Linear-Light Shading with Linearly Transformed Cosines

Eric Heitz, Stephen Hill

Linear-Light Shading with Linearly Transformed Cosines

Eric Heitz and Stephen Hill

1.1 Introduction

We recently introduced a new real-time area-light shading technique dedicated to lights with polygonal shapes [Heitz et al. 16]. In this chapter, we extend this area-lighting framework to support *linear* (line-shaped) lights in addition to polygons. Linear lights are cheaper to shade than polygons and they provide a good approximation for thin emitting cylinders (fluorescent tubes, lightsabers, etc.), as shown in Figure 1.1.

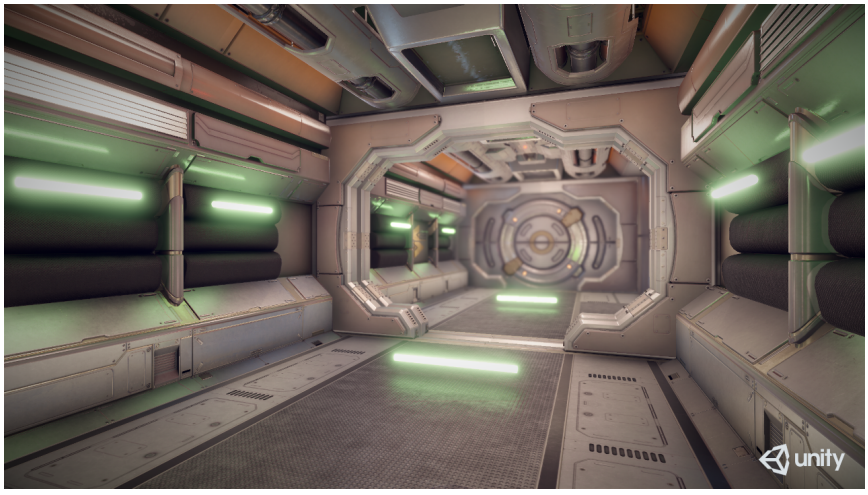

Figure 1.1. We use linear lights to approximate thin cylindrical light shapes.

Our area-lighting framework is based on a spherical distribution called *Linearly Transformed Cosines (LTCs)* introduced in our previous article [Heitz et al. 16]. While not required, we encourage anyone interested in the mathematical details to consult it (or the associated slides) before reading this chapter.

The Linear-Light Shading Model Linear lights share a limitation with point lights in that they cannot be found in the real world, since no real emitter is infinitely thin. However, in Section 1.2 we show that, in many scenarios, linear lights are a good approximation for cylindrical lights with a small but non-zero radius. We describe how to approximate these lights with linear lights that have similar power and shading, and discuss the validity of this approximation.

Note that in this chapter we use the classic definition of linear lights[1], where they are modeled as cylinders with a diffuse emission profile and an infinitely small radius [Nishita et al. 85, Bao and Peng 93]. Since these linear lights only model the lengths of the cylinders, we discuss the addition of their emitting end caps in Section 1.5.

Finally, in Section 1.6, we briefly discuss an alternative definition of a linear light that models a rectangle with an infinitely small width instead of a cylinder with an infinitely small radius. The shading of these rectangle-like linear lights remains essentially the same as the cylinder-like linear lights: it is just modulated by the orientation of the rectangle's normal.

The Material Model The key to real-time shading is the ability to integrate the product of the material (i.e., the BRDF) and the light. In the case of linear lights, we need to compute a line integral over the spherical distribution given by the BRDF. In the literature, the only spherical distributions with analytic line integrals are the diffuse distribution (we recall its derivation Section 1.3) and the Phong distribution for glossy materials [Nishita et al. 85, Bao and Peng 93]. However, the Phong distribution is an inaccurate approximation for recent physically based shading models. Furthermore, the complexity of the line integral over a Phong distribution grows linearly with the exponent of the distribution. Hence, integrating almost specular materials over a linear light can be prohibitively costly for real-time shading.

These limitations were the same for polygonal lights, and we overcame them thanks to LTCs [Heitz et al. 16]. LTC distributions yield good approximations for physically based materials based on the GGX microfacet distribution [Walter et al. 07] that are considered state of the art today in the video game industry [Hill et al. 15] and can be analytically integrated over arbitrary polygons in constant time due to their polygon-integral invariance property. In Section 1.3, we recall the definition of LTCs and show

[1] An alternative definition of linear lights can be found in the literature [Poulin and Amanatides 91, Picott 92]. With this definition, the linear light is modeled as an infinite series of point lights instead of as a cylinder with an infinitely small radius. Hence, this model lacks the Jacobian that accounts for the light emission profile and its inclination, which yields incorrect shading behavior. Because of this, it cannot be used as an approximation for actual geometric shapes such as a cylinder.

that they have a similar line-integral invariance property. Again, thanks to this property, we can integrate them analytically over linear lights in constant time. Figure 1.2 shows the result obtained by shading materials based on the GGX distribution with linear lights.

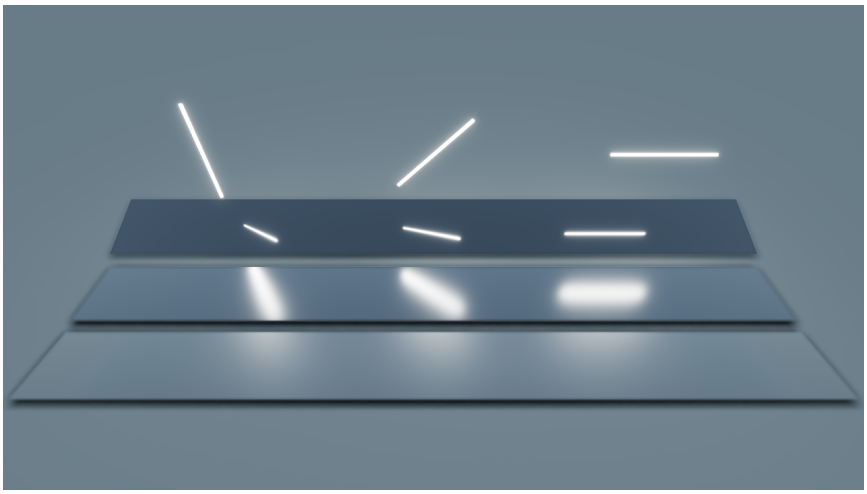

Figure 1.2. We shade linear lights with physically based materials that use the GGX microfacet BRDF.

Associated Demo This book chapter is distributed with a WebGL demo that implements all the code snippets from the chapter (Figure 1.3). Note that the code provided in the chapter and the demo prioritizes readability over performance and is not meant to be shipped as-is to production.

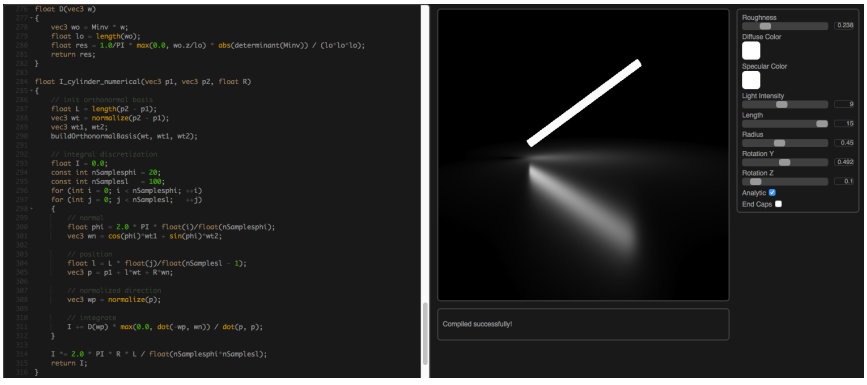

Figure 1.3. Our WebGL demo.

1.2 The Linear-Light Shading Model

In this section, we show that linear lights yield a good approximation to cylinder lights in many configurations. In order to validate the domain of validity of this approximation, we recall how the integrals of the BRDF over cylinders and lines are defined, and we implement numerical integration shaders to compute them. Thus, the goal of this section is to provide shader code for validating the approximation; the actual code one would use in practice is covered in the subsequent sections.

