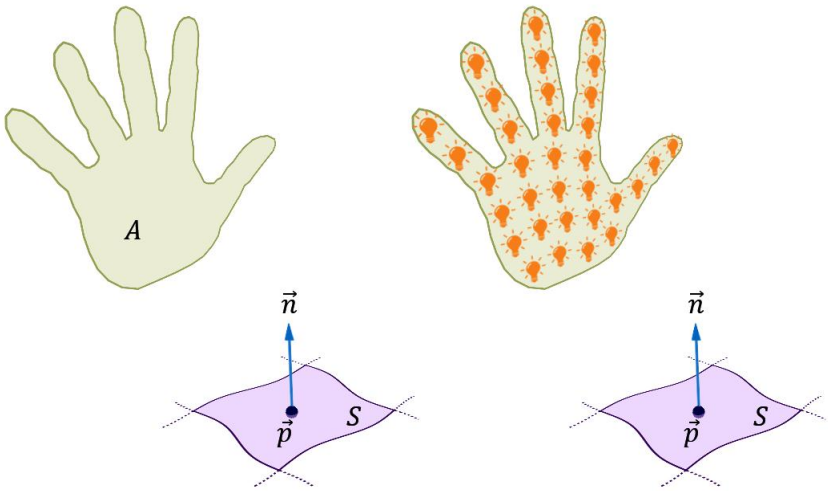
Polygons as lights: algebraic approximation of polygonal area lights.

Alexander Petryaev.

In CG, the realism of illumination from area lights is due to their naturalness - because most of the natural light sources we observe in everyday life are in fact area lights. Yet in real-time computer graphics, area lights are represented mostly in some sort of pre-calculated form because of their computational expensiveness.

The method presented below allows us to easily introduce illumination from arbitrary planar polygonal area lights in real-time shading.

Approximation of diffuse illumination.

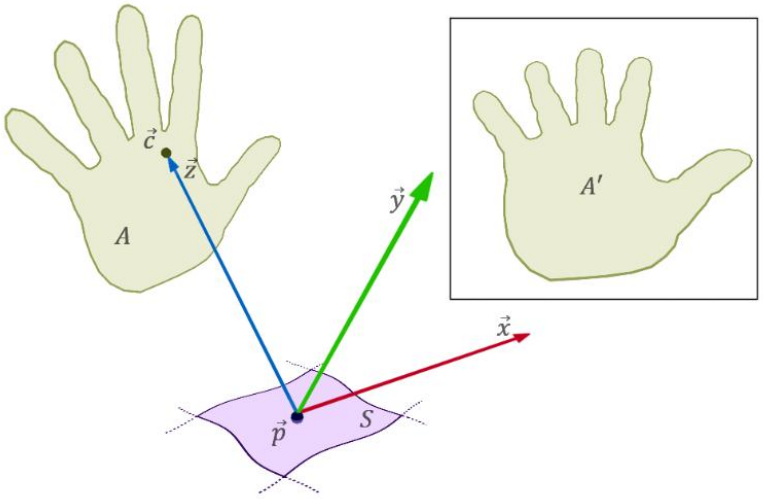


Let us consider a simple (non-self-intersecting) planar polygon A, consisting of m vertices $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_m$, and point $\vec{p}$ on some arbitrary surface S, having a vector of normal $\vec{n}$ in that point.

The naïve approach is to approximate the illumination L_A of polygonal area light, represented by polygon A, with a number of point lights:

$$L_A(\vec{p}) = \sum_{i=1}^{k} \varphi_i(\vec{p})$$

Here k is a number of point lights used in approximation, and $\varphi_i(\vec{p})$ is the illumination at point $\vec{p}$ from a specific light source.



Despite the naivety of this approach, let us look on it from another point of view. First, define a virtual basis with the origin at the point $\vec{p}$, with axis $\vec{z} = \frac{\vec{c}-\vec{p}}{|\vec{c}-\vec{p}|}$, where $\vec{c}$ is the centroid of polygon A.

Then, project polygon A on to plane $\{\vec{p}, \vec{x}, \vec{y}\}$ using perspective transformation:

$$A' = \{\vec{v}'_1, \vec{v}'_2, \dots, \vec{v}'_m\},$$

$$\vec{v}'_i = \left\{\frac{vx'_i}{vz'_i}, \frac{vy'_i}{vz'_i}\right\},$$

$$vx'_i = (\vec{v_i} - \vec{p}) \cdot \vec{x},$$

$$vy'_i = (\vec{v_i} - \vec{p}) \cdot \vec{y},$$

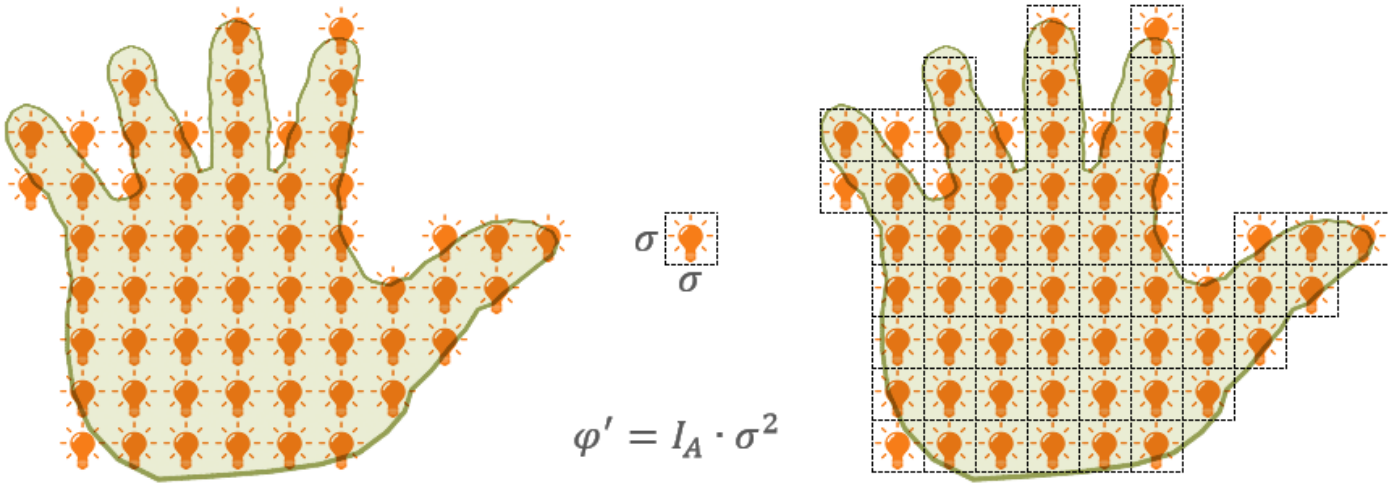
$$vz'_i = (\vec{v_i} - \vec{p}) \cdot \vec{z}.$$

Now, consider applying the similar approximation approach to the transformed polygon A'. Let's try to approximate the illumination from this polygon by a number of light sources, arranged on a regular square grid with step σ. In addition, assume that all light sources will contribute to the illumination equally, and that this contribution is proportional to the area of the grid cell:

$$\varphi'(\vec{p}) = I_A \cdot \sigma^2,$$

$$L_{A'}(\vec{p}) = \sum_{i=1}^{k} \varphi'(\vec{p}) = I_A \cdot \sum_{i=1}^{k} \sigma^2$$

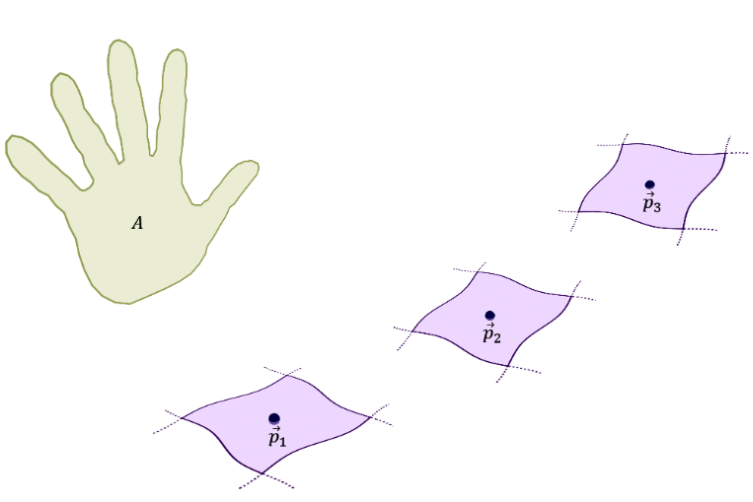
Here I_A is coefficient of proportionality, semantically close to the intensity of the approximated area light source.

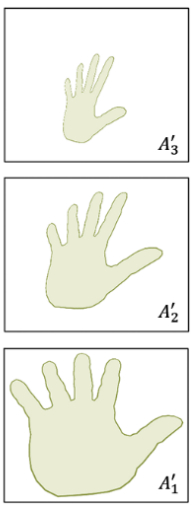


Obviously, with the value of $k \to \infty$ and the value of $\sigma \to 0$ the sum above will converge to the area of polygon A', which is trivially calculated:

$$\lim_{\sigma \to 0} L_{A'}(\vec{p}) = I_A \cdot area(A')$$

$$area(A') = \frac{1}{2}(vx'_1 \cdot vy'_2 - vx'_2 \cdot vy'_1 + \cdots + vx'_{m-1} \cdot vy'_m - vx'_m \cdot vy'_{m-1} + vx'_m \cdot vy'_1 - vx'_1 \cdot vy'_m)$$





To summarize this part: the approximated value of diffuse illumination from the planar polygonal area light at the given point is proportional to the area of the polygon in the perspective projection originated at the given point.

Local occlusion of area light.

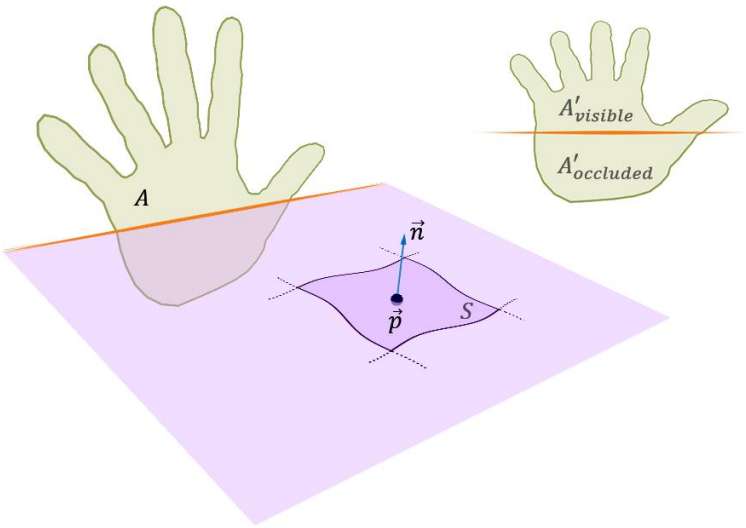
Since the polygon A is planar, the normal of its plane may be easily calculated:

$$\vec{n}_A = \frac{(\vec{v}_2 - \vec{v}_1) \times (\vec{v}_3 - \vec{v}_1)}{|(\vec{v}_2 - \vec{v}_1) \times (\vec{v}_3 - \vec{v}_1)|}$$

The equation above supposes that vertices are in a clockwise order and do not lie on the same line.

