ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20140324202037

REAL-TIME SHADOWS IN OPENGL CAUSED BY THE PRESENCE OF MULTIPLE LIGHT SOURCES

Dženan Avdić, Dejan Rančić, Petar Spalević, Aldina Avdić, Edin Dolićanin

Preliminary communication

In modern computer graphics, the emphasis is on the details of the scene, and taking into account the improvements in hardware performances, it is not allowed to make compromises when it comes to the reality of scenes. Each reflection, shadow, rounded corner and transparency must be brought to perfection and presented in order to make a depicted scene more realistic. In one of the most widely used API for rendering 3D objects, OpenGL, there is nothing similar to a library for adding those phenomena that exist in reality. If the possibility of existence of multiple light sources is considered, rendering all these details becomes a real challenge. The aim of this paper is to provide a method for generating shadows in an efficient way, for the objects commonly used as components of complex 3D objects, in conditions of the presence of moving light sources.

Keywords: *circle; cone; moving point light source; OpenGL; shadows; sphere; triangle*

Sjene u realnom vremenu u OpenGL-u uzrokovane prisutnošću više izvora svjetlosti

Prethodno priopćenje

U suvremenoj računalnoj grafici naglasak je na detaljima prizora, a uzimajući u obzir poboljšanja hardverskih svojstava, nije dopušteno raditi kompromise kada je riječ o stvarnosti scena. Svaki odraz, sjena, zaobljeni kut ili prozirnost mora biti doveden do savršenstva i prikazan u cilju da se scena koja se oslikava učini što realističnijom. U jednom od najšire rabljenih API-ja za renderiranje 3D objekata, OpenGL-u, ne postoji nešto poput knjižnice za dodavanje ovih pojava koje postoje u stvarnosti. Ako se razmatra mogućnost postojanja više izvora svjetlosti, renderiranje svih ovih detalja postaje pravi izazov. Cilj ovog rada je obezbjeđivanje metoda za generiranje sjene na efikasan način, za objekte često rabljene kao komponente složenih 3D objekata, u uvjetima prisutnosti više svjetlosnih izvora s mogućnošću kretanja.

Ključne riječi: *krug; kugla; OpenGL; sjene; stožac; pokretni točkasti svjetlosni izvor; trokut*

1 Introduction

Bearing in mind its ability of creating virtual reality, computer graphics is widely used in designing and programming computer games, CAT software, the software dedicated to various types of process simulations, in a modeling of scientific phenomena and for many other purposes. The existence of shadows and reflections in a 3D scene is crucial in enhancing the scene realism and in providing important visual cues.

In recent years, many important contributions have been made in modeling shadows, as phenomena caused by the presence of light sources and reflections, which are the consequences of presence of objects made of special material, with the mirror effect. Shadows are created by a light source which is placed above the shadowed object. Those shadows are placed on the surface where the shadowed object is placed. But, if a point light source is placed below a shadowed object, made of the material which reflects light, the phenomenon thus created on such surface is a reflection. Their presence on the scene improves a visual perception. Shadows enhance the 3D impression in order to make better immersive 3D feeling. Also, they depict relations between objects on a scene, so the distance between two virtual objects can be sensed more clearly [1].

A shadow is an area where direct light from a light source cannot reach due to obstruction by an object. It occupies the whole surface behind an opaque object with light in front of it. The cross section of a shadow is a two-dimensional silhouette or reversed projection of the object blocking the light.

Physically, a shadow consists of the umbra and the penumbra. The umbra is a result of the existence of an area created by a shadowed object that is not visible from any part of the light source. This is a dark part of the shadow. With their own area, light sources can make a penumbra, which is less dark than umbra. It happens when the area of a shadowed object can receive some, but not all light from the light source. As it cannot be the case with point light source (it does not have significant area), the shadows made in this way have only an umbra [2, 3].

When a light source is added to the 3D scene drawn in OpenGL workspace, some changes on the objects in the scene and their material can be noticeable, but new creations caused by this introduction of lighting, such as shadows, are not made, which is not the case in reality. The current methods are faced with problems of high demands for rendering scenes; otherwise, if they do not require a lot of resources for implementation and rendering, they do not make realistic shadows. The methods vary in difficulty of their implementation, their performance, and the quality of their results. These qualities depend on two parameters: the complexity of the observed object and the complexity of the scene that is being shadowed [4÷8].

The approach described in this paper promotes facilitation of the formation of shadows, taking into account the events that occur when objects are exposed to light in reality. In this example, the light sources are point sources of light, and the surface, where the light rays create the shadows, is flat. These terms are used in complex scenes, where rendering speed is important, but we use them to demonstrate our shading method and to simplify its understanding.

This paper presents a new method for the formation of shadows in the case of two 2D primitives (circle and triangle) and 3D object (a ball and loads). The aim is to show that the shadow can be drawn if we know the necessary number of important points of primitives. Also, apparently different forms have their common features that facilitate calculation of these characteristic points. Also, the dependence of the color of the shadow from the distance of the light source is displayed. In the end, it is shown that making shadow of a complex 3D object is related with 2D objects which are its basis.

