In-Class Exercises for Shadow Algorithms

Alex Wiens and Gitta Domik, University of Paderborn, Germany

Abstract: We are describing two exercises to deepen the understanding of two popular real-time shadow algorithms, namely shadow mapping and shadow volumes. Each of the exercises takes less than 10 minutes to perform during lecture time and leads to a profounder understanding and serves as a basis for further discussions of improvements of these algorithms. By using this exercise we noticed an evident increase in implementing shadow algorithms for later real-time graphics projects.

1. Background of Course

The course "Computer Graphics Rendering" at the Computer Science Department at the University of Paderborn is for advanced Bachelor students who already had some fundamental Computer Graphics course or are quick learners. This rendering course emphasizes Shader programming and finishes with group projects in real-time rendering. As a basis for teaching this course the lecturer uses the books [AHH11] and [L12], and takes background and ideas from [AS12] and [BC12]. Shadow algorithms are discussed in one 1.5 hour lecture together with other global illumination topics for real-time rendering (such as caustics and ambient occlusion), so time is limited and should be used well. Because a profound understanding of algorithm and math underlying a real-time graphics implementation is of essence to us, we stop each lecture for one or two suitable In-Class exercises. It took us some thought to invent two short In-Class Exercises for Shadow algorithms – thus we decided to share these with the community.

2. In-Class Exercise for Shadow Mapping Algorithms

This exercise (ShadowMapExercise.pdf) can be downloaded from www.uni-paderborn.de/cs/vis. Here is a step-by-step explanation.

The Shadow Mapping algorithm has been proposed first by Williams [W78]. In the lecture, we use a modern description of the algorithm as it follows the books [AHH11] or [L12]. The idea is based on the fact that –considering one ray of light - object points in shadow are farther away from the light source than illuminated object points. Setting the viewpoint of a camera into the light position, all visible object points are lit and non-visible object points are in shadow. We store the distances between the light source and visible points (from the light source) in an array (called "shadow map") for later use. From the view of the camera, for each visible object point (such as vertices v_a and v_b in Figure 1) we can calculate the distance to the light source and compare it to the recovered distance value stored in our array. As we can see in Figure 1, the distance for object point v_a to the light source will be the same as recovered in the shadow map. The distance for object point v_b will be longer than the value recovered from the shadow map, indicating that it is in shadow. Let us use this idea for the following procedure and accompany it with the first In-Class exercise:

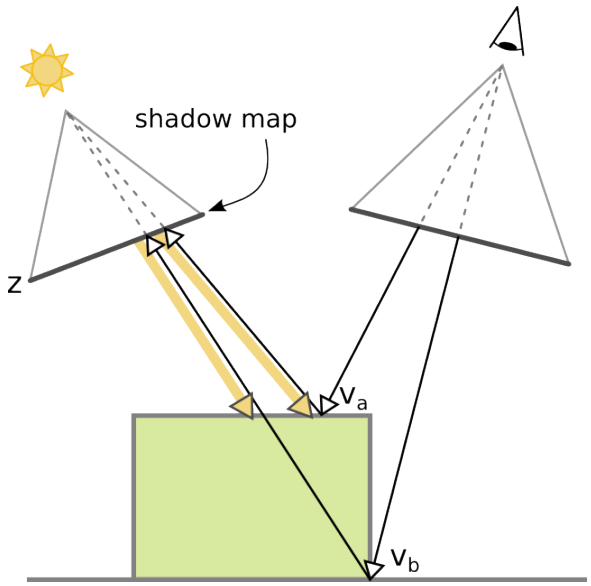

Figure 1. Basic idea of Shadow Mapping: The shadow map z contains distances from object points to light source. Two vertices v_a and v_b are seen from the camera. Vertex v_a has the same distance to the light source as recovered from the shadow map at the corresponding location (s,t). Vertex v_b has a longer distance to the light source than recovered from the shadow map at the corresponding location (s,t). The shadow mapping algorithm therefore decides that v_b is in shadow and v_a is not in shadow.

First render the scene from the position of the light source. This results in the projection of each object point p onto the near plane of the light using the ModelViewProjection Matrix M^L:

$$M^L \cdot p = p^L = (p_x^L, p_y^L, p_z^L)^T$$

Only keep the content of the z-buffer and store it in a map (=*shadow map*). Every other result of the rendering process, e.g. color buffer, is dispensable. Let's take a look at this 2d shadow map and name it z(s,t): The shadow map z(s,t) now contains depth values z (=distances between light source and lit object points) at each equally spaced 2d location (s,t), where (s,t) describe the sampling rate of this rendering procedure.