1.2.1 The Local Illumination Integral

Shading with area lights requires computing the illumination integral over the spherical domain Ω_L covered by the light:

$$I = \int_{\Omega_L} \rho(\boldsymbol{\omega}_v, \boldsymbol{\omega}_l) \cos\theta_l \, \mathrm{d}\boldsymbol{\omega}_l, \tag{1.1}$$

where $\boldsymbol{\omega}_v$ is the view direction, $\boldsymbol{\omega}_l$ is the light direction, and ρ the BRDF.

1.2.2 The Spherical Distribution

We use $D(\boldsymbol{\omega}_l) = \rho(\boldsymbol{\omega}_v, \boldsymbol{\omega}_l) \cos\theta_l$ to denote the cosine-weighted BRDF, and we show how to compute the integral of Equation (1.1) for cylindrical lights or lights composed of line segments.

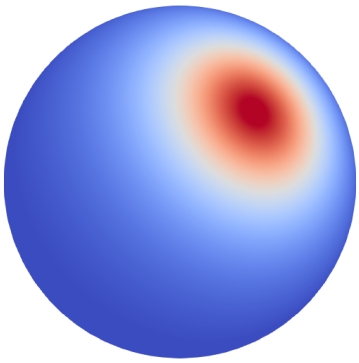

Figure 1.4. The spherical distribution D to be integrated.

```
float D(vec3 w)
{
    // .. brdf*cos
}
```

Listing 1.1. The distribution, D. This is typically a cosine-weighted BRDF, but the result of this section holds for any arbitrary spherical function D.

1.2.3 The Cylinder-Light Integral

Configuration A cylinder is defined by two end points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ and a radius R, as shown in Figure 1.5. We use $L = \|\boldsymbol{p}_2 - \boldsymbol{p}_1\|$ to denote the length of the cylinder, $\boldsymbol{\omega}_t = \frac{\boldsymbol{p}_2 - \boldsymbol{p}_1}{\|\boldsymbol{p}_2 - \boldsymbol{p}_1\|}$ the tangent direction of the cylinder, and $(\boldsymbol{\omega}_t^{\perp}, \boldsymbol{\omega}_t^{\top})$ two orthonormal directions such that $(\boldsymbol{\omega}_t, \boldsymbol{\omega}_t^{\perp}, \boldsymbol{\omega}_t^{\top})$ forms an orthonormal basis.

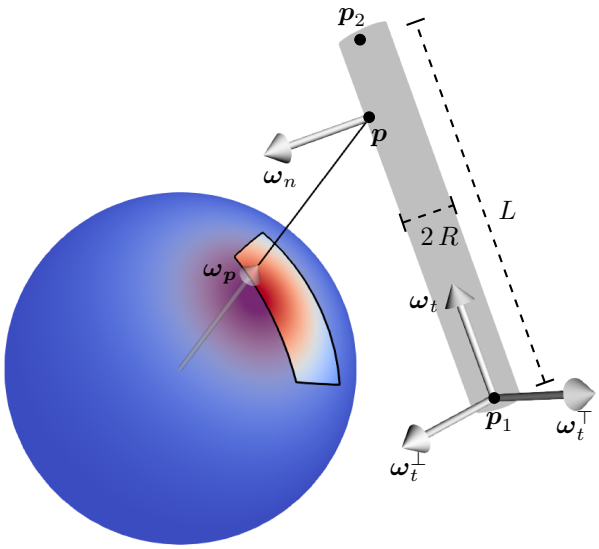

Figure 1.5. The cylinder-light integral.

Integral We rewrite Equation (1.1) in the space of the light instead of the sphere. With a cylinder parameterization (ϕ, l) for the cylinder surface the integral is

$$I_{\text{cyl}} = \int_0^L \int_0^{2\pi} D\left(\boldsymbol{\omega}_{\boldsymbol{p}}\right) \frac{|-\boldsymbol{\omega}_{\boldsymbol{p}} \cdot \boldsymbol{\omega}_n|}{\|\boldsymbol{p}\|^2} R \, d\phi \, dl. \tag{1.2}$$

With this parametrization, a point (ϕ, l) on the cylinder surface has a normal

$$\boldsymbol{\omega}_n(\phi, l) = \cos\phi \, \boldsymbol{\omega}_t^{\perp} + \sin\phi \, \boldsymbol{\omega}_t^{\top}, \tag{1.3}$$

3D coordinates

$$\boldsymbol{p}(\phi, l) = \boldsymbol{p}_1 + l \, \boldsymbol{\omega}_t + R \, \boldsymbol{\omega}_n(\phi), \tag{1.4}$$

and the normalized direction towards this point is

$$\boldsymbol{\omega}_{\boldsymbol{p}}(\phi, l) = \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}. \tag{1.5}$$

Implementation We provide the shader code for integrating the distribution over a cylinder in Listing 1.2.

```
float I_cylinder_numerical(vec3 p1, vec3 p2, float R)
{
    // init orthonormal basis
    float L = length(p2 - p1);
    vec3 wt = normalize(p2 - p1);
    vec3 wt1, wt2;
    buildOrthonormalBasis(wt, wt1, wt2);

    // integral discretization
    float I = 0.0;
    const int nSamplesphi = 20;
    const int nSamplesl   = 100;
    for (int i = 0; i < nSamplesphi; ++i)
    for (int j = 0; j < nSamplesl;   ++j)
    {
        // normal
        float phi = 2.0 * PI * float(i)/float(nSamplesphi);
        vec3 wn = cos(phi)*wt1 + sin(phi)*wt2;

        // position
        float l = L * float(j)/float(nSamplesl - 1);
        vec3 p = p1 + l*wt + R*wn;

        // normalized direction
        vec3 wp = normalize(p);

        // integrate
        I += D(wp) * max(0.0, dot(-wp, wn)) / dot(p, p);
    }

    I *= 2.0 * PI * R * L / float(nSamplesphi*nSamplesl);
    return I;
}
```

Listing 1.2. Numerical integration for the cylinder.

The helper function `buildOrthonormalBasis(in vec3 n, out vec3 b1, out vec3 b2)` takes a normalized direction as input and computes two orthonormal directions. We use the code snippet from [Frisvad 12].

```
// code from [Frisvad2012]
void buildOrthonormalBasis(
in vec3 n, out vec3 b1, out vec3 b2)
{
    if (n.z < -0.9999999)
    {
        b1 = vec3( 0.0, -1.0, 0.0);
        b2 = vec3(-1.0,  0.0, 0.0);
        return;
    }
    float a = 1.0 / (1.0 + n.z);
    float b = -n.x*n.y*a;
    b1 = vec3(1.0 - n.x*n.x*a, b, -n.x);
    b2 = vec3(b, 1.0 - n.y*n.y*a, -n.y);
}
```

Listing 1.3. Code for Building an Orthonormal Basis from a 3D Unit Vector.

1.2.4 The Linear-Light Integral

Configuration If the radius of the cylinder is zero ($R = 0$), we obtain a line segment defined by its end points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, as shown in Figure 1.6. Here, $L = \|\boldsymbol{p}_2 - \boldsymbol{p}_1\|$ is the length of the line segment and $\boldsymbol{\omega}_t = \frac{\boldsymbol{p}_2 - \boldsymbol{p}_1}{\|\boldsymbol{p}_2 - \boldsymbol{p}_1\|}$ is the tangent direction of the line.

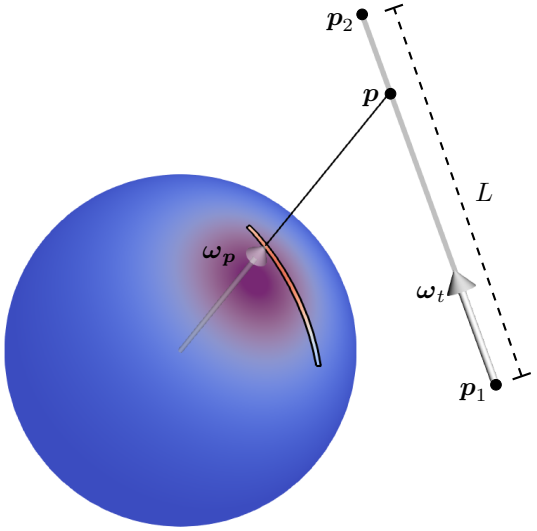

Figure 1.6. The linear-light integral.

