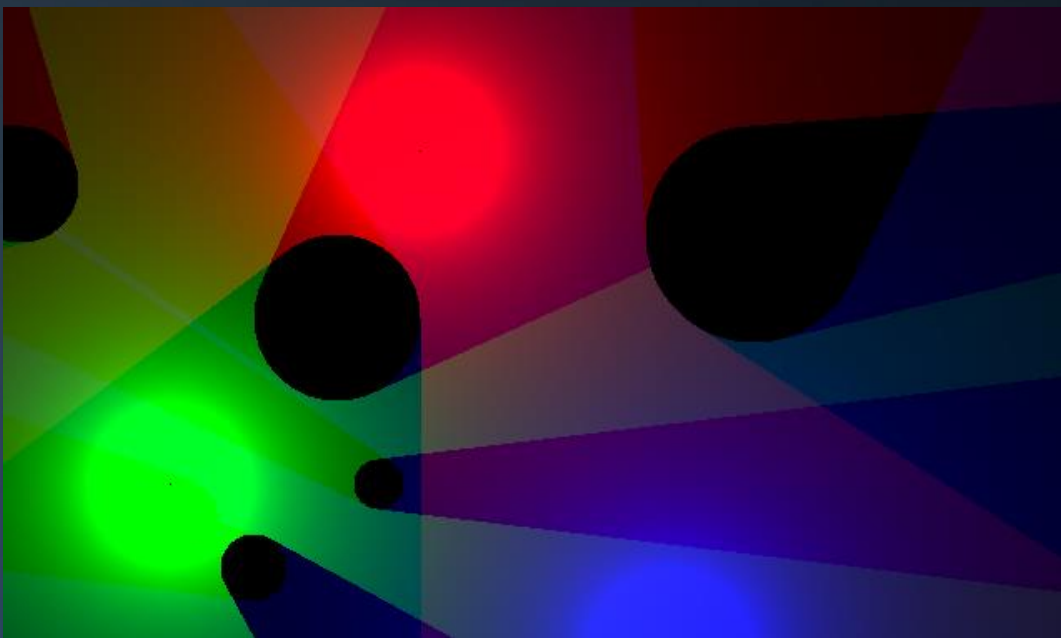
random walk - done properly, c
vive)

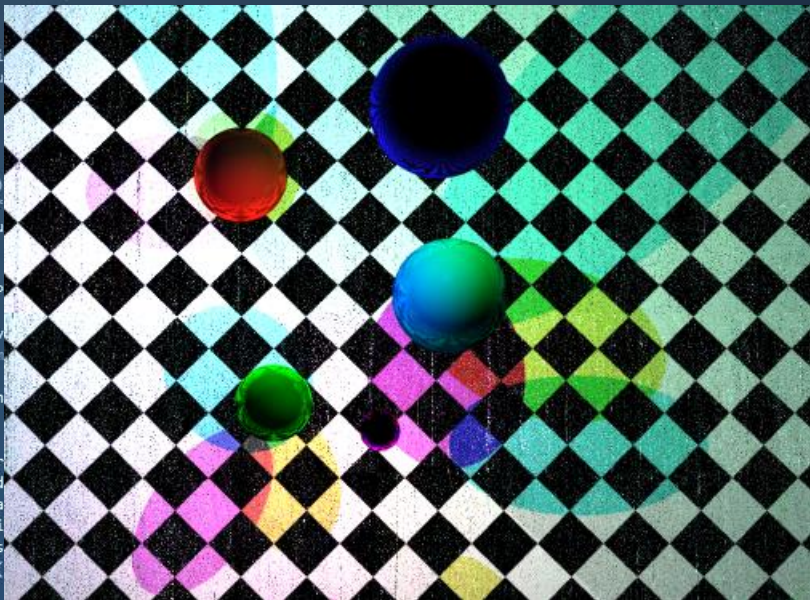
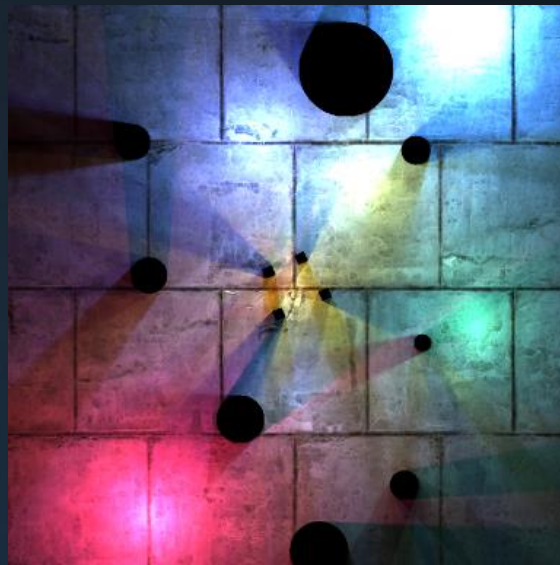
```

```

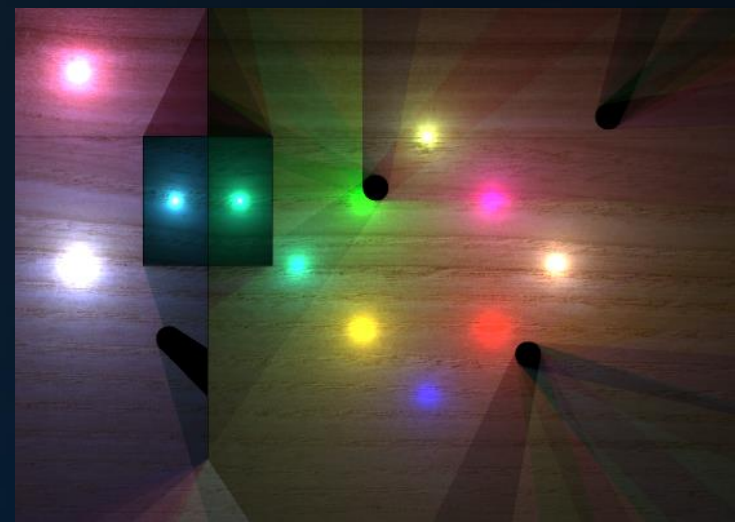
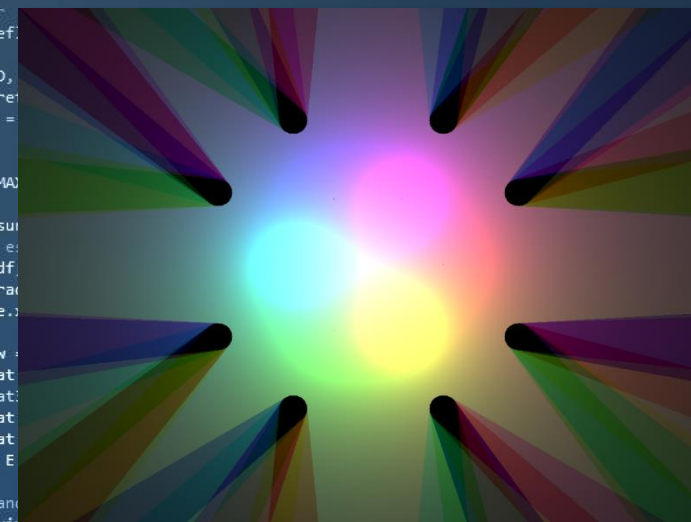
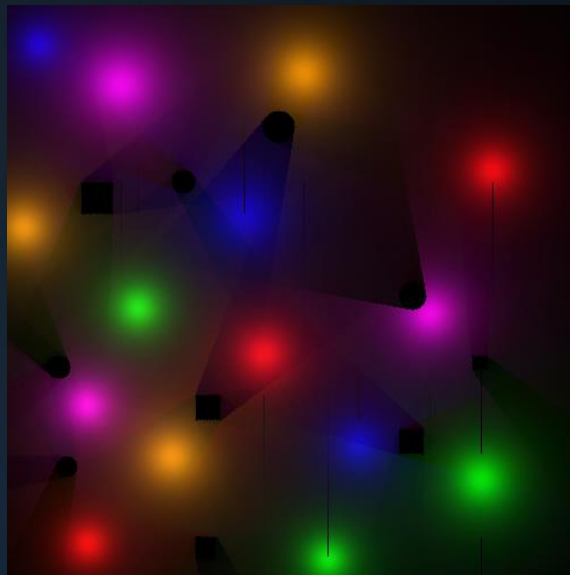
;
at3 brdf = SampleDiffuse( dif
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```





```
ics  
& (depth < N  
t = inside ?  
nt = nt / nd  
os2t = 1.0f  
D, N );  
0)  
at a = nt -  
at Tr = 1 -  
Tr) R = (D *  
E * diffuse;  
= true;
```



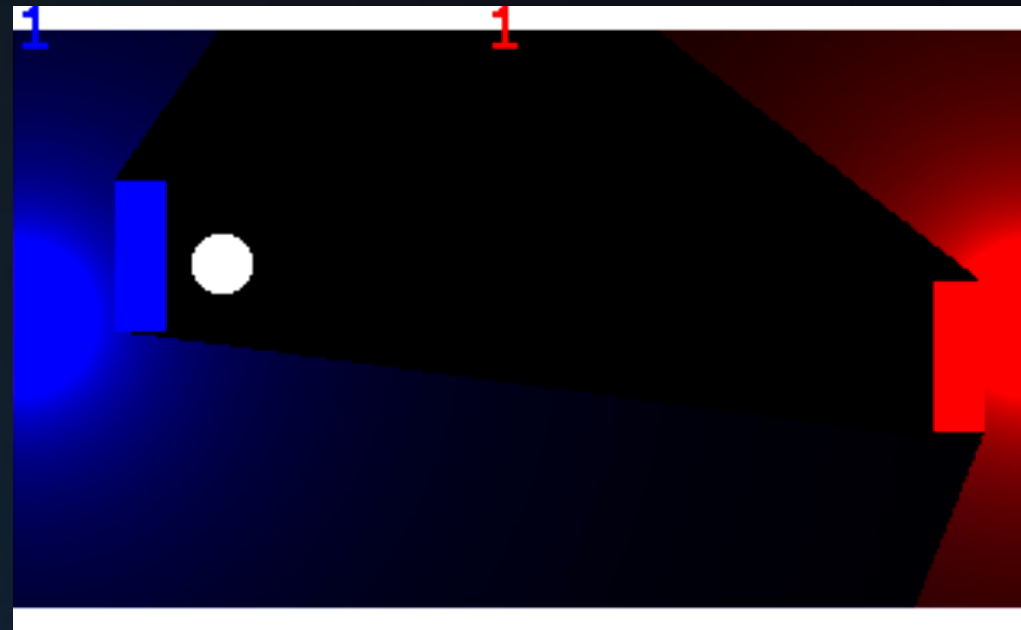
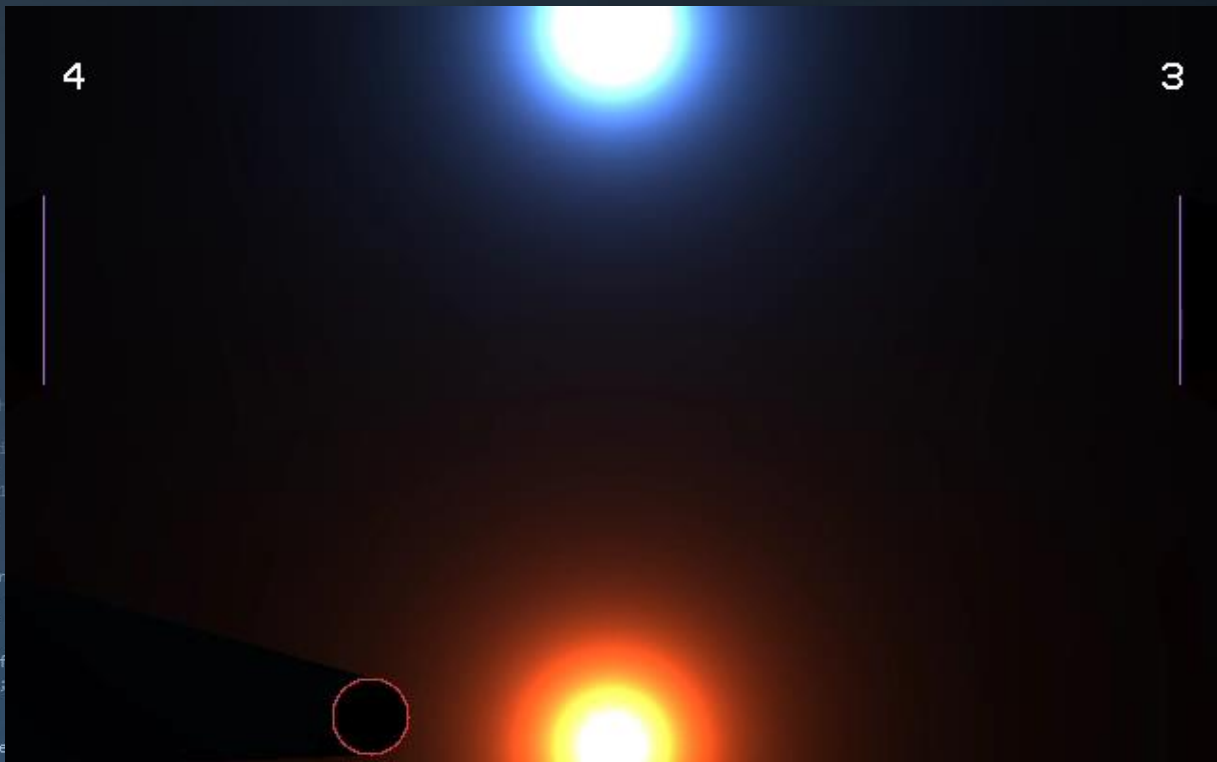
```
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );  
urvive;  
pdf;  
n = E * brdf * (dot( N, R ) / pdf);  
ision = true;
```



