

Fast Analytic Soft Shadows from Area Lights

Aakash KT¹, Parikshit Sakurikar^{1,2}, P. J. Narayanan¹

¹CVIT, KCIS, IIIT-Hyderabad
²DreamVu Inc.



Figure 1: Our method computes analytic shading and soft shadows for physically based BRDFs from arbitrary area lights. The scene shown here has two area lights (Quad at the top, lamp at the left). Our result is completely noise free and has lesser MAE (Mean Absolute Error) and RMSE (Root Mean Squared Error) as compared to direct illumination ray tracing and the Ratio Estimator [HHM18]. The reference is rendered for a large number of samples per pixel (~ 400 spp).

Abstract

In this paper, we present the first method to analytically compute shading and soft shadows for physically based BRDFs from arbitrary area lights. We observe that for a given shading point, shadowed radiance can be computed by analytically integrating over the visible portion of the light source using Linearly Transformed Cosines (LTCs). We present a structured approach to project, re-order and horizon-clip spherical polygons of arbitrary lights and occluders. The visible portion is then computed by multiple repetitive set difference operations. Our method produces noise-free shading and soft-shadows and outperforms ray-tracing within the same compute budget. We further optimize our algorithm for convex light and occluder meshes by projecting the silhouette edges as viewed from the shading point to a spherical polygon, and performing one set difference operation thereby achieving a speedup of more than $2\times$. We analyze the run-time performance of our method and show rendering results on several scenes with multiple light sources and complex occluders. We demonstrate superior results compared to prior work that uses analytic shading with stochastic shadow computation for area lights.

CCS Concepts

• **Computing methodologies** \rightarrow **Visibility**; Ray tracing;

1. Introduction

Achieving photorealism in rendered images requires intricate geometric objects, rich material models, and versatile lighting. Computer graphics has come a long way in rendering detailed 3D scenes with rich and complex material and lighting models. Accurate rendering of lighting effects, including accurate soft shadows is possible using ray tracing but requires huge computational effort. This

has spurred further research in reducing computational complexity by tracing those rays to light sources that contribute the most to the rendered image. In this paper, we address the problem of analytically computing shading and soft shadows with emissive 3D shapes as area lights.

Ray tracing area light sources involves shooting multiple light rays for each shading point. It would be convenient if the light-

ing effects at a scene point can be computed analytically from the geometry of the light source. Analytic solutions for direct lighting from specific area light sources have been proposed before. Recently, Heitz et al. 2016[HDHN16] proposed a method to analytically compute shading from a polygonal light source and later extended it to spherical lights [DHB17]. These methods, however, do not directly handle general emissive 3D meshes for lights. They also do not consider occlusions and shadows. Other work such as [HHM18] incorporates soft shadows computed from stochastic Monte Carlo (MC) techniques with analytic solutions. To our knowledge, no prior work exists that analytically computes soft shadows from area lights for physically based materials.

We propose a structured approach to analytically compute shading and soft shadows from light sources with arbitrary geometry. We extend the method of Heitz et al. 2016[HDHN16] to light sources of arbitrary 3D shapes. We compute a spherical polygon of the *visible* part of the light source by performing a set difference operation on spherical polygons of light sources and occluders. This polygon can be directly integrated over to compute the occluded irradiance using Linearly Transformed Cosines (LTCs) [HDHN16]. Since we always integrate over polygonal domains that are visible, our method can compute soft shadows. Finding the visible spherical polygon of an arbitrary light source is a non-trivial operation. We consider individual triangles of lights and occluders to find visible regions. Since this is computationally expensive, we further propose an optimization for the case of convex lights and occluders. The contributions of this paper are: (a) A structured approach to analytically compute soft shadows from spherical projections of lights and occluders with any 3D shape and (b) A method to efficiently obtain spherical polygons of arbitrary convex geometry, resulting in a speedup of more than $2\times$.

2. Related Work

Polygonal Area Light Shading. An analytical formula for integrating clamped-cosine distributions over spherical polygons was given by Lambert[Lam60] and Baum et al.[BRW89]. This was later extended by Arvo[Arv95] to Phong distributions [Pho75], which provide control over the sharpness using an exponent. Heitz et al. 2016[HDHN16] proposed a method to analytically integrate spherical polygons over GGX microfacet distributions [WMLT07] using Linearly Transformed Cosines (LTCs). Their method transforms arbitrary GGX distributions to the clamped cosine distribution using a pre-computed transformation matrix. They show that the same matrix can be applied to the direction vectors of spherical polygons to obtain a transformed spherical domain of integration. The rendering equation can be analytically integrated over the transformed domain to obtain shading. Concurrently, Lecocq et al.[LDSM16] proposed an approximation for Phong-polygon integration with $O(n)$ complexity instead of $O(e \cdot n)$ as proposed by Arvo[Arv95] (n being the number of polygon vertices and e the phong exponent). A similar approach using 4D Möbius transformations for the specific case of spherical area lights was proposed by Dupuy et al. 2017[DHB17]. In this work, we extend the method of Heitz et al. 2016[HDHN16] in two directions: (1) Analytically compute soft shadows from general 3D lights and occluders, and (2) Efficient computation for the case of convex meshes.

PRT with Polygonal Area lights. Recent research has enabled real-time polygonal area lighting in Pre-computed Radiance Transfer (PRT) [SKS02] frameworks. Specifically, the work of Wang et al.[WR18] and Belcour et al.[BXH*18] propose methods based on zonal harmonic decomposition to compute spherical harmonics (SH) coefficients for polygonal area lights. As is the norm, the Light Transport Function (LTF) is pre-computed and projected to SH and is combined at run-time with the SH computed for area lighting. These methods were extended to efficiently handle many lights in real-time [WCZR20]. PRT methods produce real-time shading and soft-shadows from area lights but require pre-computation and storage. Further, the PRT framework is limited by the representational power of SH and performs poorly on specular materials. Our method in contrast requires no pre-computation and produces soft shadows on the fly and can handle glossy and specular materials correctly.

On the fly soft shadows. Dupuy et al. 2017[DHB17] use a pre-computed representation of visibility (bent cones) to incorporate soft shadows with an analytic solution for the specific case of spherical lights. Such pre-computation based methods however do not produce soft-shadows on the fly. Mora et al.[MAAG12] propose to use the Plücker space to analytically represent visible and non-visible portions of an area light source, which is then used to analytically compute soft shadows. Note that their method works only for diffuse materials, and its extension for general materials or LTCs is non-trivial. Heitz et al. 2018[HHM18] proposed to combine analytic solutions with stochastic methods using a *ratio* estimator for shadow computation. Their ratio estimator uses Monte Carlo (MC) estimation and combines it with the analytical solution of Heitz et al. 2016[HDHN16]. The stochastic ratio estimator represents shadowed regions, which are independently denoised to preserve texture and high frequency details. They focus on real-time applications and hence only a few samples can be used. In contrast, we compute shadows analytically, not stochastically and produce higher quality shadows in equal compute budgets.

