ECE 594Q – Project Report

Real-Time Polygonal-Light Shading with Linearly Transformed Cosines

Rahul Vishwakarma

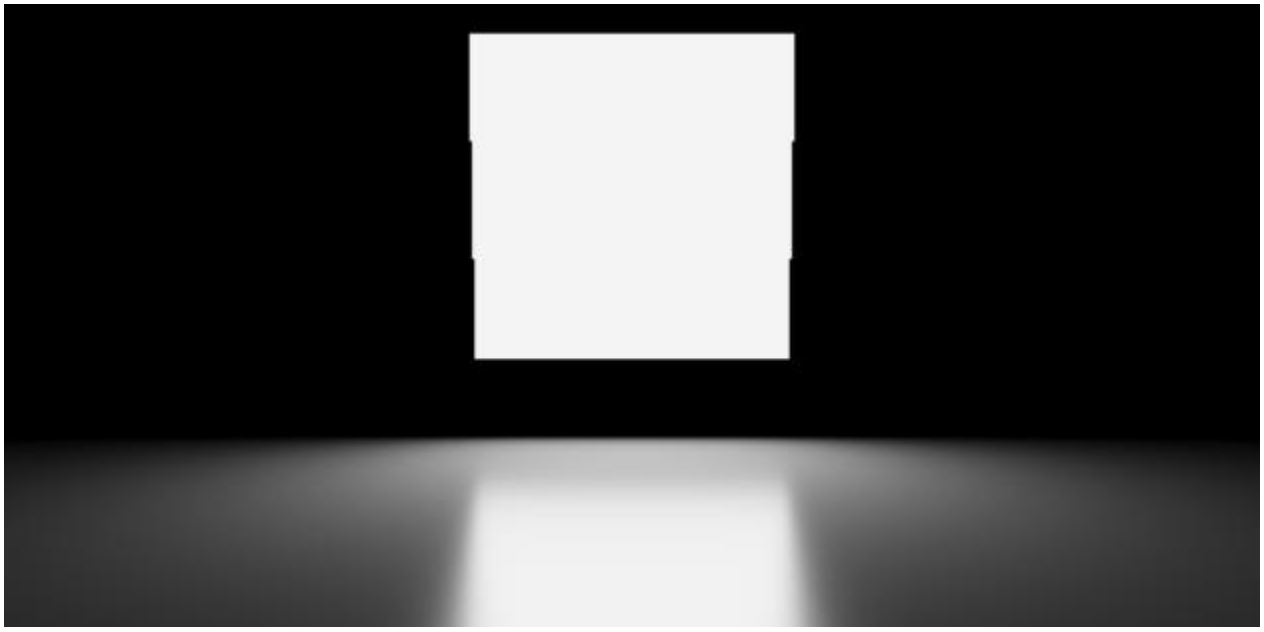


Figure 1: *Sample lighting output.based on pre computed linearly transformed cosines*

Introduction

In this report, we describe the lighting with polygonal sources. This has been a common practice using in offline rendering however real-time rendering has mostly used point light sources due to computational challenges.

Firstly, we have the BRDF: a spherical function that describes how the material scatters light at a particular shading point. The shading result (outgoing radiance) is the integral of the BRDF over this spherical polygon.

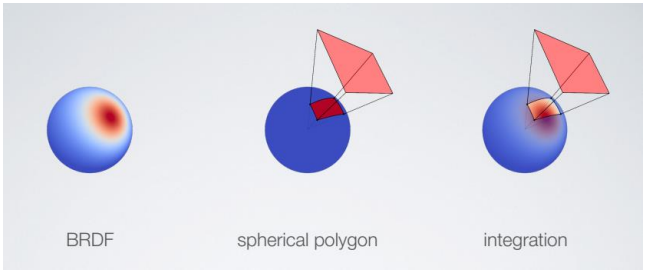


Figure 2: *Basic method for computation*

This is much harder than point lighting, where we only need to evaluate the BRDF for a single light direction. We now need to consider many directions.
In offline rendering, we might solve this with Monte Carlo sampling, but for real-time this isn't a viable option – it would either be too slow or too noisy.
We'd like to find a closed-form solution instead, i.e. an equation we can simply evaluate that will give us the right answer immediately without needing to sample.
Since this is tricky we approach this problem based on the spherical distribution.