With normal $\vec{n}_A$ of polygon A and position of its centroid $\vec{c}$ (though any of polygon's vertices may be used instead of the centroid) it is possible to rule out the illumination of points behind the polygon:

$$\vec{n}_A \cdot (\vec{c} - \vec{p}) > 0 \ \rightarrow point\ \vec{p}\ lays\ behind\ the\ polygon\ A$$

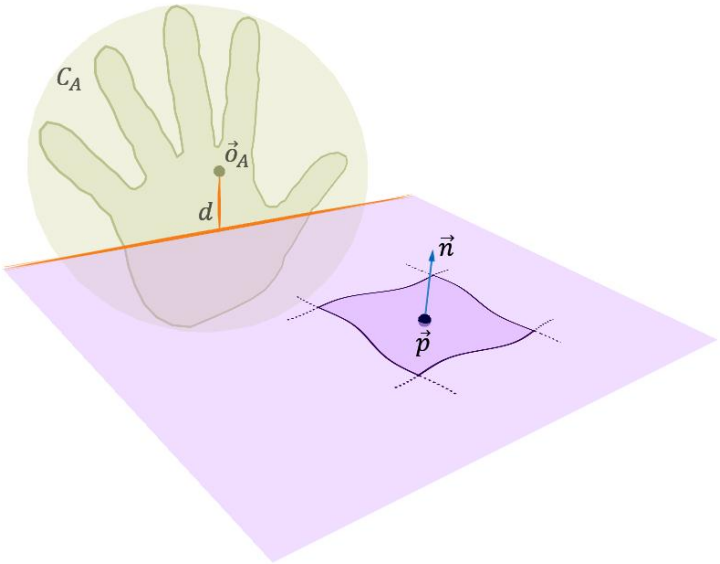


At the same time, the illumination of points ahead of the polygon still depends on surface S, because the surface itself may occlude - partially or fully - the polygon A.

Of course, knowing only the vector of normal $\vec{n}$ of the surface S at the point $\vec{p}$ it is impossible to determine global occlusion of the polygon by the surface. Yet there is enough data to calculate local occlusion, caused by the surface in the immediate vicinity of point $\vec{p}$.

Any point $\vec{a}$ of polygon A is locally occluded by surface S, if the distance from this point to the plane, defined by point $\vec{p}$ and normal $\vec{n}$, is negative:

$$\vec{a} \cdot \vec{n} - \vec{p} \cdot \vec{n} < 0 \ \rightarrow point\ \vec{a}\ is\ locally\ occluded\ by\ surface\ S$$



Though algorithmically such splitting of the polygon is trivial, it is still challenging in shaders because it involves allocation and manipulation of data structures in imperative form, which is unnatural for shading languages, restricted by register-based memory management.

Suppose C_A is a circumscribed circle of polygon A with circumcenter at point $\vec{o}_A$ and with radius $r_{C_A} = |\vec{o}_A - \vec{v}_k|$ (k is index of the farthest vertex). Then it is possible to roughly approximate the value of local occlusion as following:

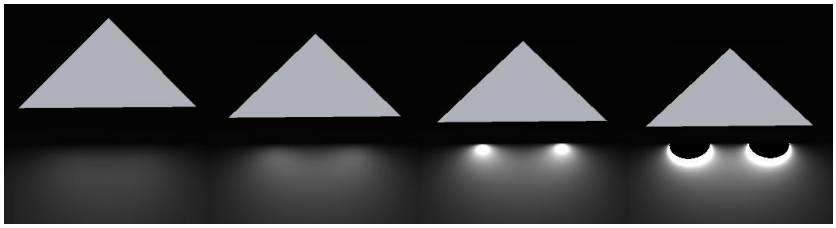
$$loc(\vec{p}, \vec{n}, \vec{o}_A, r_{C_A}) = \begin{cases} d \le -r_{C_A} \rightarrow 0 \\ d \ge r_{C_A} \rightarrow 1 \\ -r_{C_A} < d < r_{C_A} \rightarrow \dfrac{d + r_{C_A}}{2 \cdot r_{C_A}} \end{cases}$$

$$d = \vec{o}_A \cdot \vec{n} - \vec{p} \cdot \vec{n}$$

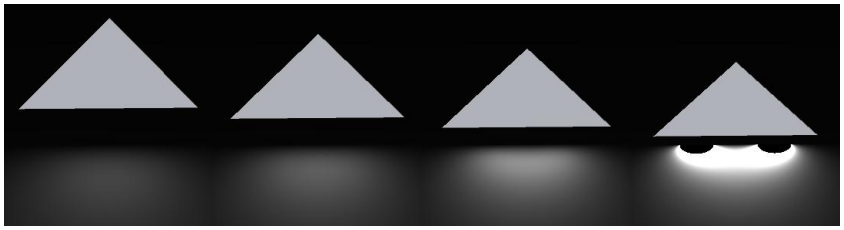
Here value of $loc(\vec{p}, \vec{n}, \vec{o}_A, r_{C_A}) = 0$ means that polygon A is fully occluded, value of $loc(\vec{p}, \vec{n}, \vec{o}_A, r_{C_A}) = 1$ means that polygon A is fully visible and value of $0 < loc(\vec{p}, \vec{n}, \vec{o}_A, r_{C_A}) < 1$ means that polygon A is partially occluded. Note that the approximation above is not good for polygons with an actual area that greatly differs from the area of corresponding circumscribed circles.