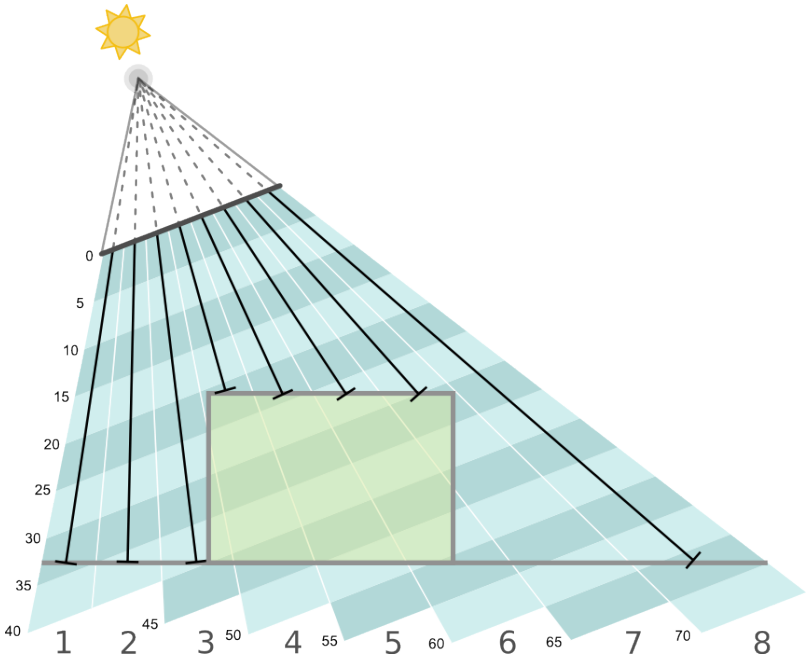

Figure 2. A cube is rendered from the view of the light source; the resulting content of the z-buffer is saved in a "shadow map".

For the set-up of Figure 2, students can fill one line of the shadow map z(at the various
by looking up the distances from the near plane (relative to the light source):

z	1	2	3	4	5	6	7	8
t	1	2	3	4	5	6	7	8

resulting in (or similar to):

z	34	37	39	21	24	27	30	62
t	1	2	3	4	5	6	7	8

Next render the scene from the position of the camera. This will result in filling the z-buffer (=distance between camera and visible object points) and the color buffer for our rendering process. We can calculate the distance from each visible point q to the light source by using

$$S \cdot M^L \cdot q = q^L = (q_x^L, q_y^L, q_z^L)^T$$

(where S is a scaling matrix ensuring the correct scaling) and compare this distance (call it z') with the previously stored z values in the shadow map. If the distances correspond, the visible point is lit, if the new value z' is larger than z, point q is in shadow.

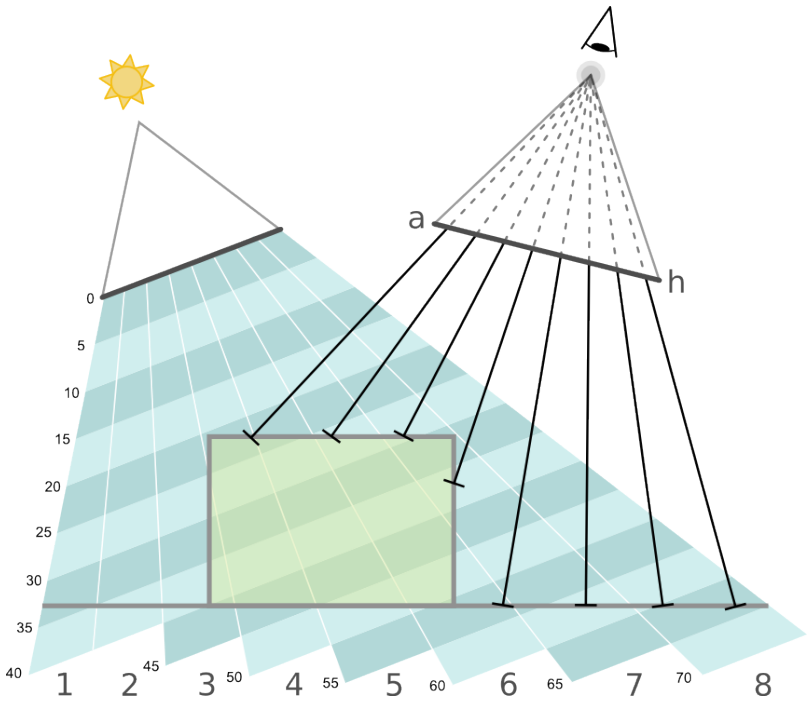

Figure 3. The cube is now rendered from the position of the camera. Object points visible from the camera are equally spaced in the corresponding image plane.

