Real-Time Polygonal-Light Shading with Linearly Transformed Cosines

Supplemental Material: Properties and MATLAB validation of Linearly Transformed Spherical Distributions

Eric Heitz
Unity Technologies

Jonathan Dupuy
Unity Technologies

Stephen Hill
Ubisoft

David Neubelt
Ready At Dawn Studios

Abstract

In this document, we show how *Linearly Transformed Spherical Distributions* inherit normalization, polygonal integration and sampling of their original distribution. We illustrate these properties with the uniform spherical, uniform hemispherical, clamped cosine, and squared clamped cosine distributions and we provide MATLAB code to validate these properties numerically.

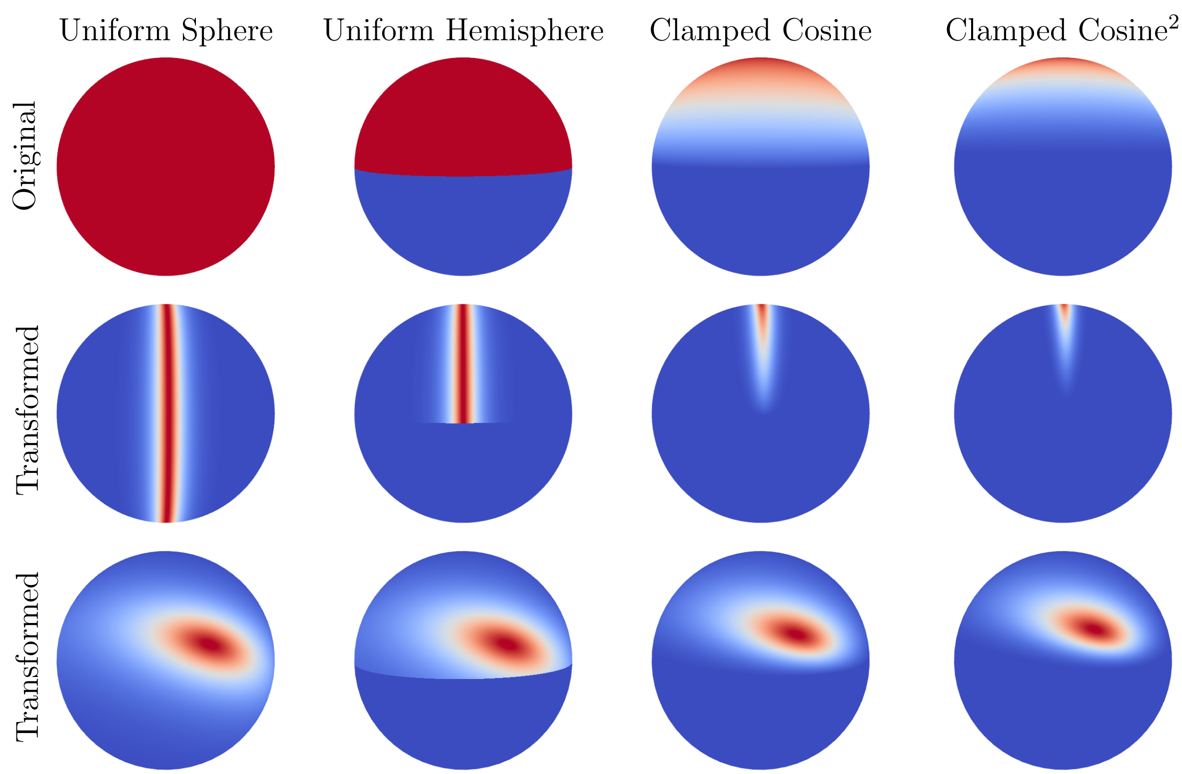

Figure 1: We apply linear transformations to different original spherical distributions.

Contents

1 Original Spherical Distributions

In this section, we introduce several normalized distributions that can be used for the original distribution $D_{\boldsymbol{o}}(\boldsymbol{\omega_o} = (x_{\boldsymbol{o}}, y_{\boldsymbol{o}}, z_{\boldsymbol{o}}))$. We provide their evaluation functions and their importance sampling methods.

