Ray Traced Copperplate Engravings

Alexander M. Morais (alexmm179@gmail.com)
Mississippi State University

Abstract

The illustration technique of applying ink to paper with engraved copperplates can be simulated in a ray tracer, as introduced by Leister in [1]. It uses a three phase method to generate hatching lines and silhouette edges. A method to generate this same effect in a single pass of my ray tracer and with stochastic super-sampling([2]), is introduced here. This paper describes how to hatch scene objects and draw silhouette edges on them. I ultimately reveal that computer generated copperplate engravings can be drawn with all of the effects presented in Leister's paper, but in a single pass and anti-aliased.

1. Introduction

Copperplate engraving illustrations came about around the Middle Ages or earlier. While it started as a convenient way to generate technical illustrations, it evolved into more artistic employments over time. The resulting images produced by this process are always bi-level since a point on the medium (or pixel in the final image) either has ink or does not. [3] My method even allows for hatchings of colors other than black.

Leister presented a method in [1] that generates hatches by mapping parallel planes through the scene objects and calculating kappa, the intensity at the point of intersection. There can be more than one hatch set and kappa is a gray value based on several parameters to sum all hatches together for a visually appealing result without artifacts. Leister's method would determine the color at a point using the usual ray tracing method, compute the hatching lines, and perform image-processing techniques in the post-process to interpret the intensity values, draw the edge lines, and combine the two images. No one else has attempted to ray trace computer generated engravings, but Ostromoukhov presented a method in [4] that uses purely 2D techniques to produce engraved textures without the use of 3D scene descriptions.

In this method, kappa is calculated exactly as Leister did, but then I threshold its value to black or white to avoid a post-processing operation. So, if kappa is >= 0.5, then the intensity is 1 (white). Otherwise, the intensity at the point is 0 (black).

Silhouette edges are drawn in the method of this paper for spheres and polygons using two very different methods; both different from Leister's method. This method also avoids the edge-drawing post-process. Whether the point of intersection is incident on a silhouette edge of a sphere is determined by the angle between the normal of the surface and the ray that intersected with the object. If the angle is within the threshold, described by the edge width, then black is returned as the color. Otherwise, the color is calculated as usual. For polygon silhouettes an edge-polygon, which is larger copy of the original polygon by a factor(edge width) of the face normals, determines when to draw the edge. If an object with edges is not hit by the current ray, then it's edge-polygon is checked for intersection.

Copperplate engravings have the potential to produce severe aliasing artifacts. Anti-aliasing of these images can greatly improve there quality. The method presented here uses stochastic super-sampling, as described in [2]. The method works by sub-dividing each pixel in the output image to an NxN grid. It, then, randomly jitters the center position of each sub-pixel and casts rays from those points. The final color of the pixel is produced by averaging all of the colors returned by the sub-pixels. By increasing the sampling size of the image one can achieve greater results but at the cost of computation.

2. Ray Tracing Simulated Copperplate Engravings.

A synopsis of the process is as follows for each ray that is cast into the scene:

1. Use the standard ray tracing process to produce the RGB intensities at the point.
2. Either of the two:

    1. Calculate Kappa
    2. Threshold to hatch-color or white

    Or

    1. Return the edge-color

The characteristic hatching lines produced by this technique are a visualization of parallel planes. Objects in this approach can only have one or two sets of planes. This limits the possibility of messy artifact-ridden hatchings. An almost infinite array of interesting patterns can still be produced using only 2, though.

The silhouette edges improve the perceptability of the image in that overlapping objects will be separated by a stimulus that humans . This helps not to confuse the viewer.

2.1. Hatching lines

The ray tracer needs several parameters to generate the hatching lines; the plane coefficients (*e_1* – *e_4*), the distance between planes (*a*), two weights for balancing multiple hatchings (*c*, *d*), and two line thickness factors (*h_f*, *h_b*). All of these parameters are specified in the material properties of the scene description. Below, is an example of the syntax for specifying hatching lines in my ray tracer. Leister did not provide any example inputs.

```
begin_object
 ....
 begin_material
  ....
  begin_hatch
       hb 1.0
       hf 1.0
       begin_lamina
            a 1.0
            e -15 7 15 25
            c 1.0
            d 1.0
       end_lamina
       # OPTIONAL 2nd plane set
       begin_lamina
            a 0.6
            e 3 -1 3 30
            c 1.0
            d 1.0
       end_lamina
   end_hatch
 end_material
end_object
```

A plane is defined by the plane equation (1). It is a function of the *e*-coefficients and the 3D point of intersection on the surface.

$$f = e_1 t_x + e_2 t_y + e_3 t_z + e_4 \qquad (1)$$

This plane has to be defined for each hatching set and copies of this plane are transposed, parallel to the original. A word of advice when defining planes, the fourth component, e_4, should be relatively high so that the hatching pattern extends far enough to cover the object. Experiments showed that lower numbers between 0 and 10 were not sufficient for larger objects.