Optimal projection basis.

Some of the readers have probably already found a flaw in the described approximation method. Yes, it is about the choosing of the basis for projection. Choosing the centroid of the polygon as a control point for the basis is fine while the polygon is far enough from the illuminated point. At a close distance, some of the transformed vertices may appear behind the projection plane, which makes further calculations invalid.



Slightly better results give choosing the control point using weighting coefficients:



$$\vec{c}(\vec{p}) = \frac{1}{\sum_{i=1}^{m} \frac{1}{|\vec{v}_i - \vec{p}|}} \cdot \sum_{i=1}^{m} \vec{v}_i \cdot \frac{1}{|\vec{v}_i - \vec{p}|}$$

Still the optimal basis for projection must preserve all transformed vertices ahead of the projection plane. There is a problem similar to this one in analytical geometry. It is finding the right irregular pyramid by given directions of its edges. In the right pyramid an apex lies directly above the centroid of the base polygon and therefore the vector of height is collinear to the normal of the base plane of the pyramid.

For the apex take the origin and for directions of edges - subtraction of point $\vec{p}$ from each of polygon vertices $\vec{v}_i$ then the problem is finding of scalars t_i such that:

$$\frac{\vec{C}(t_1, t_2, \dots, t_m)}{|\vec{C}(t_1, t_2, \dots, t_m)|} \cdot \vec{N}(t_1, t_2, \dots, t_m) \equiv -1$$

$$\vec{C}(t_1, t_2, \dots, t_m) = centroid\ of\ polygon\ \{(\vec{v}_1 - \vec{p}) \cdot t_1, (\vec{v}_2 - \vec{p}) \cdot t_2, \dots, (\vec{v}_m - \vec{p}) \cdot t_m\}$$

$$\vec{N}(t_1, t_2, \dots, t_m) = normal\ of\ polygon\ \{(\vec{v}_1 - \vec{p}) \cdot t_1, (\vec{v}_2 - \vec{p}) \cdot t_2, \dots, (\vec{v}_m - \vec{p}) \cdot t_m\}$$

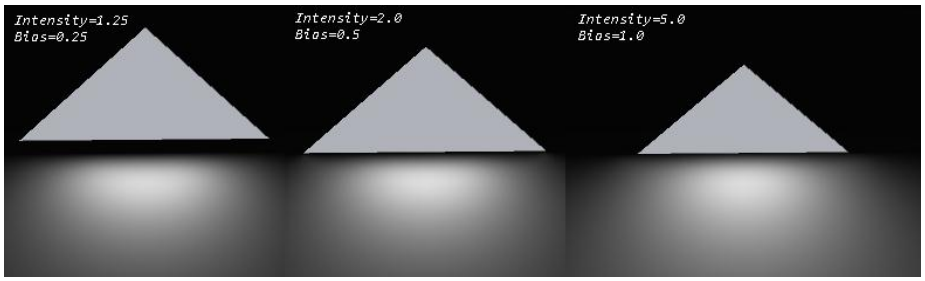
For the case of simplex (polygon A is triangle), the problem is less abstract:

$$\frac{\frac{1}{3}(\vec{e}_1 + \vec{e}_2 \cdot t_2 + \vec{e}_3 \cdot t_3)}{\left|\frac{1}{3}(\vec{e}_1 + \vec{e}_2 \cdot t_2 + \vec{e}_3 \cdot t_3)\right|} \cdot \frac{(\vec{e}_2 \cdot t_2 - \vec{e}_1) \times (\vec{e}_3 \cdot t_3 - \vec{e}_2 \cdot t_2)}{|(\vec{e}_2 \cdot t_2 - \vec{e}_1) \times (\vec{e}_3 \cdot t_3 - \vec{e}_2 \cdot t_2)|} \equiv -1$$

$$\vec{e}_i = (\vec{v}_i - \vec{p})$$

$$t_1 = 1$$

As you can see, even the case of simplex is rather tough. For the time being, I adopt biased projection basis as the solution for the area lights located too close to the illuminated surface:

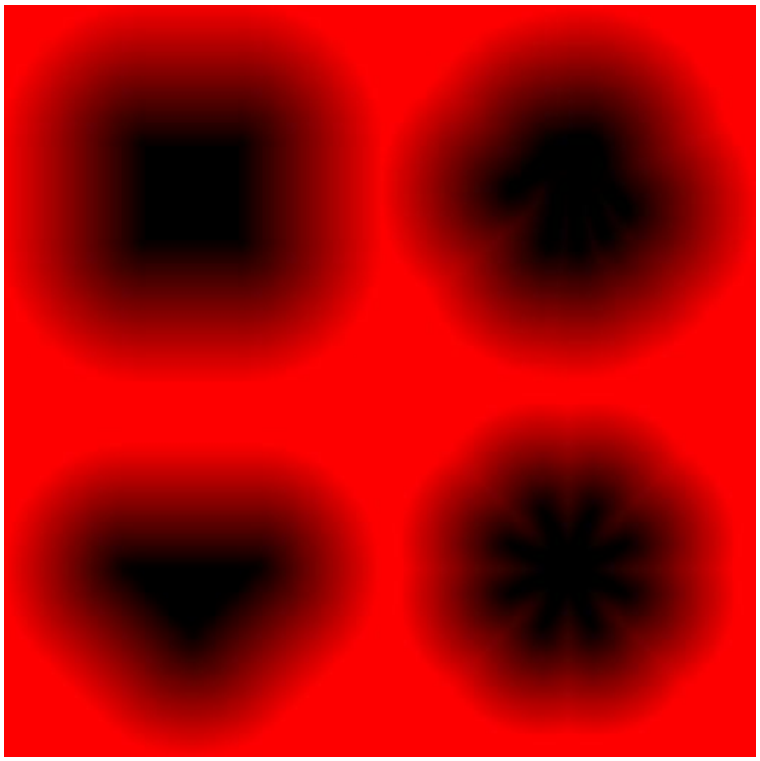


$$\vec{p}_{biased} = \vec{p} - \vec{z} \cdot bias$$
$$vx'_i = (\vec{v_i} - \vec{p}_{biased}) \cdot \vec{x}$$
$$vy'_i = (\vec{v_i} - \vec{p}_{biased}) \cdot \vec{y}$$
$$vz'_i = (\vec{v_i} - \vec{p}_{biased}) \cdot \vec{z}$$