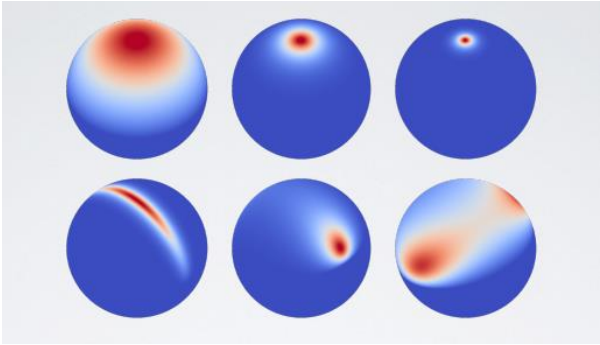


Figure 3: *General requirements*

Authors approached this using a Linearly Transformed Cosines function where the main idea is to take a simple distribution and apply a linear transform to it. In doing so, we can create a wide range of more sophisticated 'shapes' (spherical functions).

All the behaviors required for the functions can be replicated by using cosines whether it is roughness, anisotropy, skewness or a random distribution. All this is possible using a linear transformation of the cosines function.

Here we see the various linear transform needed with the effect.

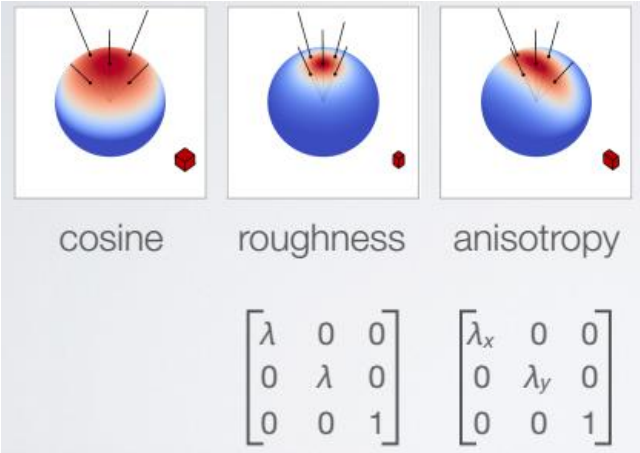


Figure 4: *Goodness in linear transforms*

We can even create all sorts of wacky behaviour with random transforms, some of which even lead to bimodal distributions. So, as you can see, this approach is very expressive.

1 Approach

In general we can take a cosine distribution and apply an arbitrary 3x3 matrix, M, producing a new distribution.

In the paper this is referred to as the family of distributions generated in this way as Linearly Transformed Cosines.

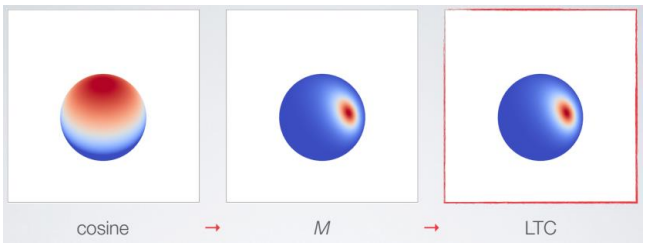


Figure 5: *Linear Transform of Cosines (LTC)*

In order to use this method in practice we want to compute the integral of a GGX-based BRDF with a polygonal light source.

First we find a linear transform M that best approximates this BRDF with an LTC, for a given roughness and view angle. We do this ahead of time for all roughnesses and view angles, and store the resulting matrices in a table (=texture).

At runtime we take our BRDF-polygon configuration, and apply the inverse transform to the vertices of the polygon, based on our LTC fitting of that BRDF (for the view angle and roughness at the current shading point).