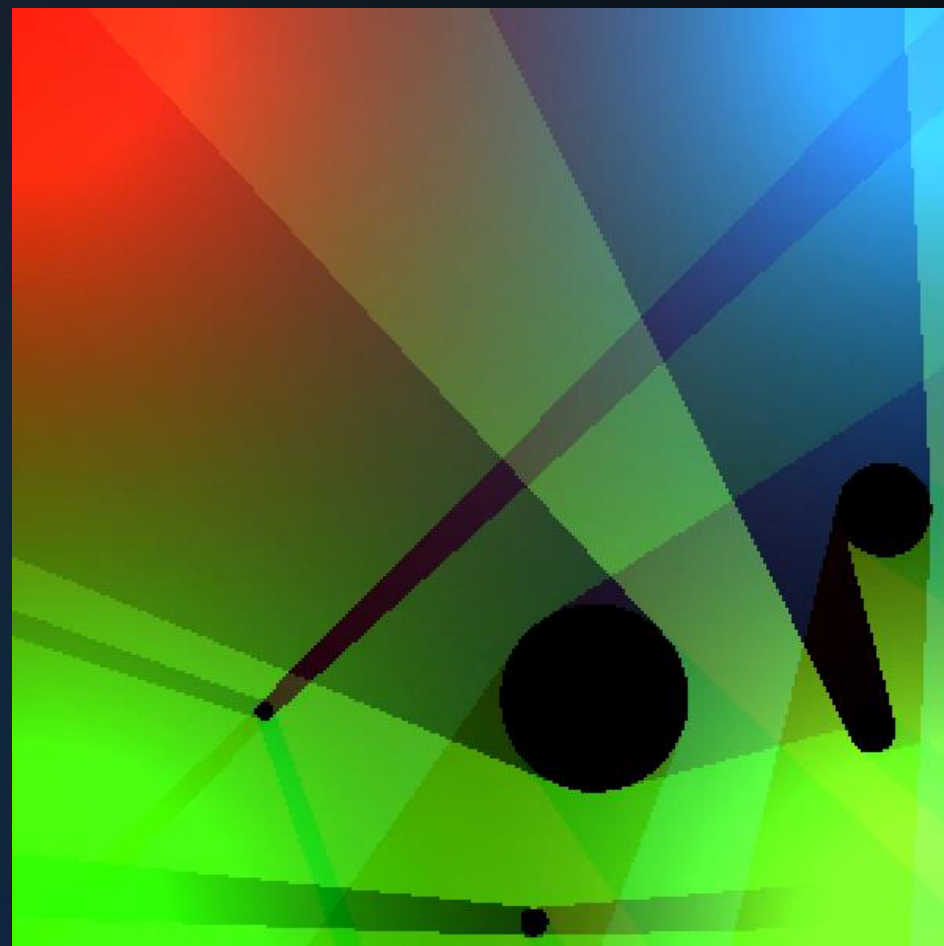
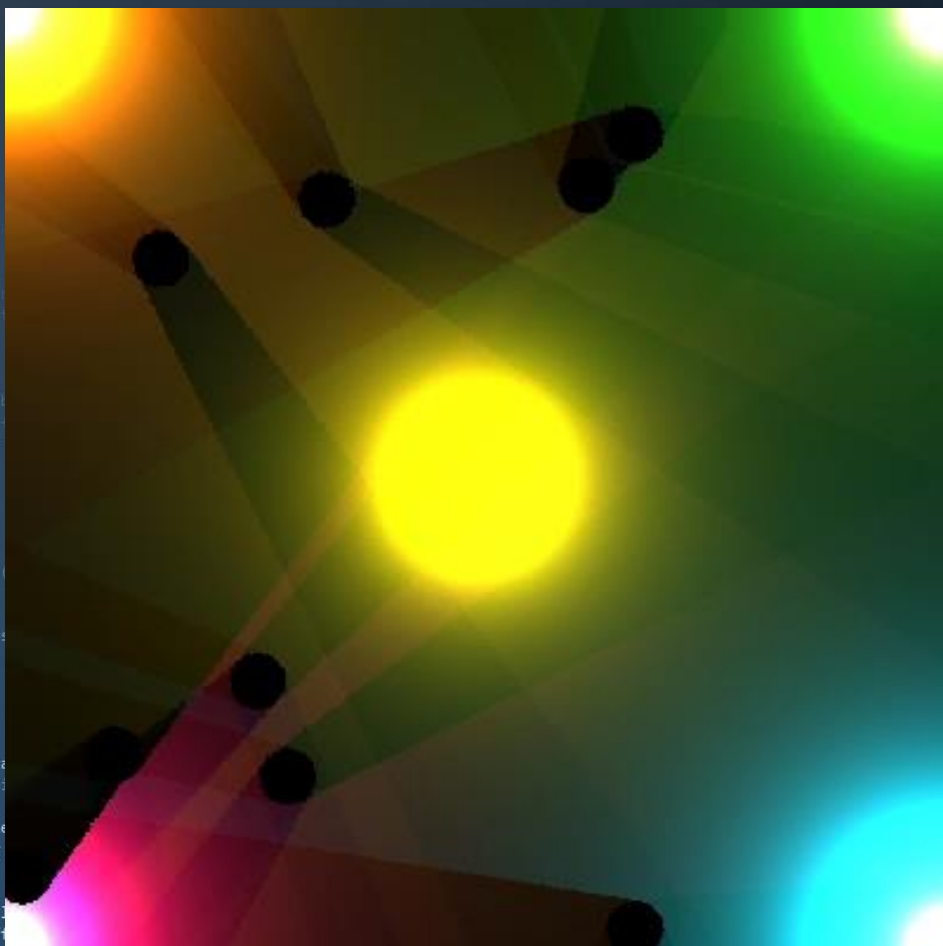
```
ics
& (depth
t = insi
nt = nt
ps2t = 1
D, N );
0);
at a = r
at Tr =
Tr) R =
E * diff
= true;
-
efl + re
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
```

```
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following Small's
df;
radiance = SampleLight( &rand, I, &L, &lightPos );
e.x + radiance.y + radiance.z > 0) && (dot( N, L ) > 0)
w = 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
andom walk - done properly, closely following Small's
ive)
```

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



```
ics
& (depth < MAXDEPTH)
{
    if (nt < inside ? 1 : 0)
    {
        nt = nt / nc; ddr = ddr * ddr;
        cos2t = 1.0f - nnt;
        D, N );
    }
    else
    {
        at a = nt - nc;
        at Tr = 1 - (R0 + R1 * a);
        (Tr) R = (D * nnt);
        E * diffuse;
        = true;
        = true;
        (refl + refr)) && (
        D, N );
        refl * E * diffuse;
        = true;
        MAXDEPTH)
        survive = Survive;
        estimation - do;
        df;
        radiance = Sample
        e.x + radiance.y
        w = true;
        at brdfPdf = Eval
        at3 factor = diff
        at weight = Mis2( directPdf, brdfPdf );
        at cosThetaOut = dot( N, L );
        E * ((weight * cosThetaOut) / directPdf) * (radiance
        random walk - done properly, closely following Small's
        vive)
        ;
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }
}
```



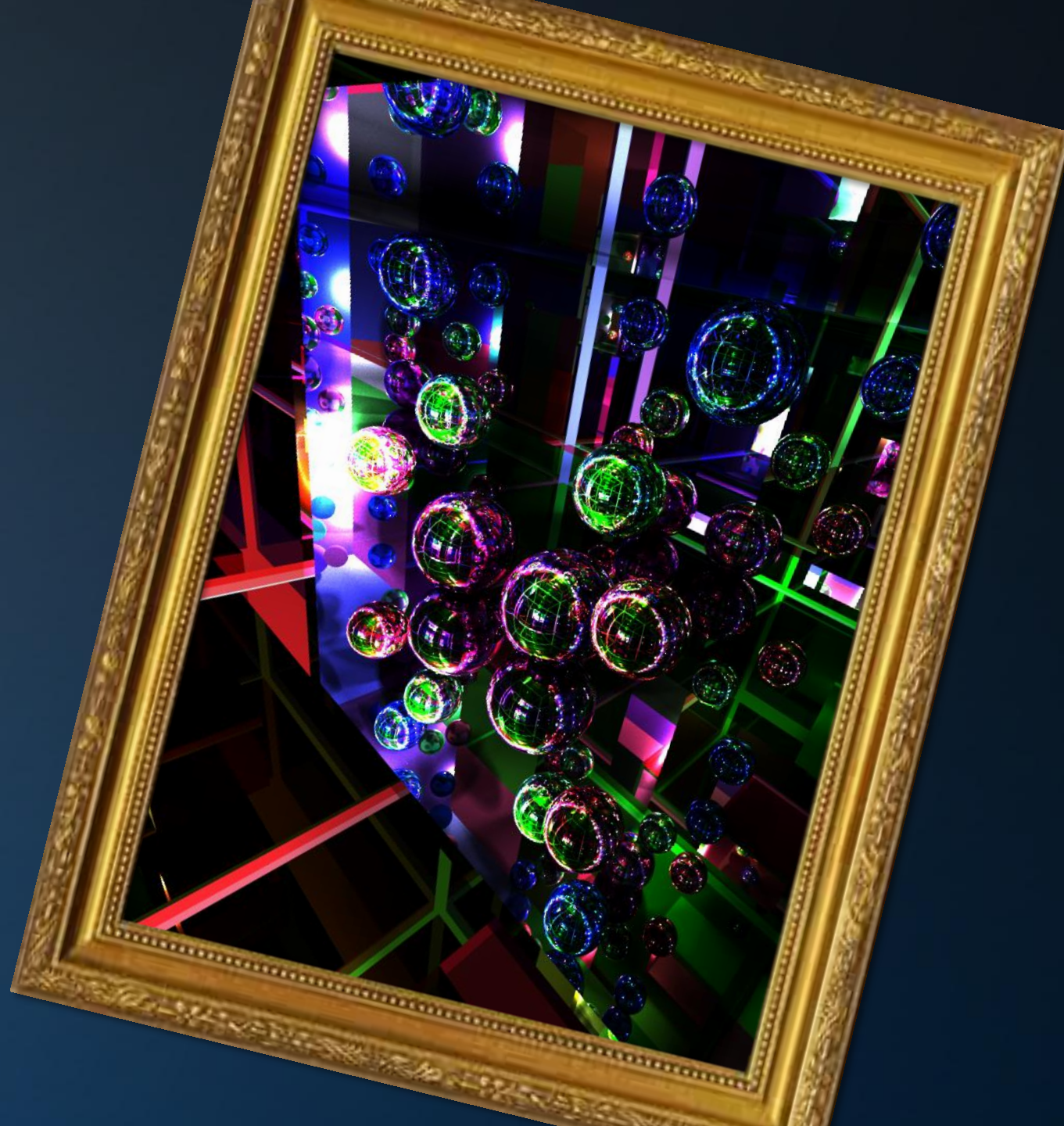
```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.25 * nnt)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    {
        a = nt - nc; b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * ddn);
        Tr) R = (D * nnt - N * (ddn > 0 ? 1 : -1));
    }
    E * diffuse;
    = true;
    {
        refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    {
        survive = SurvivalProbability( diffuse );
        estimation - doing it properly, closely following
        df;
        radiance = SampleLight( &rand, I, &L, &lightPos,
        e.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
    {
        w = 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 Small's
    (survive)
    {
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }
}