Unfortunately, biased projection affects the shape of the light spot. It is especially noticeable in the case of elongated polygons.

Specular reflections.

The straight approach is to use the vector of reflection in order to find the point on the polygon plane, which is reflected onto the given position of illuminated surface, and then to check the inclusion of that point into the polygon. The latter problem is planar and have a number of effective solutions, for example - the Crossing Number method (1) or the Winding Number method (2).



However, the more effective method is to bake the whole polygon to a texture and use cheap fetch instruction instead of complicated calculations involving a loop and multiple branches. Moreover, by baking a distance to a polygon instead of Boolean value such as "point is inside of a polygon" it is possible to simulate smooth specular reflections on rough surfaces (I deliberately have used here "simulate" since such technique will not be physically corrected).

Baking is based on local coordinate system of the polygon, originated at the circumcenter $\vec{o}_A$ with basis vectors $\{\vec{n}_A, \vec{t}_A, \vec{b}_A\}$, where tangent $\vec{t}_A = \left(n_y^A, n_z^A, -n_x^A\right)$ and bi-tangent $\vec{b}_A = \vec{t}_A \times \vec{n}_A$.

In addition, transformation of a vertex to the texture space involves scaling proportional to the circumradius of the polygon r_{C_A} and to the coefficient Φ that controls how much of area around the polygon must be baked in to texture:

$$v_x^{uv} = \frac{(\vec{v} - \vec{o}_A) \cdot \vec{t}_A}{r_{C_A} \cdot \Phi}, v_y^{uv} = \frac{(\vec{v} - \vec{o}_A) \cdot \vec{b}_A}{r_{C_A} \cdot \Phi}$$

Thus, the baking function for arbitrary point $\vec{p}^{uv}$ in texture space look like:

$$bake(A,\vec{p}^{uv}) = min\begin{cases} \min\limits_{i=1\ldots m-1} dist(\vec{v}_i{}^{uv}, \vec{v}_{i+1}{}^{uv}, \vec{p}^{uv}) \\ dist(\vec{v}_m{}^{uv}, \vec{v}_1{}^{uv}, \vec{p}^{uv}) \\ (1 - crn(A,\vec{p}^{uv})\ mod\ 2)\cdot \Phi \end{cases}$$

Function $crn(A,\vec{p}^{uv})$ returns the crossing number:

$$crn(A,\vec{p}^{uv}) = \begin{cases} odd \ \rightarrow \vec{p}^{uv}\ is\ inside\ of\ A \\ even \ \rightarrow \vec{p}^{uv}\ is\ outside\ of\ A \end{cases} \quad (1,2)$$

Function $dist(\vec{v}_i{}^{uv}, \vec{v}_{i+1}{}^{uv}, \vec{p}^{uv})$ returns unsigned distance from point $\vec{p}^{uv}$ to the line segment defined by points $\vec{v}_i{}^{uv}$ and $\vec{v}_{i+1}{}^{uv}$ (3):

$$dist(\vec{a},\vec{b},\vec{p}) = \left|\overrightarrow{pa} - \overrightarrow{ba}\cdot h\right|$$

$$h = clamp\left(\frac{\overrightarrow{pa}\cdot\overrightarrow{ba}}{\overrightarrow{ba}\cdot\overrightarrow{ba}}, 0,1\right)$$

$$\overrightarrow{pa} = \vec{p} - \vec{a}\ ;\ \overrightarrow{ba} = \vec{b} - \vec{a}$$

Having the texture containing baked polygon S_A, the intensity of specular reflection at arbitrary point $\vec{p}$ in world space may be approximated as:

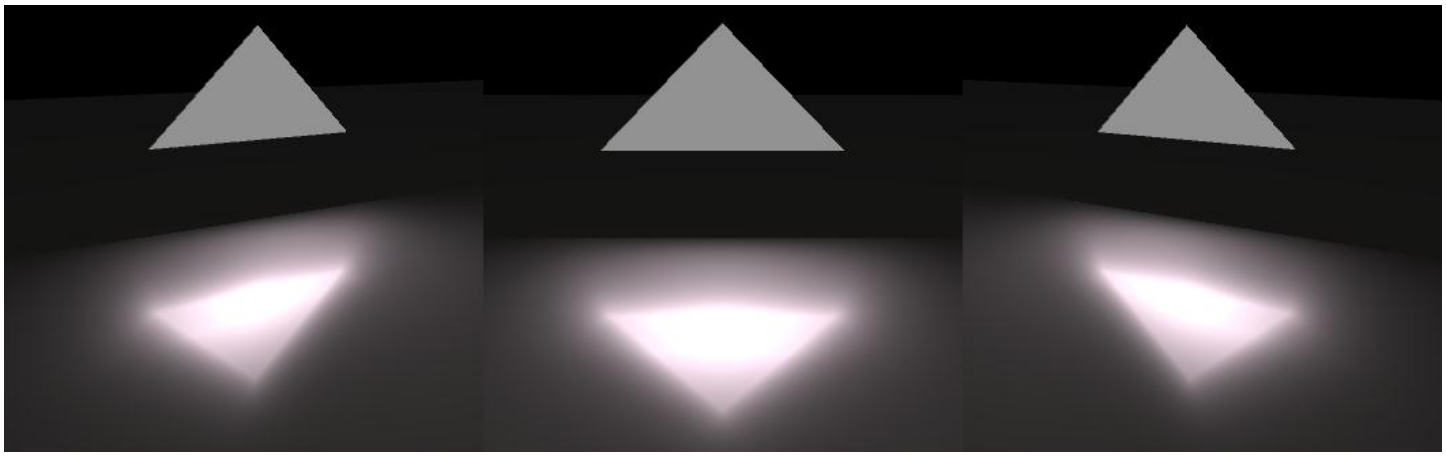
$$k_{spec} = \frac{1}{(tex2D(S_A,\vec{p}^{uv}) + 1)^{k_s}}$$