The concurrent work of Zhou et al. [ZWRY21] presents a similar approach of using polygon operations for visibility computation. They demonstrate that such exact visibility computations along with LTC can not only be used to obtain analytic and noise free soft-shadows but also noise-free gradients for differentiable rendering. We note that their approach works on individual triangles (similar to our per-triangle approach, Sect. 4.4). In this work, in addition to the per-triangle approach, we also present an efficient algorithm for convex meshes and demonstrate a $2\times$ speedup in rendering.

3. Fast Analytic Soft Shadows

We first give a brief overview of the method of Heitz et al. 2016[HDHN16] to analytically compute shading from polygonal light sources. Note that their method does not take occlusions into account. Given a 3D point x to be shaded, the spatially constant radiance L emitted by a diffuse area light source and the Bidirectional Reflectance Distribution Function (BRDF) ρ , the radiance I at x towards the viewing direction ω_v is defined as an integral over the spherical polygon P of the light source as [Kaj86]:

$$I = L \int_P \rho(\omega_v, \omega_l, x) \cos \theta_l d\omega_l, \quad (1)$$

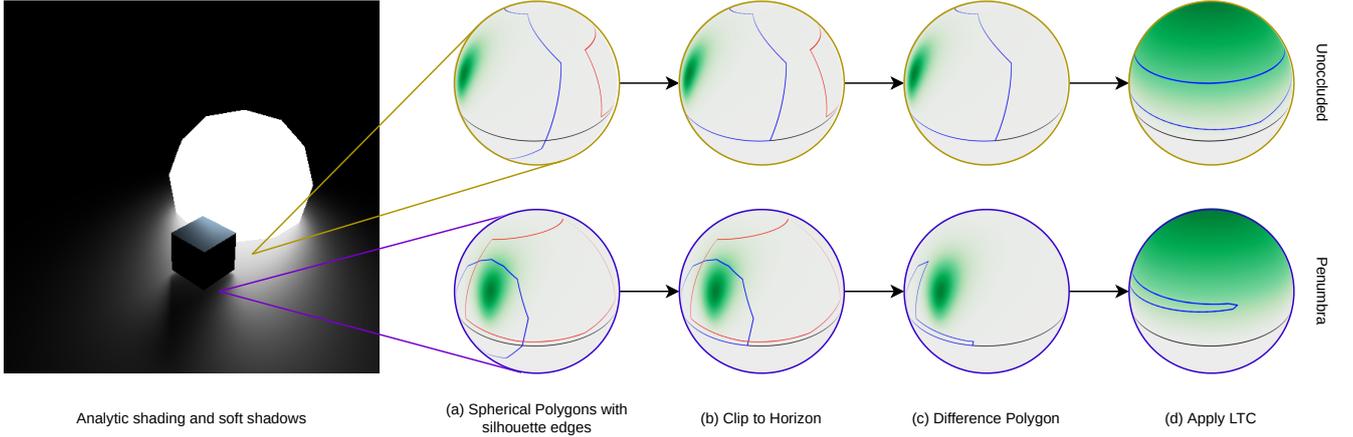


Figure 2: High-level steps of our algorithm to compute shading and soft shadows from area lights. Unit spheres at an unoccluded shading point (yellow) and at a shading point in the penumbra (purple) are visualized with the corresponding spherical polygons of the light source (icosphere, marked as blue polygon) and occluder (cube, marked as red polygon). (a) Obtain spherical polygons from silhouette edges, (b) Clip the spherical polygons to the horizon, (c) Compute difference of the light source and the occluder polygon, (d) Apply Linearly Transformed Cosines (LTC) [HDHN16], clip the result to the horizon and compute shading with analytic formula [BRW89].

where $\cos\theta_l = \omega_l \cdot n$, n being the surface normal and ω_l being outgoing direction. Heitz et al. 2016[HDHN16] showed that given a linear transformation matrix M that transforms the direction vectors of the clamped cosine distribution to the direction vectors of the a microfacet GGX BRDF ρ , the integral in Eq. 1 can be written as:

$$I = L \int_{P_o} \frac{1}{\pi} \cos\theta_o d\omega_o = L E(P_o), \quad (2)$$

where $P_o = M^{-1}P$ is the transformed spherical polygon, $\cos\theta_o = \omega_o \cdot n$ and $\omega_o = \frac{M^{-1}\omega_l}{\|M^{-1}\omega_l\|}$ are the transformed direction vectors. The irradiance integral E can be analytically computed using the closed form expression given by Baum et al.[BRW89]. The matrix M is precomputed and stored for a fixed number of samples of ω_v and α (roughness of the BRDF ρ). Heitz et al. 2016[HDHN16] then use Eq. 2 with the precomputed matrix M to analytically compute shading from diffuse polygonal light sources with no occlusion.

We consider generalizing the light source to an arbitrary 3D shape given by a polygonal mesh. We observe that the radiance I at a point x due to such a mesh can be computed by replacing P in Eq. 1 with the spherical polygon P' which is the projection of the light source object on the unit sphere centered at x . The problem thus reduces to finding the projected area represented as a spherical polygon P' for a given arbitrary light source. Our second observation concerns the impact on I of an arbitrary 3D mesh occluder O , which also has a corresponding projected spherical polygon O' . The light source is occluded in regions where O' overlaps with the light's projection P' . Thus, $\hat{P} = P' - O'$ defines the projected area of the visible portion of the light source. Transforming \hat{P} with M and applying Eq. 2 yields the correct radiance at the shading point x due to the light source.

Directly obtaining the projected area for a non-convex polygonal mesh is non-trivial. We can instead treat individual triangles of

the mesh as its own separate object and project them to the unit sphere. This can be done for both light sources and occluders one polygon at a time to obtain \hat{P} . This produces correct results but is inefficient if larger objects are used (per-triangle method, Sect. 4.4). Estimating the projection of a shape onto the sphere can be efficiently done for convex shapes, achieving more than $2\times$ speedup from the per-triangle method. Therefore, for most of this paper, we assume both light sources and occluders to be convex 3D meshes. As mentioned earlier, simple non-convex meshes can still be represented by a combination of independent convex parts, for example the chair in Fig. 1 is made up of four rectangular cuboids. We discuss the per-triangle method in Sect. 4.4.

3.1. Overview

We now present an overview of our method for convex area light and convex occluder meshes. Note that the steps outlined below can also be used on non-convex meshes by iterating over individual triangles of the mesh (Sect. 4.4). Fig. 2 and Alg. 1 show high-level steps of our algorithm. The key observation is that for a convex 3D light source, the spherical polygon P in Eq. 1 can be obtained using its silhouette edges as viewed from x . The silhouette edges can be efficiently computed using front and back facing polygon detection [IFH*03]. They are then projected to the unit sphere to obtain a spherical polygon of the silhouette (Fig. 2(a) blue polygon, Alg. 1 line 4). We then clip the spherical polygon to the horizon (Fig. 2(b), Alg. 1 line 5). Further, to obtain a polygon P that represents only the visible region of the light source, we first compute silhouette edges and corresponding spherical polygons for all potential occluders, clip them to the horizon (Fig. 2(a) red polygon). The clipped light and occluder polygons are then projected to a plane (Alg 1 line 6, lines 8-11). The set difference between the light occluder polygon represents the visible region of the light source from the point x (Fig. 2(c), Alg 1 line 12). We perform this set difference for each

ALGORITHM 1: Overview of our algorithm.