```

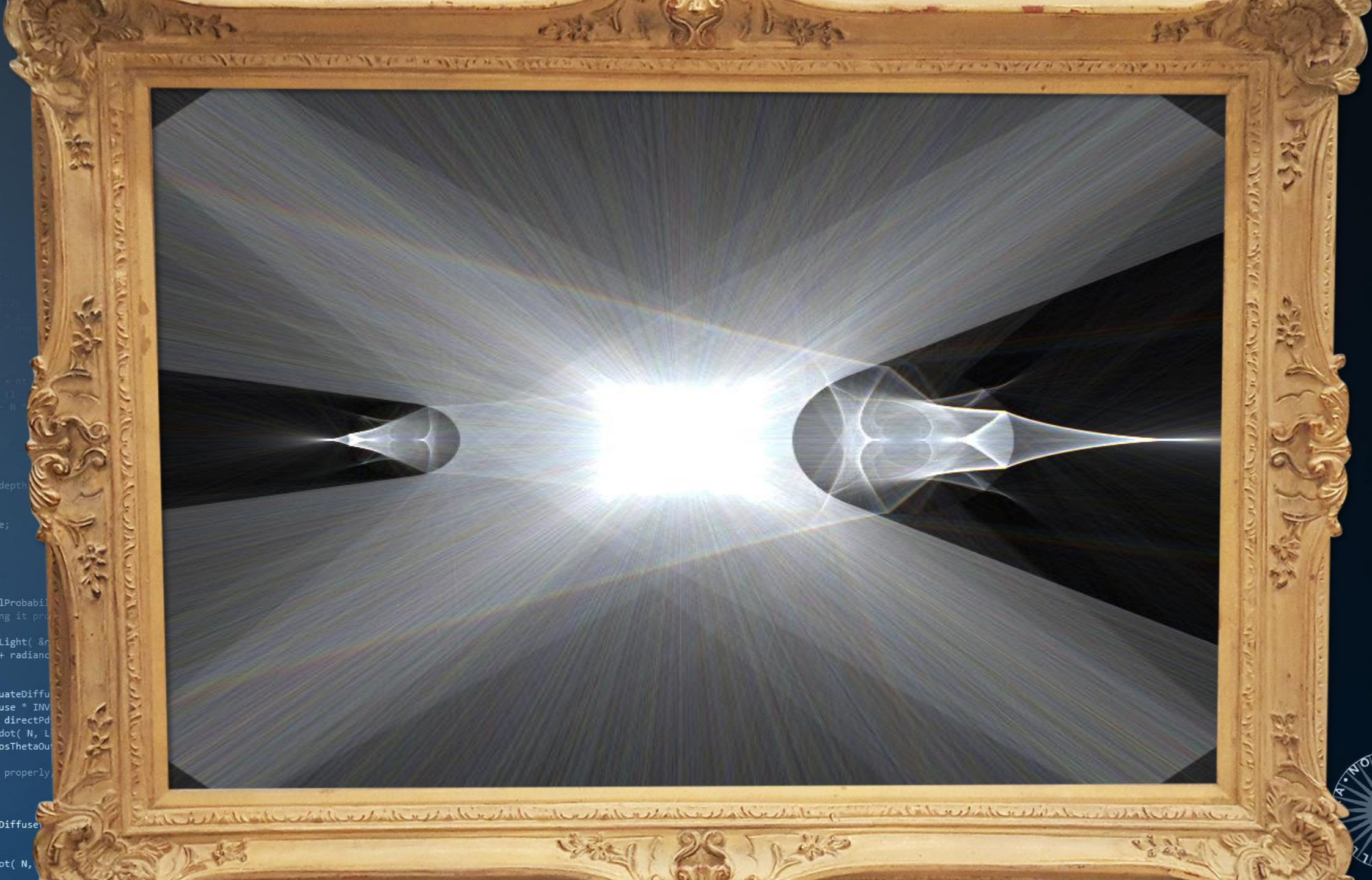


```
ics
& (depth < MAXDEPTH) {
    if (nt < 0) {
        nt = inside ? 1 : -1;
        nnt = nt / nc; ddn = dot(N, N);
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    // Russian roulette
    float a = nt - nc, b = nt + nc;
    float r0 = rand(), r1 = rand();
    float Tr = 1 - (R0 + (1 - R0) * r1 * r1);
    float R = (D * nnt - N * (ddn < 0 ? 1 : 0));
    float E * diffuse;
    bool = true;
    // Russian roulette
    float refl + refr)) && (depth < MAXDEPTH) {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    // Russian roulette - doing it properly, closely following
    pdf;
    radiance = SampleLight( &rand, I, &L, &light );
    if (radiance.x + radiance.y + radiance.z > 0) && (depth <
    w = true;
    float brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    float3 factor = diffuse * INVPI;
    float weight = Mis2( directPdf, brdfPdf );
    float cosThetaOut = dot( N, L );
    float E * ((weight * cosThetaOut) / directPdf) * (radiance
    random walk - done properly, closely following Small's
    (survive)
    ;
    float3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
```





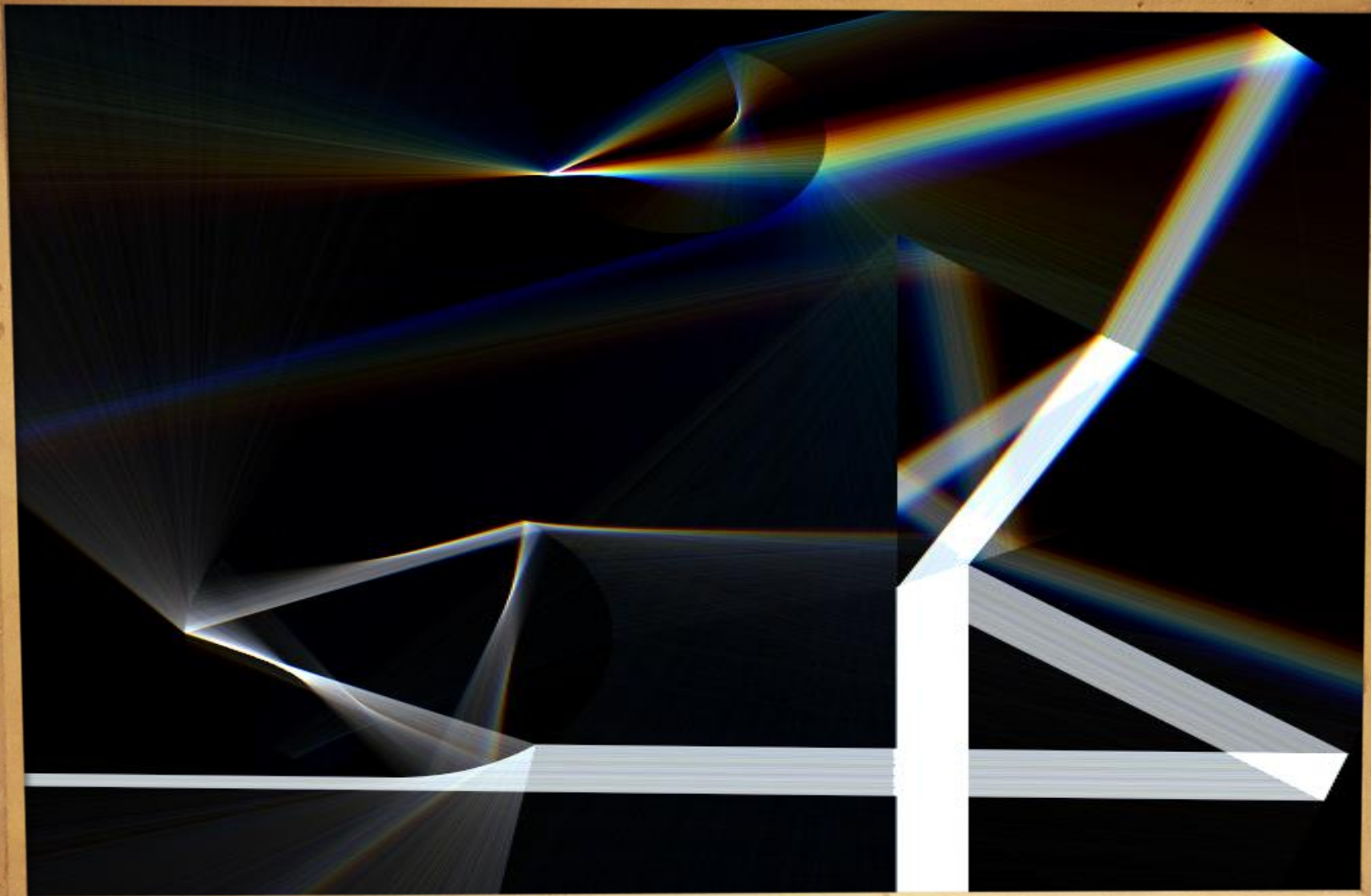

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nt;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    if (a = nt - nc, b = nt)
    {
        Tr = 1 - (R0 + (1 - R0) * nnt);
        R = (D * nnt - N * nnt);
    }
    if (E * diffuse;
        = true;
    )
    {
        refl + refr)) && (depth
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbabil
    estimation - doing it pro
    df;
    radiance = SampleLight( &r
    e.x + radiance.y + radianc
    w = true;
    at brdfPdf = EvaluateDiffu
    at3 factor = diffuse * INV
    at weight = Mis2( directPd
    at cosThetaOut = dot( N, L
    E * ((weight * cosThetaOut
    random walk - done properly
    ve)
    ;
    at3 brdf = SampleDiffuse
    survive;
    pdf;
    n = E * brdf * (dot( N,
    sion = true;
}
```



```

ics
& (depth < MAXDEPTH)
{
    nt = inside ? 1.0f : 0.0f;
    nt = nt / nc; ddn = nt *
    cos2t = 1.0f - nnt * nnt;
    D, N );
    0);
    at a = nt - nc, b = nt;
    at Tr = 1 - (R0 + (1 -
    Tr) R = (D * nnt - N
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbabil
    estimation - doing it pro
    df;
    radiance = SampleLight( &r
    e.x + radiance.y + radianc
    w = true;
    at brdfPdf = EvaluateDiffu
    at3 factor = diffuse * INV
    at weight = Mis2( directPd
    at cosThetaOut = dot( N, L
    E * ((weight * cosThetaOut
    random walk - done properly
    ve)
    ;
    at3 brdf = SampleDiffuse
    survive;
    pdf;
    n = E * brdf * (dot( N,
    sion = true;

```



```
ics
& (depth < MAXDEPTH)
c = inside ? 1 : 0;
nt = nt / nc; do {
  os2t = 1.0f - nm;
  D, N );
0);
at a = nt - nc; do {
  at Tr = 1 - (R0 + R1 + R2);
  (Tr) R = (D * nnt -
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
D, N );
refl * 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 <
w = 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 Small's
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

```
O=console.log;O("P2 80 80 99");for(P=6400;C=0,L=4,P--
;O(~~(C>3?99:C*33)))for(;S=3,o=0,L--;C+=o?0:1/1)while(S--
)a=P/80*.1,b=d=P%80*.1,c=a+~S,d-=S>1?1:2<<S,a-=5*L&5,b-
=3*L&6,l=a*a+b*b,i=a*c+b*d,i+=(i*i-(c*c+d*d)*l+1)**.5,o|=1>i&i>0
```

229 bytes



INFOGR – Computer Graphics

P2 - 2019



```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nt;
        pos2t = 1.0f - nnt * ddn;
        D, N );
    }
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + (1 - R0) * pos2t);
    (Tr) R = (D * nnt - N * (1 - Tr));
    E * diffuse;
    = true;
    (refl + refr)) && (d < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse, nt );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &radiance, &L, Align(
    e.x + radiance.y + radiance.z ); && (d < MAXDEPTH)
    w = 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
    ive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



$$M_{final} = T_{toTorso} \cdot R \cdot T_{down}$$

$R =$

x	y	z	
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Operations:

1. Translate head down
2. Rotate towards snowspeeder
3. Translate towards torso

$$\hat{z} = \text{normalize}(P_{speeder} - P_{trooper})$$

$$\hat{x} = \text{normalize}(\hat{z} \times (0,1,0))$$

$$\hat{y} = \hat{x} \times \hat{z}$$



Today's Agenda:

- Recap: Diffuse Materials
- The Phong Shading Model
- Environment Mapping
- Normal Mapping
- Rendering Short Fur



Basics of Shading