The setup in Figure 3 shows eight equally spaced image/object locations (a…. h) when rendering our cube. For one line of the current color buffer we fill in the distances z' from object point to light source by looking them up in Figure 3.

z'	a	b	c	d	e	f	g	h

We also have to remember the corresponding index t of the shadow map in relation to the new spacing in image (camera) space:

a	b	c	d	e	f	g	h
t(a)	t(b)	t(c)	t(d)	t(e)	t(f)	t(g)	t(h)

Resulting in the following z' values and corresponding shadow map index (locations t):

z'	23	26	29	37	54	57	60	64
t	4	6	7	7	6	7	8	8

Next compare the values z and z' and make a shadow decision. Where z= z' we expect lit points and where z < z' we expect shadow points for this indicates that there is another object point closer to the light source than the current object point under investigation.

This final decision can be expressed by the Heaviside function

$$H(z(t)-z') = \begin{cases} 1, for(z-z') \geq 0 \\ 0, for(z-z') < 0 \end{cases}$$

If H(z(t)-z') = 0 (or slightly larger than zero because of inaccuracies), then q lies in shadow, if H(z(t)-z') is negative, q is lit.

For the next two lines of exercise we compare z(t) and z', but remember to use the corresponding shadow map index t resulting in

z(t) - z'	-2	1	1	-7	-27	-27	2	-2
Corresponding index in image buffer (a…h)	a	b	c	d	e	f	g	h
Corresponding index in shadow map (t)	4	6	7	7	6	7	8	8
Shadow decision H(z(t)-z')	0	1	1	0	0	0	1	0
s(hadowed) or lit?	s	lit	lit	s	s	s	lit	s

Comparing this result with Figure 3 we see that the decision is not correct for image index b and h. This phenomenon is called self-shadow aliasing and is treated in the next section.

Improvement of self-shadow aliasing. One can easily imagine that the precise comparison of z and z' leads to errors. Each z entry in the shadow map represents an area in object space whose size is dependent on the applied sampling rate during the first rendering from the light source. Depending on where p or where q are located within this area, the distance of q might be larger than the distance of p to the light source and erroneously determine shadow for this location. This so called self-shadowing can be corrected by adding a constant bias to the entries of the shadow map. Therefore allow students to add a constant bias c (e.g. c=2) to the entries of the shadow map, resulting in a comparison z(t) + c-z'

z(t) +2 - z'	0	3	3	-5	-25	-25	4	0

and a new (correct) shadow decision

H(z(t) +2 - z')	1	1	1	0	0	0	1	1
s(hadowed) or lit?	lit	lit	lit	s	s	s	lit	lit

The solution sheet (ShadowMapSolution.pdf) can also be downloaded from www.uni-paderborn.de/cs/vis.

This exercise is easy to extend by other algorithm improvements, e.g. the percentage-closer filtering [AHH11].

3. In-Class Exercise for Shadow Volume Algorithms

This exercise (ShadowVolumesExercise.pdf) can be downloaded from www.uni-paderborn.de/cs/vis. Here is the step-by-step explanation.

The Shadow Volume Algorithm originates by Franklin Crow, 1977 [C77]. In the lecture, we use a modern description of the algorithm as it follows the books [AHH11] or [L12]. Modern hardware increases the speed of this algorithm so that it is now in common use for dynamic real-time shadowing. The basic idea behind this algorithm is that the shadow of an object is constructed as a volume by extruding the object's silhouette (with respect to a light source) away from this light source (grey area extruding from the green object in Figure 4).