1.1 Uniform Spherical Distribution

$$D_{\boldsymbol{o}}(\boldsymbol{\omega_o}) = \frac{1}{4\,\pi}$$

```
function [res] = Do_uniform_sphere(w)
	res = 1 / (4*pi);
endfunction
```

```
function [wo] = sample_Do_uniform_sphere()
	theta = acos(2*rand-1);
	phi = 2*pi*rand;
	wo = [cos(phi)*sin(theta) sin(phi)*sin(theta) cos(theta)];
endfunction
```

1.2 Uniform Hemispherical Distribution

$$D_{\boldsymbol{o}}(\boldsymbol{\omega_o}) = \begin{cases} \frac{1}{2\,\pi} & \text{if} \quad z_{\boldsymbol{o}} \geq 0 \\ 0 & \text{if} \quad z_{\boldsymbol{o}} < 0 \end{cases}$$

```
function [res] = Do_uniform_hemisphere(w)
	if w(3)<0 res = 0; else res = 1 / (2*pi); endif
endfunction
```

```
function [wo] = sample_Do_uniform_hemisphere()
	theta = acos(rand);
	phi = 2*pi*rand;
	wo = [cos(phi)*sin(theta) sin(phi)*sin(theta) cos(theta)];
endfunction
```

1.3 Clamped Cosine Distribution

$$D_{\boldsymbol{o}}(\boldsymbol{\omega_o}) = \begin{cases} \frac{z_{\boldsymbol{o}}}{\pi} & \text{if} \quad z_{\boldsymbol{o}} \geq 0 \\ 0 & \text{if} \quad z_{\boldsymbol{o}} < 0 \end{cases}$$

```
function [res] = Do_clamped_cosine(w)
	res = max(0, w(3)) / pi;
endfunction
```

```
function [wo] = sample_Do_clamped_cosine()
	theta = acos(rand^(1/2));
	phi = 2*pi*rand;
	wo = [cos(phi)*sin(theta) sin(phi)*sin(theta) cos(theta)];
endfunction
```

1.4 Squared Clamped Cosine Distribution

$$D_{\boldsymbol{o}}(\boldsymbol{\omega_o}) = \begin{cases} \frac{3\,z_{\boldsymbol{o}}^2}{2\,\pi} & \text{if} \quad z_{\boldsymbol{o}} \geq 0 \\ 0 & \text{if} \quad z_{\boldsymbol{o}} < 0 \end{cases}$$

```
function [res] = Do_squared_clamped_cosine(w)
	res = max(0, w(3))^2 * 3 / (2*pi);
endfunction
```

```
function [wo] = sample_Do_squared_clamped_cosine()
	theta = acos(rand^(1/3));
	phi = 2*pi*rand;
	wo = [cos(phi)*sin(theta) sin(phi)*sin(theta) cos(theta)];
endfunction
```

Normalization Note that these distributions are all normalized:

$$\int_{\Omega} D_{\boldsymbol{o}}(\boldsymbol{\omega_o})\,\mathrm{d}\boldsymbol{\omega_o} = 1.$$

2 Linearly Transformed Distributions

2.1 Definition

The Linearly Transformed Distributions are defined by

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

```
function [res] = D(w, M, Do)

	% invert matrix
	invM = inv(M);
	% Jacobian
	Jacobian = abs(det(invM)) / norm(invM * w')^3;
	% value of original distribution
	wo = invM * w' / norm(invM * w');
	a = feval(Do, wo);
	% result
	res = a * Jacobian;

endfunction
```

2.2 Normalization

We verify that the Linearly Transformed Distributions are normalized:

$$\begin{aligned}\int_\Omega D(\boldsymbol{\omega})\,d\boldsymbol{\omega} &= \int_0^\pi \int_0^{2\,\pi} D\left(\boldsymbol{\omega} = (\sin\theta\ \cos\phi, \cos\theta\ \cos\phi, \cos\theta)\right)\,\sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi \\ &= 1.\end{aligned}$$

```
function [res] = test_normalization(M, Do)

	res = 0;

	dangle = 0.01; % decrease for improved precision
	for theta = 0 : dangle : pi
	for phi = 0 : dangle : 2*pi

		w = [cos(phi)*sin(theta) sin(phi)*sin(theta) cos(theta)];

		res += dangle*dangle*sin(theta) * D(w, M, Do);

	end
	end

endfunction
```

Example

```
 >> M = [0.1 0.5 1 ; 0.2 1 1 ; 0 2 0.5 ]
M =

   0.10000   0.50000   1.00000
   0.20000   1.00000   1.00000
   0.00000   2.00000   0.50000

>> test_normalization(M, 'Do_uniform_sphere')
ans =  1.0003
>> test_normalization(M, 'Do_uniform_hemisphere')
ans =  1.0019
>> test_normalization(M, 'Do_clamped_cosine')
ans =  1.0006
>> test_normalization(M, 'Do_squared_clamped_cosine')
ans =  1.0006
```