```

        nc = inside ? 1.0f : 0.0f;
        nt = nt / nc; ddn = ddn * nt;
        nnt = nt * nnt; dds2t = 1.0f - nnt * nnt;
        D, N );
    )
}

float R = (D * nnt - N * (ddn * dds2t));

E * diffuse;
= true;

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

MAXDEPTH)

survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely follow
df;
radiance = SampleLight( &rand, I, &L, &ll
.r.x + radiance.y + radiance.z) > 0) && (d
v = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psu
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf)
random walk - done properly, closely follow
ive)

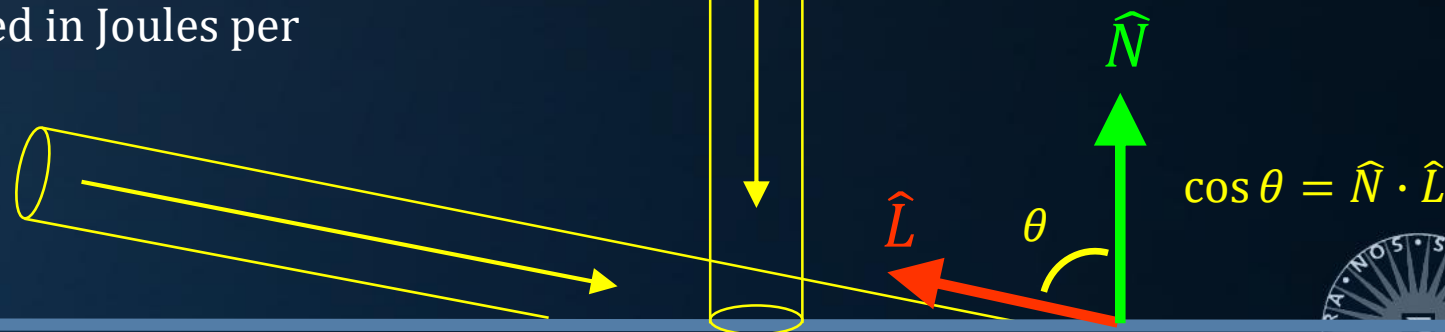
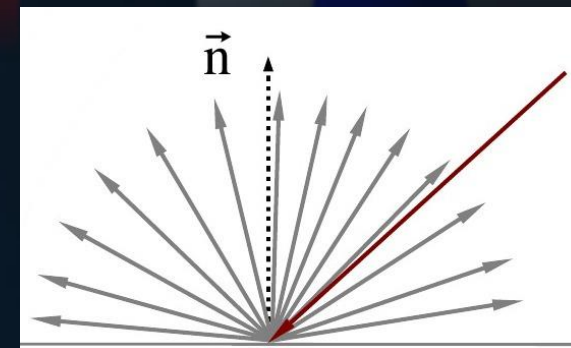
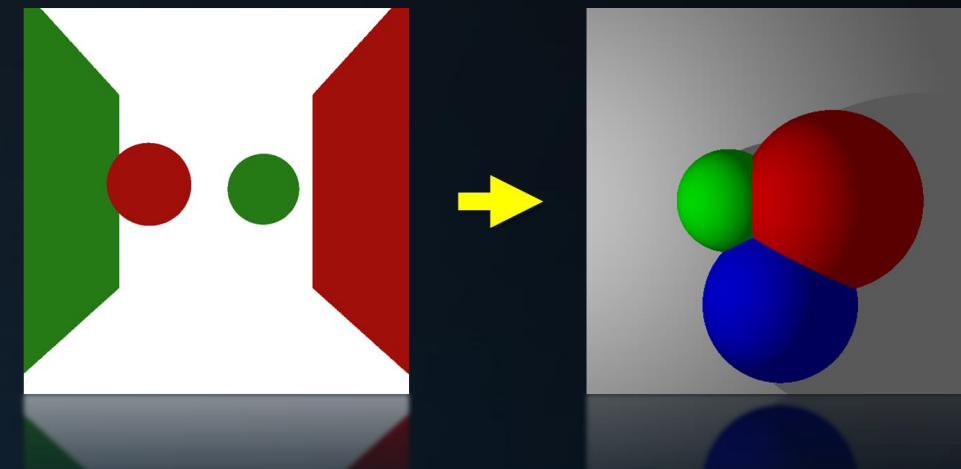
;
at3 brdf = SampleDiffuse( diffuse, N, r1,
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

Two aspects to this:

1. Diffuse materials scatter light uniformly in all directions. (*well... details in ADVGR*)
2. Arriving light however depends on angle.

Arriving: *irradiance*, expressed in Joules per second per m^2 .



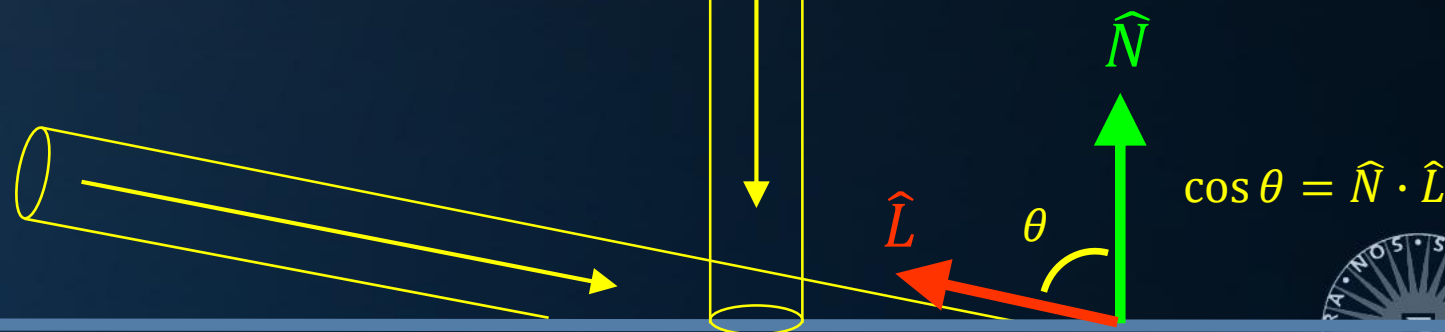
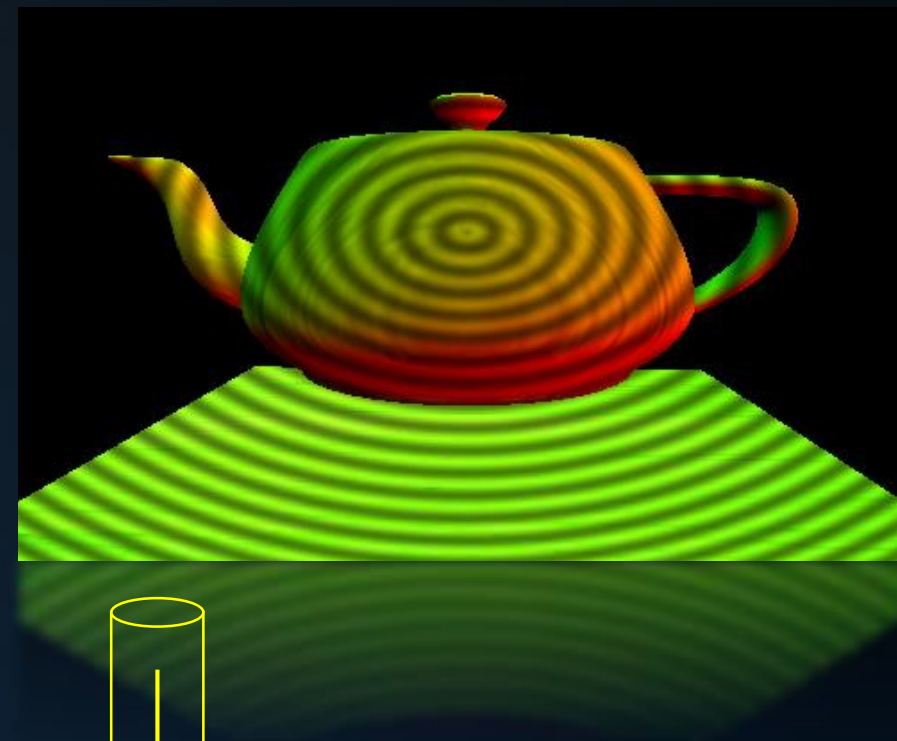
Diffuse

Basics of Shading

So, in the fragment shader:

- Calculate L : $L = \text{lightPos} - \text{intersectionPoint}$
- Use N : already passed via vertex shader
- Apply attenuation, scale by material color
- Multiply by light color
- Emit fragment color.

But wait...



Diffuse

Basics of Shading

So, in the fragment shader:

- Calculate L : $L = \text{lightPos} - \text{intersectionPoint}$
- Use N : already passed via vertex shader
- Apply attenuation, scale by material color
- Multiply by light color
- Emit fragment color.

But wait...

We have significant problems:

1. How do we specify a light position
2. In *which space* do we operate?

Ehm...

That one?



```
#version 330

// shader input
in vec2 vUV;
in vec3 vNormal;
in vec3 vPosition;

// shader output
out vec4 normal;
out vec2 uv;
uniform mat4 transform;

// vertex shader
void main()
{
    // transform vertex using supplied matrix
    gl_Position = transform * vec4( vPosition, 1.0 );

    // forward normal and uv coordinate
    normal = transform * vec4( vNormal, 0.0f );
    uv = vUV;
}
```

model space

model space
to
camera space?

normal now
also in camera
space?



Diffuse

Spaces

Default matrix in MyApplication.cs:

```
Matrix4 Tpot = Matrix4.CreateScale( 0.5f ) * Matrix4.CreateFromAxisAngle(0, 1, 0, a );
Matrix4 Tcamera = Matrix4.CreateTranslation(0, -14.5f, 0 )
                  * Matrix4.CreateFromAxisAngle( 1, 0, 0, angle90degrees );
Matrix4 Tview = Matrix4.CreatePerspectiveFieldOfView( 1.2f, 1.3f, .1f, 1000 );
```

(ignoring the floor for clarity)

This produces:

- a teapot (scaled by 0.5) that spins around it's pivot
- a camera located at (0, -14.5, 0) *or* the objects spins at (0, 14.5, 0) and the camera is at (0, 0, 0).

The last line adds perspective.

➔ We need a *'base system'* in which we can define a light position: *world space*.



Diffuse

Spaces

Getting *model space* coordinates to *world space*:

```
Matrix4 Tpot = Matrix4.CreateScale( 0.5f ) * Matrix4.CreateFromAxisAngle(0, 1, 0, a );
Matrix4 toWorld = Tpot;
Matrix4 Tcamera = Matrix4.CreateTranslation(0, -14.5f, 0 )
                  * Matrix4.CreateFromAxisAngle( 1, 0, 0, angle90degrees );
Matrix4 Tview = Matrix4.CreatePerspectiveFieldOfView( 1.2f, 1.3f, .1f, 1000 );
```

We need some additional changes now:

```
public void Render(
    Shader shader,
    Matrix4 transform, // final transform, includes perspective
    Matrix4 toWorld,  // matrix that takes us to world space
    Texture texture)
```

...