Integral We rewrite Equation (1.1) in the space of the light instead of the sphere. With a 1D parameterization of variable l for the line, the integral is

$$I_{\text{line}} = \int_0^L D\left(\boldsymbol{\omega}_{\boldsymbol{p}}\right) \frac{2\,\|\boldsymbol{\omega}_{\boldsymbol{p}} \times \boldsymbol{\omega}_t\|}{\|\boldsymbol{p}\|^2}\, dl. \tag{1.6}$$

We set the origin of the parameterization to $\boldsymbol{p}_1$, and it increases in the direction $\boldsymbol{\omega}_t$, such that at abscissa l on the line, the 3D coordinates of the point are

$$p(l) = \boldsymbol{p}_1 + l\,\boldsymbol{\omega}_t, \tag{1.7}$$

and the normalized direction towards this point is

$$\boldsymbol{\omega}_{\boldsymbol{p}}(\phi, l) = \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}. \tag{1.8}$$

Implementation We provide the shader code for integrating the distribution over a line segment in Listing 1.4.

```
float I_line_numerical(vec3 p1, vec3 p2)
{
    float L = length(p2 - p1);
    vec3 wt = normalize(p2 - p1);

    // integral discretization
    float I = 0.0;
    const int nSamples = 100;
    for (int i = 0; i < nSamples; ++i)
    {
        // position
        vec3 p = p1 + L * float(i)/float(nSamples - 1) * wt;

        // normalized direction
        vec3 wp = normalize(p);

        // integrate
        I += 2.0 * D(wp) * length(cross(wp, wt)) / dot(p, p);
    }

    I *= L / float(nSamples);
    return I;
}
```

Listing 1.4. Numerical integration for the line.

1.2.5 Approximating the Cylinder-Light Integral by the Linear-Light Integral

As illustrated in Figure 1.7, for any continuous distribution D and a cylinder of radius R, the integral over the cylinder of Equation (1.2) converges towards the line segment integral of Equation (1.6) as R goes to zero:

$$\lim_{R\to 0} \frac{I_{\text{cyl}}(R)}{R} = I_{\text{line}}. \tag{1.9}$$

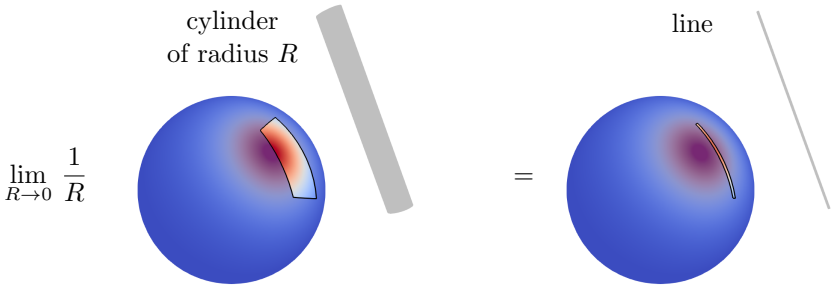

Figure 1.7. The linear-light integral is the limit of the cylinder-light integral when its radius tends towards zero.

Approximation If the radius R is small enough, we can use the line integral as an accurate approximation for the cylinder integral:

$$I_{\text{cyl}}(R) \approx R\, I_{\text{line}}. \tag{1.10}$$

In cases where the radius is too large, the approximation can be inaccurate and produce overly high values (for instance if D is a specular material and if the linear light overlaps with the specular peak). In order to avoid this, we use the property that I_{cyl} cannot be greater than the integral of D over the sphere:

$$I_{\text{cyl}} \leq \int_{\Omega} D(\boldsymbol{\omega})\, \mathrm{d}\boldsymbol{\omega}. \tag{1.11}$$

This property is intuitive: I_{cyl} represents the integral of D over the spherical domain covered by the cylinder, so it can only be smaller than the integral of D over the entire sphere. Hence, we can prevent the approximation from overshooting by clamping it to $\int_{\Omega} D(\boldsymbol{\omega})\, \mathrm{d}\boldsymbol{\omega}$. In practice, we use distributions D that are normalized, so we clamp the approximation to 1.

Implementation We provide the shader code for approximating the result of `I_cylinder` (from Listing 1.2) in Listing 1.5.

```
float I_cylinder_approx(vec3 p1, float p2, float R)
{
    return min(1.0, R * I_line(p1, p2));
}
```

Listing 1.5. Approximation of the cylinder-light integral by the linear-light integral.

Results of the Approximation In Figure 1.8, we compare the results obtained by the cylinder-light integral and the linear-light integral approximation with a GGX BRDF. We can see that the approximation is most accurate with:

- cylinders of small radius,
- cylinders far from the shading point, or
- low-frequency (large roughness parameter α) materials.

In summary, the approximation works well when the width of the solid angle covered by the cylinder is small compared to the variation of the distribution. However, the approximation typically cannot be used with specular materials and very large/close cylindrical light sources.

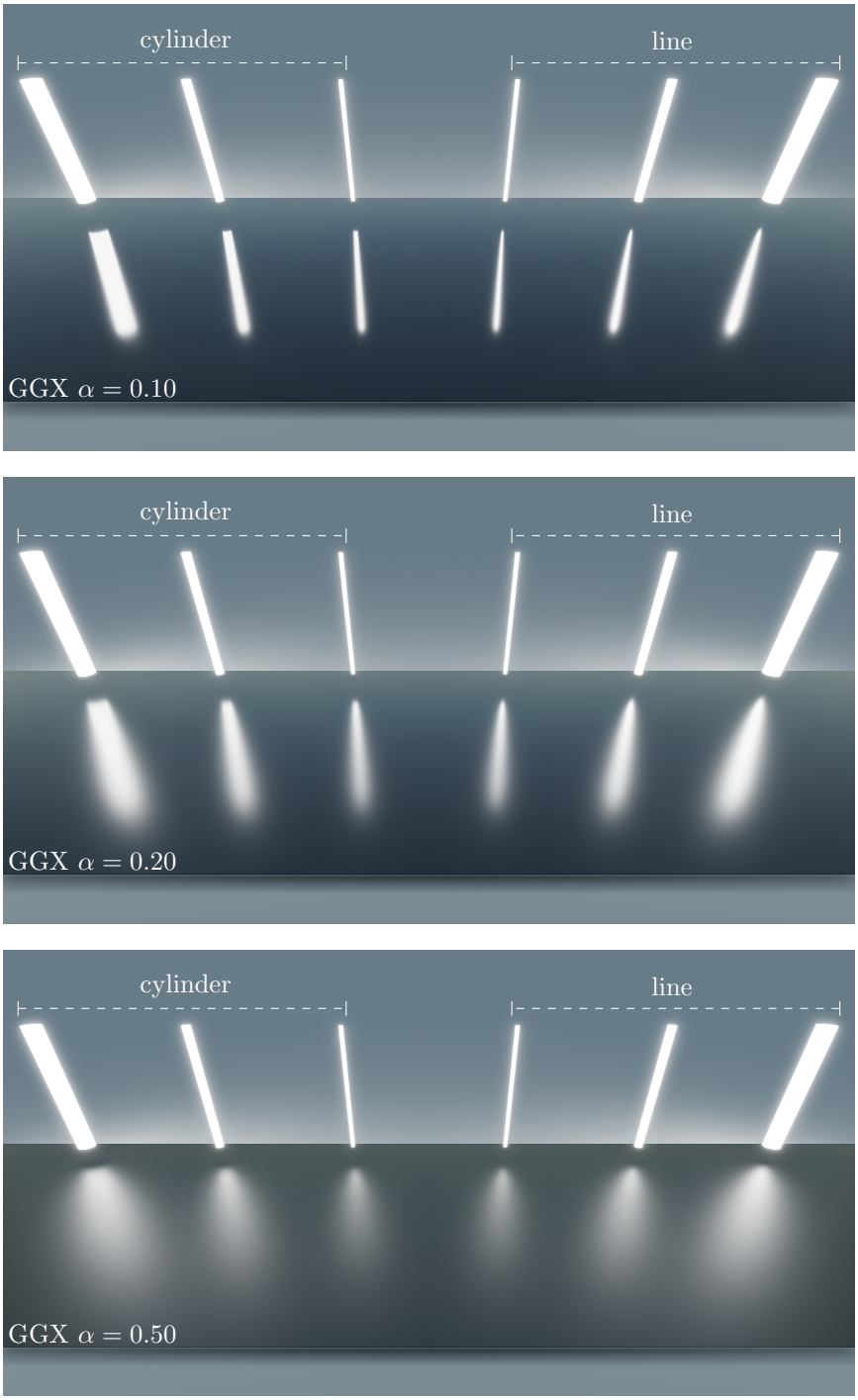

Figure 1.8. Results of the approximation with a GGX BRDF.