Input: $\mathcal{L}, \mathcal{B}, x, n, \rho$: List of light sources \mathcal{L} , list of occluders \mathcal{B} , shading point x , normal vector n , BRDF ρ

Output: I : Outgoing radiance (Eq. 1)

```

1  $I \leftarrow \text{Rgb}(0,0)$  // Init. with black col.
2  $p \leftarrow \text{vec3}(0,0,1)$  // LookAt for plane proj.
  /* Consider each light source separately */
3 for  $l$  in  $\mathcal{L}$  do
4    $l' = \text{SphPolySilhouette}(l, x, n)$  // Light sph. poly
5    $l_{clip}' = \text{ClipToHorizon}(l')$ 
6    $l_{xy}' = \text{project}(l_{clip}', p)$  // Project to plane
7    $\mathcal{B}_{bw} = \text{GetBetween}(x, l, \mathcal{B})$  // Occluders b/w  $x$  &  $l$ 
  /* Set diff. b/w light and occluders */
8   for  $b$  in  $\mathcal{B}_{bw}$  do
9      $b' = \text{SphPolySilhouette}(b, x, n)$ 
10     $b_{clip}' = \text{ClipToHorizon}(b')$ 
11     $b_{xy}' = \text{project}(b_{clip}', p)$ 
12     $l_{xy}' = \text{SetDifference}(l_{xy}', b_{xy}')$ 
  /* Apply LTC and integrate */
13   $l' = \text{reproject}(l_{xy}', p)$  // Project back to sphere
14   $l_{lrc} = \text{ApplyLTC}(l', \rho)$ 
15   $l_{lrc} = \text{ClipToHorizon}(l_{lrc})$ 
16   $I = I + \text{AnalyticIntegrate}(l_{lrc}, \rho)$ 

```

occluder polygon. Finally, we reproject the resultant polygon to the unit sphere, apply LTC and clip the result to the horizon, to obtain P_o for analytic evaluation of Eq. 2 (Fig. 2(d), Alg. 1 lines 13-16). These steps are repeated for each light source in the scene (Alg. 1 line 3).

We elaborate on the following functions in the next sections: *project* and *reproject* to project spherical polygons to a plane and reproject planar polygons to the unit sphere (Sect. 3.2), *SphPolySilhouette* to obtain ordered spherical polygons (Sect. 3.3), *ClipToHorizon* to clip spherical polygons to the horizon (Sect. 3.4), *GetBetween* to prune objects that do not occlude the light source (Sect. 3.5), *SetDifference* to obtain a polygon representing the visible area of the light (Sect. 3.6).

3.2. Projecting spherical polygons to a plane

Ordering edges and performing set operations on spherical polygons have, to our knowledge, not been explored in the literature and are non-trivial. Our purpose, however, is served by performing the ordering and set operations on a plane to which we project spherical polygons. The modified polygon is then reprojected back to the unit sphere. In the *project* function, the projection is done using an appropriate look-at camera matrix \mathcal{M} . The look-at vector needs to be carefully chosen to avoid transformed vertices having negative z-coordinates. We then apply perspective division to obtain a correct planar projection and discard the z-coordinate. We visualize this process in Fig. 3(a), for an arbitrary look-at vector p . To reproject polygon vertices to the unit sphere, we first apply the inverse camera matrix \mathcal{M}^{-1} , and then normalize the vectors of each polygon vertex.

Projecting a sphere on to a plane has been previously explored, in the context of map projections or sphere mapping. The closest

mapping operation to ours is the Gnomonic projection [Sny87]. However, for simplicity and ease of implementation, we opt for the look-at transform with perspective camera projection.

ALGORITHM 2: SphPolySilhouette.

Input: \mathcal{G}, x : Convex geometry \mathcal{G} , shading point x

Output: \mathcal{G}_{sil} : Ordered silhouette edges

```

1  $\mathcal{G}' = \text{ComputeSilhouette}(\mathcal{G}, x)$ 
2  $\mathcal{G}' = \mathcal{G}' - x$  // Shift edges to  $x$  as origin
3  $\mathcal{G}' = \text{LocalShadingFrame}(\mathcal{G}', x)$  // Normal vec aligned
  with z-axis
4  $\mathcal{G}' = \text{ToUnitSphere}(\mathcal{G}')$  // Spherical Poly
5  $c = \text{centroid}(\mathcal{G}')$ 
6  $\mathcal{G}_{xy} = \text{project}(\mathcal{G}', c)$ 
  /* Order edges clockwise or anti-clockwise */
7  $\mathcal{G}_{ord} \leftarrow []$ 
8  $\mathcal{G}_{ord}.append(\mathcal{G}_{xy}[0])$ 
9 for  $e$  in  $\mathcal{G}_{ord}$  do
10   $\mathcal{E} = [\{Len(e.v2 - e'.v1), Len(e.v2 - e'.v2)\} \mid \forall e' \in \mathcal{G}_{xy} - \mathcal{G}_{ord}]$ 
11   $next = \min(\mathcal{E})$ 
12   $nextEdge = \text{argmin}(\mathcal{E})$ 
13  if  $next[0] < next[1]$  then
14     $\mathcal{G}_{ord}.append(nextEdge)$ 
15  else if  $next[0] > next[1]$  then
16     $\mathcal{G}_{ord}.append(nextEdge.reverse())$ 
17  $\mathcal{G}_{sil} = \text{reproject}(\mathcal{G}_{ord}, c)$ 

```

3.3. Spherical Polygons of Convex Shapes

We now describe *SphPolySilhouette* function from Alg. 1 which performs two tasks: (1) Compute the spherical polygon of the silhouette in the local shading frame and (2) order the spherical polygon (clockwise or anti-clockwise). The second task is important for the correct functioning of the *ClipToHorizon* algorithm, *SetDifference* operation and analytic LTC integration.

Lines 1-4 of Alg. 2 compute and project the silhouette edges to the unit sphere to get a spherical polygon. They also shift the origin to the shading point x and transform them to the local shading frame. In lines 4-6, we use the centroid of the geometry G as the look-at vector for the *project* function. This projection results in X-Y co-ordinates for all edges, which we order by finding consecutive edges and adding them to a list (lines 7-16). For each edge e , we find another edge which has either of its two vertices closest to the second vertex of e (line 10). Note that since edges may not have vertices in the same direction, we reverse edges based on which vertex follows the second vertex of e (lines 13-16). Finally, we re-project the ordered edges to the unit sphere (line 17).