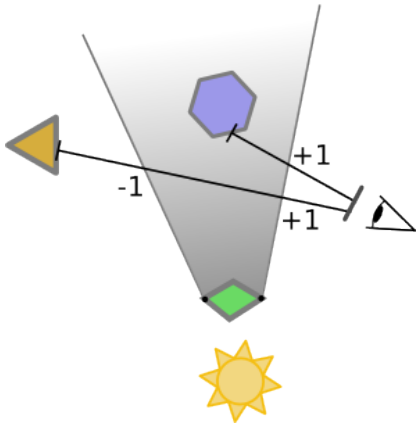

Figure 4. A light source lights a green object that will block out light for other objects within its "shadow volume". (The shadow volumes for the orange and blue object are not drawn in this Figure.) A line of sight from the camera will intersect a front facing polygon of the shadow volume when rendering a point on the blue object; a line of sight from the camera will intersect a front facing and a back facing polygon of the shadow volume when rendering a point on the orange object. Intersecting (crossing) a front facing polygon adds one to the stencil buffer, intersecting a backfacing polygon subtracts one to the stencil buffer, leaving the stencil buffer with +1 for the corresponding image location of the object point on the blue object and with 0 for the corresponding image location of the object point on the orange object.

Those parts of other objects lying inside this object's shadow volume are in shadow. For a typical scene, many shadow volumes will exist. A ray of sight sent out from the camera to an object point might cross a shadow volume's surface on entering and exiting the volume. When increasing a counter on entering and decreasing it on exiting the volume, the counter will have a non-zero value if the ray ends on a shadowed object. This procedure can be realized by (non-visibly) drawing the volume's front and back faces and using the stencil-buffer as the counter.

The classic algorithm called "z-pass Shadow Volume algorithm" consists of two parts:

Part 1: initialize all buffers (including the stencil buffer) and **render the scene** as seen from the camera using only ambient illumination of the light source (this makes the resulting color buffer look like a scene in full shadow). Keep the depth (z) buffer.

Part 2: Set the depth test to **pass** for z values smaller and equal values than in the depth buffer. Now *in a first pass* render all the **front faces** of the shadow volumes without drawing to the color buffer or changing the depth buffer. Instead, this and the next rendering pass will aid only in filling the stencil buffer. Whenever a depth test is performed the stencil buffer is increased if the test is **passed**. Nothing happens if the depth test fails.

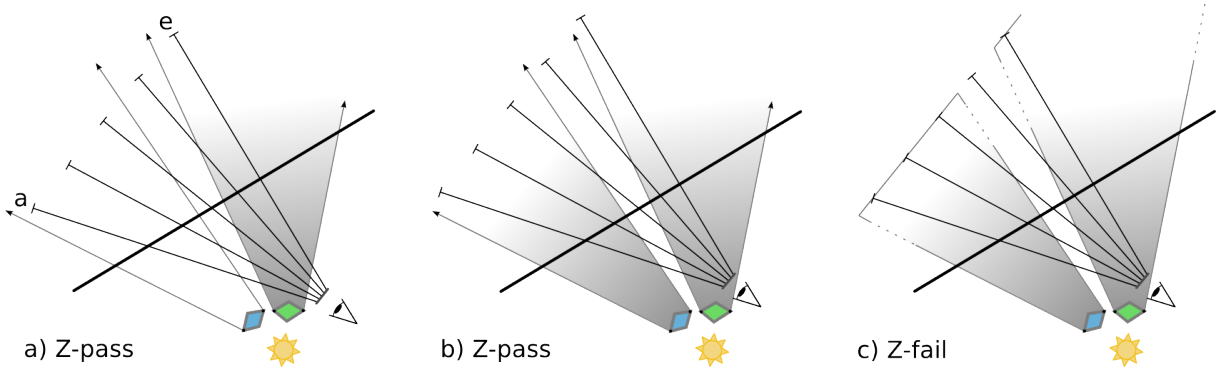

Figure 5. Two objects and their shadow volumes in front of a surface being rendered into the image buffer with image locations a through e.

Applying this procedure, students can use Figure 5 (a) to track the intersection of the camera rays and the volume faces to determine the changes to the stencil buffer:

Change in stencil buffer…	+2	+2	+1	+1	+1
… at location	a	b	c	d	e

Now render the **back faces** in a *second pass*, and decrease the value in the stencil buffer whenever the depth test is being **passed.** Again the changes to the stencil buffer are recorded.

Change in stencil buffer	-1	-1	-1	0	0

By summing up the changes the resulting state of the stencil can be computed.