1.3 Line-Integral of a Diffuse Material

In this section, we show how to integrate a line against a diffuse BRDF, i.e., with $D(\boldsymbol{\omega}) = \frac{1}{\pi}\max(0, \boldsymbol{\omega} \cdot \boldsymbol{z})$. In this case, the integral of Equation (1.6) is also called the *irradiance* $I_{\text{line}} = E[\boldsymbol{L}]$ of the line $\boldsymbol{L}$.

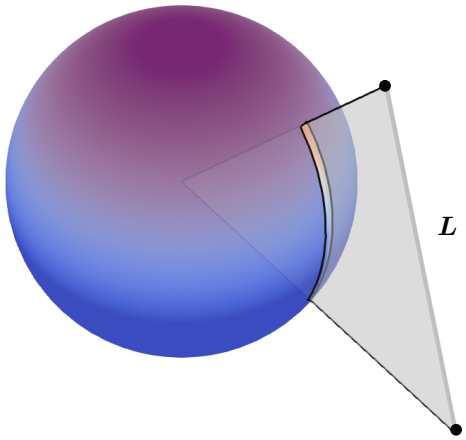

Figure 1.9. The diffuse-line integral (or the irradiance of the line).

Clamping the Line Below the Horizon The first step is to ensure that the parts of the light contributing to the diffuse integral are limited to the upper hemisphere ($z \geq 0$). To achieve this, we start by clamping the line to the upper part of the hemisphere. If one of the vertices is below the horizon—i.e., its z component is less than zero—we replace it with the intersection of the line with the plane $z = 0$, as illustrated in Figure 1.10.

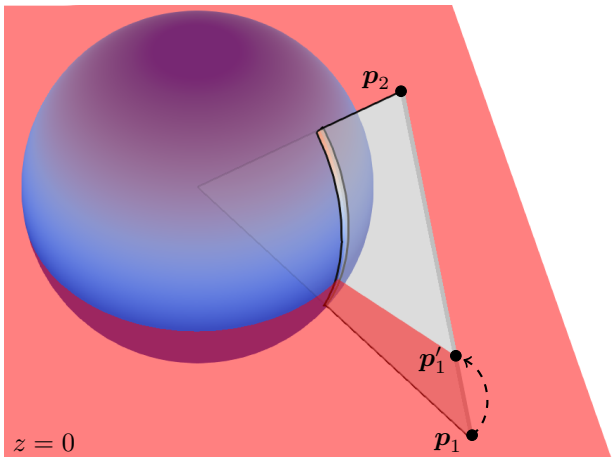

Figure 1.10. Clamping the linear light below the horizon.

Parameterization of the Line In order to compute the diffuse-line integral, we need a 1D parameterization for the linear light. Our parameterization is shown in Figure 1.11 and explained below.

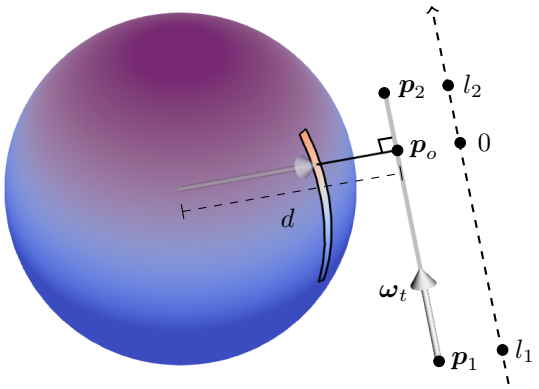

Figure 1.11. Line-integral parameterization.

The abscissas of the end points of the linear light are

$$l_1 = \boldsymbol{p}_1 \cdot \boldsymbol{\omega}_t, \tag{1.12}$$
$$l_2 = \boldsymbol{p}_2 \cdot \boldsymbol{\omega}_t. \tag{1.13}$$

The 0 abscissa is the orthonormal projection of the shading point onto the line, denoted $\boldsymbol{p}_o$ (it is not a problem if $\boldsymbol{p}_o$ is outside segment $[\boldsymbol{p}_1, \boldsymbol{p}_2]$). The distance between the line and the shading point is the norm of this point:

$$\boldsymbol{p}_o = \boldsymbol{p}_1 - l_1 \, \boldsymbol{\omega}_t, \tag{1.14}$$
$$d = \|\boldsymbol{p}_o\|. \tag{1.15}$$

To simplify the line integral, we parameterize the points on the line relative to $\boldsymbol{p}_o$, with an abscissa l

$$\boldsymbol{p}(l) = \boldsymbol{p}_o + l \, \boldsymbol{\omega}_t. \tag{1.16}$$

Integration of the Line We rewrite Equation (1.6) using the new parameterization. The terms in the integrand become

$$\|\boldsymbol{p}(l)\| = \sqrt{d^2 + l^2}, \tag{1.17}$$
$$\|\boldsymbol{\omega}_{\boldsymbol{p}}(l) \times \boldsymbol{\omega}_t\| = \frac{d}{\sqrt{d^2 + l^2}}, \tag{1.18}$$
$$D\left(\boldsymbol{\omega}_{\boldsymbol{p}}(l)\right) = \frac{1}{\pi} \frac{(\boldsymbol{p}_o + l \, \boldsymbol{\omega}_t)}{\sqrt{d^2 + l^2}} \cdot \boldsymbol{z}, \tag{1.19}$$

and the integral becomes

$$I_{\text{line}} = \frac{2\,d}{\pi} \int_{l_1}^{l_2} \frac{(\boldsymbol{p}_o + l\,\boldsymbol{\omega}_t) \cdot \boldsymbol{z}}{(d^2 + l^2)^2}\, dl, \tag{1.20}$$

which has the analytic closed form:

$$I_{\text{line}} = \frac{1}{\pi} \left\{ \left[F_{\boldsymbol{p}_o}(l_2) - F_{\boldsymbol{p}_o}(l_1) \right] \boldsymbol{p}_o + \left[F_{\boldsymbol{\omega}_t}(l_2) - F_{\boldsymbol{\omega}_t}(l_1) \right] \boldsymbol{\omega}_t \right\} \cdot \boldsymbol{z}, \tag{1.21}$$

with

$$F_{\boldsymbol{p}_o}(l) = \frac{l}{d(d^2 + l^2)} + \frac{1}{d^2} \operatorname{atan}\left(\frac{l}{d}\right), \tag{1.22}$$

$$F_{\boldsymbol{\omega}_t}(l) = \frac{l^2}{d(d^2 + l^2)}. \tag{1.23}$$

Implementation We provide the shader code for clamping, parameterizing and integrating the linear light against the diffuse BRDF in Listing 1.6.

```
float Fpo(float d, float l)
{
    return l/(d*(d*d + l*l)) + atan(l/d)/(d*d);
}

float Fwt(float d, float l)
{
    return l*l/(d*(d*d + l*l));
}

float I_diffuse_line(vec3 p1, vec3 p2)
{
    // tangent
    vec3 wt = normalize(p2 - p1);

    // clamping
    if (p1.z <= 0.0 && p2.z <= 0.0) return 0.0;
    if (p1.z < 0.0) p1 = (+p1*p2.z - p2*p1.z) / (+p2.z - p1.z);
    if (p2.z < 0.0) p2 = (-p1*p2.z + p2*p1.z) / (-p2.z + p1.z);

    // parameterization
    float l1 = dot(p1, wt);
    float l2 = dot(p2, wt);

    // shading point orthonormal projection on the line
    vec3 po = p1 - l1*wt;

    // distance to line
    float d = length(po);

    // integral
    float I = (Fpo(d, l2) - Fpo(d, l1)) * po.z +
              (Fwt(d, l2) - Fwt(d, l1)) * wt.z;
    return I / PI;
}
```

Listing 1.6. Analytic line-diffuse integration.