3.4. Clipping to Horizon

Heitz et al. 2016[HDHN16] assumed a four sided spherical polygon, which on clipping results in either three or five edges. We need a more general clipping algorithm since silhouette edge projections could result in an N sided spherical polygon. We present a general algorithm to clip arbitrary spherical polygons to the horizon *ClipToHorizon* in Alg. 3. This algorithm takes as input a spherical polygon with ordered edges and outputs a horizon clipped spherical

ALGORITHM 3: Clip to Horizon.

Input: \mathcal{S} : Spherical polygon \mathcal{S}
Output: \mathcal{S}_{clip} : Spherical polygon clipped to horizon

```

1  $\mathcal{S}_{clip} \leftarrow []$ 
2  $n \leftarrow \text{vec3}(0,0,1)$  // Normal vec (local shad. fra.)
  /* Iterate over edges of sph. poly */
3 for  $e$  in  $\mathcal{S}$  do
4    $dp_1 = \text{Dot}(e.v1, n)$ 
5    $dp_2 = \text{Dot}(e.v2, n)$ 
6   if  $dp_1 > 0.0$  and  $dp_2 > 0.0$  then
7      $\mathcal{S}_{clip}.append(e)$ 
8   else if  $dp_1 \leq 0.0$  and  $dp_2 \leq 0.0$  then
9     continue
10  else
11     $newEdge \leftarrow \text{Edge}()$ 
12    /* Intersect. of edge & horizon arc */
13     $i = \text{IntersectHorizon}(e)$ 
14    if  $dp_1 < 0.0$  then
15       $newEdge.v1 = i$ 
16       $newEdge.v2 = e.v2$ 
17    else
18       $newEdge.v1 = e.v1$ 
19       $newEdge.v2 = i$ 
20     $\mathcal{S}_{clip}.append(newEdge)$ 
21  $\mathcal{S}_{clip} = \text{CloseVertices}(\mathcal{S}_{clip})$  // Add edges on horizon

```

polygon while preserving the edge order. For each edge that crosses the horizon, we check whether its vertices are above or below the horizon. If the edge is completely above the horizon (line 6), we add it to the final list (line 7) and if it is completely below (line 8), we ignore it (line 9). For an edge intersecting with the horizon (line 10), we find the intersection point using standard algorithms for intersection of spherical arcs (line 12). Note that we find intersection between the edge arc (defined by its two vertices) and the full horizon arc. We then replace one vertex of this edge with the intersection point. Which vertex gets replaced depends on which one of the two vertices are below the horizon (lines 13-18). In the final step on line 20, we close vertex pairs that are consecutively on the horizon, which in effect adds missing edges on the horizon. The algorithm preserves the order of edges and results in a spherical polygon clipped to the horizon.

3.5. Pruning Non-Occluding Objects

In this section, we describe the *GetBetween* function of Alg 1 to prune out objects that cannot occlude the radiance from the light source. This avoids extra processing for objects that have no contribution to the occluded radiance. As shown in Fig. 3(b), we discard objects that are outside the frustum defined by the shading point and the light source. Note that in our implementation, we use frustum culling with spherical bounding boxes and a tetrahedral conic frustum, but any other suitable method could be used for this step.

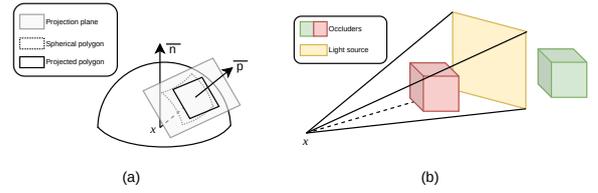


Figure 3: (a) Projection of a spherical polygon (dotted) to a camera plane defined by the look-at vector p . The camera origin is at the shading point x . (b) Pruning objects that do not occlude the light source (green box) using a tetrahedral frustum defined by the shading point x and a plane at the light source (yellow).

3.6. Set operations on spherical polygons

We apply *SetDifference* operation after projecting spherical polygons to a plane, using the *projection* function defined in Sect. 3.2. Note that we use the z-axis as the look-at vector for this projection. The reason for this is that all spherical polygons are clipped to the horizon before projection, and using z-axis as the look-at vector ensures that no vertex has a negative z-coordinate after the look-at camera transform. Choosing any other look-at vector could potentially result in negative z-coordinates for either the light or occluder polygon. Negative z-coordinates result in wrong projections which distort the polygon shape, thus affecting the set operations. After projection, we take the set difference between the light polygon and each occluder polygon, progressively modifying the same light polygon. The result of these operations is a polygon, which represents the region of the light source which is visible.

4. Results, Evaluation and Comparisons

The entire approach presented in this paper is implemented as an integrator plugin in PBRT-v3 [PJH16]. We use the publicly available pre-computed LTC matrices M from [HDHN16]. Our implementation requires that light sources and occluders are marked appropriately, which is done with a flag in the scene description file. This is necessary since it avoids wasteful computations, for example, the walls in a room scene are not occluders for light sources within the room. We use standard frustum culling to get occluders that lie between the shading point and the light source (Alg. 1 line 5). The frustum is a five sided tetrahedral cone, with the apex being the shading point and the bottom face at the light source (Fig. 3 (b)). To compute the set difference operation on polygons, we use the Greiner-Hormann polygon clipping algorithm [GH98]. The main advantage of this algorithm over other algorithms such as [Vat92] is the ease of implementation, fast run-time and out-of-the-box support for set operations. Note that although we use Greiner-Hormann in our implementation, our method is independent of the choice of the polygon clipping algorithm. We plan to release the full source code and scenes used in this paper.

To study the run-time of our method, we first analyze its behaviour with varying number of light and occluder vertices. We then provide a comparison of our results with direct illumination ray tracing and the control variate ratio estimator of Heitz et al.