Diffuse

Changes

The vertex shader now takes two matrices:

```
// transforms
uniform mat4 transform; // full transform: model space to screen space
uniform mat4 toWorld;   // model space to world space
```

...and uses them:

```
gl_Position = transform * vec4( vPosition, 1.0 );
worldPos = toWorld * vec4( vPosition, 1.0f );
normal = toWorld * vec4( vNormal, 0.0f );
```



Diffuse

Changes

The shader class needs to know about the two matrices:

```
public int uniform_mview;
public int uniform_2wrld;
```

```
...
```

```
uniform_mview = GL.GetUniformLocation( programID, "transform" );
uniform_2wrld = GL.GetUniformLocation( programID, "toWorld" );
```

And the mesh class needs to pass both to the shader:

```
// pass transforms to vertex shader
GL.UniformMatrix4( shader.uniform_mview, false, ref transform );
GL.UniformMatrix4( shader.uniform_2wrld, false, ref toWorld );
```



Diffuse

Changes

The new fragment shader, complete:

```
#version 330

in vec2 uv;           // interpolated texture coordinate
in vec4 normal;        // interpolated normal, world space
in vec4 worldPos;      // world space position of fragment
uniform sampler2D pixels; // texture sampler

out vec4 outputColor;  // shader output

uniform vec3 lightPos; // light position in world space

void main()            // fragment shader
{
    vec3 L = lightPos - worldPos.xyz;
    float dist = L.length();
    L = normalize( L );
    vec3 lightColor = vec3( 10, 10, 8 );
    vec3 materialColor = texture( pixels, uv ).xyz;
    float attenuation = 1.0f / (dist * dist);
    outputColor = vec4( materialColor * max( 0.0f, dot( L, normal.xyz ) ) *
        attenuation * lightColor, 1 );
}
```

In MyApplication.cs, Init():

```
// set the light
int lightID = GL.GetUniformLocation(
    shader.programID,
    "lightPos"
);
GL.UseProgram( shader.programID );
GL.Uniform3(
    lightID,
    0.0f, 10.0f, 0.0f
);
```



Diffuse

```

rics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.0f - 0.5f)
    {
        nt = nt / nc, ddn = ddn * ddn;
        r = 1.0f - nnt * ddn;
        D, N );
    }
    {
        a = nt - nc, b = nt + nc;
        Tr = 1 - (R0 + (1 - R0) * r);
        R = (D * nnt - N * (ddn * ddn));
    }
    E * diffuse;
    = true;
}
{
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    {
        survive = SurvivalProbability( diffuse );
        estimation - doing it properly, closely following Small's
        if;
        radiance = SampleLight( &rand, I, &L, &lightDir );
        e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
        {
            w = 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 Small's
        vive)
    };
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;
}

```



Today's Agenda:

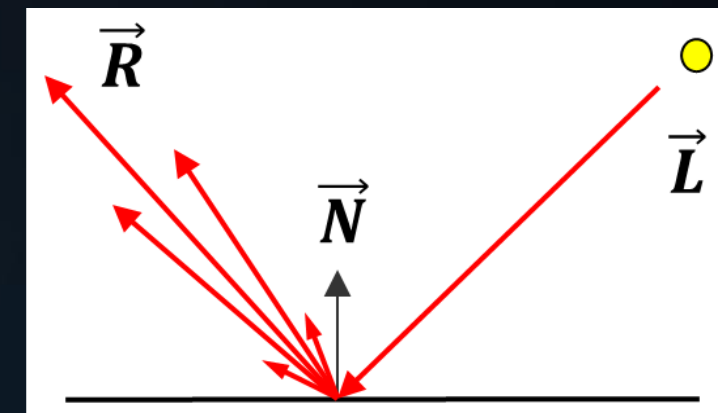
- Recap: Diffuse Materials
- The Phong Shading Model
- Environment Mapping
- Normal Mapping
- Rendering Short Fur



Phong

Glossy Materials

A glossy material reflects, but the reflection is somewhat fuzzy:



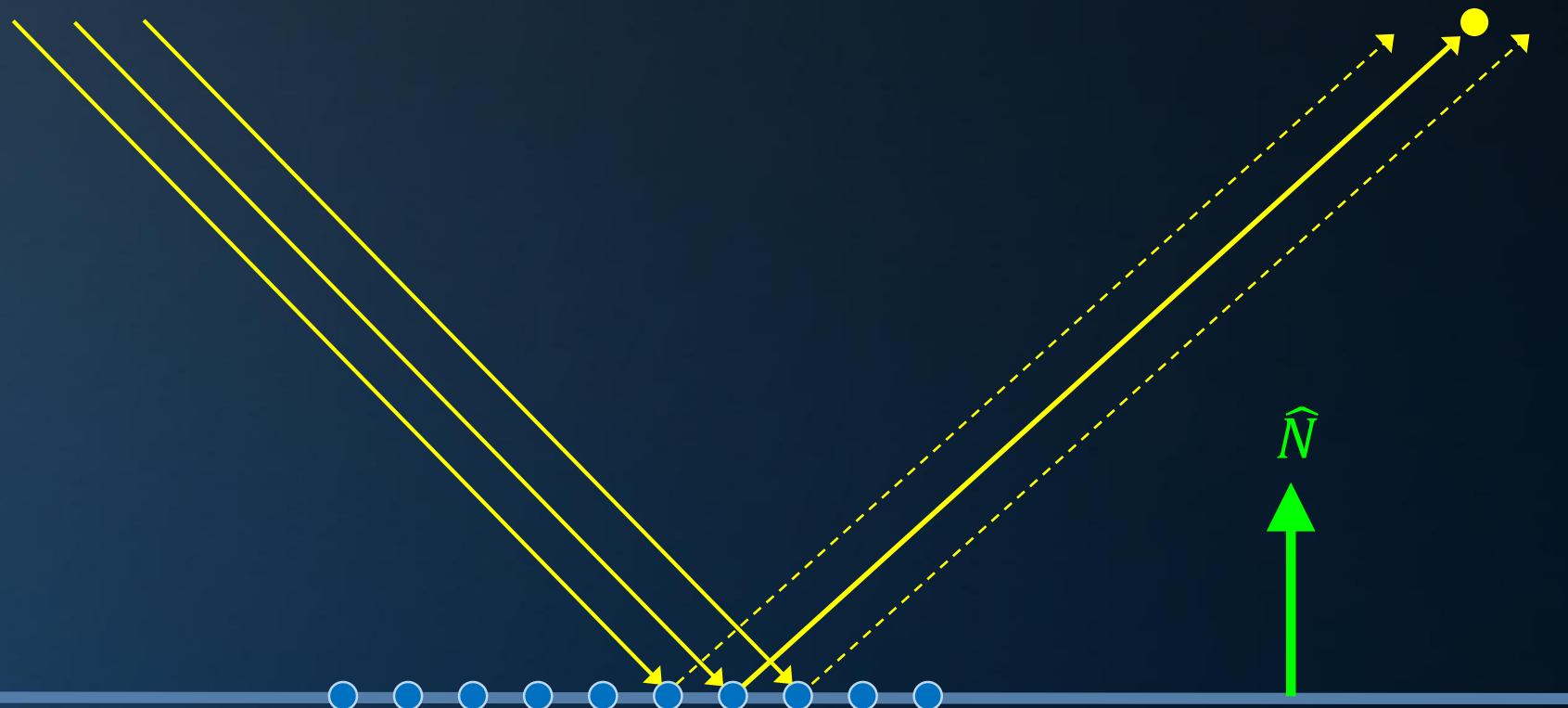
Using a ray tracer we achieve this effect by sending multiple rays in directions close to the reflection vector $\hat{R}$.



Phong

Glossy Materials

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 1.01)
    {
        nt = nt / nc, ddn = ddn * nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * cos2t);
        Tr) R = (D * nnt - N * (ddn < 0 ? 1 : -1));
    }
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following Small's
    df;
    radiance = SampleLight( &rand, I, &L, &light, &N, &N );
    e.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
    {
        w = 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 Small's
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
```



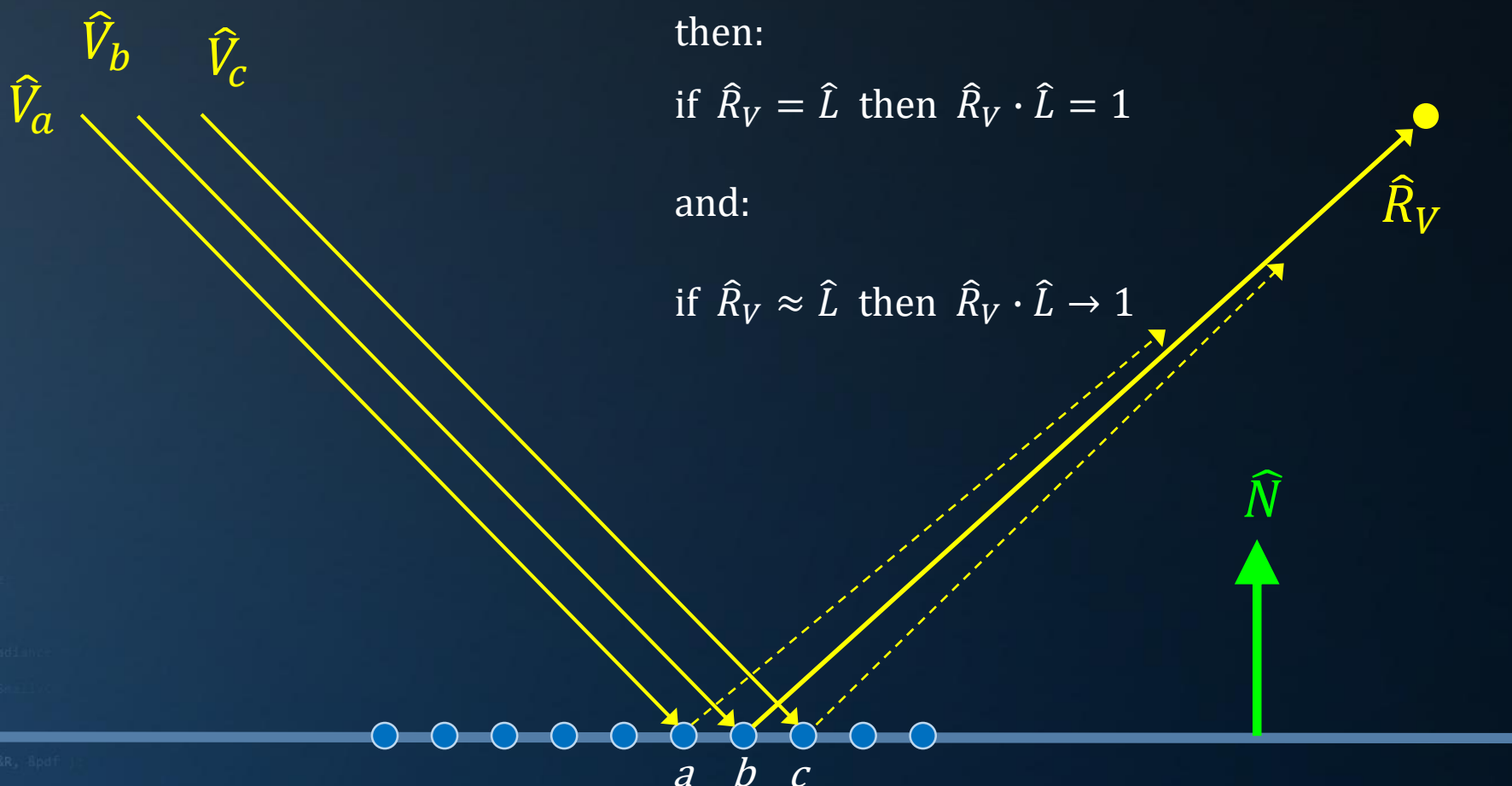
Phong

Glossy Materials

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        r = 1.0f - nnt * ddn;
        D, N );
    }
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * r);
    (Tr) R = (D * nnt - N * (ddn * r));
    E * diffuse;
    = true;
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * 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 <
    w = 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;

```



Let:

$\hat{L}$ be a vector from the fragment position b to the light;
 $\hat{R}_V$ be the vector $\hat{V}_b$ reflected in the plane with normal $\hat{N}$

then:

if $\hat{R}_V = \hat{L}$ then $\hat{R}_V \cdot \hat{L} = 1$

and:

if $\hat{R}_V \approx \hat{L}$ then $\hat{R}_V \cdot \hat{L} \rightarrow 1$

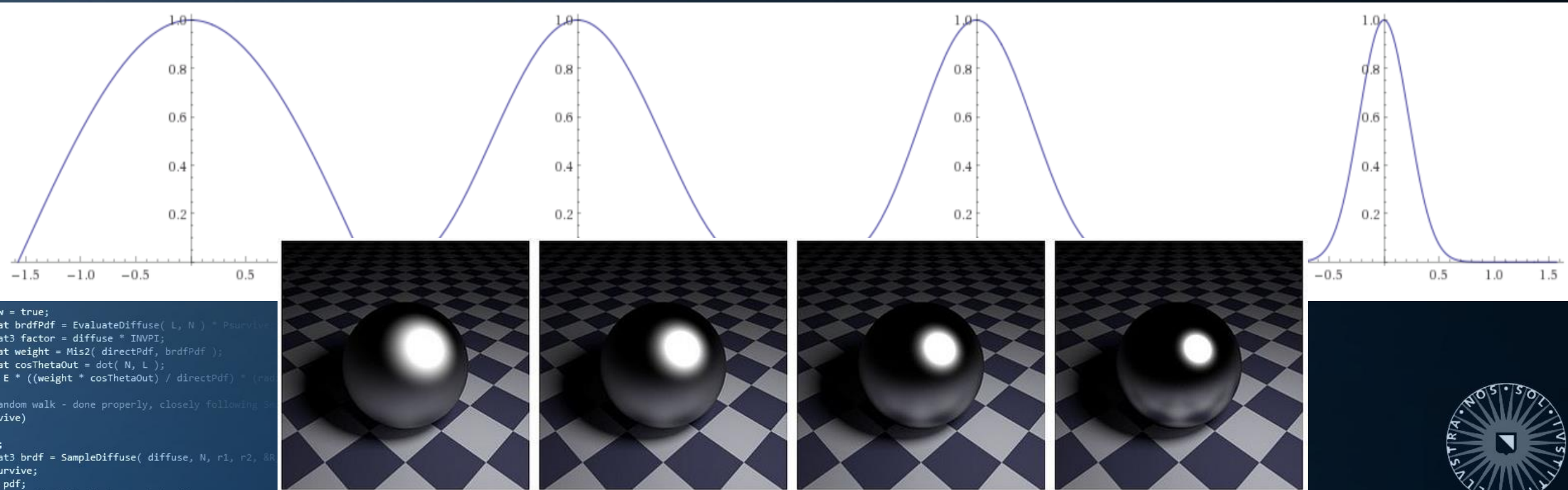


Phong

Glossy Materials

“Locations near b receive almost as much light as b .”