1.4 Line-Integral of a Glossy Material with LTCs

1.4.1 Linearly Transformed Cosines (LTCs)

Linearly Transformed Cosines are the distributions obtained by applying a linear transformation, represented by a 3×3 matrix M, to the direction vectors associated with a clamped cosine distribution denoted D_o. Fig. 1.12 shows how the choice of M affects the properties of the distribution. The matrix M provides control over roughness (b), anisotropy (c) and skewness (d) of the transformed distribution. The effect of the linear transformation can be seen on the lines and the red cube.

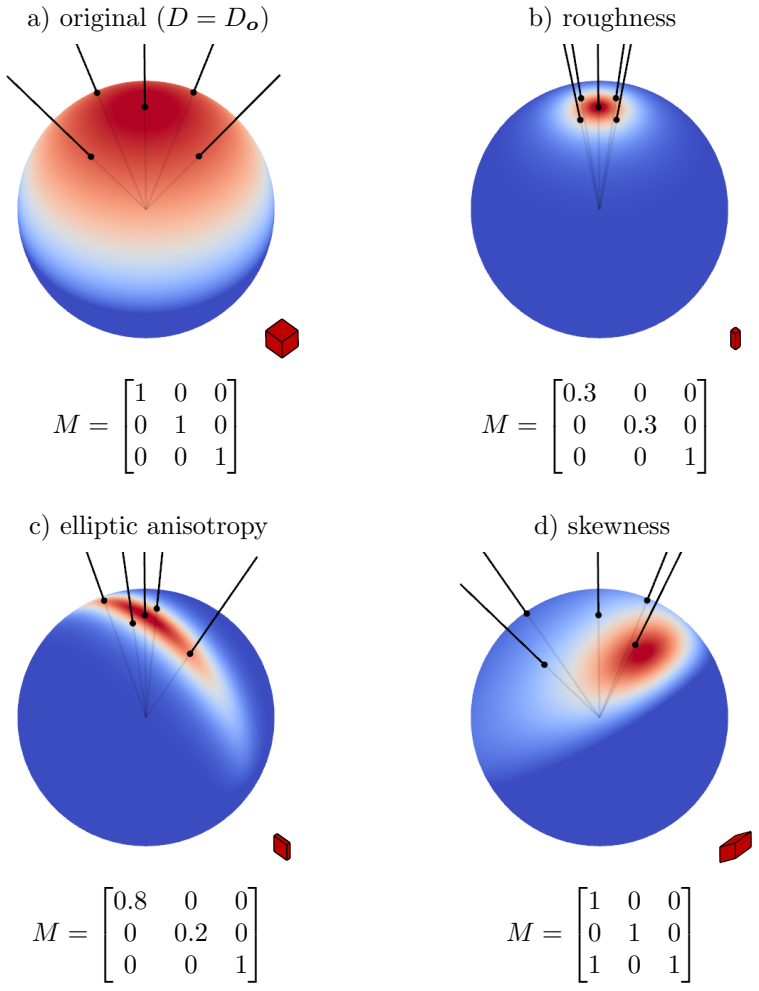

Figure 1.12. The parameterization of Linearly Transformed Cosines (LTCs).

Closed-Form Expression The magnitude of an LTC is the magnitude of the original distribution D_o in the original direction $\boldsymbol{\omega_o}$ multiplied by the change of solid angle measure due to the distortion of the spherical transformation. It has the closed-form expression:

$$D(\boldsymbol{\omega}) = D_o\left(\frac{M^{-1}\,\boldsymbol{\omega}}{\|M^{-1}\,\boldsymbol{\omega}\|}\right)\frac{|M^{-1}|}{\|M^{-1}\boldsymbol{\omega}\|^3}. \tag{1.24}$$

Note that this closed form is never used at runtime in the shader. It is only used to fit physically based materials with LTCs in an offline precomputation. In our shader, we approximate a GGX BRDF with LTCs whose parameters are stored in a look-up table that we access at runtime. The look-up table is the same as in [Heitz et al. 16].

1.4.2 LTC-Polygon Integral Invariance

LTCs are invariant to linear transformations, i.e., if a linear transformation is applied to both the polygon and the distribution, the value of the integral remains the same:

$$\begin{aligned}\int_{\boldsymbol{P}} D\left(\boldsymbol{\omega_p}\right)\frac{|-\boldsymbol{\omega_p}\cdot\boldsymbol{\omega}_n|}{\|\boldsymbol{p}\|^2}\,d\boldsymbol{p} &= \int_{\boldsymbol{P_o}} D_o\left(\boldsymbol{\omega_p}\right)\frac{|-\boldsymbol{\omega_p}\cdot\boldsymbol{\omega}_n|}{\|\boldsymbol{p}\|^2}\,d\boldsymbol{p}\\ &= E[\boldsymbol{P_o}].\end{aligned} \tag{1.25}$$

Thanks to this invariance, an LTC can be integrated over a polygon by multiplying the (vertices of the) polygon by the inverse linear transformation $\boldsymbol{P_o} = M^{-1}\,\boldsymbol{P}$ and computing the irradiance $E[\boldsymbol{P_o}]$ of this new polygon, as shown in Figure 1.13.

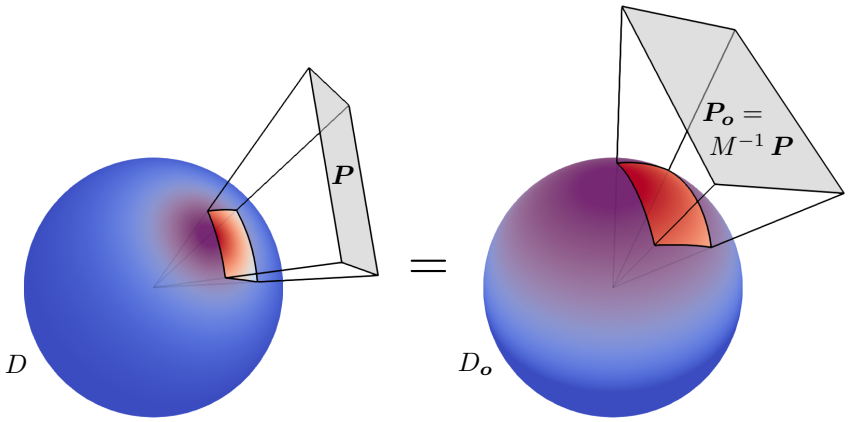

Figure 1.13. *Invariance of the polygonal integration.* The configuration on the right is the left configuration multiplied by matrix M^{-1}.

1.4.3 LTC-Line Integral Invariance

The invariance for linear lights is similar to the invariance for polygons given by Equation (1.25). If a linear transformation is applied to both the line segment and the distribution, then the value of the integral remains the same:

$$\int_{\boldsymbol{L}} D\left(\boldsymbol{\omega_p}\right) \frac{2\left\|\boldsymbol{\omega_p}\times\boldsymbol{\omega_t}\right\|}{\|\boldsymbol{p}\|^2}\, d\boldsymbol{p} = \frac{1}{\|M^T\,\boldsymbol{\omega}_\perp\|} \int_{\boldsymbol{L_o}} D_{\boldsymbol{o}}\left(\boldsymbol{\omega_p}\right) \frac{2\left\|\boldsymbol{\omega_p}\times\boldsymbol{\omega_t}\right\|}{\|\boldsymbol{p}\|^2}\, d\boldsymbol{p}$$
$$= \frac{1}{\|M^T\,\boldsymbol{\omega}_\perp\|}\, E[\boldsymbol{L_o}] \tag{1.26}$$

except for the additional width factor $\frac{1}{\|M^T\,\boldsymbol{\omega}_\perp\|}$. Thanks to this invariance, an LTC can be integrated over a line segment by multiplying the (vertices of the) line segment by the inverse linear transformation $\boldsymbol{L_o} = M^{-1}\,\boldsymbol{L}$, computing the irradiance $E[\boldsymbol{L_o}]$ of this new line segment using the method presented in Section 1.3, and multiplying the result by the width factor.