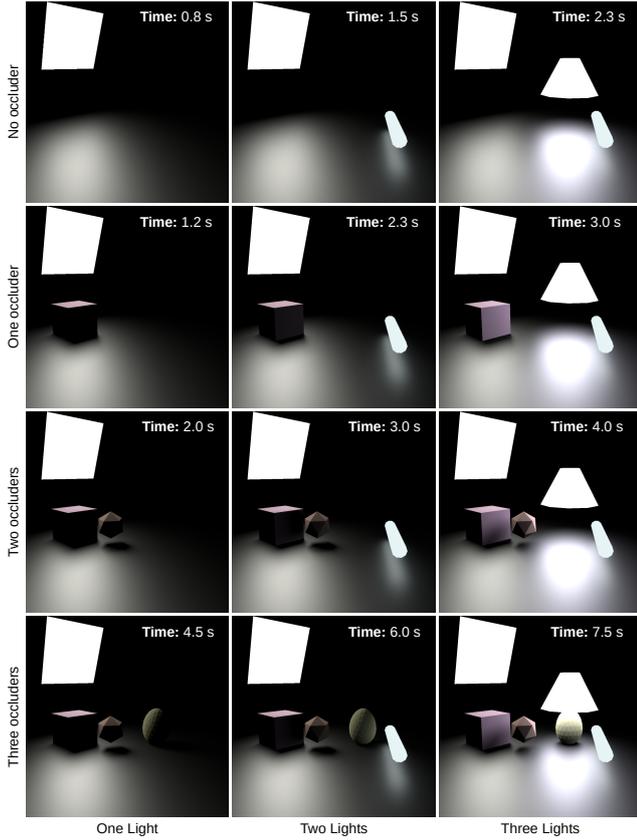
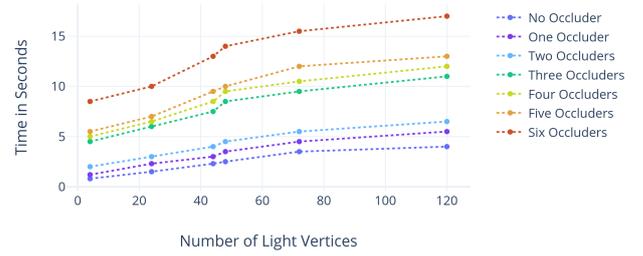


Figure 4: Test scenes with varying number and geometric complexity of light sources and occluders. Each image is rendered at a resolution of 1000×1000 using our method. Run-times are reported at the top right. We use three kinds of light sources: Plane (four vertices), Cylinder and Truncated cone (20 vertices each). We similarly use three kinds of occluders: Cube (eight vertices), Icosphere (16 vertices) and Ellipsoid (154 vertices)

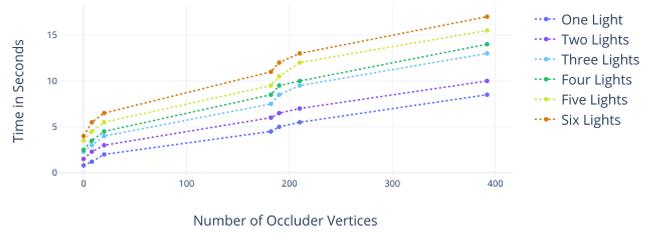
2018[HHM18] on four realistic scenes having varying complexity. In addition, we compare our method to a naïve extension of Heitz et al. 2016[HDHN16] for general 3D meshes to highlight the need for silhouette edge computation. Lastly, we compare to a variation of our method that can handle non-convex 3D meshes. All scenes are rendered on a workstation with a AMD Ryzen 5 CPU having eight cores. PBRT correspondingly utilizes all eight cores for rendering.

4.1. Run-time Analysis

We first demonstrate results and analyze our method’s run-time on simple test scenes, as shown in Fig. 4. We vary the number and geometric complexity of light sources along the columns, starting from one to three (first to third column). We use three 3D meshes with varying complexity for light sources: Plane (four vertices), Cylinder and Truncated cone (20 vertices each). We follow the same variation for occluders along the rows, starting from zero to three (first to fourth row). We use three occluders: Cube (eight vertices), Icosphere (16 vertices) and Ellipsoid (154 vertices). Thus, the bot-



(a) Run-time v/s No. of light vertices



(b) Run-time v/s No. of occluder vertices

Figure 5: Plot of #vertices vs. runtime for light sources (a) and occluders (b), for scenes like in Fig. 4. Our method is roughly linear in the #vertices of light sources and occluders.

tom right scene of Fig. 4 has the maximal complexity (three light sources and three occluders). We render 1000×1000 images for each variation. Note that the shadows become softer on addition of new light sources, and in the penumbra region on the ground and on occluders. Our method is able to produce plausible, realistic and accurate soft-shadows in presence of multiple area lights and occluders.

We further plot the run-times for each of the test scenes against the total number of vertices of the light sources and occluders. Fig. 5(a) shows the plot of number of light source vertices against the run-time and 5(b) shows the plot of number of occluder vertices against the run-time. To obtain more data points for the plot, we duplicate each light source and occluder and place them at a new location in the scene. Both plots show that our method is linear in the number of light source and occluder vertices.

4.2. Comparisons

We now show results of our method and compare with direct illumination ray tracing (RT) and the control variate ratio estimator (Ratio est.) of Heitz et al. 2018[HHM18]. We implement the Heitz et al. 2018 method as an integrator plugin in PBRT. Specifically, we use three integrators which separately compute S_N , U_N and U . We then denoise S_N and U_N separately with a bilateral filter, and combine all three terms. The run-time is a summation of the time taken to render S_N plus the time taken to render U and denoising. Note

