Light and Shadow Volume Construction for Sprite Sheet Objects using Outline Mesh Generation

Kurtis Slagle
Department of Electrical Engineering and Computer Science
University of Kansas
Lawrence, KS
kslagle@ku.edu

***Abstract*—This paper covers an approach to generating outlines of objects in sprite sheets for simulating 2D lighting based on the object's general shape, using either shadow volume generation or light volume generation. A trivial method for storing this information on disk to be loaded at program runtime is also covered.**

Keywords—sprite sheet, shadow volume, light volume, ray casting, edge detection, corner detection, occluder, attenuation

I. Introduction

One of many important aspects of real-time computer graphics are shadows. In a flat, 2-dimensional space, lights and shadows are planar in nature. Though algorithms in 3-dimensional space are complex, this need not be the case in 2D space. A mesh forming the outline of an object can be used to compute dynamic shadows for use in real-time graphics applications. 2D lighting is less computationally intensive than the analogous 3D case but is often ignored or not emphasized due to the typical method of generating 2D graphics.

2D graphics typically uses sprite sheets. Sprite sheets do not inherently contain any outline or mesh information. They are typically used for animation, but the only special information that is known is that the sprite sheet is rendered as a quad on screen. This quad does not provide necessary information to produce shadows.

Using existing computer vision algorithms for feature detection and extraction, a mesh representing the outline of a sprite can be generated (multiple meshes in the case of an animated sprite) and used in conjunction with a lighting model to generate realistic, dynamic 2D lighting directly affected by the sprite.

The general approach follows the following steps (in no particular order):

A. *Edge Detection:* The edges of the object within the sprite must be determined

B. *Outline Generation:* With the edges known, the edges must be connected to form a closed, connected outline of the object.

C. *Outline Mesh Generation:* The outline can be stored as a mesh to be later used in a lighting model.

D. *Light/Shadow Volume Construction:* Depending on the desired effects, the generated outline mesh can be used to generate either light volumes or shadow volumes.

Because the generated mesh will be used to simulate lighting and shadows, the lighting model is discussed prior to the outline generation to better convey the purpose of extracting the outline.

II. Lighting Model

2D lights simulate a planar environment, and thus the lights and shadows will exist only in a flat plane. Both point light sources and directional light sources need to be able to be modeled. Point light sources may assume a point with or without a given size, but the light is emitted radially in all directions. Directional light sources emit light in the same direction for all objects.

In 2D, an "occluder" is "any object in a scene that casts shadows"[2]. A **shadow volume**, then, is the volumetric region of space subsumed by the shadow cast by an occluding object [1][2]. In contrast, a **light volume** is the volumetric region of space illuminated by the light source. These two regions of space are separated by the occluding object. The shadow volume can either extend infinitely far or only encapsulate a determinate amount of space. For 3-dimensional graphics, the viewpoint of the light source is used to generate the silhouette of the object [1].

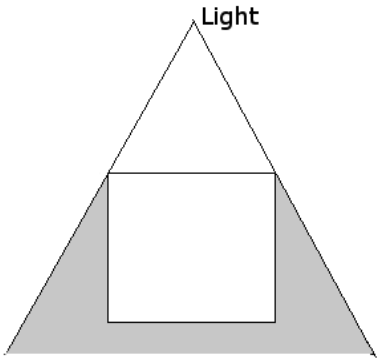


Figure 1 – point light with occluder – Shadow Volume

In Figure 1, the point light source is located at the origin of the two outgoing rays. The occluder is generating a silhouette, shown as the darkened region behind the object. The area of space subsumed by this silhouette will be referred to as the **shadow volume**.

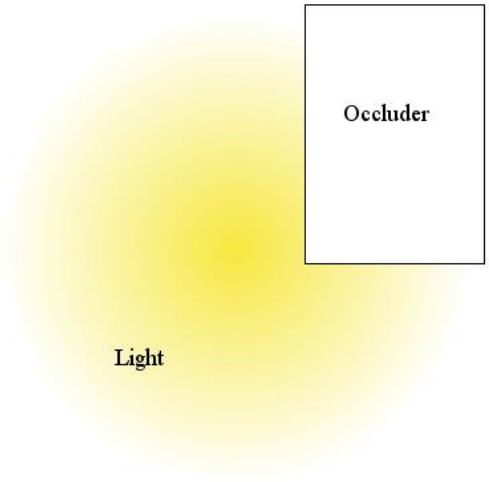


Figure 2 – point light with occluder – Light Volume

In Figure 2, the point light source is emitting a light, signified by the yellow color radiating from the center. The occluding object is blocking any light from protruding beyond the edge closest to the light source. The region of space that remains illuminated by the remaining light is the **light volume**. It can be observed that the illuminating effect of the light extends into a limited region. This distance from the center of the light to the end of this region is the **attenuation** distance.

To generate these volumes, the outlining mesh of the occluder must be known. Once this is known, the region of space contained within both the light volume and shadow volume can be determined.

Determining these regions of space requires casting light rays through the scene to determine which objects in the scene will be blocking a light source. The shadow volumes for occluders that are behind other occluders need not be considered. If light volumes are to be generated, it must be determined if an unobstructed path exists from a light source to the edge of any object. Due to the planar nature of 2D lighting, ray casting can be leveraged to very quickly compute the intersection of a light ray and the outline mesh of an object.

III. Ray Casting

The proposed methods utilize a modified ray casting method designed for use in a homogenous coordinate system.

A. *2D World Space Coordinate System*

All points in the 2D world-space coordinate system are represented as $\binom{x}{y}$ where x is the x-coordinate and y is the y-coordinate of a given point in space.

B. *Homogenous Coordinates*

For 2 given points in 2D space, $\binom{x_1}{y_1}$ and $\binom{x_2}{y_2}$, an edge is formed connecting the two points. The points can be modeled in homogenous coordinates as $(x, y, 1)^T$. For 2 given points $\boldsymbol{x_1} = (x_1, y_1, 1)^T$ and $\boldsymbol{x_2} = (x_2, y_2, 1)^T$, the line through the two points is $l = \boldsymbol{x_1} \times \boldsymbol{x_2}$.

C. *Raycasting Implementation*

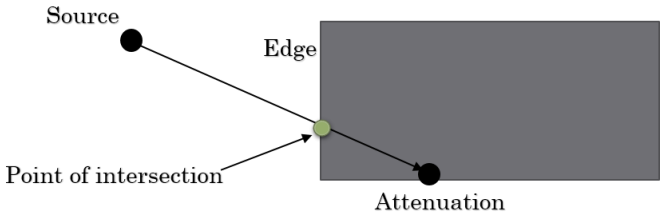


Figure 3 – casting a ray from a source to an attenuation point, intersecting an edge

In Figure 3, a ray is cast from a source (the light source) toward an attenuation point (the point in the ray's direction that is at the edge of the lights attenuation) and this ray intersects an edge. The exact point of this intersection must be determined.

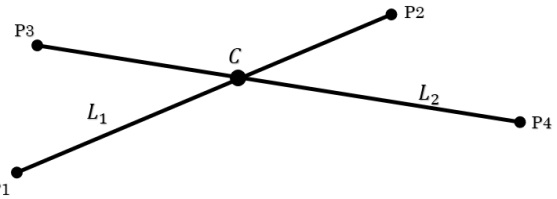


Figure 4 – intersection of rays

In Figure 4, the intersection of two rays is seen. If the lines *L1* and *L2* are represented in homogenous coordinates, then the intersection is:

$$C = L_1 \times L_2$$

This assumes a line extending infinitely in both directions. The meshes for objects are made up of finite rays, so a test must be used to determine if the ray intersects between the two endpoints of the edge.

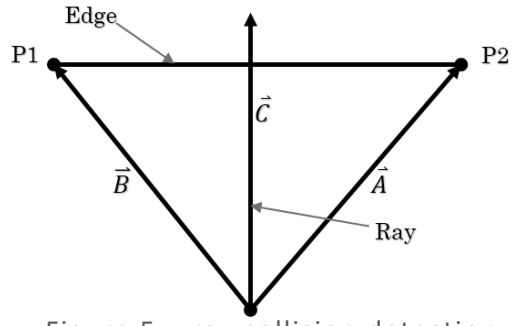


Figure 5 – ray collision detection

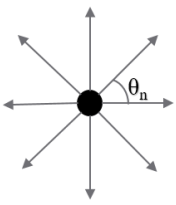
In Figure 5, a ray is cast through an edge. If this ray is to intersect this edge, it must pass in-between its endpoints. Let $\vec{A}$ and $\vec{B}$ be rays from the source to the two endpoints, and $\vec{C}$ be the aforementioned ray. It can be determined if $\vec{C}$is between the rays $\vec{A}$ and $\vec{B}$ by testing if $(\vec{A}\times\vec{B}) \cdot (\vec{A}\times\vec{C}) \geq 0$ and $(\vec{C}\times\vec{B}) \cdot (\vec{C}\times\vec{A}) \geq 0$. If either fail, the ray does not intersect the finite edge and no intersection used. This method of ray casting is all that is necessary for the proposed methods. A demonstration of this with many polygons is demonstrated later.

IV. Illumination Methods

There are two main methods of illumination proposed: Additive Illumination and Subtractive Illumination.

A. *Additive Illumination*

Light illuminates the world and shadows are a result of the obstruction of this illumination. Additive Illumination simulates this by adding light to the scene, in effect brightening it and altering its color. This uses the earlier discussed light volume by determining what regions of space should be illuminated by a light source. Simulating this produces the most visually accurate lighting, but can be very expensive for complex scenes.



Light volumes must be generated. The sum of these light volumes will be referred to as a **light map**. To construct these light volumes, rays are cast in all directions outward from the light source. $\theta_n = 2\pi/n$ – the "resolution" of this method – is the angle between consecutive rays.

Every pair of consecutive rays in the scene form a triangular region. These collectively form the light map. This map can be used to render the light and properly illuminate the scene.

B. *Subtractive Illumination*

Subtractive illumination simulates removing light from regions of a scene that should be subsumed in shadow rather than adding light. This is less expensive, and requires no ray casting, but results in less accurate lighting and can generate artifacts.

For this method, the entire light is rendered into a framebuffer. Then, for every object in the scene, shadow volumes are extruded from every edge of the object in the direction away from the light source. These shadow volumes are then rendered into the framebuffer in black, thus overwriting the light in the buffer and preventing it from illuminating the area.

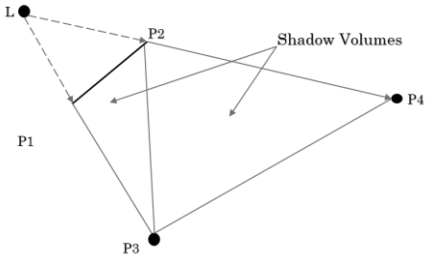


Figure 8 – Shadow Volumes extruded

Figure 8 demonstrates the extrusion of shadow volumes behind an edge. The cumulative extruded shadow volumes is the **shadow map**, which is subsequently rendered on the scene to remove the light from the framebuffer.

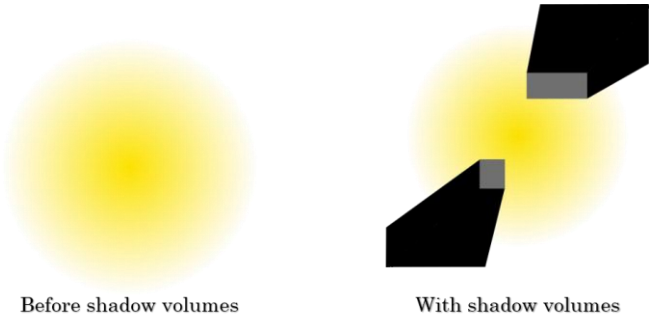


Figure 9 – Rendered shadow volumes

In Figure 9, the left gradient is the light source immediately after being rendered into the framebuffer. The right image is the result after rendering shadow volumes over the light in the framebuffer.

The purpose is to remove light, not the scene entirely, so this framebuffer cannot be directly rendered on top of the scene. It must be blended with the scene so that the parts of the

framebuffer that retain the light's original color will illuminate the scene but regions in black do nothing. There are many ways to do this. One method is to use the framebuffer as a render texture and draw it as a quad covering the entire scene using a shader (if using OpenGL) to blend the texture with the existing scene.

V. Outline Mesh Generation

The object in the sprite sheet must be outlined. To do this, an edge detection method will be used. This method requires control over the rendering system, so the sprite can be rendered in all black on a white background to force an extremely high contrast render as a binary image which can then be contoured.

Figure 10 – Sprite before processing

Figure 11 – Sprite rendered in all black

At this point, the contour of the object must be extracted. OpenCV's `cv::findContours` can generate a contour of this and the positions of every point along the contour can be retrieved from the output. This outline is the sprite's outline mesh.

Figure 11 – OpenCV render of the sprite's contour

VI. Experimental Results

For testing, the following libraries were used:

- SFML[3] – For OpenGL rendering and implementation of the lighting model
    - SFML binaries can be downloaded from sfml-dev.org
- OpenCV[4] – For sprite contouring and contour extraction

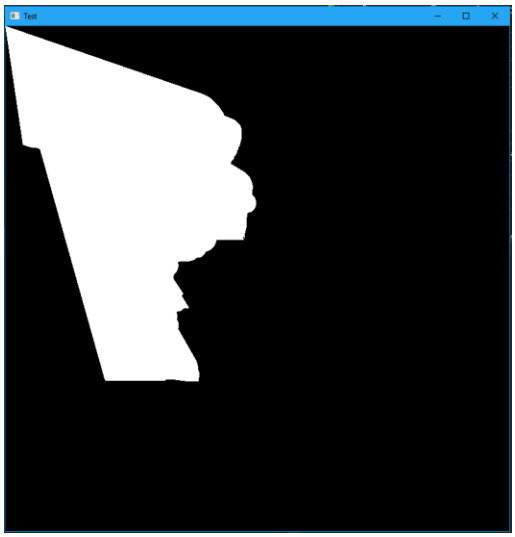


Figure 12 – Rendering the contour as a light volume

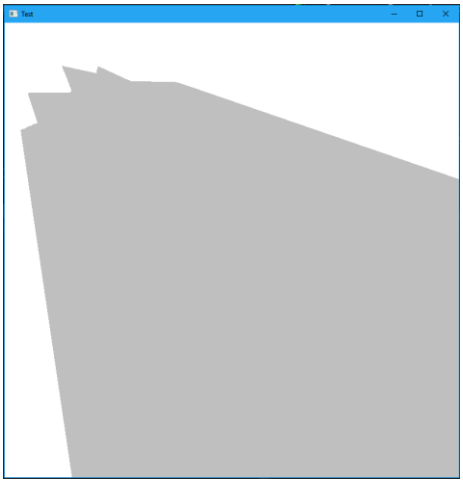


Figure 13 – The object's shadow volume, extruded from the extracted contour

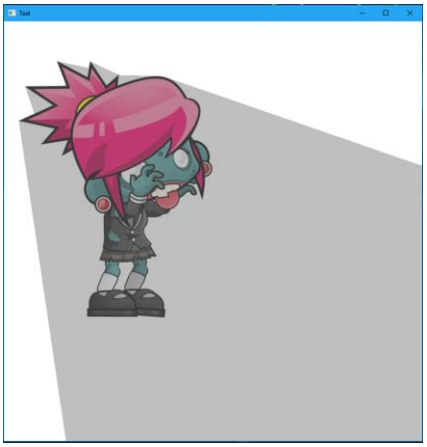


Figure 14 – Rendering shadow volume with original sprite

Figure 12 shows a render of the sprite's partial light volumes. Figure 13 shows a render of the sprite's shadow volume. The shadow volume in fig. 13 was generated assuming a light at (0, 0) and the shadow volumes were extruded back behind every edge along the contour. Figure 14 shows a render of the original sprite with its shadow volume rendered as well. The shadow volume was made partially transparent.

Source code for the examples can be accessed at:

REFERENCES

[1] Frank Crow. Shadows Algorithms for Computers Graphics. Computer Graphics, Vol. 11, No.3, Proceedings of SIGGRAPH 1977, July 1977

[2] Yen, Hun (2002-12-03). "The Theory of Stencil Shadow Volumes". GameDev.net. Retrieved 2017-10-12.

[3] Gomillam Laurent. 2017. *SFML (Simple Fast Multimedia Library)*, v.2.4.2 [Online] Available: https://github.com/SFML/SFML, Accessed on: Dec. 12, 2017.

[4] OpenCV team. 2017. *OpenCV*, v. 3.3.1 [Online] Available: https://opencv.org/, Accessed on: Dec. 13, 2017.