Where

$$p_x^{uv} = \frac{(\vec{p}_{int} - \vec{o}_A)\cdot\vec{t}_A}{r_{C_A}\cdot\Phi}\ , p_y^{uv} = \frac{(\vec{p}_{int} - \vec{o}_A)\cdot\vec{b}_A}{r_{C_A}\cdot\Phi}$$

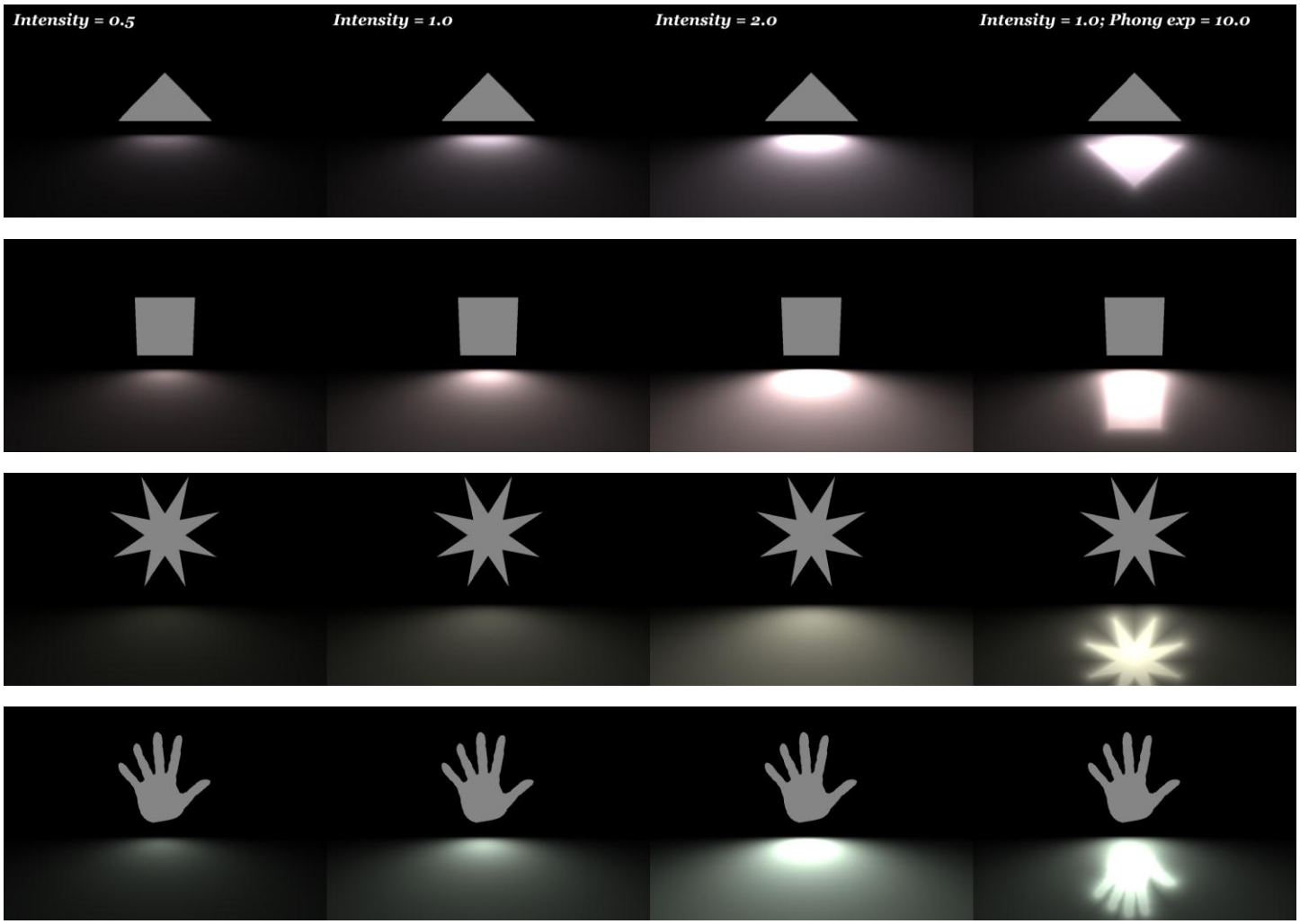
Point $\vec{p}_{int}$ is intersection point of the plane of the polygon A and the vector of reflection of view direction at point $\vec{p}$.

The number k_s is a user-chosen value that controls the apparent smoothness of the surface.