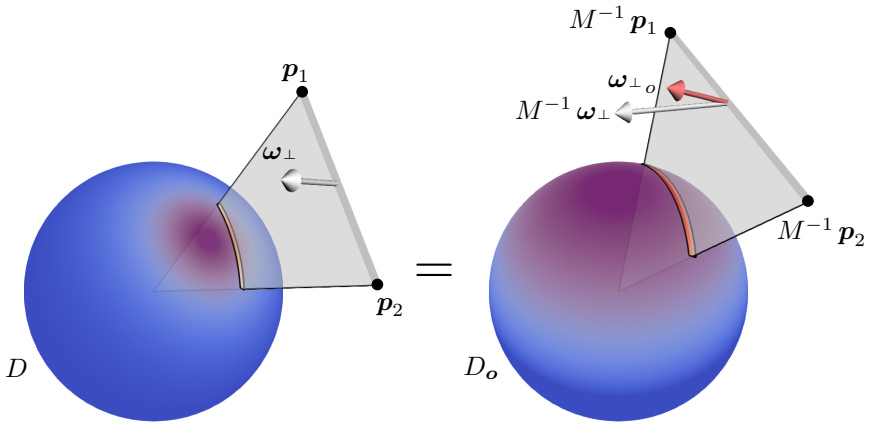

Figure 1.14. *Invariance of the linear integration.* The configuration on the right is the left configuration multiplied by matrix M^{-1}.

Proof of the Line-Integral Invariance We can see in Figure 1.14 that the infinitely small width of a linear light is defined by vector $\boldsymbol{\omega}_\perp = \frac{\boldsymbol{p}_1 \times \boldsymbol{p}_2}{\|\boldsymbol{p}_1 \times \boldsymbol{p}_2\|}$. After the linear transformation, this vector can be scaled and/or no longer orthonormal. The actual orthonormal vector—illustrated in red in the figure—is defined by

$$\begin{aligned} \boldsymbol{\omega}_{\perp_o} &= \frac{(M^{-1}(\boldsymbol{p}_1 - \boldsymbol{p}_2)) \times M^{-1}\boldsymbol{p}_1}{\|(M^{-1}(\boldsymbol{p}_1 - \boldsymbol{p}_2)) \times M^{-1}\boldsymbol{p}_1\|} \\ &= \frac{M^T[(\boldsymbol{p}_1 - \boldsymbol{p}_2) \times \boldsymbol{p}_1]}{\|M^T[(\boldsymbol{p}_1 - \boldsymbol{p}_2) \times \boldsymbol{p}_1]\|} \\ &= \frac{M^T\boldsymbol{\omega}_\perp}{\|M^T\boldsymbol{\omega}_\perp\|}, \end{aligned} \tag{1.27}$$

and is different from the transformed orthonormal vector $M^{-1}\,\boldsymbol{\omega}_\perp$. The transformation of this vector (its length and orientation) affects the evaluation of the linear light proportional to the width factor $\frac{1}{\|M^T\boldsymbol{\omega}_\perp\|}$. Indeed, the effective width after the transformation is the dot product between the transformed orthonormal vector and the actual orthonormal vector:

$$\begin{aligned} \boldsymbol{\omega}_{\perp_o} \cdot (M^{-1}\boldsymbol{\omega}_\perp) &= \frac{1}{\|M^T\boldsymbol{\omega}_\perp\|}\,(M^T\boldsymbol{\omega}_\perp) \cdot (M^{-1}\boldsymbol{\omega}_\perp) \\ &= \frac{1}{\|M^T\boldsymbol{\omega}_\perp\|}\,(M^{-T}\,M^T\boldsymbol{\omega}_\perp) \cdot \boldsymbol{\omega}_\perp \\ &= \frac{1}{\|M^T\boldsymbol{\omega}_\perp\|}, \end{aligned} \tag{1.28}$$

which is the expression of the width factor in Equation (1.26).

Implementation We provide the analytic LTC-line integral shader code in Listing 1.7. Note that, in practice, recovering the matrix M by inverting M^{-1} can be done in an optimized way, because the matrix is sparse (see demo and previous publication).

```
float I_ltc_line(vec3 p1, vec3 p2)
{
    // transform to diffuse configuration
    vec3 p1o = Minv * p1;
    vec3 p2o = Minv * p2;
    float I_diffuse = I_diffuse_line(p1o, p2o);

    // width factor
    vec3 ortho = normalize(cross(p1, p2));
    float w =  1.0 / length(inverse(transpose(Minv)) * ortho);

    return w * I_diffuse;
}
```

Listing 1.7. Analytic line-LTC integration.

1.5 Adding the End Caps

1.5.1 End Caps

So far, we have been using a line segment to approximate a cylindrical emitter. However, our line is only an approximation of the length of the cylinder, so it doesn't account for emission from the end caps of the cylinder. The shading with and without them is shown in Figure 1.15. We can see that a black spot shows up when the ends are not emitting.

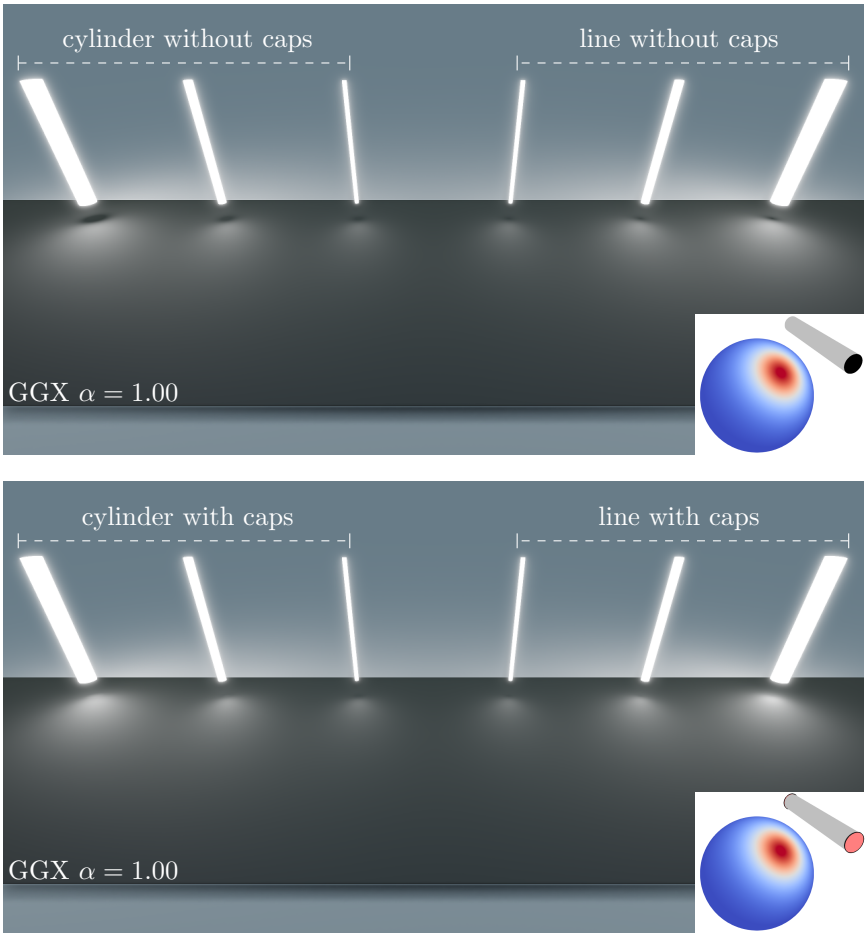

Figure 1.15. Test of the approximation with a GGX BRDF.

1.5.2 Integrating the End Caps

In order to remain consistent with the line-segment approximation of the cylinder's length, we approximate the two caps with infinitely small one-sided disks located at $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ and with normals $-\boldsymbol{\omega}_t$ and $\boldsymbol{\omega}_t$, respectively.