Content of stencil buffer	1	1	0	1	1

Now in a final rendering pass of the objects in the scene (where only fragments with stencil value 0 are being lit) renders the scene correctly. From the stencil buffer the students can derive the shadow decision for each pixel.

s(hadowed) or lit?	s	s	lit	s	s

This works well when the camera is outside any shadow volume. When the camera's near plane intersects the shadow volume, this algorithm fails. The students reproduce this problem by repeating the algorithm using a slightly changed camera position (Figure 5 b) and observe the wrong results in the final shadow decision:

Change after front face	1	1	0	0	0

rendering					
Change after back face rendering	-1	-1	-1	0	0
Resulting stencil buffer	0	0	-1	0	0
s(hadowed) or lit?	lit	lit	s	lit	lit

Changes to the original “z-pass” algorithm lead to a camera position robust solution, called the “z-fail Shadow Volume algorithm”. A robust solution [EK03] demands several changes in the approach: first, the silhouette's extrusion has to be changed to extrude the vertices (mathematically) onto the camera's far plane, by using a special projection matrix. This ensures that all normal geometry resides inside the shadow volume. Second, a cap for the shadow volume has to be added onto the back face – and front face! For our example, Figure 6 points out the relevant front faces, back faces and back face cap.

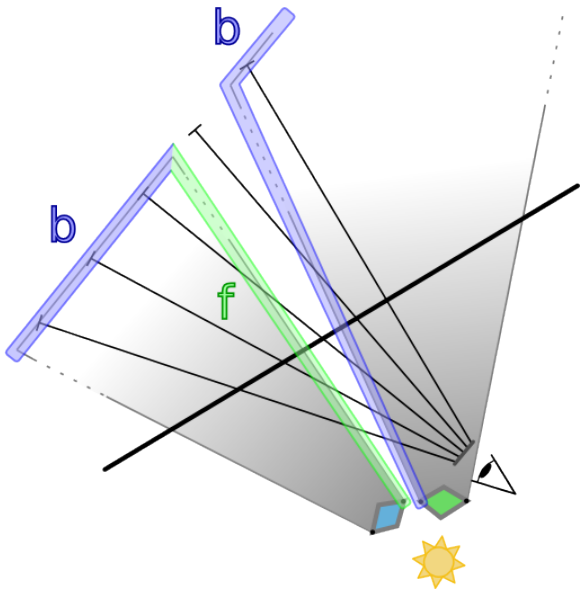

Figure 6. For the Z-Fail algorithm, the silhouette's extrusion has to be changed to include a cap at the end of the shadow volume. This figure emphasizes the relevant front faces (f”), back faces and back caps (“b”) for rendering a surface from the camera’s point of view.

Also the drawing decision and the stencil operation have to be changed to the following:

Part 1: initialize all buffers (including the stencil buffer) and **render the scene** as seen from the camera using only ambient illumination of the light source. Keep the depth (z) buffer.

Part 2: Set the depth test to **pass** for z values smaller than in the depth buffer. Now *in a first pass* render the **front faces** of the shadow volumes without drawing to the color buffer or changing the depth buffer. Whenever a depth test is performed the stencil buffer is **decreased** if the test **fails**. Again, the changes can be tracked by checking the camera ray in Figure 5(c). This time, however, the scene behind the visible geometry has to be considered, resulting in

Change in stencil buffer…	0	0	-1	0	0

… at location	a	b	c	d	e

Now render the **back faces** in a *second pass*, and **increase** the value in the stencil buffer whenever the depth test **fails**.

Change in stencil buffer	+1	+1	+1	+1	+1

The resulting stencil buffer and shadow decision demonstrate how the z-fail method solves the preceding problems.

Content of stencil buffer	1	1	0	1	1
s(hadowed) or l(it)?	s	s	lit	s	s

The solution sheet (ShadowVolumesSolution.pdf) can also be downloaded from www.uni-paderborn.de/cs/vis.

4. Discussion

Students produce a final (group) project for this course, where they are free to construct an animation of a virtual world of their choice and making. They are being graded for the real-time visual effects they implement and the models they generate for their world. They are free to use additionally imported models to fill their world as they see suited. In 2012 we had 19 students submitting 6 projects (with roughly 3 students per group), only one project group implemented a shadow algorithm; in 2013 we had 8 group projects, again only one project included a shadow algorithm; in 2014 (when we implemented the In-Class shadow exercises) we had 26 projects, where 10 projects implemented a real-time shadow algorithm.

Over the last years we have submitted the best of the group projects resulting from this course to ACM SIGGRAPH's "Faculty Submitted Student Work" Competition. The following two snapshots (Figure 7 and Figure 8) are from such submitted animations.