2.3 Polygonal Integration

We verify that the integral of the Linearly Transformed Distributions over a polygon is the integral of their original distribution over the transformed polygon:

$$\int_{(\boldsymbol{p}_1,..,\boldsymbol{p}_n)} D(\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega} = \int_{(\boldsymbol{p}'_1,..,\boldsymbol{p}'_n)} D_{\boldsymbol{o}}(\boldsymbol{\omega}_{\boldsymbol{o}})\,\mathrm{d}\boldsymbol{\omega}_{\boldsymbol{o}},$$

with the transformed vertices $\boldsymbol{p}'_i = M^{-1}\,\boldsymbol{p}_i$.

Integration of D over $\boldsymbol{P}$ To compute the integral numerically, we integrate over the polygon and account for the Jacobian $\left|\left\langle \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}, \boldsymbol{n} \right\rangle \frac{1}{\|\boldsymbol{p}\|^2}\right|$ of the spherical projection:

$$\int_{(\boldsymbol{p}_1,..,\boldsymbol{p}_n)} D(\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega} = \int_{(\boldsymbol{p}_1,..,\boldsymbol{p}_n)} D\left(\frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}\right) \left|\left\langle \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}, \boldsymbol{n} \right\rangle \frac{1}{\|\boldsymbol{p}\|^2}\right| \mathrm{d}\boldsymbol{p}$$

where $\boldsymbol{n}$ is the normal of the polygon.

```
function [res] = test_polygonal_integration_reference(P1, P2, P3, M, Do)

	% triangle's normal
	N = cross(P1-P2, P1-P3) / norm(cross(P1-P2, P1-P3));
	% triangle's area
	A = 0.5*norm(cross(P1-P2, P1-P3));

	res = 0;
	dx = 0.0025; % decrease for improved precision
	for x = 0 : dx : 1
	for y = 0 : dx : 1

		% generate point on triangle
		if x > y
			P = (1.0-x) * P1 + (x-y) * P2 + y * P3;
		else
			P = (1.0-y) * P1 + (y-x) * P2 + x * P3;
		endif

		% direction to point
		w = P / norm(P);
		% squared distance
		d2 = norm(P)^2;

		% integrate D
		res += dx*dx*A * abs(dot(w, N)) / d2 * D(w, M, Do);

	end
	end

endfunction
```

Integration of $D_{\boldsymbol{o}}$ over $\boldsymbol{P}'$ First, we transform the vertices $\boldsymbol{p}_i' = M^{-1}\boldsymbol{p}_i$. Then, to compute the integral numerically, we integrate over the polygon $\boldsymbol{P}'$ and account for the Jacobian $\left\langle \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}, \boldsymbol{n} \right\rangle \frac{1}{\|\boldsymbol{p}\|^2}$ of the spherical projection:

$$\int_{(\boldsymbol{p}_1',..,\boldsymbol{p}_n')} D_{\boldsymbol{o}}(\boldsymbol{\omega_o})\,\mathrm{d}\boldsymbol{\omega_o} = \int_{(\boldsymbol{p}_1',..,\boldsymbol{p}_n')} D_{\boldsymbol{o}}\left(\frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}\right) \left| \left\langle \frac{\boldsymbol{p}}{\|\boldsymbol{p}\|}, \boldsymbol{n} \right\rangle \frac{1}{\|\boldsymbol{p}\|^2} \right| \mathrm{d}\boldsymbol{p},$$

where $\boldsymbol{n}$ is the normal of the polygon.

```
function [res] = test_polygonal_integration(P1, P2, P3, M, Do)

        % transform triangle
        Minv = inv(M);
        P1 = (Minv * P1')';
        P2 = (Minv * P2')';
        P3 = (Minv * P3')';

        % triangle's normal
        N = cross(P1-P2, P1-P3) / norm(cross(P1-P2, P1-P3));
        % triangle's area
        A = 0.5*norm(cross(P1-P2, P1-P3));

        res = 0;
        dx = 0.0025; % decrease for improved precision
        for x = 0 : dx : 1
        for y = 0 : dx : 1

                % generate point on triangle
                if x > y
                        P = (1.0-x) * P1 + (x-y) * P2 + y * P3;
                else
                        P = (1.0-y) * P1 + (y-x) * P2 + x * P3;
                endif

                % direction to point
                wo = P / norm(P);
                % squared distance
                d2 = norm(P)^2;

                % integrate Do
                res += dx*dx*A * abs(dot(wo, N)) / d2 * feval(Do, wo);

        end
        end

endfunction
```