But how much? $(\hat{L} \cdot \hat{R}_V)^\alpha$



Phong

The Full Phong

$$L = c_{ambient} + c_{diff}(\hat{N} \cdot \hat{L})l_{diff} + c_{spec}(\hat{L} \cdot \hat{R}_V)^\alpha l_{spec}$$

The Phong material model is a combination of:

- ‘Specular’ illumination: $(\hat{L} \cdot \hat{R})^\alpha$, times the ‘specular color’ of the light, times the ‘specular color’ of the material;
- Diffuse illumination: $(\hat{N} \cdot \hat{L})$, times the ‘diffuse color’ of the light, times the ‘diffuse color’ of the material;
- An ‘ambient color’.



Phong

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0.2)
    {
        nt = nt / nc; ddn = dot(N, R);
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    at a = nt - nc; b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) * cos2t);
    R = (D * nnt - N * (ddn * cos2t));
    E * diffuse;
    = true;
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diff
    estimation - doing it properly,
    df;
    radiance = SampleLight( &rand, I,
    e.x + radiance.y + radiance.z) > 0
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N,
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdf
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf);
    random walk - done properly, closely following a
    ve)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```



Diffuse Lighting used in
Half-Life 2

+

Specular Lighting

=

High Quality Lighting in
Half-Life 2: Episode One



Today's Agenda:

- Recap: Diffuse Materials
- The Phong Shading Model
- Environment Mapping
- Normal Mapping
- Rendering Short Fur



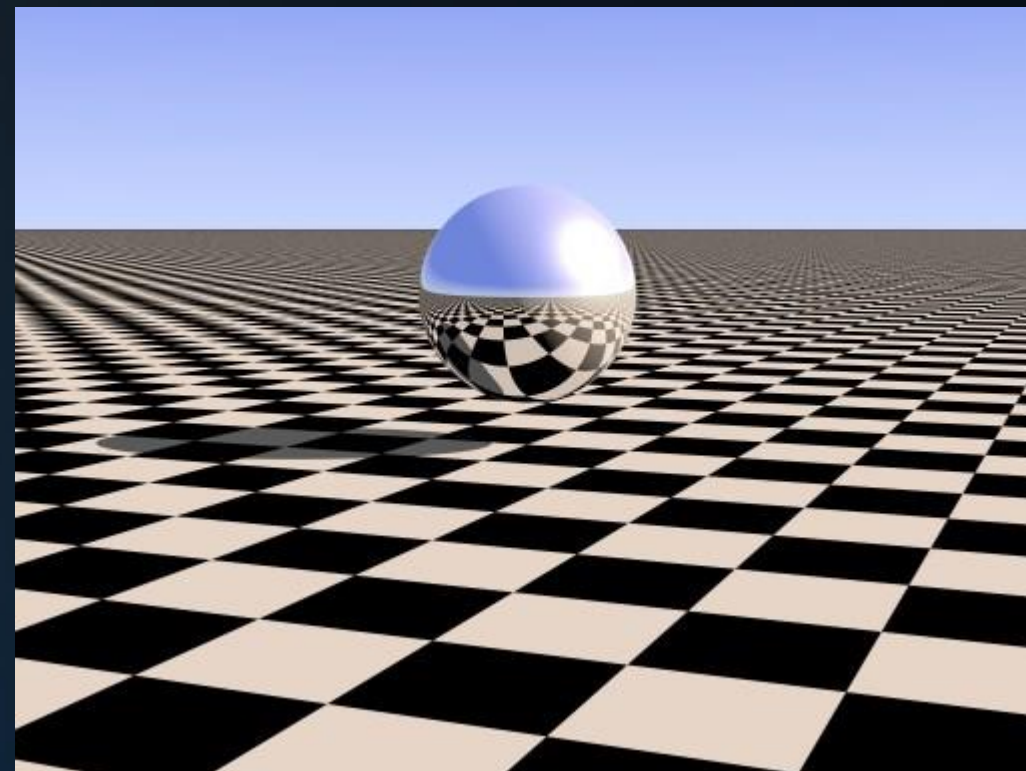
Mirrors

Reflections

Reflections in a ray tracer are easy:

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

But what about rasterizers?





Mirrors

```

ics
& (depth < MAXDEPTH)
{
    nt = inside ? 1.0 : 0.0;
    nt = nt / nc; ddn = dot(N, D);
    cos2t = 1.0f - nnt * nnt;
    D, N );
    0);
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) *
    at Tr) R = (D * nnt - N * (ddn > 0) ?
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    df;
    radiance = SampleLight( &rand, I, &L, &lightPos,
    e.x + radiance.y + radiance.z) > 0) && (depth <
    w = 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 Small
    ve)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



Mirrors

Planar Reflections

We can fake reflections in a rasterizer by duplicating the scene:

The mirror is not really there; it's just a hole through which we see a copy of the scene.



Mirrors

```

    // Ray-sphere intersection
    float t;
    if (depth < MAXDEPTH)
    {
        // Inside?
        bool inside = inside ? 1 : -1;
        float nt = nt / nc, ddn = ddn * inside;
        float r = 1.0f - nnt * ddn;
        float D, N );
    }

    // Ray-sphere intersection
    float a = nt - nc, b = nt * nc;
    float Tr = 1 - (R0 + (1 - R0) * r);
    float R = (D * nnt - N * (ddn * r));

    // Ray-sphere intersection
    E * diffuse;
    = true;

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

    MAXDEPTH)

    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, close
    if;
    radiance = SampleLight( &rand, I, &L,
    e.x + radiance.y + radiance.z) > 0) &&
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) *
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPd

    random walk - done properly, closely following the path
    (survive)