Numerical Integration The contribution of the disk located at $\boldsymbol{p}_1$ is

$$I_{\text{disk1}} = \int_0^R \int_0^{2\pi} D\left(\boldsymbol{\omega}_{\boldsymbol{p}}\right) \frac{\left|\boldsymbol{\omega}_{\boldsymbol{p}} \cdot \boldsymbol{\omega}_t\right|}{\|\boldsymbol{p}\|^2} r \, d\phi \, dr, \tag{1.29}$$

where we use a polar parameterization (ϕ, r) for the disk surface. With this parametrization, a point (ϕ, r) on the disk surface has 3D coordinates $\boldsymbol{p}$ and normalized direction $\boldsymbol{\omega}_{\boldsymbol{p}}$

$$\boldsymbol{p}(\phi, r) = \boldsymbol{p}_1 + r \cos\phi \, \boldsymbol{\omega}_t^{\perp} + r \sin\phi \, \boldsymbol{\omega}_t^{\top}, \tag{1.30}$$

$$\boldsymbol{\omega}_{\boldsymbol{p}}(\phi, r) = \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}. \tag{1.31}$$

Implementation In Listing 1.8, we provide the numerical integration shader code for Equation (1.29).

```
float I_disks_numerical(vec3 p1, vec3 p2, float R)
{
    // init orthonormal basis
    float L = length(p2 - p1);
    vec3 wt = normalize(p2 - p1);
    vec3 wt1, wt2;
    buildOrthonormalBasis(wt, wt1, wt2);

    // integration
    float Idisks = 0.0;
    const int nSamplesphi = 20;
    const int nSamplesr   = 200;
    for (int i = 0; i < nSamplesphi; ++i)
    for (int j = 0; j < nSamplesr;   ++j)
    {
        float phi = 2.0 * PI * float(i)/float(nSamplesphi);
        float r = R * float(j)/float(nSamplesr - 1);
        vec3 p, wp;

        p = p1 + r * (cos(phi)*wt1 + sin(phi)*wt2);
        wp = normalize(p);
        Idisks += r * D(wp) * max(0.0, dot(wp, +wt)) / dot(p, p);

        p = p2 + r * (cos(phi)*wt1 + sin(phi)*wt2);
        wp = normalize(p);
        Idisks += r * D(wp) * max(0.0, dot(wp, -wt)) / dot(p, p);
    }

    Idisks *= 2.0 * PI * R / float(nSamplesr*nSamplesphi);
    return Idisks;
}
```

Listing 1.8. Evaluating the end caps.

1.5.3 Approximating the End Caps with Point Lights

As the radius of the cylinder tends towards zero, the end caps converge towards disk-like point lights. The integral of Equation (1.29) converges towards

$$\lim_{R\to 0} \frac{I_{\text{disk1}}}{\pi R^2} = D(\boldsymbol{\omega}_{\boldsymbol{p}_1}) \frac{\max(0, \boldsymbol{\omega}_{\boldsymbol{p}_1} \cdot \boldsymbol{\omega}_t)}{\|\boldsymbol{p}_1\|^2}. \tag{1.32}$$

Hence, if the radius is small enough, the integral can be approximated by

$$I_{\text{disk1}} \approx \pi R^2 D(\boldsymbol{\omega}_{\boldsymbol{p}_1}) \frac{\max(0, \boldsymbol{\omega}_{\boldsymbol{p}_1} \cdot \boldsymbol{\omega}_t)}{\|\boldsymbol{p}_1\|^2}. \tag{1.33}$$

Similarly, we approximate the integral of the second end by

$$I_{\text{disk2}} \approx \pi R^2 D\left(\boldsymbol{\omega}_{\boldsymbol{p}_2}\right) \frac{\max(0, -\boldsymbol{\omega}_t \cdot \boldsymbol{\omega}_{\boldsymbol{p}_2})}{\|\boldsymbol{p}_2\|^2}. \tag{1.34}$$

Implementation In Listing 1.9, we provide the analytic approximation of Equations (1.33) and (1.34).

```
float I_ltc_disks(vec3 p1, vec3 p2, float R)
{
    float A = PI * R * R;
    vec3 wt  = normalize(p2 - p1);
    vec3 wp1 = normalize(p1);
    vec3 wp2 = normalize(p2);
    float Idisks = A * (
    D(wp1) * max(0.0, dot(+wt, wp1)) / dot(p1, p1) +
    D(wp2) * max(0.0, dot(-wt, wp2)) / dot(p2, p2));
    return Idisks;
}
```

Listing 1.9. Evaluating the end disks.

For this we need the evaluation of D for an LTC, which is provided in Equation (1.24) and implemented in Listing 1.10.

```
mat3 Minv;
float D(vec3 w)
{
    vec3 wo = Minv * w;
    float lo = length(wo);
    float res = 1.0/PI * max(0.0, wo.z/lo) * abs(determinant(Minv)) / ↩
        (lo*lo*lo);
    return res;
}
```

Listing 1.10. LTC Evaluation.

Summing the Contributions In order to account for the contribution of the caps, we simply add $I_{\text{disk1}} + I_{\text{disk2}}$ to I_{line} of Equation (1.6), and we clamp the sum to 1 as explained in Section 1.2.5.

1.5.4 Discussion

In practice, we found out that the caps approximation is less robust and useful than expected. Figure 1.16 shows that

- The approximation can result in visually disturbing artifacts: the reflection of the cylinder on the left exhibits a strange bulb at one end due to the cap approximation.
- Adding the caps is not always worth the cost, nor the risk of having artifacts. The two reflections in the middle (with and without caps) have similar reflection. In Figure 1.15, we can see that the absence of caps leaves black holes that are less visible with the line integral than with the cylinder integral anyway.

Hence, we recommend shading without the caps by default and adding them only for specific needs.

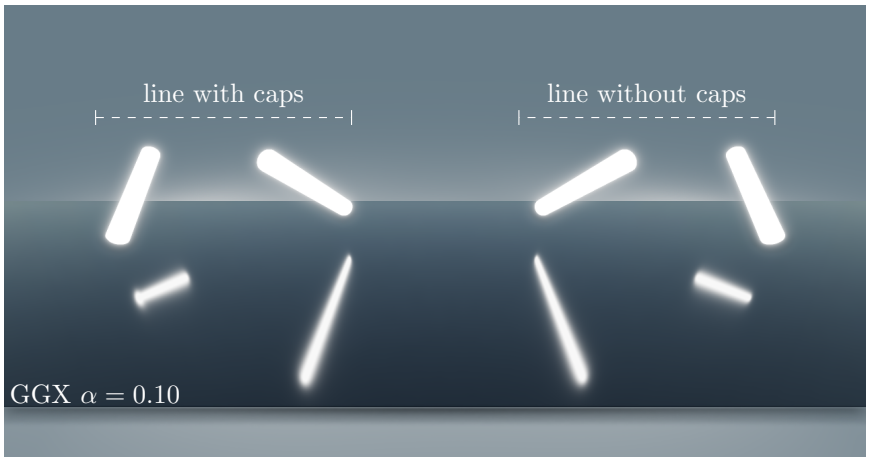

Figure 1.16. The caps approximation is not always worth it and can result in visually disturbing artifacts.

1.6 Rectangle-Like Linear Lights

Linear lights can also be used to model thin rectangular lights. In this case, the linear light parameters remain the same, with the addition of $\boldsymbol{\omega}_n$, the normal of the rectangle (Figure 1.17).

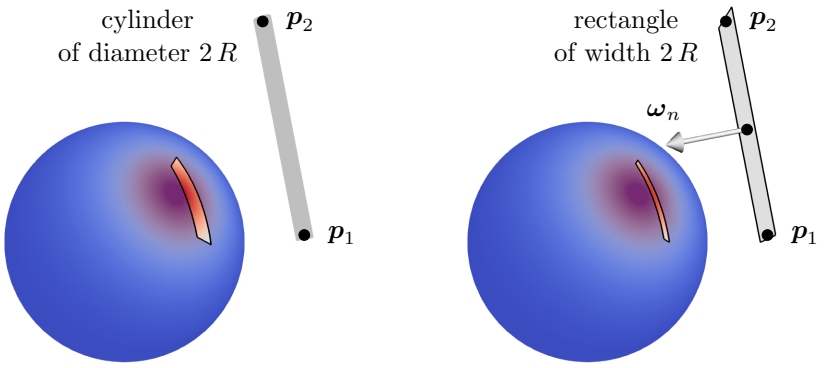

Figure 1.17. A linear light can also be used as an approximation for thin rectangular lights.