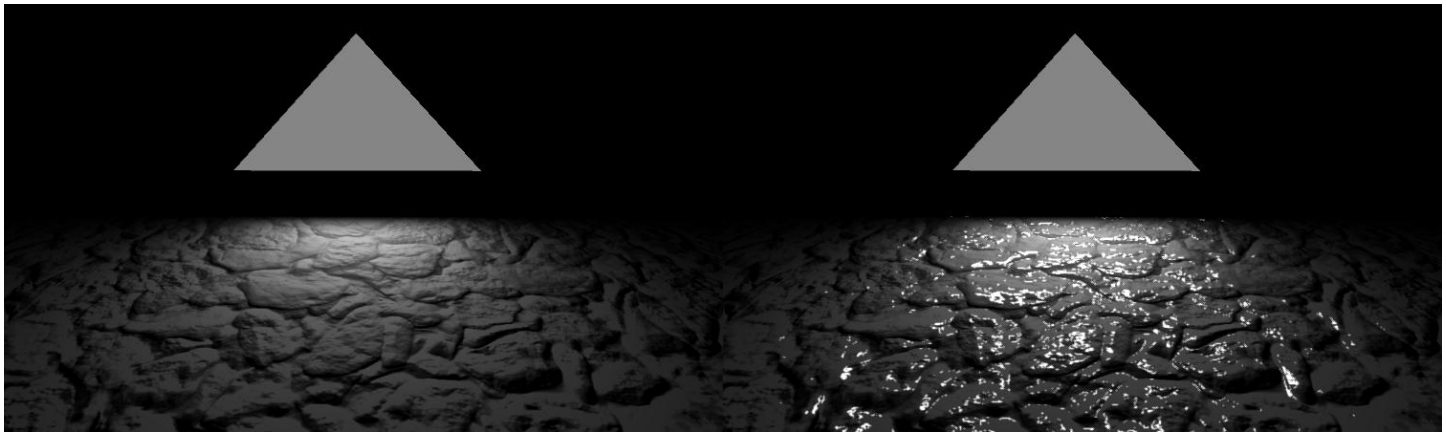


Results.

The visual results are quite promising.

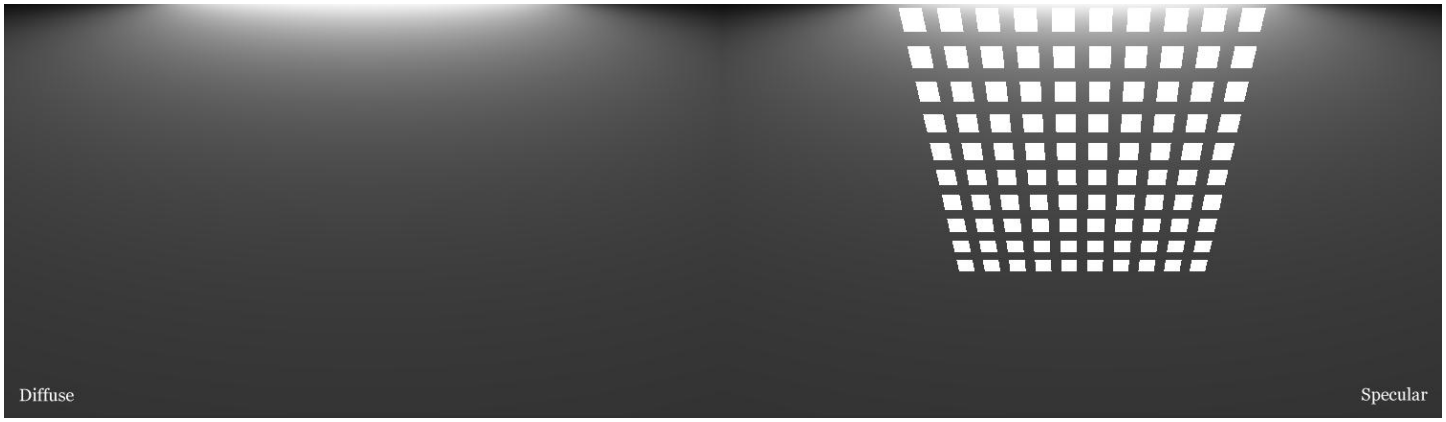


Special bonus: local occlusion works perfectly with normal map.