Example

```
>> P1 = [0 0 1]
P1 =

   0   0   1

>> P2 = [1 0 1]
P2 =

   1   0   1

>> P3 = [1 1 1]
P3 =

   1   1   1
>> M = [0.1 0.5 1 ; 0.2 1 1 ; 0 2 0.5 ]
M =

   0.10000   0.50000   1.00000
   0.20000   1.00000   1.00000
   0.00000   2.00000   0.50000

>> test_polygonal_integration_reference(P1, P2, P3, M, 'Do_clamped_cosine')
ans =  0.0059141
>> test_polygonal_integration(P1, P2, P3, M, 'Do_clamped_cosine')
ans =  0.0059141
```

2.4 Importance Sampling

We generate samples $\boldsymbol{\omega}$ from PDF $D(\boldsymbol{\omega})$ using Algorithm 1.

Algorithm 1 Importance Sampling

sample $\boldsymbol{\omega_o}$ from $D_{\boldsymbol{o}}$ ▷ $\mathrm{PDF}(\boldsymbol{\omega_o}) = D_{\boldsymbol{o}}(\boldsymbol{\omega_o})$

return $\boldsymbol{\omega} = \frac{M\,\boldsymbol{\omega_o}}{\|M\,\boldsymbol{\omega_o}\|}$ ▷ $\mathrm{PDF}(\boldsymbol{\omega}) = D_{\boldsymbol{o}}(\boldsymbol{\omega_o})\,\frac{\partial \omega_o}{\partial \omega} = D(\boldsymbol{\omega})$

```
function [w] = D_sample(M, Do_sample)

        wo = feval(Do_sample);
        w = (M * wo')' / norm(M * wo');

endfunction
```

Numerical Validation We verify that the importance sampling technique is consistent with the distribution. We check that the moments of the direction components computed via numerical integration and importance sampling are equal.

$$E[\boldsymbol{\omega}] = \int_{\Omega} D(\boldsymbol{\omega})\,\boldsymbol{\omega}\,\mathrm{d}\boldsymbol{\omega}$$

$$E[\boldsymbol{\omega}] = \int_0^{\pi}\int_0^{2\,\pi} D\left(\boldsymbol{\omega} = (\sin\theta\ \cos\phi, \cos\theta\ \cos\phi, \cos\theta)\right)\,(\sin\theta\ \cos\phi, \cos\theta\ \cos\phi, \cos\theta)\right)\,\sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi$$

$$E[\boldsymbol{\omega}] = \frac{1}{N}\sum_{i}^{N}\boldsymbol{\omega}_i.$$

```
function [res] = test_sampling(M, Do, Do_sample)

        %%% Reference
        moment_x = 0;
        moment_y = 0;
        moment_z = 0;
        dangle = 0.01; % decrease for improved precision
        for theta = 0 : dangle : pi
        for phi = 0 : dangle : 2*pi

                w = [cos(phi)*sin(theta) sin(phi)*sin(theta) cos(theta)];

                weight = dangle*dangle*sin(theta) * D(w, M, Do);
                moment_x += weight * w(1);
                moment_y += weight * w(2);
                moment_z += weight * w(3);

        end
        end
        disp('reference')
        moment_x
        moment_y
        moment_z

        %%% Sampling
        moment_x = 0;
        moment_y = 0;
        moment_z = 0;
        for sample = 1 : 10000

                w = D_sample(M, Do_sample);
                moment_x += 0.0001 * w(1);
                moment_y += 0.0001 * w(2);
                moment_z += 0.0001 * w(3);

        end
        disp('sampling')
```

```
        moment_x
        moment_y
        moment_z

endfunction
```

Example

```
>> test_sampling(M, 'Do_squared_clamped_cosine', 'sample_Do_squared_clamped_cosine')
reference
moment_x =  0.54400
moment_y =  0.49160
moment_z =  0.10410
sampling
moment_x =  0.54477
moment_y =  0.49372
moment_z =  0.10481
```