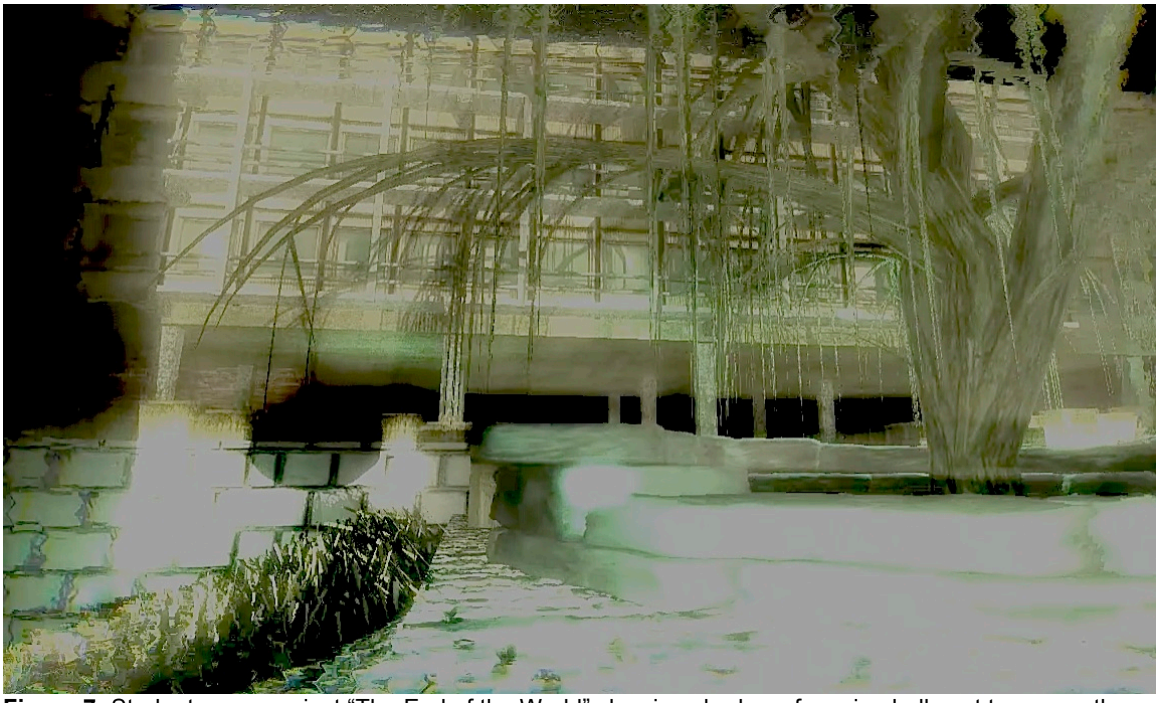

Figure 7: Student group project “The End of the World” showing shadow of moving ball next to many other real-time effects, such as blooming or particle systems.

Figure 8: Student group project “Sheep at a Picnic” showing a candle flame throwing shadows of objects on table.

Acknowledgement: Peter Wagener, Stanislaw Eppinger, Tobias Rojahn, Thomas Löcke, Kathlén Kohn for their projects; Stephan Arens as Lab coordinator; Mike Bailey and Scott Owen for Review.

References

[AHH11] Tomas Akenine-Möller, Eric Haines and Naty Hoffman, *Real-Time Rendering*, Third Edition. Boca Raton, FL, Taylor & Francis, 2011.

[AS12] Edward, Angel and Dave Shreiner, *Interactive Computer Graphics: A Top-down Approach with Shader-based OpenGL,* 6. Boston, MA, Addison-Wesley, 2012.

[BC12] Mike Bailey and Steve Cunningham, *Graphics Shaders: Theory and Practice,* 2nd. Boca Raton, FL, Taylor & Francis, 2012.

[C77] Franklin C. Crow. 1977. Shadow algorithms for computer graphics in *Proceedings of the 4th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '77). New York, NY, ACM, 1977, pp. 242-248.

[EK03] Cass Everitt and Mark J. Kilgard (2003), *Practical and robust stenciled shadow volumes for hardware-accelerated rendering*. In: arXiv:cs/0301002

[L12] Eric Lengyel, *Mathematics for 3D Game Programming and Computer Graphics*, Third Edition. Boston, MA, Cengage Learning, 2012.

[W78] Lance Williams. 1978. Casting curved shadows on curved surfaces in *Proceedings of the 5th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '78). New York, NY, ACM, pp. 270-274.