This turns the configuration into an equivalent but simpler integration problem. You can think of this as transforming back to 'cosine space'

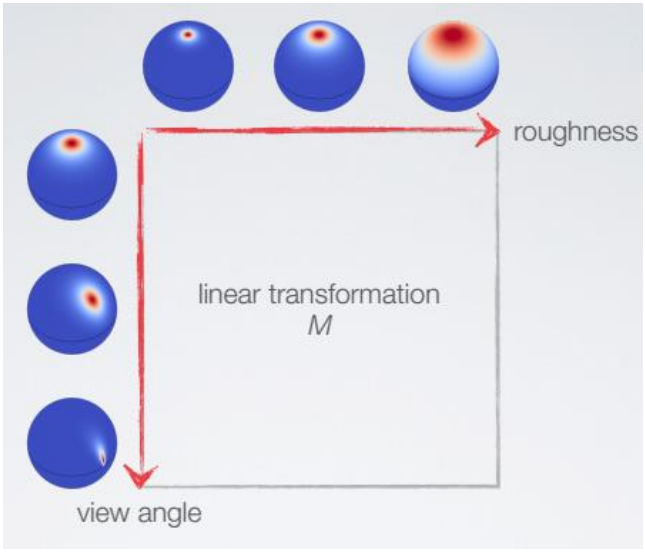


Figure 6: *Pre computed LTC*

2 Implementation

2.1 Lookup M-1, based on roughness & v. angle

Read the precomputed linear transformations based on the roughness and view angles. This is typically read as a texture and fed to the fragment code.

2.2 Transform polygon by M-1

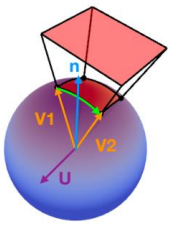
Fit the matrices into a four component texture. However, we still needed a fifth component for the magnitude of the BRDF, so a second texture fetch in the shader was unavoidable

2.2 Clip polygon to upper hemisphere

To get the correct result (form factor) of the polygon, we need to clip the polygon to the upper hemisphere. We implement a method "void ClipQuadToHorizon(inout vec3 L[5], out int n)" in the fragment shader to do this clipping.

2.2 Compute edge integrals

Here is the edge integral computation for the vector v1 and v2. We need to do this for all the four edges of the square.




```
float IntegrateEdge(vec3 v1, vec3 v2)
{
   float cosTheta = dot(v1, v2);
   float theta = acos(cosTheta);
   float res = cross(v1, v2).z * ((theta > 0.001) ? theta/sin(theta) : 1.0);
   return res;
}
```

In practice we don't calculate the area integral directly but instead evaluate a series of line/edge integrals over the boundary of the spherical polygon (one for each edge).

3 Conclusion

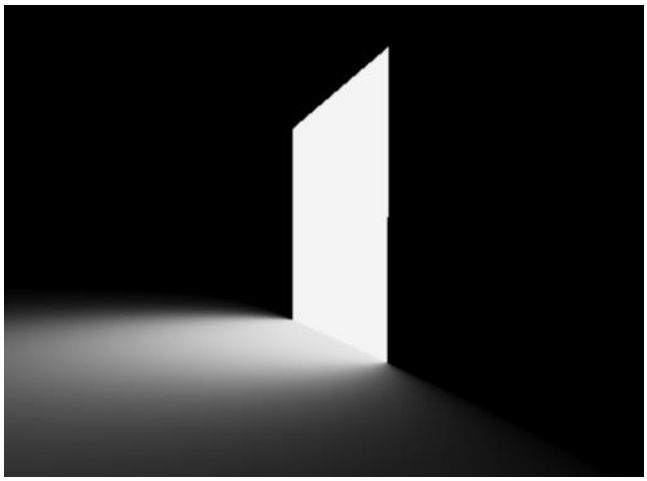


Figure 8: *Lighting results*

The final lighting can be computed in real time and has all the good property that we can extract. However, the caveat that this solution has is that, as the edges and the number of light sources grow our computations will grow accordingly. This limits our capability to have complex shapes with multiple edges in the scene.

References

ERIC HEITZ, JONATHAN DUPUY, STEPHEN HILL & DAVID NEUBELT. Real-time polygonal-light shading with linearly transformed cosines (Proceedings of ACM SIGGRAPH 2016)