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

```



Mirrors

Environment Mapping

Reflections on complex surfaces are faked using an environment map.

This is done exactly as in a ray tracer:

- at the fragment position, we have $\hat{V}$ and $\hat{N}$;
- based on these we calculate the reflected vector $\hat{R}$;
- we use $\hat{R}$ to look up a value in the skydome texture.

Limitations:

- we will not reflect anything but the skydome;
- the reflection is static.



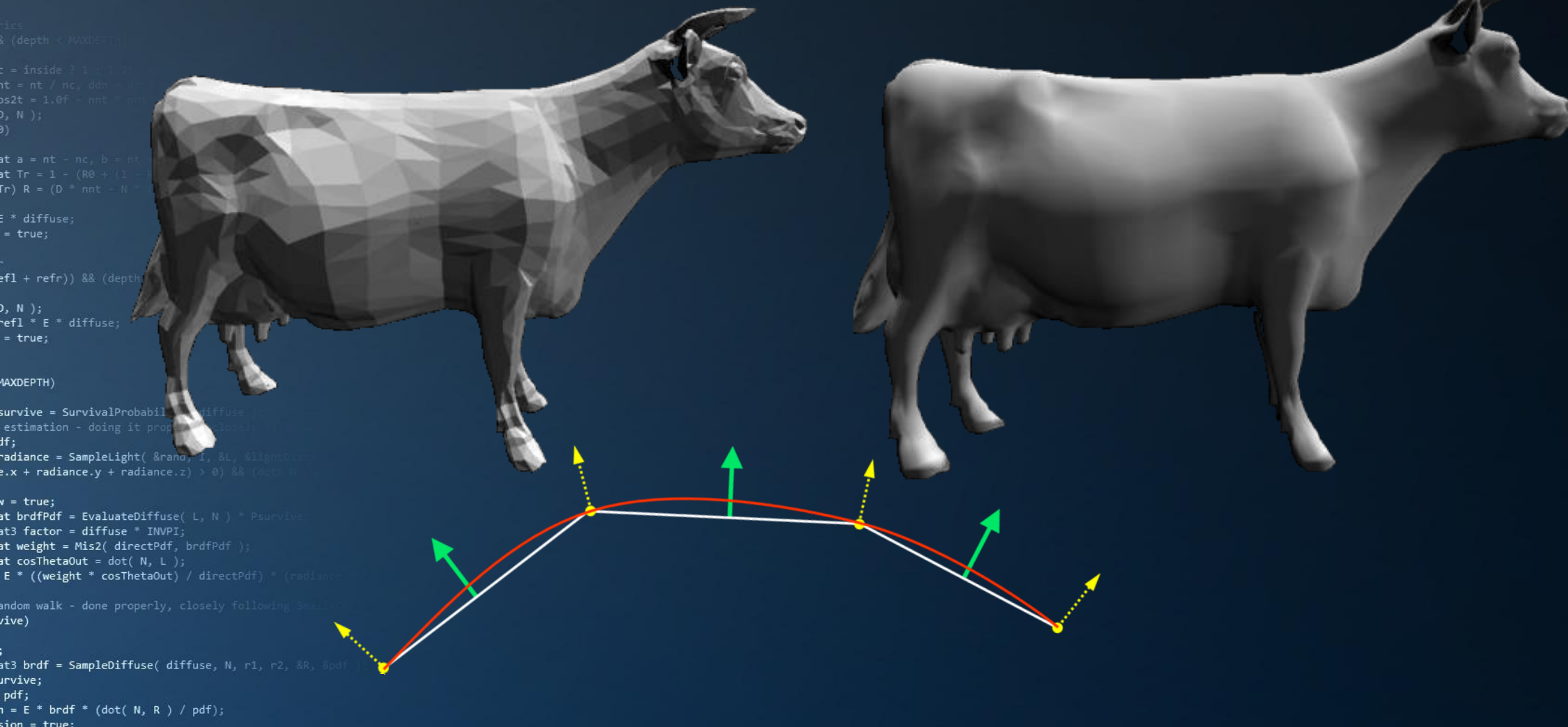
Today's Agenda:

- Recap: Diffuse Materials
- The Phong Shading Model
- Environment Mapping
- Normal Mapping
- Rendering Short Fur



Bumps

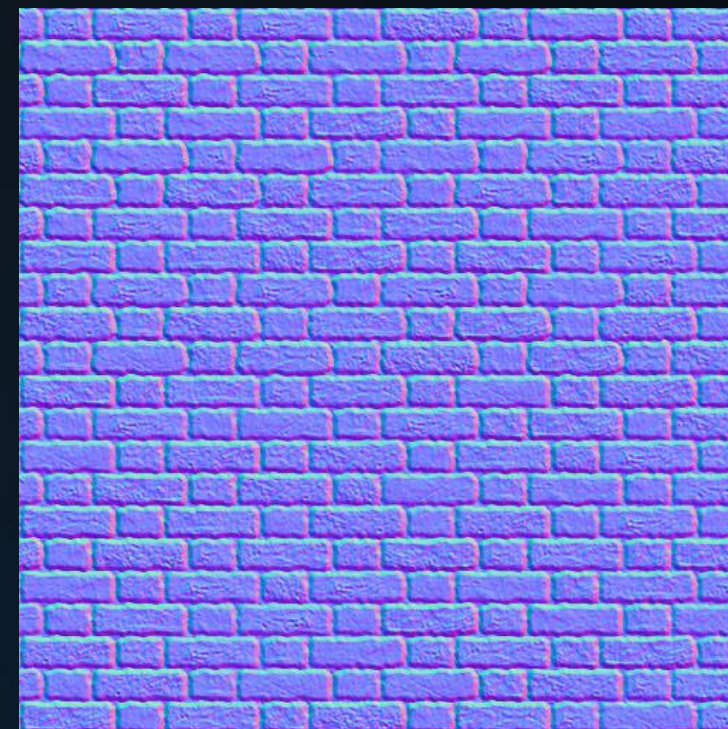
Recap: Normal Interpolation



Normal Maps

- we use textures to lookup a color at a particular position on a mesh;
- we use normal maps to lookup a normal at a particular position.

Normal maps generally store normals in *tangent space*.



Bumps

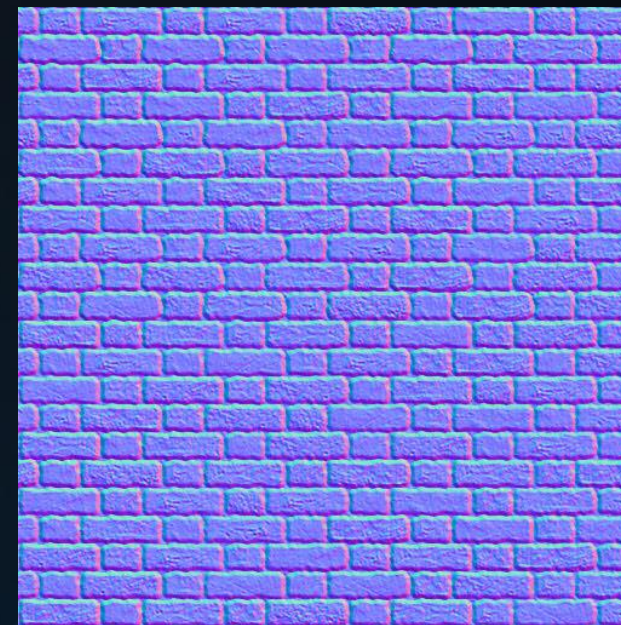
Normal Map Data

A normal map stores 2 or 3 components per texel.

For 3 components: $x, y, z \in [0..255]$.

To get this to the correct range:

- Cast to float
- Divide by 128 $\rightarrow x, y, z \in [0..2]$
- Subtract 1 $\rightarrow x, y, z \in [-1..1]$



Bumps

Normal Map Data

A normal map stores 2 or 3 components per texel.

For 2 components: $x, y \in [0..255]$.

To reconstruct the normal, we first apply the same conversion as we did for three components. Then:

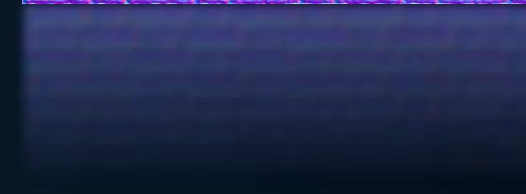
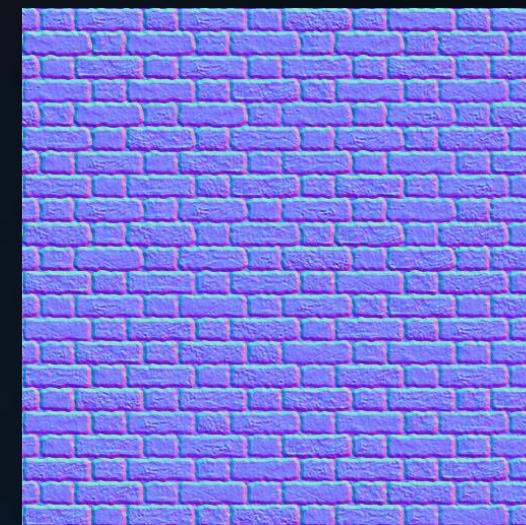
$$\sqrt{x^2 + y^2 + z^2} = 1$$

$$x^2 + y^2 + z^2 = 1$$

$$z^2 = 1 - (x^2 + y^2)$$

$$z = \pm\sqrt{1 - (x^2 + y^2)}$$

$$z = \sqrt{1 - (x^2 + y^2)}$$



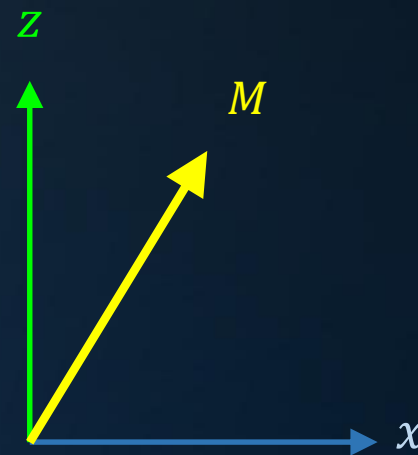
Bumps

Tangent Space

Things are easy when $x = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ and $z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$:

$$N = M_x x + M_y y + M_z z$$

$$= M_x \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + M_y \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + M_z \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix}$$



Bumps

Tangent Space

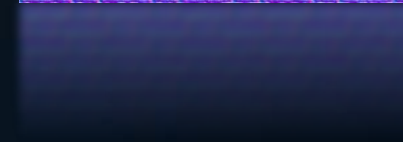
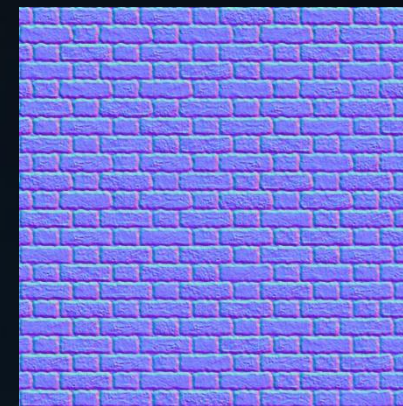
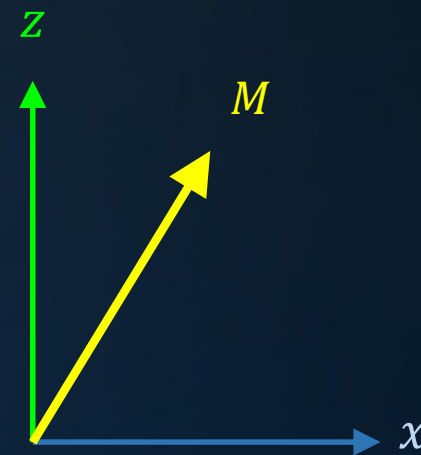
Things are still easy for arbitrary x, y and z :

$$N = M_x x + M_y y + M_z z$$

Obtaining this x, y and z:

- z = (interpolated) surface normal
- x is perpendicular to z
- y is perpendicular to z and x

See Crytek for details*.



<http://docs.cryengine.com/display/SDKDOC4/Tangent+Space+Normal+Mapping>

<https://fenix.tecnico.ulisboa.pt/downloadFile/845043405449073/Tangent%20Space%20Calculation.pdf>



Bumps

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nt;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * cos2t);
        (Tr) R = (D * nnt - N * (ddn *
    }
    {
        E * diffuse;
        = true;
    }
    {
        refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    {
        MAXDEPTH)
    {
        survive = SurvivalProbability( diffuse,
        estimation - doing it properly, closely following
        df;
        radiance = SampleLight( &rand, I, &L, &light,
        e.x + radiance.y + radiance.z) > 0) && (depth <
    }
    {
        w = 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) * (re
    }
    {
        random walk - done properly, closely following S
        (survive)
    }
    {
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }
}

```

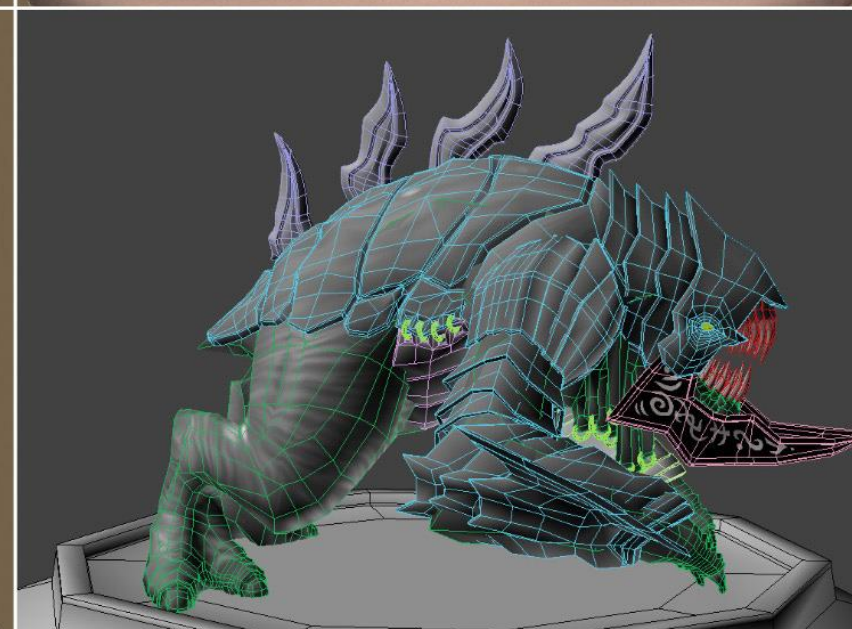


Bumps

```

ics
& (depth < MAXDEPTH)
{
    nt = inside ? 1.0 : 0.0;
    nt = nt / nc; ddn = dot(N, D);
    cos2t = 1.0f - nnt * nnt;
    D, N );
    )
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) *
    (Tr) R = (D * nnt - N * (ddn
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    -refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, closely
    df;
    radiance = SampleLight( &rand, I, &L, &lightP,
    e.x + radiance.y + radiance.z) > 0) && (dot(N,
    w = 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) * (ra
    random walk - done properly, closely following S
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



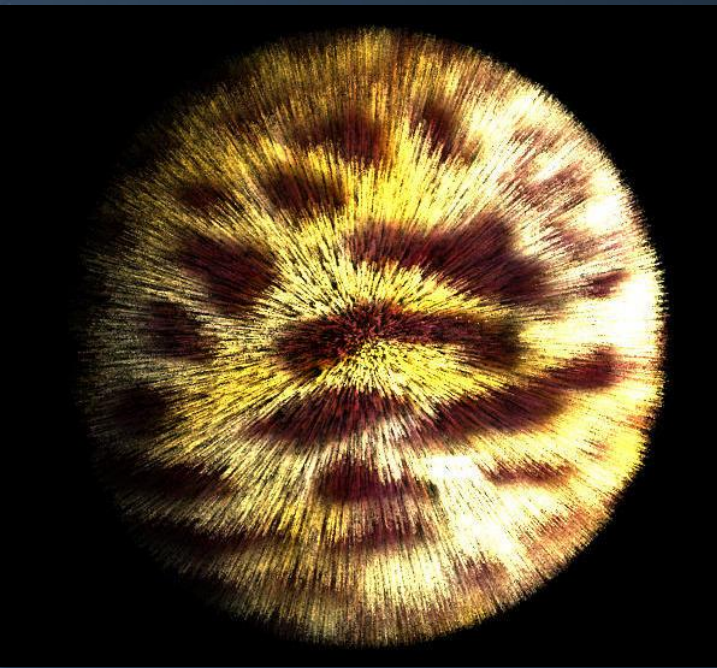
Today's Agenda:

- Recap: Diffuse Materials
- The Phong Shading Model
- Environment Mapping
- Normal Mapping
- Rendering Short Fur



Fur

Fur, Grass etc.



```
w = true;  
at brdfPdf = EvaluateDiffuse( L, N ) * 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 *Survive*
(*Survive*)

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



Fur

Fur, Grass etc.