The line integral for the rectangle is the line integral for the cylinder of Equation (1.6) adjusted by the orientation of the line with respect to the shading point

$$I_{\text{lineRectangle}} = |\boldsymbol{\omega}_\perp \cdot \boldsymbol{\omega}_n| \, I_{\text{lineCylinder}}, \tag{1.35}$$

where $\boldsymbol{\omega}_\perp = \frac{\boldsymbol{p}_1 \times \boldsymbol{p}_2}{\|\boldsymbol{p}_1 \times \boldsymbol{p}_2\|}$ is the orthonormal vector introduced in Figure 1.14.

Implementation We provide the analytic LTC-line integral shader code for rectangular lines in Listing 1.11.

```
float I_ltc_line_rectangle(vec3 p1, vec3 p2, vec3 wn)
{
    vec3 wortho = normalize(cross(p1, p2));
    float I = abs(dot(wortho, wn)) * I_ltc_line(p1, p2);
    return I;
}
```

Listing 1.11. Analytic line-LTC integration for a rectangular line.

Test of the Approximation In Figure 1.18, we compare the results obtained by the rectangle-light integral and the linear-light integral approximation with a GGX BRDF. The approximation has the same properties as the cylinder light.

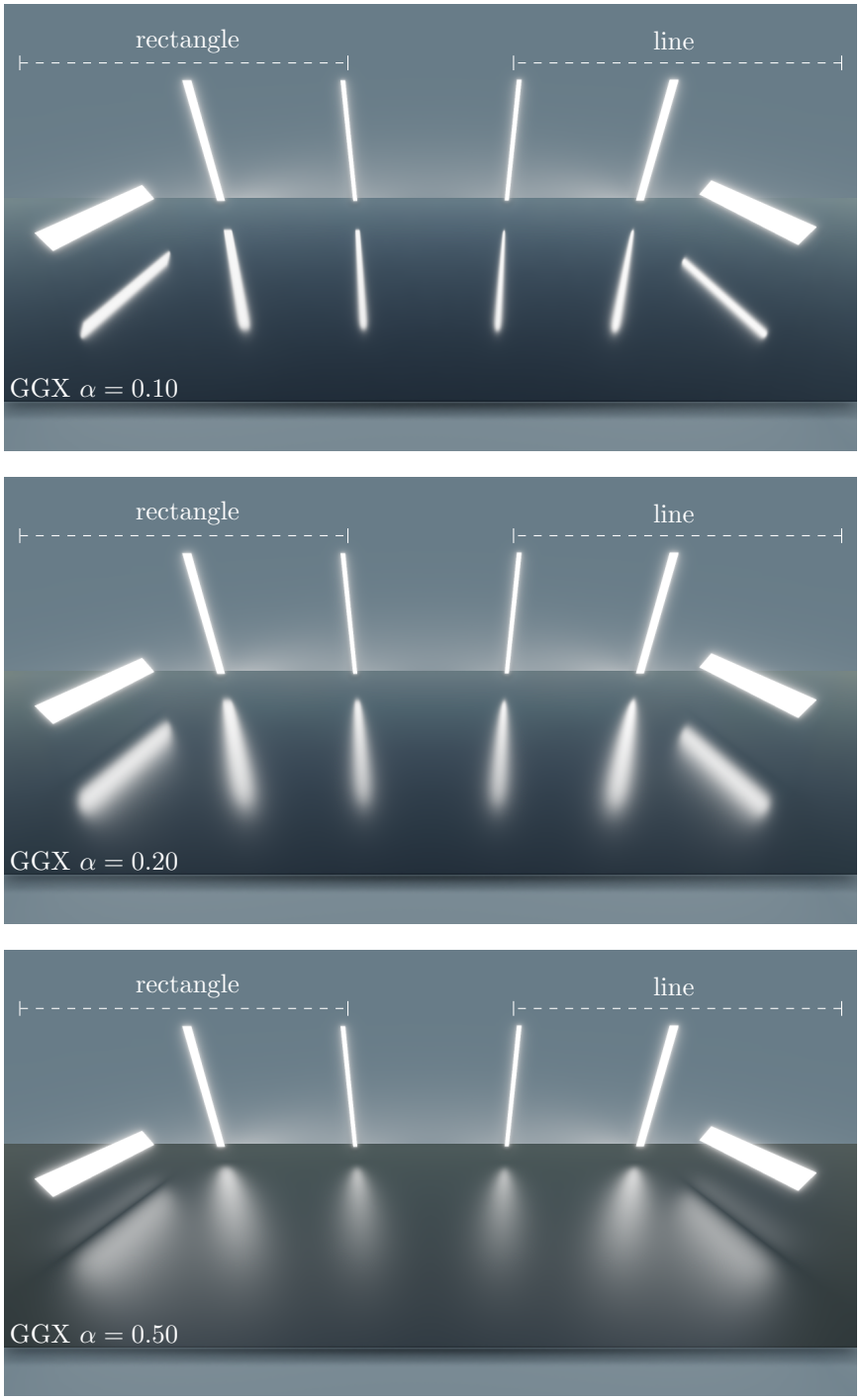

Figure 1.18. Test of the rectangle-light approximation with a GGX BRDF.

1.7 Performance

To assess the performance of our linear-light technique, we used the same Sponza scene (including viewpoint) as the original article. Although this is not a real production game environment, it is a suitable proxy in terms of pixel-shader workload and divergence, on account of the range of materials and surface orientations.

In our timings, using an NVIDIA Quadro M6000 GPU and a screen resolution of 1920×1080 pixels[2], the primary lighting pass took 0.42ms for a linear light without end caps, compared to 0.58ms for a quadrilateral light. This demonstrates, as expected, that linear lights are cheaper to evaluate than their polygonal counterparts, since only a single line integral is involved.

1.8 Conclusion

We have presented an extension to our existing area-lighting framework to support linear light sources, which can be used to model common real-world lights such as fluorescent bulbs. As we have shown, this approximation works well in many cases (with the exception of wide cylindrical lights or highly specular materials) and is cheaper than a full polygonal area light solution.

1.9 Acknowledgments

This work was supported by Unity Technologies (Eric Heitz) and Lucasfilm (Stephen Hill).

[2]We used an RGBA 8-bit render target and 1x MSAA to minimize bandwidth overhead and additional shading work.

Bibliography

[Bao and Peng 93] Hujun Bao and Qunsheng Peng. "Shading models for linear and area light sources." *Computers & Graphics* 17:2 (1993), 137–145.

[Frisvad 12] J. R. Frisvad. "Building an Orthonormal Basis from a 3D Unit Vector Without Normalization." *Journal of Graphics Tools* :16 (2012), 151–159.

[Heitz et al. 16] Eric Heitz, Jonathan Dupuy, Stephen Hill, and David Neubelt. "Real-time Polygonal-light Shading with Linearly Transformed Cosines." *ACM Trans. Graph.* 35:4 (2016), 41:1–41:8.

[Hill et al. 15] Stephen Hill, Stephen McAuley, Brent Burley, Danny Chan, Luca Fascione, MichałIwanicki, Naty Hoffman, Wenzel Jakob, David Neubelt, Angelo Pesce, and Matt Pettineo. "Physically Based Shading in Theory and Practice." In *ACM SIGGRAPH Courses 2015*, 2015.

[Nishita et al. 85] Tomoyuki Nishita, Isao Okamura, and Eihachiro Nakamae. "Shading Models for Point and Linear Sources." *ACM Trans. Graph.* 4:2 (1985), 124–146.

[Picott 92] Kevin P. Picott. "Extensions of the Linear and Area Lighting Models." *IEEE Comput. Graph. Appl.* 12:2 (1992), 31–38.

[Poulin and Amanatides 91] Pierre Poulin and John Amanatides. "Shading and shadowing with linear light sources." *Computers and Graphics* 15:2 (1991), 259–265.

[Walter et al. 07] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. "Microfacet Models for Refraction Through Rough Surfaces." In *Proceedings of EGSR 2007*, pp. 195–206, 2007.