The *a* parameter is an offset between the parallel planes. In other words, this value will effect the frequency of hatching lines. For higher resolution images with many samples per pixel, this can be low, but it is generally more desirable to use higher numbers to prevent aliasing patterns, especially on curved objects when the dot product of the normal of the surface and the ray direction is beyond 45°. In a simpler implementation of copperplates, this can serve as the only parameter for generating hatching lines and it would produce circular patterns [3]. Since my method strives for more variance in the pattern, it uses the more sophisticated approach. Nonetheless, the value, *l*, is utilized in the final intensity calculation and must be calculated. It is as follows:

$$l = (\lambda \, mod \, a_0)/a_0 \tag{2}$$

Lambda, in this implementation, is the depth value at the point of intersection and a_0 is the the first(possibly only) hatch-set's offset, *a*.

Since the plane is defined by the *e*-coefficients and the point of intersection, and *a* is the offset until the next plane, we can determine the distance this point is from a hatch plane, *b*, effectively producing an intensity value. The system of equations is as follows and it is very similar to *l*:

$$b = \begin{Bmatrix} (f \, mod \, a)/a & if \, f \geqslant 0 \\ a - (f \, mod \, a)/a & if \, f < 0 \end{Bmatrix} \tag{3}$$

This value could be thresholded and output to the image if there are no other hatches. This would effectively produce hatching lines, but without any control over line thickness.

The line thickness parameters *hf* and *hb* are an obvious addition to this model. Just as artisans had control over the thickness of the lines being carved into the copperplates, so shall we. It uses two scaling factors to, first subtract a portion of the ray traced color at a point by h_b, and then scale that difference by h_f, as seen in (4). These factors are floating point numbers and can be any value, but some values do not produce satisfactory results. Unfortunately, Leister did not discuss recommended values for these parameters and finding decent ones requires some experimentation. The equation for line thickness is as follows:

$$(h - h_b) h_f \tag{4}$$

To combine multiple hatches, one cannot simply add the two hatch sets' intensities together, as it will produce unwanted artifacts. Instead, a union operation, below (5), must be performed to combine them. [1]

$$x_1 \cup x_2 = \begin{Bmatrix} max(x_1, x_2) \\ min(x_1, -x_2) \end{Bmatrix} \tag{5}$$

This union operation is used at two points in the calculation of the final intensity, kappa.

The calculation of the final intensity, *K* (*Kappa*), combines all of the previous equations. It introduces the c and d, scaling parameters. These are useful when there are two hatch-sets if the one hatch set has positive weights and the other has negative weights. One can cancel the other instead of them both being drawn in black. It overlays two hatchings by union-ing the two together. Leister did not explicitly describe the order of operations, but this is the formula with proper parentheses the way I implemented it.

$$K = ((c_0 \cdot b_0) \cup (l \cdot c_0)) \cup (c_1 \cdot b_1 \cup (l \cdot c_0)) + \sum (d_i \cdot b_i + l \cdot d_0 + (h - h_b) \cdot h_f) \tag{6}$$

The first half before the sum is a union between two unions related to the *c* scaling factors and normalized distance, *b*, of each lamina. Take note that the middle union symbol combines the first and last unions. If there is only one lamina set, then only one of the two “inner” unions need to be calculated and not the outer union. The sum scales the line thickness using the scaling factor, *d*, and the normalized distance from a hatch plane. This adds the two hatches together (but not so simply to produce those artifacts mentioned before). Finally, by thresholding the value to either white or the hatch color, one gets the final image result. No post-processing necessary.

2.2. Silhouette Edges

Leister's method for producing silhouette edges uses a post-processing technique that utilizes the z-depth of the 2D intensity image. Remember, he is not thresholding *kappa*, but, rather, is using the intensity image to make the hatching and edge images which are combined into one afterward. In my method, all edge drawing is done within the single process of the ray tracer. Spheres and polygons in this implementation are not treated the same with respect to silhouette edges.

2.2.1 Sphere Silhouettes

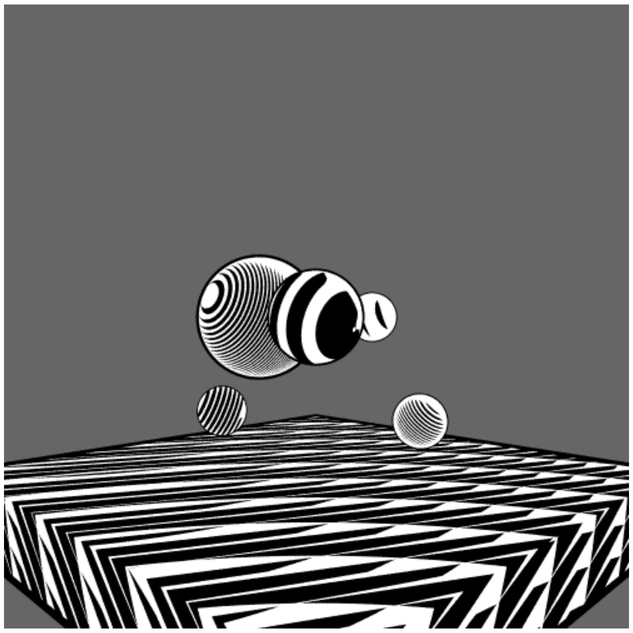
Spheres use geometry inherent in the scene to determine the the viewing angle is within some threshold indicative of the “edge.” A defining characteristic of an edge on a sphere is that it is at the point where the normal is perpendicular to the viewing direction(or ray direction). In other words, the dot product of the sphere, surface normal and the ray direction will provide the angle between those two vectors. One cannot simply draw an edge only when the cos(theta) = 0. By allowing the angle to be within 0 and the edge-width it is thicker because you are drawing all pixels within some angle black.