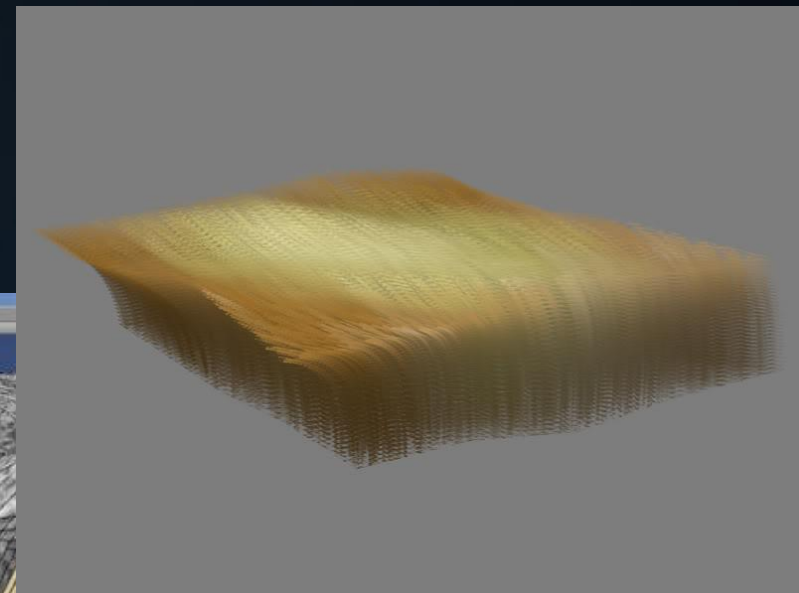
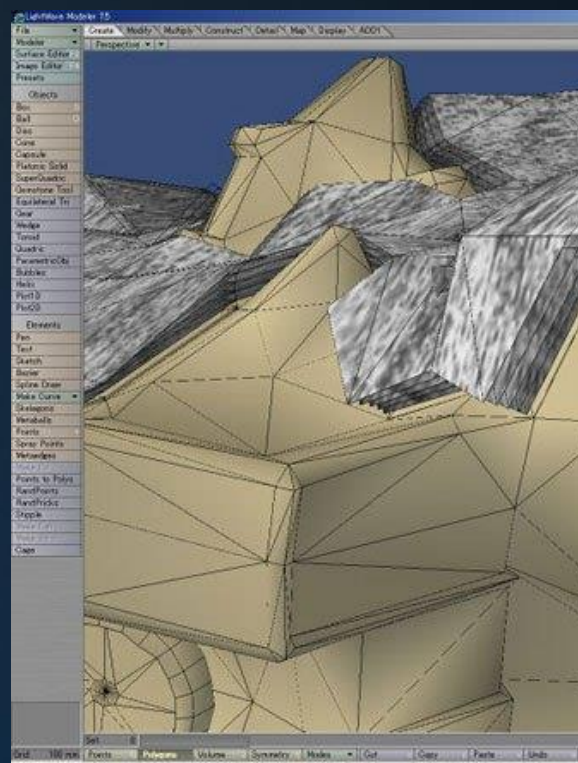
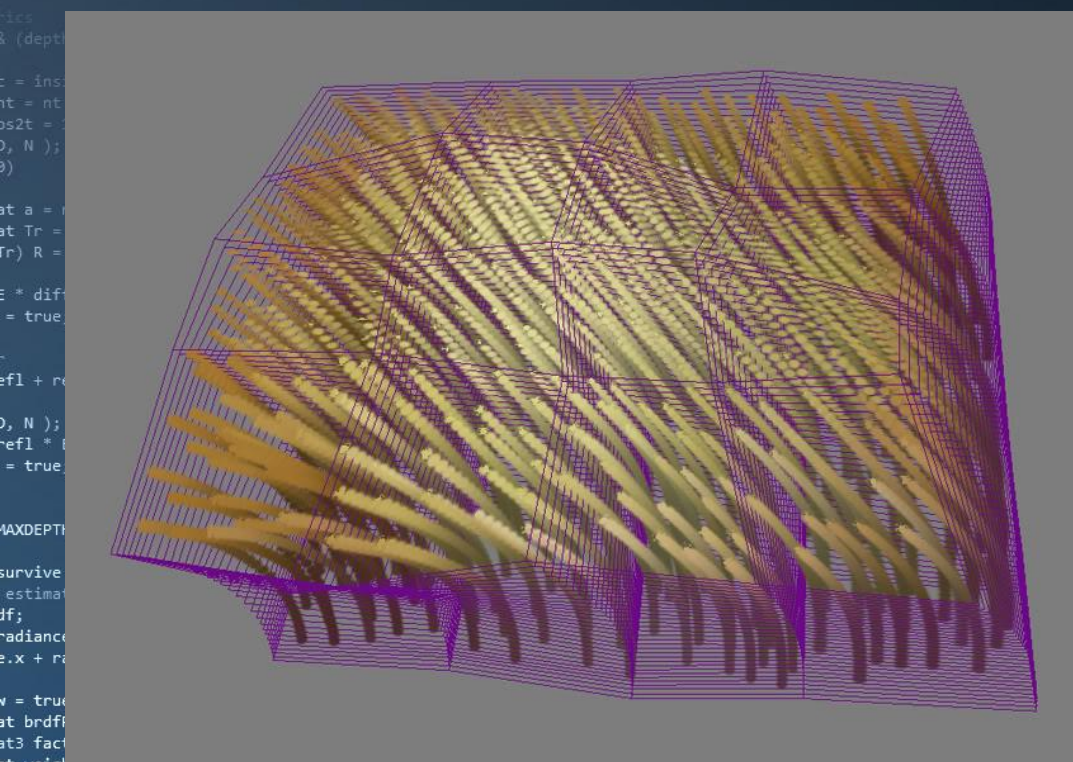
```
if (v == true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurface;
    at3 factor = diffuse * INVPI;
    at weight = Mix2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance);
    random walk - done properly, closely following Bee-Bee's path
    (vive)
```

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



Fur

Fur, Grass etc.



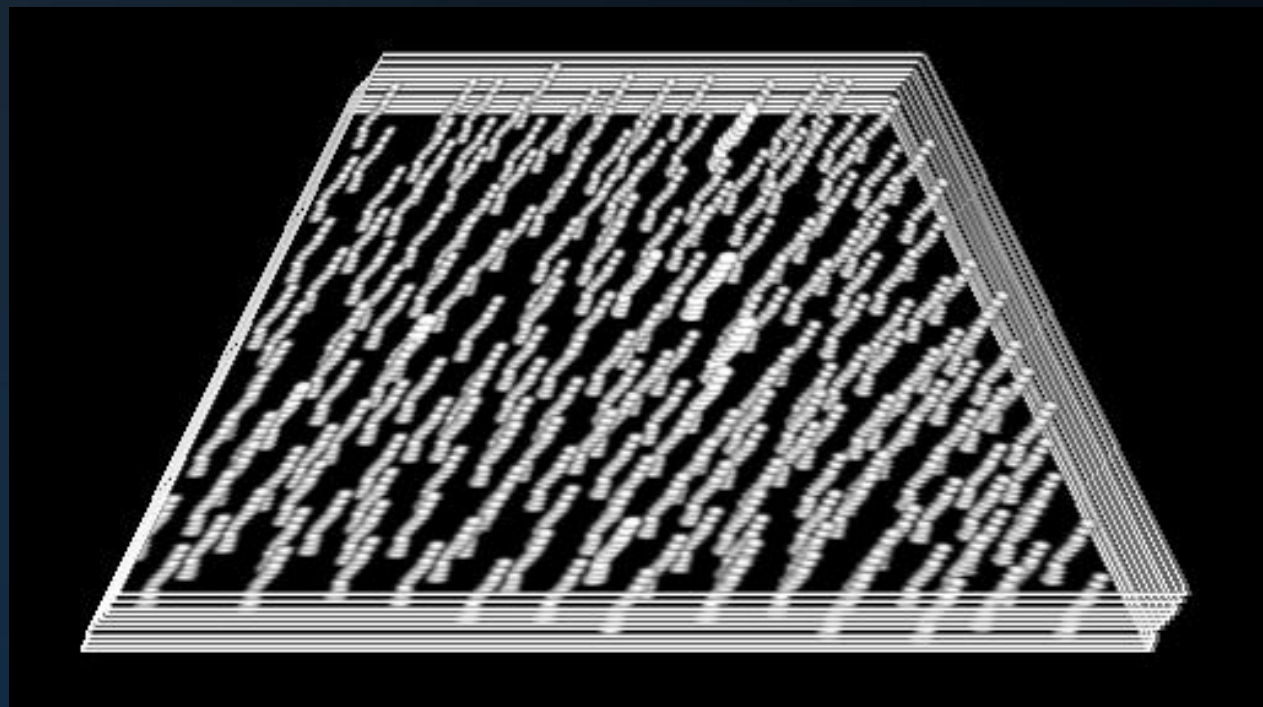
Fur

Fur, Grass etc.

```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) * ddn);
    R = (D * nnt - N * (ddn * cos2t));
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, close
    df;
    radiance = SampleLight( &rand, I, &L,
    e.x + radiance.y + radiance.z) > 0) &&
    w = 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 Small's
    vive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



Fur, Grass etc.

```
#version 330
```

• • •

```
// shell offset
```

```
uniform float shellOffset;
```

```
// vertex shader
```

```
void main()
```

{

```
gl_Position = transform * vec4( vPosition + shellOffset * vNormal, 1.0 );
```

```
worldPos = toWorld * vec4( vPosition + shellOffset * vNormal, 1.0f );
```

```
normal = toWorld * vec4( vNormal, 0.0f );
```

$$uv = vUv;$$


Fur

Fur, Grass etc.

In game.cs:

```
for( int i = 0; i < 30; i++ )
{
    GL.UseProgram( shader.programID );
    GL.Uniform1( offsetID, (float)i * 0.02f );
    mesh.Render( shader, prevMat[19 - i / 4], prevWld[19 - i / 4], fur );
}
```

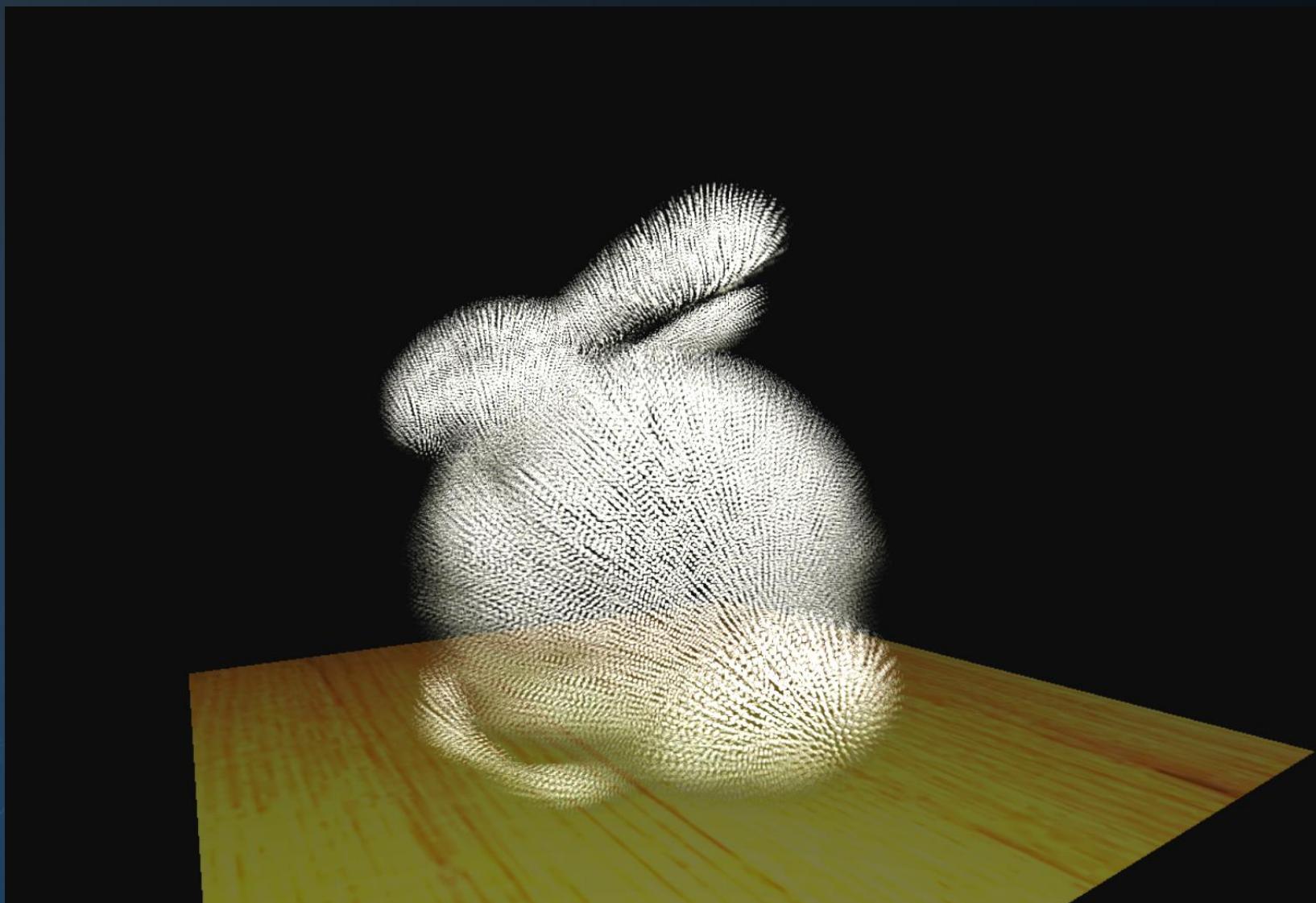


Fur

```

rics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0.25)
    {
        nt = nt / nc, ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    else
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * ddn);
        Tr) R = (D * nnt - N * (ddn * ddn));
    }
    E * diffuse;
    = true;
}
-
refl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;
}
MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
df;
radiance = SampleLight( &rand, I, &L, &light;
e.x + radiance.y + radiance.z > 0) && (depth <
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psum;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) *
random walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```



INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2019

END OF lecture 10: “Shaders”

Next lecture: “Visibility”