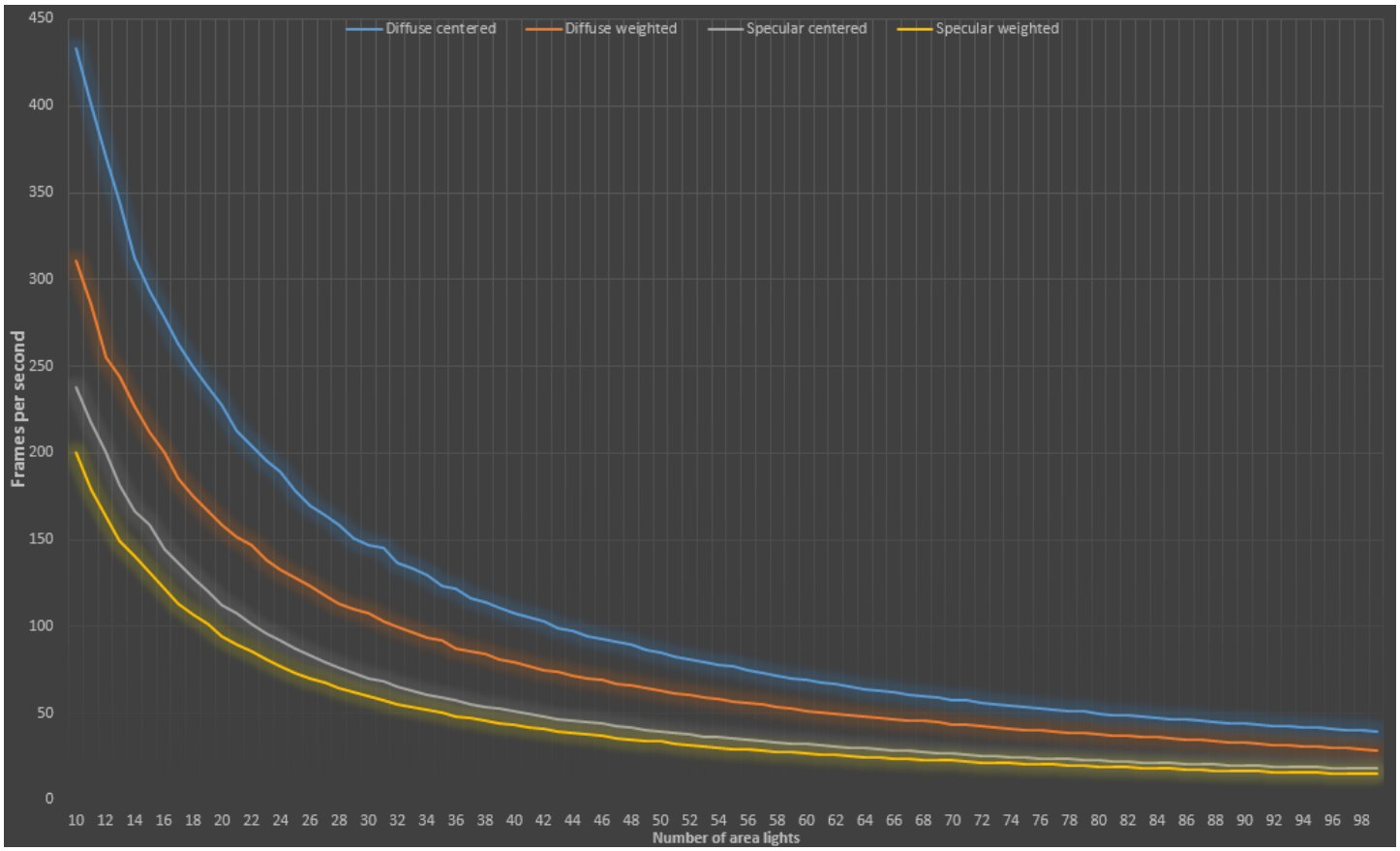


Performance results.

To measure performance I have filled the whole screen (1440x900) with fragments illuminated by a different number of quadrilateral area lights. I have used GeForce GTX 960 as the hardware to run the performance tests.

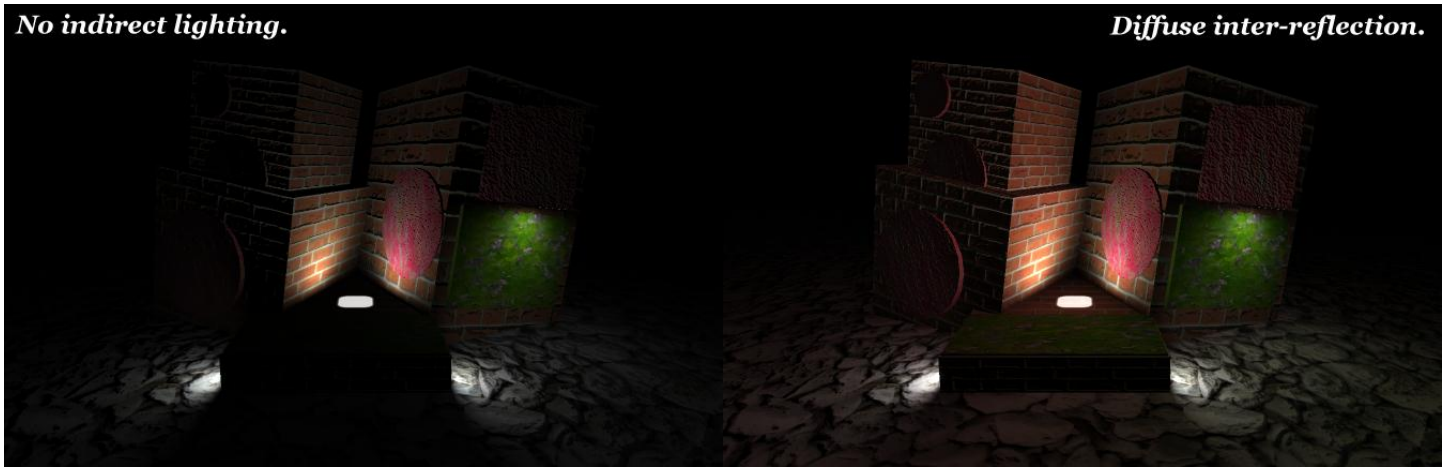


I have tested both centered and weighted projections.



Future research.

Now I am in research of the applicability of technique for real-time global illumination, specifically - for diffuse inter-reflections. It is too early to make conclusions but I already have some visible results.



I foresee there are still a number of ways to improve the presented technique:

- Find a better solution for optimal projection basis.
- Develop a proper algorithm for the calculation of a precise local occlusion.
- Research substitution of constant coefficient of proportionality I_A by functions.

External resources.

Link to GitHub repository (Unity3D project): https://github.com/bad3p/PAL

References.

1. Wm. Randolph Franklin, "PNPOLY - Point Inclusion in Polygon Test" (2000).
2. Inclusion of a point in a polygon; http://geomalgorithms.com/a03-_inclusion.html
3. Inigo Quilez; “Modeling with distance functions”; http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm