



Creating Shadow Volumes on Mainstream Graphics Hardware

by David Bookout

The information contained in this document is provided for informational purposes only and represents the current view of Intel Corporation Intel on the date of publication. Intel makes no commitment to update the information contained in this document, and Intel reserves the right to make changes at any time, without notice.

DISCLAIMER. THIS DOCUMENT AND ALL INFORMATION CONTAINED HEREIN IS PROVIDED AS IS. INTEL MAKES NO REPRESENTATIONS OF ANY KIND WITH RESPECT TO PRODUCTS REFERENCED HEREIN, WHETHER SUCH PRODUCTS ARE THOSE OF INTEL OR THIRD PARTIES. INTEL EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES, IMPLIED OR EXPRESS, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, AND ANY WARRANTY ARISING OUT OF THE INFORMATION CONTAINED HEREIN, INCLUDING WITHOUT LIMITATION, ANY PRODUCTS, SPECIFICATIONS, OR OTHER MATERIALS REFERENCED HEREIN. INTEL DOES NOT WARRANT THAT THIS DOCUMENT OR THE INFORMATION CONTAINED HEREIN IS FREE FROM ERRORS, OR THAT ANY PRODUCTS OR OTHER TECHNOLOGY DEVELOPED IN CONFORMANCE WITH THIS DOCUMENT WILL PERFORM IN THE INTENDED MANNER, OR WILL BE FREE FROM INFRINGEMENT OF THIRD PARTY PROPRIETARY RIGHTS, AND INTEL DISCLAIMS ALL LIABILITY THEREFOR.

INTEL DOES NOT WARRANT THAT ANY PRODUCT REFERENCED HEREIN OR ANY PRODUCT OR TECHNOLOGY DEVELOPED IN RELIANCE UPON THIS DOCUMENT, IN WHOLE OR IN PART, WILL BE SUFFICIENT, ACCURATE, RELIABLE, COMPLETE, FREE FROM DEFECTS OR SAFE FOR ITS INTENDED PURPOSE, AND HEREBY DISCLAIMS ALL LIABILITIES THEREFOR. ANY PERSON MAKING, USING OR SELLING SUCH PRODUCT OR TECHNOLOGY DOES SO AT HIS OR HER OWN RISK.

Licenses may be required. Intel and others may have patents or pending patent applications, trademarks, copyrights or other intellectual proprietary rights covering subject matter contained or described in this document. No license, express, implied, by estoppels or otherwise, to any intellectual property rights of Intel or any other party is granted herein. It is your responsibility to seek licenses for such intellectual property rights from Intel and others where appropriate.

Limited License Grant. Intel hereby grants you a limited copyright license to copy this document for your use and internal distribution only. You may not distribute this document externally, in whole or in part, to any other person or entity.

LIMITED LIABILITY. IN NO EVENT SHALL INTEL HAVE ANY LIABILITY TO YOU OR TO ANY OTHER THIRD PARTY, FOR ANY LOST PROFITS, LOST DATA, LOSS OF USE OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, OR FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF YOUR USE OF THIS DOCUMENT OR RELIANCE UPON THE INFORMATION CONTAINED HEREIN, UNDER ANY CAUSE OF ACTION OR THEORY OF LIABILITY, AND IRRESPECTIVE OF WHETHER INTEL HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES. THESE LIMITATIONS SHALL APPLY NOTWITHSTANDING THE FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDY.

Intel, the Intel logo, Pentium, Intel Xeon, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2005 Intel Corporation

Introduction

Shadows give computer-generated images additional naturalness and provide the viewer with important cues about the relative sizes and positions of the objects and light in the scene. Realistic shadows are difficult to produce, especially within the confines of real-time graphics, and most shadows in 3D graphics are approximations of real shadows. Two shadow algorithms in common use today are shadow maps and shadow volumes. This article focuses on shadow volumes, which were introduced by Frank Crow in 1977 (1).

This article provides:

- A general overview of shadow volume algorithm, and describe some of its strengths and weaknesses.
- A detailed implementation of the shadow volumes on mainstream graphics hardware using Microsoft DirectX*.
- A simple example of threading to improve performance.

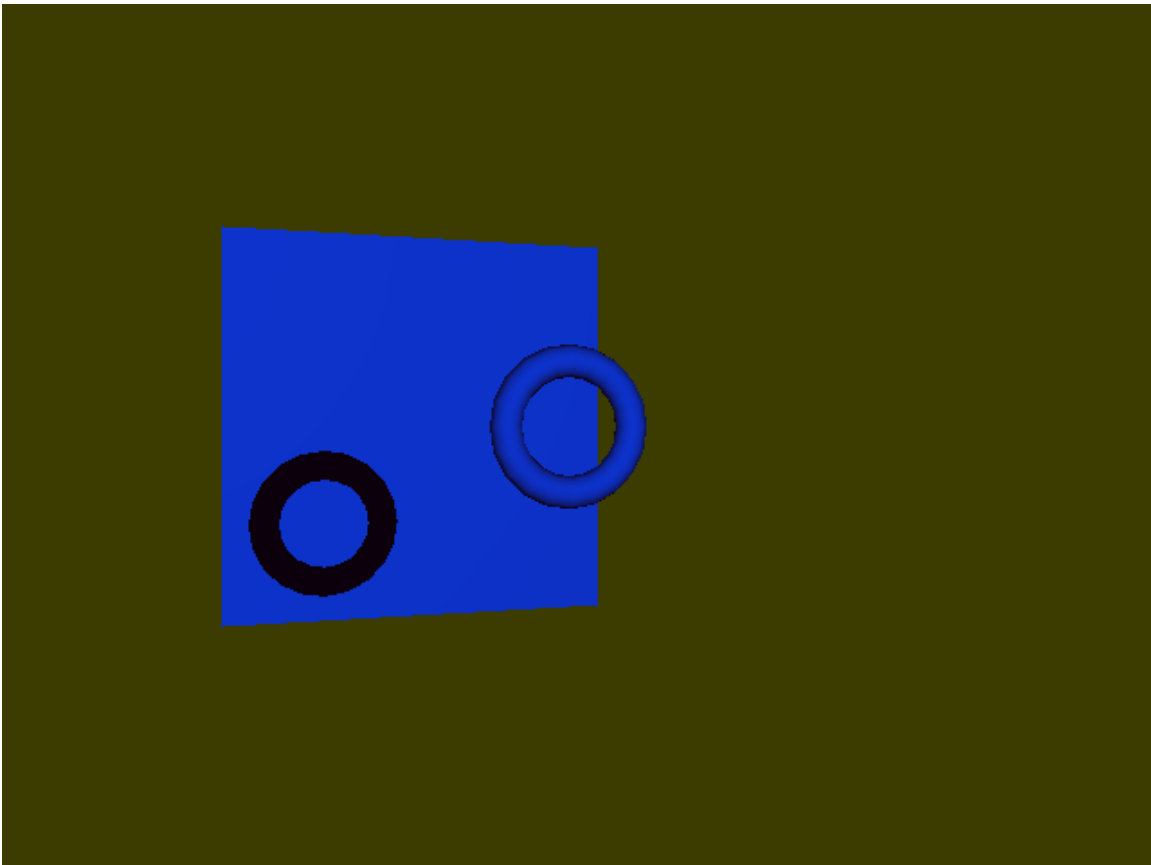


Figure 1: A simple scene rendered with a shadow generated by a shadow volume

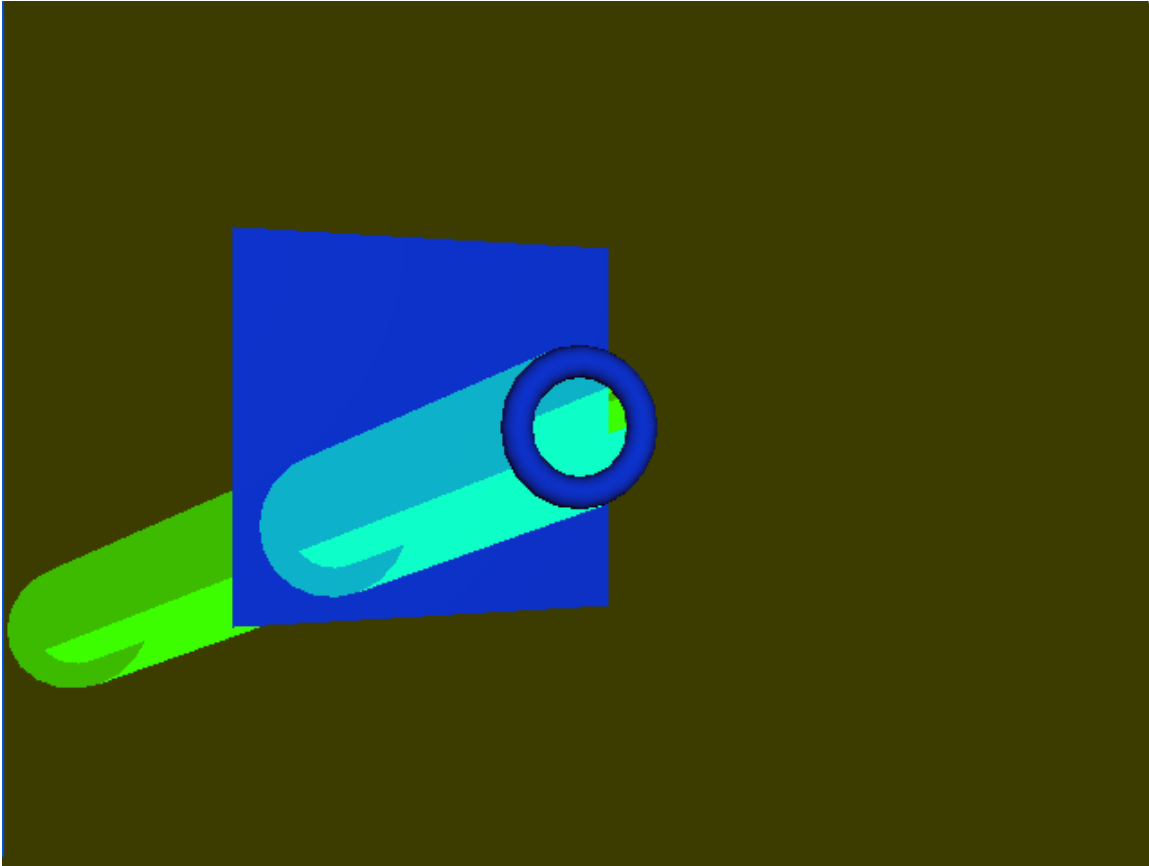


Figure 2: The scene rendered again with the shadow volume visible

Shadow Volume Overview

Shadows are produced when light from a light source is blocked by an object (the occluder) before it reaches a second object (the receiver). The darkest area of the shadow on the receiver is called the umbra. Because lights usually have a significant volume, there is an area around the edge of the shadow where some, but not all, of the light from the light source reaches the receiver. This area, called the penumbra, blends the edge of the shadow to the fully lit section of the receiver. Real-time graphics relies on a number of simplifications to the lighting system that make accurate shadows difficult. The most significant is the point light source; point lights do not generate a penumbra on the receiver because the light is either fully occluded or not.

Point light sources will make our calculations much easier, but they result in unrealistic “hard” shadows.

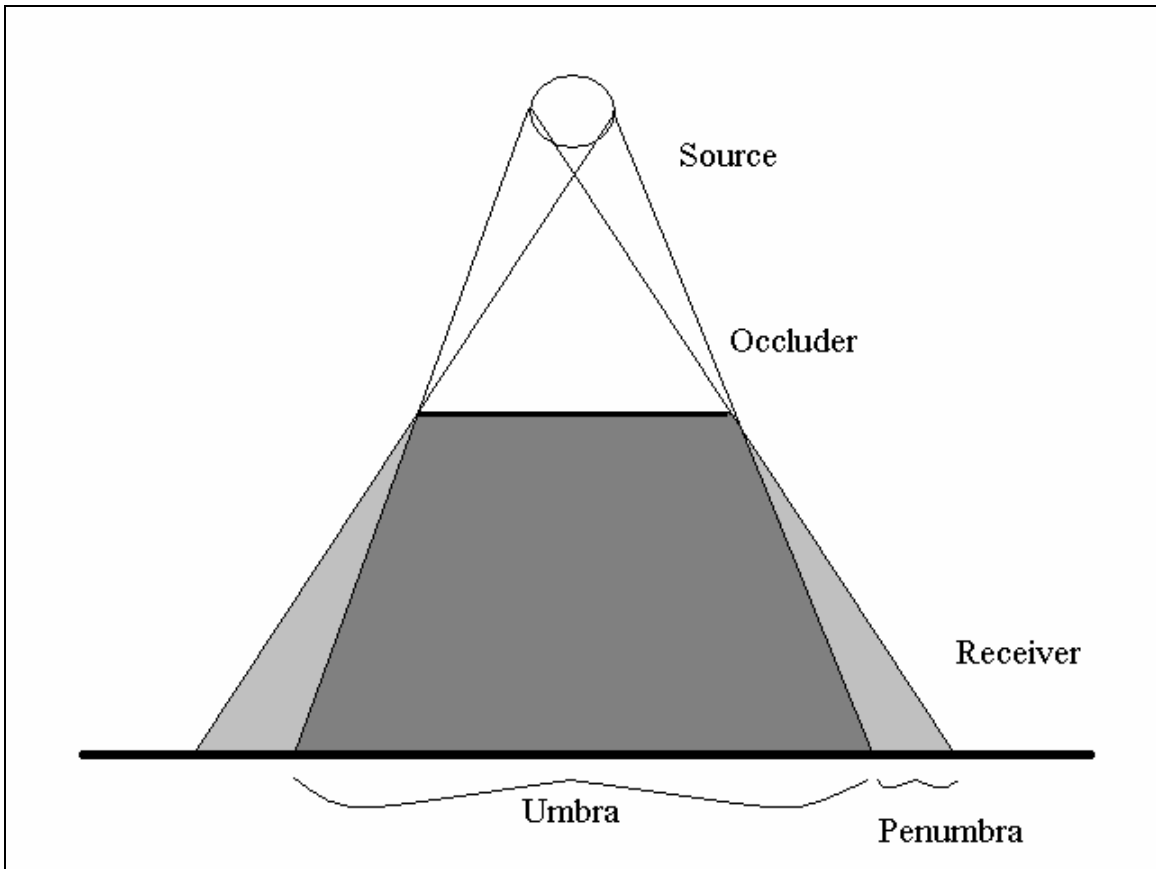


Figure 3: More realistic lighting uses soft shadows where the umbra blends into the lit region of the receiver

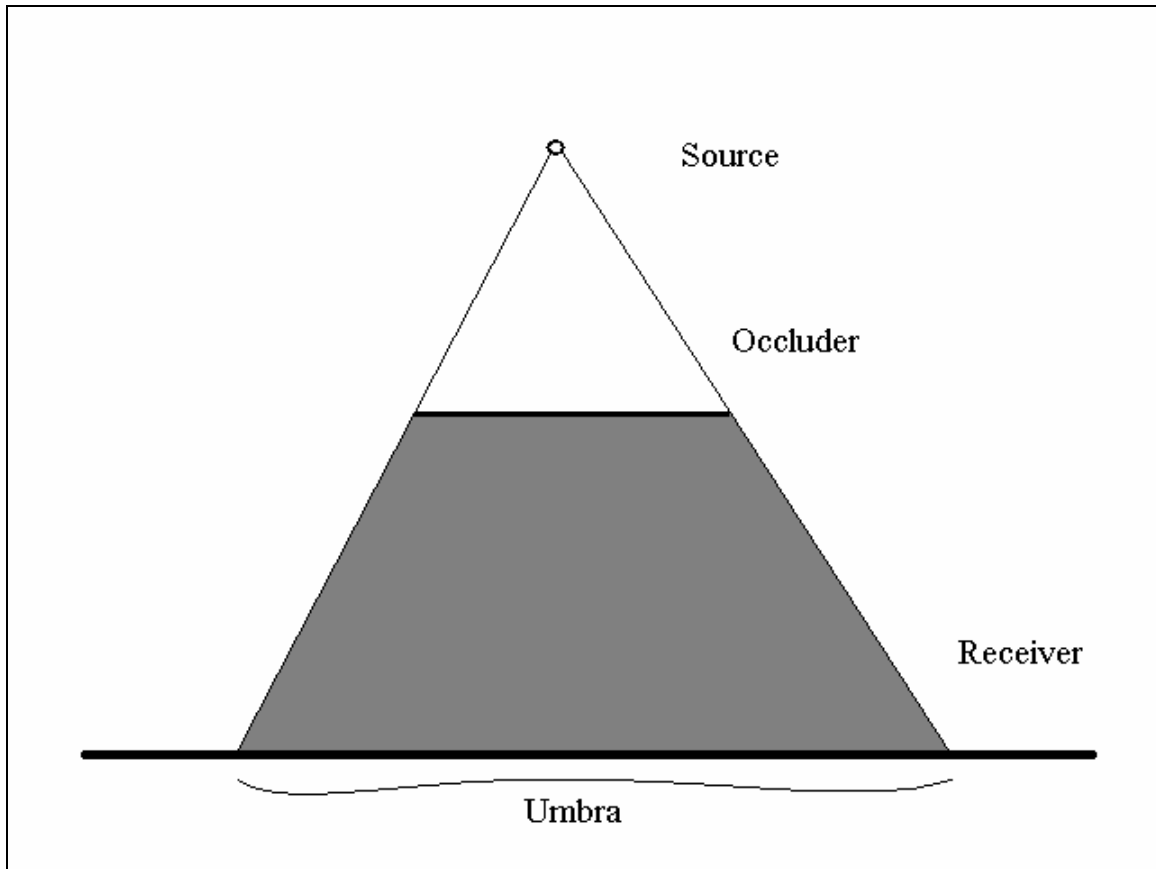


Figure 4: The simplified point light source results in hard shadows

The shadow volume algorithm uses the volume of space defined by the light source and an occluder. Anything inside the shadow volume is in shadow, and anything outside of the volume is lit. The shape of the volume is determined by the silhouette edges of the occluder relative to the light source. The shadow volume is the collection of silhouette edges projected away from the light to infinity.

When lighting a scene, we must first determine the silhouette edges of the occluders and generate the appropriate shadow volumes. Once we have the volumes we must determine what geometry is in shadow and what is not. Then we must perform the lighting calculations.

Silhouette Edge Detection

One of the most important aspects of using shadow volumes is the choice of algorithms for determining the silhouette edges of the occluder. Once the boundaries of the shadow volume are known, determining the lighting of a scene is reasonably straightforward. Silhouette edge detection algorithms vary in running time, conceptual complexity, and robustness. When choosing an algorithm, consider the models that you will use. If the models are simple, a simple algorithm will suffice.

For our implementation, we chose a simple algorithm that relies on a closed mesh, and each edge in the mesh is used by exactly two faces. The algorithm does not require any additional restrictions and runs in $O(n^2 \log n)$ complexity.

Algorithm:

```
start with an empty list of silhouette edges
for each face
    if the face is front facing
        for each edge of the face
            if the edge is in the silhouette edge list
                remove the edge from the list
            else
                add the edge to the silhouette edge list
```

The algorithm simply iterates over each triangle. If the triangle faces the light, check to see if the edges are shared with another triangle that faces the light. If an edge is not shared, then it is a silhouette edge. The included sample code contains an implementation of the above algorithm that generates the edge list for an ID3DXMesh using the vertex buffer and index buffer.

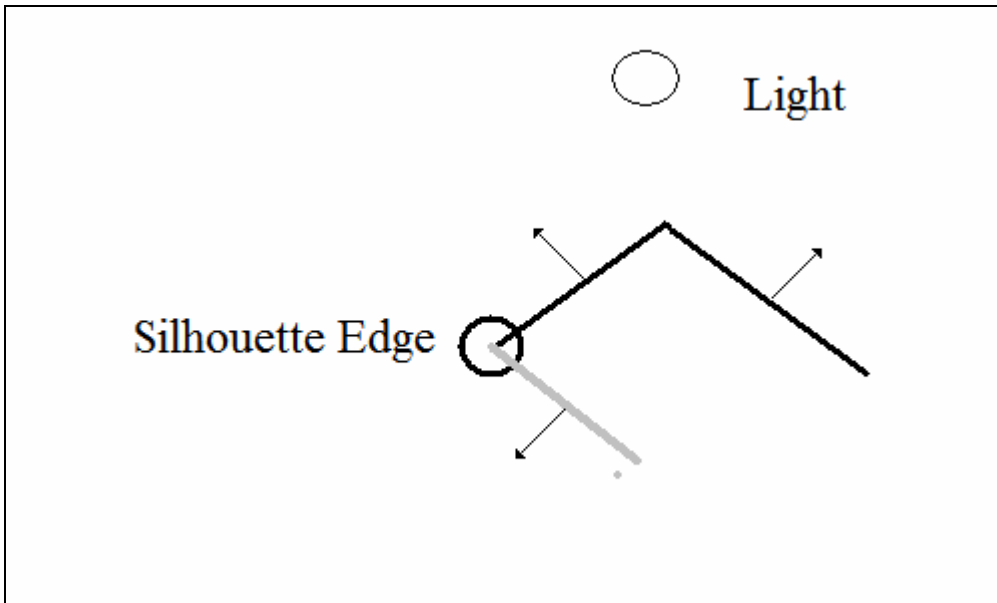


Figure 5: A silhouette edge is shared by a face that points toward the light and a face that points away from the light

Create the Shadow Volume

Now that we know the silhouette edges of the model, we must create a new mesh for the shadow volume. We create a new quad for each silhouette edge and use the vertex positions. For each position in the silhouette edges, we create two vertices: one that will be the vertex near the light source, and the other that will be the vertex projected away from the light source. We use a texture coordinate value to pass this information to the vertex

shader. When we create the quad, we must maintain the winding order of the original mesh so that we know if we are entering or leaving a shadow volume.

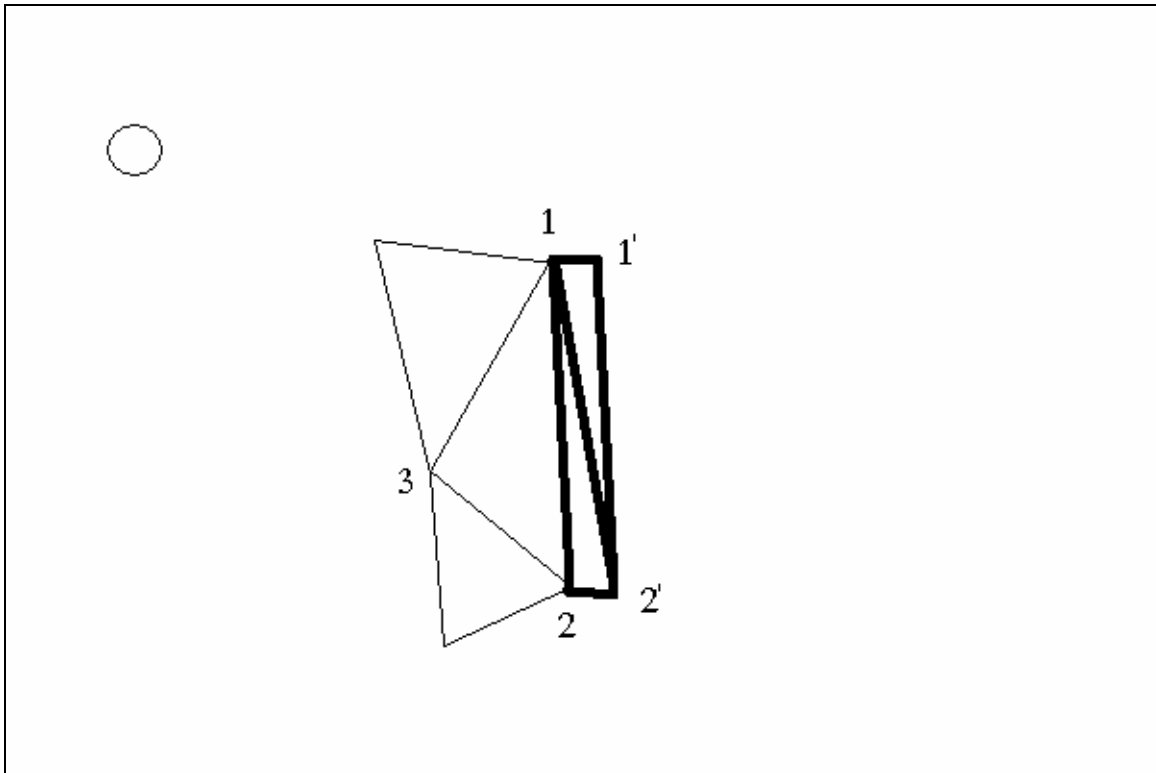


Figure 6: The new quad created has two new vertices that will be projected away from the light to form the shadow polygon

Count in the Stencil Buffer

There are two ways to use the stencil buffer and shadow volumes to determine if a point is in shadow. The simplest is the Z-Pass approach. First render the scene with only diffuse light. Then, render the shadow volumes. When we render the shadow volumes, we update the stencil buffer only if the shadow volume is closer than the point stored in the Z-Buffer from the ambient render pass, and we do not update the depth buffer. We increment the stencil buffer for each front-facing polygon, and decrement for each back-facing polygon. Then render the scene with diffuse light, and only update the render target where the stencil buffer is 0.

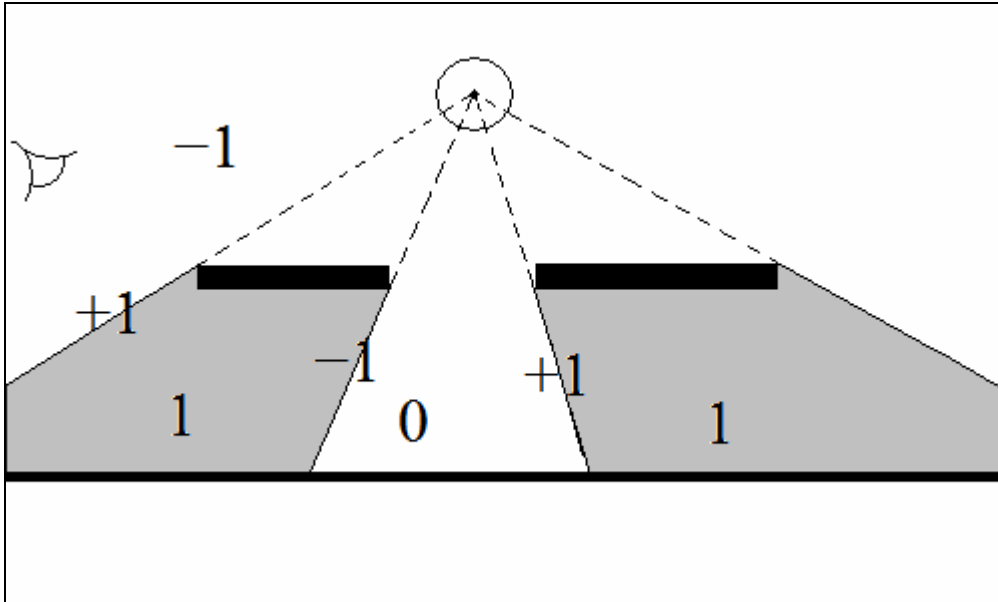


Figure 7: A simple example of the Z-Pass stencil buffer counting technique

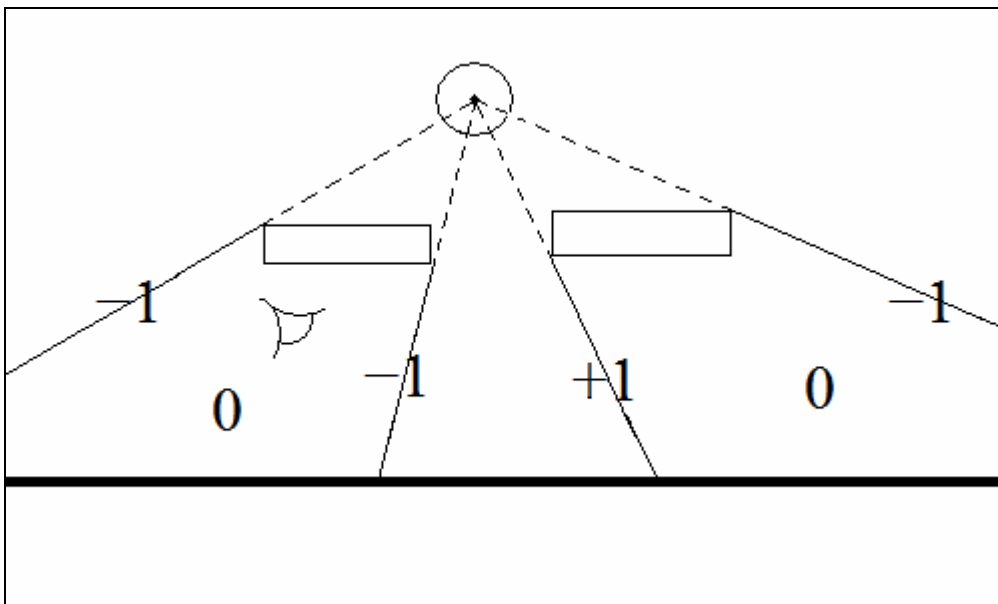


Figure 8: The Z-pass technique is incorrect when the view is inside of a shadow volume

The basic idea of Z-Pass is make sure that a ray cast from the viewer to a point in the scene exits as many shadow volumes as it enters before it reaches that point (2). Unfortunately, if the view is inside of a shadow volume, because we do not keep track of entering and leaving specific shadow volumes, the count is off by one. The Z-Fail, or Carmack's Reverse, avoids this problem by counting all of the shadows that are farther away. To use Z-Fail, render the

scene with ambient light. Then, without modifying the depth or color buffers, render the shadow volume polygons, but now update the stencil buffer only when the depth test fails. The Z-Fail technique works well, but requires that the shadow volumes are closed to maintain the correct count in the stencil buffer. We must generate caps for the shadow volumes to maintain the correct count in the stencil buffer for cases where the shadow volume is clipped by the view frustum and the volume. For simple convex shapes, capping is straightforward, but more complex shapes must be handled carefully.

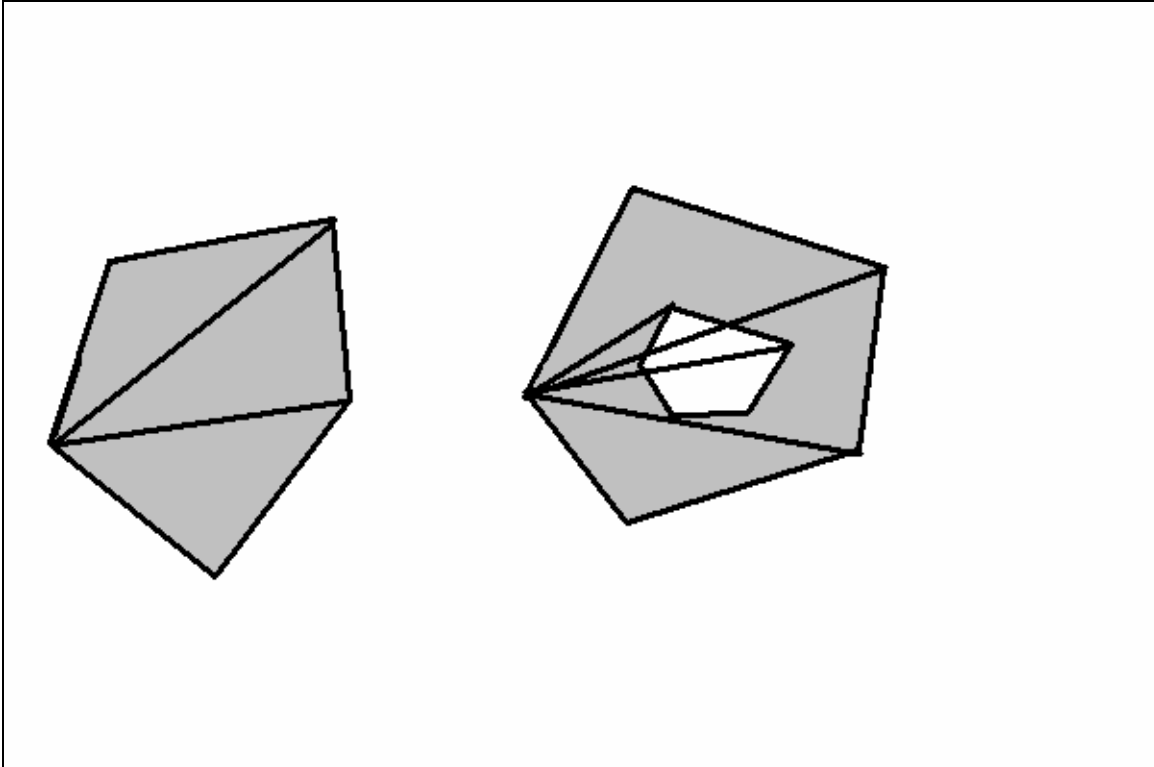


Figure 9: Simple convex shadow volumes have easy capping solutions. However, these do not apply to more complicated shapes, e.g., holes

Drawing Shadow Volumes

The code for setting up the render state to draw the shadow volume polygons is below.

```
// The RenderShadowVolume technique updates the stencil buffer using the
// Z-Pass shadow volume technique. It uses the winding order of the
// triangles to determine the face normals and projects vertices with
// texture coordinates greater than 1.0f away from the light source.
// The light source is assumed have been transformed into model space.
technique RenderShadowVolume
{
    pass p0
    {
```

```

VertexShader = compile vs_2_0 VertexShadowVolume();
PixelShader = compile ps_2_0 PixelShadowVolume();

// We will render front and back facing shadow volume
// polygons in one pass. We are using the Z-pass technique
// so increment and decrement on pass, keep on fail.

StencilRef = 1;
CullMode = None; // render both front and back
TwoSidedStencilMode = true; // faces, but treat them
// differently

StencilMask = 0xFFFFFFFF;
StencilWriteMask = 0xFFFFFFFF;

StencilFunc = Always; // clockwise faces
StencilZFail = Keep;
StencilPass = Decr;

Ccw_StencilFunc = Always; // counter clockwise faces
Ccw_StencilZFail = Keep;
Ccw_StencilPass = Incr;

StencilEnable = true;

// Do not write to the frame buffer. We want to keep the color
// and the current depth values generated from the ambient color
// render step.
ColorWriteEnable = 0x0;
ZWriteEnable = false;
ZFunc = Less;

```

Now that we have the proper render state, we can actually draw the shadow volume polygons. The vertex shader examines the vertices passed to it, and projects some of the vertices away from the light source. When creating the shadow volume mesh, we marked some of the vertices with a texture coordinate value of 0; these vertices will remain where they are. All of the others will be transformed here.

```

void VertexShowShadowVolume(float4 iPosition : POSITION,
                           float4 iNormal : NORMAL,
                           float3 iShadowProject : TEXCOORD0,
                           out float4 oPosition : POSITION,
                           out float4 oColor : COLOR)
{
    // just check the value stored in iShadowProject. If it is larger than
    // 1.0f, it should be projected away from the light source to create the
    // shadow volume polygon.
    if(iShadowProject.x > 1.0f)
    {
        iPosition = iPosition
            + 500.0f * normalize(iPosition - g_vLight0Pos);
    }
    oPosition = mul(iPosition, g_mWorldViewProjection);
    oColor = 1.0f;
}

```

Threading Shadow Volumes

Shadow volumes present an excellent opportunity for parallel execution. Each occluder/light-source pair is independent of all the other pairs. The silhouette edge

detection algorithm can be executed on a separate thread from the main rendering and other edge detection routines when the edge detection is performed on the CPU. An update manager can monitor the relative positions of light sources and occluders and update the shadow volumes when an occluder or light source moves. The shadow volumes are then regenerated on a separate thread.

To get access to the information about the model that we need, we must lock the vertex buffer and the index buffer using the LockVertexBuffer and LockIndexBuffer methods provided by the ID3DXMesh interface. The lock method allows you to specify the type of lock needed; here we just need a read-only lock that does not block the rendering of the model.

To ensure that all shadow volumes are up to date for the scene and to simplify synchronization issues, we can start threads to regenerate any necessary shadow volumes and draw the scene with ambient light at the same time. After the first pass of rendering, we render the shadow volumes, then, the scene is rendered again with diffuse light.

Additional Considerations

Shadow volumes provide you with more control over the shadows in the scene. Because only the geometry we specify as an occluder can act as an occluder, we can limit the number of occluders in a scene to help improve system performance. Unfortunately, we must also keep track of more information about the scene, so that we can regenerate any volumes associated with geometry or a light that has moved.

While shadow maps suffer aliasing on the shadow boundaries because the lighting is determined in light space (a perspective projection from the light), all shadow volume lighting is determined in view space. Visual artifacts remain a problem because the geometry casting the shadow must be of appropriate detail or the resulting shadow volume may appear too rough. Non-manifold, non-closed, and other inconsistencies within the geometry may present additional visual artifacts in the rendered scene.

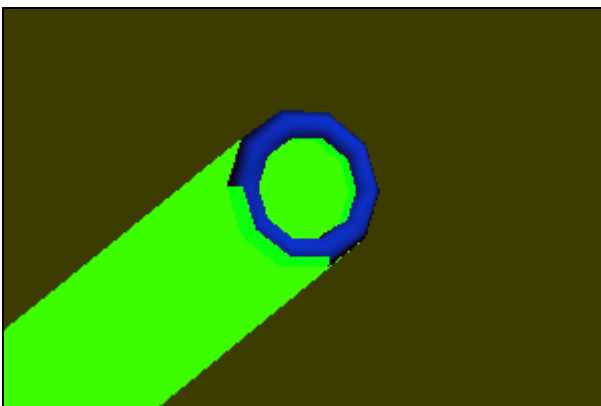


Figure 10: Low resolution geometry results in low resolution shadows

Because shadow volumes rely on the repeated drawing of many possibly large and overlapping polygons that make up the volume, the fill rate of the graphics system can be the ultimate bottleneck. (3)

Shadows are very useful in real-time graphics because they provide needed realism and important visual cues. The shadow volume technique is particularly good because it allows a high degree of control concerning which lights and occluders require attention. The technique also allows for a wide range of implementations from very simple to very sophisticated, depending on the situation. Shadow volumes also present an excellent opportunity for parallel computation that can greatly improve performance.

References and Related Articles

References

1. F. Crow "Shadow Algorithms for Computer Graphics," Computer Graphics (Proc. SIGGRAPH), Vol. 11, No. 3, Aug 1977, pp. 242-248)
2. C. Everitt and M. J. Kilgard, "Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering," developer.nvidia.com.
3. T. Akenine-Moller and E. Haines. Real-Time Rendering. A K Peters Ltd, 2nd edition, 2002.

Related Articles

A good introduction to the different techniques for generating shadows:

A. Woo, P. Poulin, A. Fournier, "A Survey of Shadow Algorithms," *IEEE Computer Graphics and Applications*, vol. 10, no. 6, November, 1990, pp. 13--32.

Most of the new work with real-time shadow volumes concerns creating soft shadows. For an excellent overview of soft shadows with shadow volumes and shadow maps see:

J.-M. Hassenfratz, M. Lapierre, N. Holzshuch, and F.X. Sillion, "A Survey of Real-Time Soft Shadows Algorithms," EuroGraphics, 2003.

For a better silhouette detection algorithm see:

G. Aldridge and E. Woods, "Robust, Geometry-Independent Shadow Volumes," GRAPHITE '04: Proceedings of the 2nd international conference of Computer graphics and interactive techniques in Australia and South East Asia, 2004, pp. 250-253.

[Distributed 3D Rendering for Gaming, Simulations and Other Rich Media Web Services](#)

[ShadowVolume code listing](#)

About the Author

David Bookout is a software engineer who has worked in 3D graphics for several years. He has three previous publications with Intel® Software Network and is currently a graduate student in computer science at Oregon Graduate Institute of Science and Technology. While at Intel, he worked on Shockwave 3D and wrote custom pixel and vertex shaders while on the graphics and 3D technologies team in the Intel® Architecture Labs. He also has a BA in English from Purdue University.