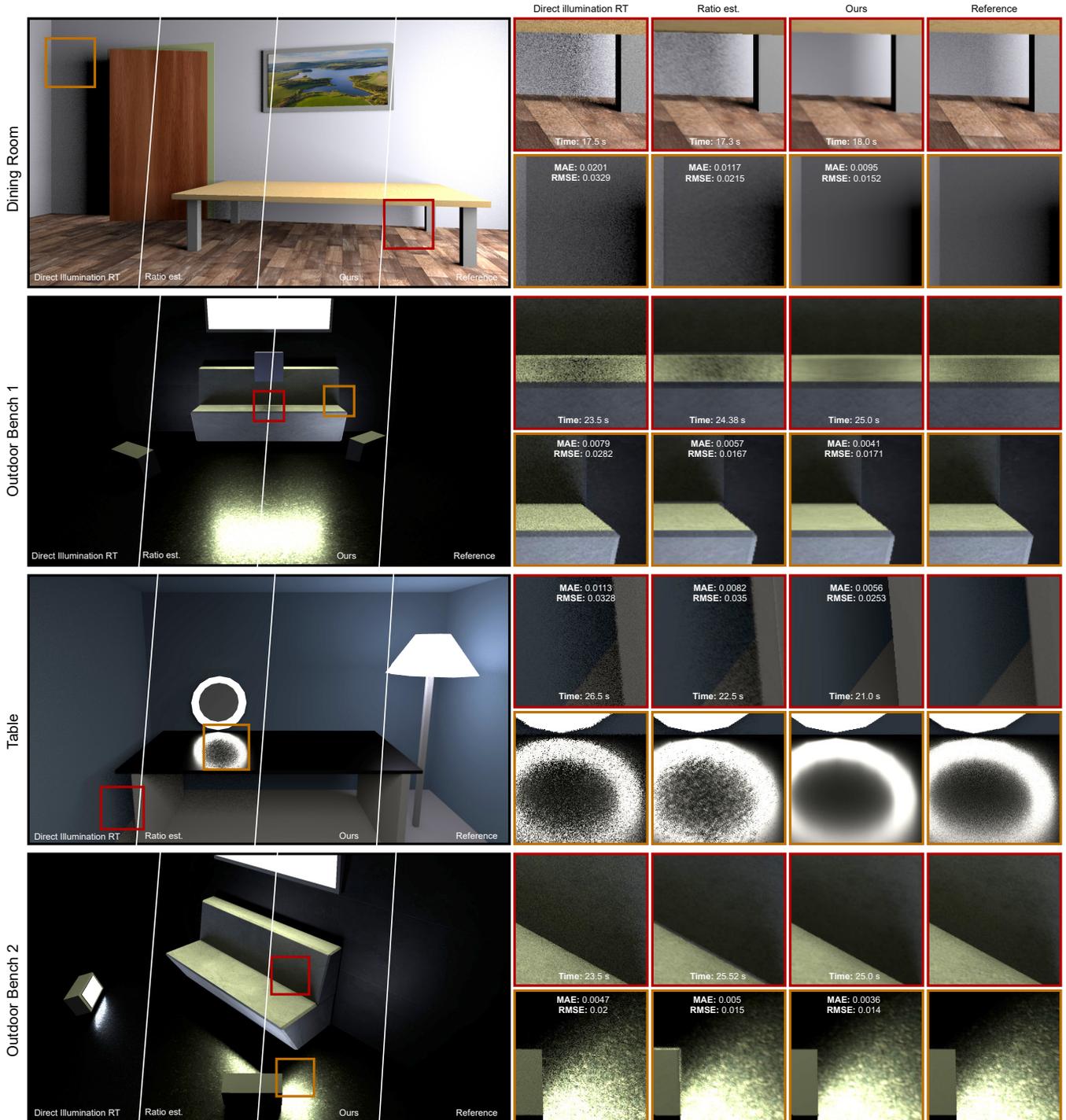


Figure 6: Roughly equal time comparison of our method with direct illumination Ray Tracing (RT) and the ratio estimator (Ratio est.) of Heitz et al. 2018[HHM18]. Run-times and quantitative values of MAE (Mean Absolute Error) and RMSE (Root Mean Squared Error) are shown in insets. Each scene is rendered at a resolution of 1920×1080. Our method outperforms RT and Ratio est. in terms of both quality and quantitative metrics.

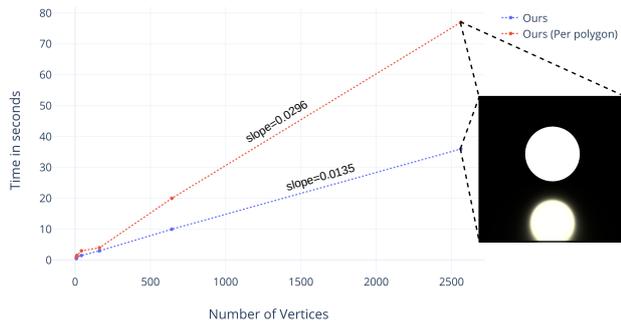


Figure 7: Run-time comparison of our method with a naïve extension of Heitz et al. 2016[HDHN16], for a simple scene having only one icosahedral light source, a specular ground plane and no occluders. We progressively sub-divide the light source geometry to plot run-time.

that we ignore the time taken to render U_N , since it could potentially use the same rays that are used for S_N . We request the reader to refer to their paper for the explanation of these terms.

We use four realistic scenes for this comparison: *Dining Room*, *Living Room*, *Outdoor Bench* and *Table*. We render each scene at a resolution of 1920×1080 . Note that we set the number of samples for RT and Ratio est. such that their rendering time is roughly equal to ours. The results and comparisons are shown in Fig. 6. The *Table* scene has a circular planar light source with a circular occluder just in front, which our method is correctly able to handle. The quantitative metrics along with roughly equal run-times are shown in the insets. Since our method analytically computes soft shadows, our renderings have no noise or blurring artefacts caused due to denoising. Our method thus also achieves lower MAE (Mean absolute error) and RMSE (Root mean squared error) as compared to RT and Ratio est. The attached video shows more renderings of these scenes.

4.3. Naïve extension of Heitz et al. 2016

In this section we highlight the necessity of computing silhouette edges for obtaining spherical polygons. The method proposed by Heitz et al. 2016[HDHN16] can be naïvely extended to arbitrary emissive meshes, by iterating over individual faces of the geometry. Each face can be treated as an independent polygonal light source for which shading can be obtained with their method. Note that in this case, a per triangle check to determine whether its normal vector points towards the shading point is necessary. We also need to sort each polygon’s edges either clock-wise or anti-clockwise for correctness of the LTC integration.

This strategy is however inefficient with a run-time linear in the number of faces of the light source. In comparison, the run-time of our method which obtains one spherical polygon for the light source using silhouette edges is linear in the number of silhouette edges of the light source, which are expected to be small (around \sqrt{e} , where e is the total number of edges [OZ06]). Note that our

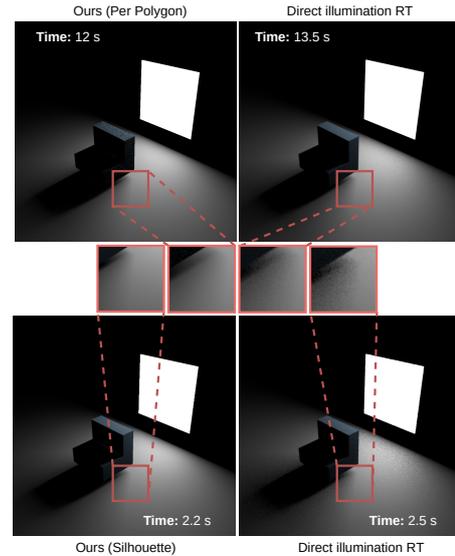


Figure 8: Comparison of the per-triangle method (Sect. 4.4) with direct illumination GT (top) and our method (silhouette edges) with direct illumination GT, run for equal time in both cases. Note that the per-triangle method exhibits instability due to degeneracies caused by vertices at the same location.

method incurs an additional fixed cost of silhouette edge computation for each shading point. We demonstrate this in Fig. 7, with a simple scene having only one icosahedral light source, a specular ground plane and no occluders. We repeatedly subdivide the icosahedral light source to increase the number of vertices and plot the run-time against it. Both methods are linear in the number of vertices, however, the rate of increase of naïve method is more than twice to ours.

4.4. Variation of our method for non-convex meshes

Fig. 9 shows the possible problems that occur when using silhouette edges for non-convex meshes. The silhouette edges are estimated wrongly, resulting in incorrect shading and shadows. To handle simple non-convex meshes, we can decompose them into a set of convex parts, as shown in Fig. 6 for the table in the Dining Room scene and bench in the Outdoor Bench scene. For direct handling of arbitrary non-convex meshes, we can modify our approach in Alg. 1, where the list \mathcal{L} will instead be a list containing all triangles of all light sources and \mathcal{B} will be a list containing all triangles of all occluders. Note that in this case, the function *SphPolySilhouette* will only shift the origin and transform the triangle to the local shading frame and exit. We refer to this approach as the *per-triangle* method.