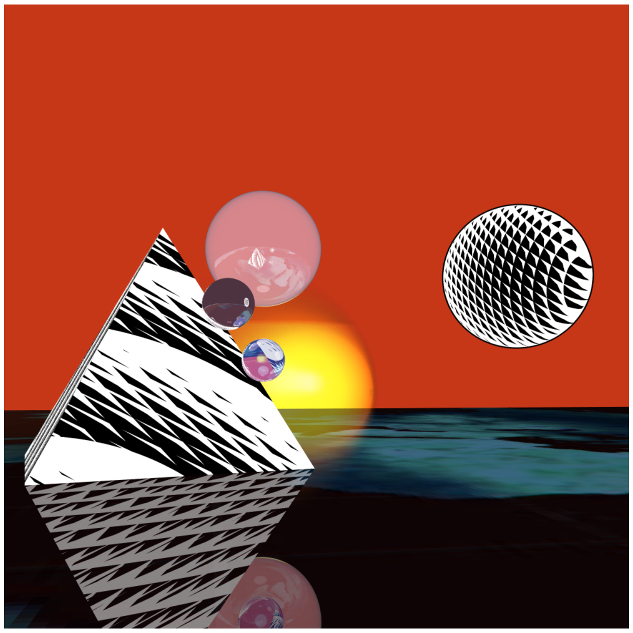
2.2.2 Polygon Silhouettes

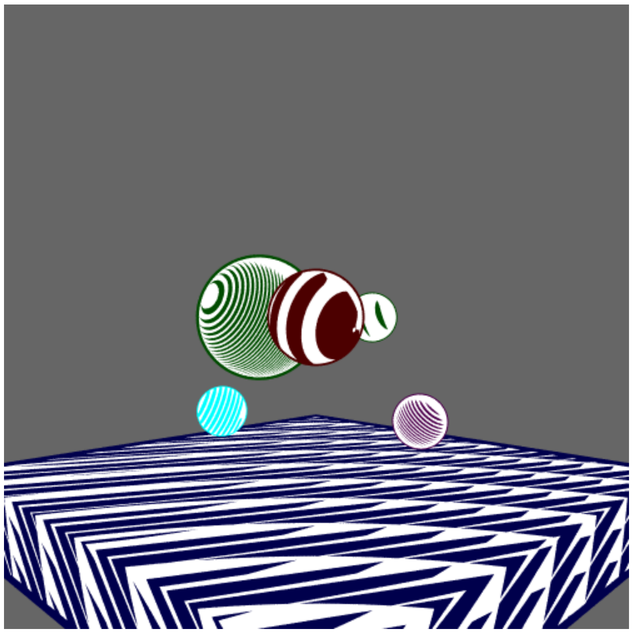
In order to get edges on polygons, a special-case trick is used. When an edge_width parameter is specified in the process of reading the scene, a copy of the original polygon is created, scaled with respect to the edge width parameter and stored for use during ray tracing. When rendering the scene the edge polygon is tested for intersection in the event that its parent polygon is not intersected by the “current ray”. While the drawback of this is that it requires extra intersection tests, it is very simple and effective. I could have implemented spheres the same way, but I avoided the unnecessary intersections by using geometry.

4. Sample Images

The following are images produced by my method. I did not convert the second image into a bi-level image to produce the first image. What is displayed is the fact that hatchings can either be uniform or any variety of user-specified colors. One can even mix non-copperplate engraved images, as shown.







5. Conclusion

Simulated copperplate engraving images can be produced in a single render pass with bi-level hatches and silhouette edges without the use of post-processes, contrary to Leister's method in [1]. Also, the ability to add color to the scene can increase perception with contrast while still looking artistic. This is an advantage over Leister's method which only used intensity values and could never apply specific color characteristics to certain objects when stepping over the 2D intensity image. The same goes for drawing edges of a particular color. One other benefit of using one render pass is being able to super-sample the image. If post-processes were used, hatching would not be super-sampled and would require additional post-processing.

5. References

[1] W. Leister, “Computer Generated Copper Plates”, *Computer Graphics Forum*, vol 13, no. 1, Mar. 1994, pp. 69-77

[2] R. L. Cook, “Stochastic Sampling in Computer Graphics,” *ACM Transactions on Graphics*, vol. 5, no. 1, Jan. 1986, pp. 51-72.

[3] T. Strothotte and S. Schlechtweg, *Non-Photorealistic Computer Graphics Modeling, Rendering, and Animation*, Morgan Kaufman Publ., San Francisco CA., 2002.

[4] V. Ostromoukhov, “Digital facial engraving,” *Proceedings: ACM SIGGRAPH CONF COMPUTER GRAPHICS*, 1999, pp. 417-424.