We compare the per-triangle approach with ours and with equal time direct illumination ray tracing. The result is shown in Fig. 8, for a scene having a chair like object and a four sided polygonal light. Note that for the bottom left rendering of our method, the object is decomposed into two convex shapes (the bottom portion and the top portion). The per-triangle method takes much more time

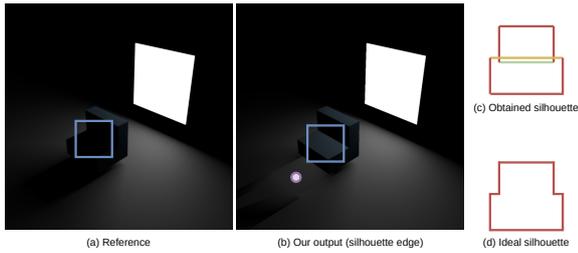


Figure 9: (a) Reference image rendered with direct illumination ray tracing. (b) Output of our method with silhouette edges. (c) shows the silhouette edges at a shading point (marked in purple) and (d) shows the ideal/expected silhouette. Due to the definition of silhouette edges, we get extra edges (marked in green and yellow), which results in incorrect outputs of our algorithm and incorrect LTC integration. Also note the missing shadow in the blue insets. Non-convex meshes result in self-occlusions, which need to be projected and handled separately, since direct silhouette edge projection will result in wrong shading.

than our method, for which equal time direct illumination (shown in the left column) is almost noise free. Furthermore, we would like to note that the per-triangle approach is unstable, mainly due to degeneracies arising from vertices at the same location (for example two triangles sharing a common edge). This ultimately leads to a failure case for the *setDifference* and other geometric operations. Thus, there is a trade-off between using our per-triangle approach with arbitrary meshes, at the cost of high run-time versus our approach for convex meshes at a lower run-time.

5. Discussion and Conclusions

We presented a method to analytically compute shading and soft shadows from light sources with general 3D shapes. We further demonstrated several examples of efficiently rendering high-quality scenes with realistic soft-shadows when light source and occluders are convex meshes. One limitation of our method is its inability to directly handle non-convex shapes, though a simple variation can handle them at much reduced speeds. Another limitation is that our approach has a memory footprint that is not predictable, especially for the *setDifference* and the *clipToHorizon* functions, which poses a challenge for efficient GPU implementation. Finally, since we use LTCs which are itself an approximation to the true BRDF, our result may not exactly match the ground truth rendered with ray-tracing.

An interesting future direction of our work is to obtain proper silhouettes even for non-convex shapes. This would lead to increase in generality of our algorithm at acceptable run-time. Another area of future work is to improve the speed of computation of each step of our algorithm. It is also possible to obtain and exploit reasonable upper bounds on the memory required for each step of our algorithm for an efficient GPU implementation. We would further like to investigate and exploit the continuity of spherical projections between adjacent shading points to improve efficiency. Lastly, we would like to investigate the integration of analytic solutions like ours with methods like [BWP*20], which re-use spatial and temporal information to drastically reduce variance.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback. This work was partially funded by the "Kohli Fellowship" of KCIS.

References

- [Arv95] ARVO, JAMES. "Applications of Irradiance Tensors to the Simulation of Non-Lambertian Phenomena". *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: Association for Computing Machinery, 1995, 335–342. ISBN: 0897917014. DOI: [10.1145/218380.218467](https://doi.org/10.1145/218380.218467).
- [BRW89] BAUM, D. R., RUSHMEIER, H. E., and WINGET, J. M. "Improving Radiosity Solutions through the Use of Analytically Determined Form-Factors". *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '89. New York, NY, USA: Association for Computing Machinery, 1989, 325–334. ISBN: 0897913124. DOI: [10.1145/74333.74367](https://doi.org/10.1145/74333.74367). URL: <https://doi.org/10.1145/74333.74367>.
- [BWP*20] BITTERLI, BENEDIKT, WYMAN, CHRIS, PHARR, MATT, et al. "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting". *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39.4 (July 2020). DOI: [10/gg8xc79](https://doi.org/10/gg8xc79).
- [BXH*18] BELCOUR, LAURENT, XIE, GUOFU, HERY, CHRISTOPHE, et al. "Integrating clipped spherical harmonics expansions". *ACM Transactions on Graphics (Presented at SIGGRAPH)* 37.2 (Mar. 2018). DOI: [10/gd52pf2](https://doi.org/10/gd52pf2).
- [DHB17] DUPUY, JONATHAN, HEITZ, ERIC, and BELCOUR, LAURENT. "A Spherical Cap Preserving Parameterization for Spherical Distributions". *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301. DOI: [10.1145/3072959.3073694](https://doi.org/10.1145/3072959.3073694). URL: <https://doi.org/10.1145/3072959.3073694>.
- [GH98] GREINER, GÜNTHER and HORMANN, KAI. "Efficient clipping of arbitrary polygons". *ACM Transactions on Graphics (TOG)* 17.2 (1998), 71–83. DOI: [10.1145/2897824.2925895](https://doi.org/10.1145/2897824.2925895).
- [HDHN16] HEITZ, ERIC, DUPUY, JONATHAN, HILL, STEPHEN, and NEUBELT, DAVID. "Real-Time Polygonal-Light Shading with Linearly Transformed Cosines". *ACM Trans. Graph.* 35.4 (July 2016). ISSN: 0730-0301. DOI: [10.1145/2897824.2925895](https://doi.org/10.1145/2897824.2925895). URL: <https://doi.org/10.1145/2897824.2925895>.
- [HHM18] HEITZ, ERIC, HILL, STEPHEN, and MCGUIRE, MORGAN. "Combining Analytic Direct Illumination and Stochastic Shadows". *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '18. Montreal, Quebec, Canada: Association for Computing Machinery, 2018. ISBN: 9781450357050. DOI: [10.1145/3190834.3190852](https://doi.org/10.1145/3190834.3190852). URL: <https://doi.org/10.1145/3190834.3190852>.
- [IFH*03] ISENBERG, TOBIAS, FREUDENBERG, BERT, HALPER, NICK, et al. "A Developer's Guide to Silhouette Algorithms for Polygonal Models". *IEEE Comput. Graph. Appl.* 23.4 (July 2003), 28–37. ISSN: 0272-1716. DOI: [10.1109/MCG.2003.1210862](https://doi.org/10.1109/MCG.2003.1210862). URL: <https://doi.org/10.1109/MCG.2003.1210862>.
- [Kaj86] KAJIYA, JAMES T. "The Rendering Equation". *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '86. New York, NY, USA: Association for Computing Machinery, 1986, 143–150. ISBN: 0897911962. DOI: [10.1145/15922.15902](https://doi.org/10.1145/15922.15902). URL: <https://doi.org/10.1145/15922.15902>.
- [Lam60] LAMBERT, I. H. "Photometria sive de mensura et gradibus luminis, colorum et umbrae". (1760). DOI: <https://doi.org/10.3931/e-rara-9488/2>.

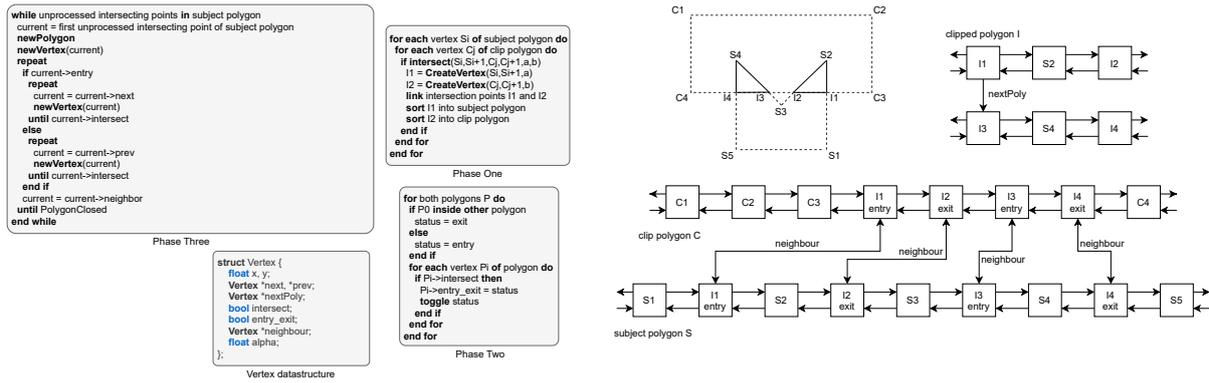


Figure 10: Left: Pseudocode for the three phases of Greiner-Hormann polygon clipping algorithm along with the vertex datastructure. Right: Example of a doubly linked list of the Vertex datastructure, representing the clip and subject polygons. The intersection points (I1, I2..) are generated by phase one, entry and exit is marked by phase two and the final clipped polygon (top right) is generated by phase three. The algorithms and the example are directly adapted from [GH98].

[LDSM16] LECOCQ, PASCAL, DUFAY, ARTHUR, SOURIMANT, GAËL, and MARVIE, JEAN-EUDES. “Accurate Analytic Approximations for Real-Time Specular Area Lighting”. *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '16. Redmond, Washington: Association for Computing Machinery, 2016, 113–120. ISBN: 9781450340434. DOI: [10.1145/2856400.2856403](https://doi.org/10.1145/2856400.2856403).

[MAAG12] MORA, F., AVENEAU, L., APOSTU, O., and GHAZANFARPOUR, D. “Lazy Visibility Evaluation for Exact Soft Shadows”. *Computer Graphics Forum* 31.1 (2012), 132–145. DOI: <https://doi.org/10.1111/j.1467-8659.2011.02089.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.02089.x>.

[OZ06] OLSON, MATT and ZHANG, HAO. “Silhouette Extraction in Hough Space”. *Comput. Graph. Forum* 25 (Sept. 2006), 273–282. DOI: [10.1111/j.1467-8659.2006.00946.x](https://doi.org/10.1111/j.1467-8659.2006.00946.x).

[Pho75] PHONG, BUI TUONG. “Illumination for Computer Generated Pictures”. *Commun. ACM* 18.6 (June 1975), 311–317. ISSN: 0001-0782. DOI: [10.1145/360825.360839](https://doi.org/10.1145/360825.360839). URL: <https://doi.org/10.1145/360825.360839>.

[PJH16] PHARR, MATT, JAKOB, WENZEL, and HUMPHREYS, GREG. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016 5.

[SKS02] SLOAN, PETER-PIKE, KAUTZ, JAN, and SNYDER, JOHN. “Pre-computed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments”. *ACM Trans. Graph.* 21.3 (July 2002), 527–536. ISSN: 0730-0301. DOI: [10.1145/566654.566612](https://doi.org/10.1145/566654.566612). URL: <https://doi.org/10.1145/566654.566612>.

[Sny87] SNYDER, JOHN PARR. *Map projections—A working manual*. Vol. 1395. US Government Printing Office, 1987 4.

[Vat92] VATTI, BALA R. “A Generic Solution to Polygon Clipping”. *Commun. ACM* 35.7 (July 1992), 56–63. ISSN: 0001-0782. DOI: [10.1145/129902.129906](https://doi.org/10.1145/129902.129906). URL: <https://doi.org/10.1145/129902.129906>.

[WCZR20] WU, LIFAN, CAI, GUANGYAN, ZHAO, SHUANG, and RAMAMOORTHY, RAVI. “Analytic Spherical Harmonic Gradients for Real-Time Rendering with Many Polygonal Area Lights”. *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: [10.1145/3386569.3392373](https://doi.org/10.1145/3386569.3392373).

[WMLT07] WALTER, BRUCE, MARSCHNER, STEPHEN R., LI, HONGSONG, and TORRANCE, KENNETH E. “Microfacet Models for Refraction through Rough Surfaces”. *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. Grenoble, France: Eurographics Association, 2007, 195–206. ISBN: 9783905673524 2.

[WR18] WANG, JINGWEN and RAMAMOORTHY, RAVI. “Analytic Spherical Harmonic Coefficients for Polygonal Area Lights”. *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: [10.1145/3197517.3201291](https://doi.org/10.1145/3197517.3201291).

[ZWRY21] ZHOU, YANG, WU, LIFAN, RAMAMOORTHY, RAVI, and YAN, LING-QI. “Vectorization for Fast, Analytic, and Differentiable Visibility”. *ACM Trans. Graph.* 40.3 (2021), 27:1–27:21 2.

A. Appendix

We provide the implementation details of the *setDifference* function using the Greiner-Hormann polygon clipping algorithm for completeness. This algorithm relies on a doubly linked list of vertices to represent polygons. Consecutive nodes in the list point to the next vertex, defining an implicit order, which is used in the algorithm. It considers two polygons, which are denoted as the *subject* and the *clip* polygon (the subject polygon is clipped) and proceeds in three phases. Detailed steps and the vertex datastructure are given in Fig. 10. The first phase is responsible for the determination and storage of edge intersection points of the subject and the clip polygon. The algorithm terminates if no-intersection points are found. Note that, the subject polygon could also be completely inside the clip polygon and vice-versa, which can be easily determined. The second phase is responsible for marking *entry* and *exit* points of intersection vertices of each polygon, which is done by looping over each vertex of a polygon and testing whether it *enters* or *leaves*. Note that this assumes contiguous ordering of polygon vertices, which we perform in the *SphPolySilhouette* function (Alg. 2). In the final phase, the previous computations are used to filter out the clipped polygon. This is done by jumping to the other polygon’s vertex at each *entry* or *exit* vertex, starting from a random vertex of the clip polygon, effectively tracing the clipped polygon edges (Fig. 10, right). For more details, please refer to the original paper [GH98].