This paper is organized as follows. The first section provides an overview of the state of art in this area, with the emphasis on their advantages and disadvantages of existing methods. The next chapter presents the mathematical basis and the pseudo code for the derivation of the formula used for finding characteristic points for drawing shadows, and their color. This is followed by the description of experiments, with an application created using derived mathematical basis, and the display of results. Finally, conclusions and trends in future work are given.

2 Related work

When it comes to displaying real phenomena using computer graphics high standards are set, since there are several techniques for creating shadows, but they are constantly improving. When OpenGL is concerned, the most important shading techniques are: projected shadows, shadow maps and shadow volumes.

The projected shadows are facile for implementation, but they are approximate to the shadows made in real circumstances. The idea is to project the shadowed object onto one of the axes. By moving the light source or the shadowed object, the form of a shadow will not change [5].

The shadow volume is built by the rays from the light source that cut off the corners in the item which casts a shadow, going ahead out of the scene. In this way, a polygonal surface which contains shaded or partially shaded items is created. This strategy uses the stencil buffer to calculate which portion of the object is within the shadow volume. Depending on whether the test depth passes, the stencil value for each pixel in the scene will increase if the limits for the shadow volume intersect on the way in, but reduce if they cut on the way out.

This method is struggling when shadowing complex objects, and its implementation is greatly onerous [1, 6, 7].

In order to create a shadow in a scene, the shadow map strategy uses depth buffer and texture projection. The scene is outlined from the light's point of view. As the objects in the scene are rendered, the depth buffer is updated. It produces the shadow map which is used for placing textures on the area in the shadow. The disadvantages of this strategy are aliased shadow edges and self-shadowing effects [8÷13].

The problem of existence of multiple light sources is considered in the literature [14, 15, 16], but there was not any solution that could be applied independently of number of light sources, or could be efficient and cheap at the same time. Also, the possibility of moving the light sources was not considered.

All the methods for implementing shadows have certain problems with rendering the scene. They all have their advantages and disadvantages. Usually, if they are facile to implement, then the reality of shadows is questionable. If shadows look like in reality, then their computing and drawing requires a lot of resources and time. Also, some of them have a problem when it comes to creating a complex object shadows. The aim of our method is to decrease these problems and to establish a single library which will make feasible the addition of a shadow after drawing primitives, independently of the number or the position of light sources. Also, complex objects consisting of 2D and 3D basic objects will have their shadow formed from the shadows of their components.

3 Calculating the shadows of the primitive objects

Our approach is based on the advent which occurs in reality, in case when the scene is illuminated by point source of light, which has the capability to move along the axis of the coordinate system [17]. The benefit of rendering shadows in this manner is in the fact that it is enough to calculate the lowest number of points which are relevant for drawing shadows. Another advantage is the presence of similarities between shading primitive shapes. The sphere is made by rotation of a circle around one of the axes; the cone is made by rotation of a triangle. So, the invention of the method for forming the shadow for 2D objects makes simple producing the shadow for the 3D object.

The paper considers producing shadows for four objects (circle, sphere, triangle and cone), located away from the surface for a certain distance. The shadow is placed on the flat surface and is created by the point light source with the ability of movement (it can change its *X* and *Y* coordinates).

If the light source is near to the *Y* axis, the shadow area is enhanced, and vice versa. When the light source shifts away from the object in *X* axis, the shadows are also going in the reverse direction to the motion of the light source.

Some of computation parameters are taken as constants, due to a great number of limitation of 3D system exposure on 2D paper, and they are:

The light source is always above shadowed object ($y_i > y_p + h_p$).

X and *Z* are constant coordinates of the object, with the value 0.

x_i, y_i, z_i are the coordinates of the position of the light source ($z_i = 0$).

h_p is the height of the object (height of the isosceles triangle and cone, distance between the diameter of circle and sphere and the light source).

r is the radius of the circle and the sphere.

d is the distance between the object and the surface.

x_p, y_p, z_p are coordinates of center point of object (for an isosceles triangle that is the center of the base, for circle it is the circle's center, for the sphere, and for the cone it is the center of its base).

3.1 Calculating the color of the shadow

In order to make the scene where the shadowed objects are placed as realistically as possible, it is not enough to draw the shadow in a certain shade of gray, but is necessary to calculate that shade.

It is known that the shadow is stronger if the light source is, with respect to the shadowed object, closer. If the source of light is further, the created shadow is fainter. The question is how to calculate the distance between the object and the light source, and how to include the position of the shadow in this calculation.

The distance of the object and the light source can be calculated as the distance between their centers.

The shadow has some shade of gray, and it is known that the shades of gray in RGB model are made if all the components of the color, red and green and blue, are set to the same value. If the value is closer to one, the shade of gray is darker, closer to black RGB (0, 0, 0). If the value of the components is closer to 255, the shade of gray color is closer to white RGB (255, 255, 255).

It is necessary to define the coefficient which would multiply 255, and which would be $0 \le k \le 1$, to give a realistic shade of the shadow; in accordance with the conditions set above, the shade is darker if the distance of the light source is smaller, and vice versa.

In [17] it is explained that the requirement to form a shadow on the surface is that the light source should be above the observed object, and if it is below, then it forms a reflection. This means that the distance between the light source and the surface (center of the shadow) is always greater than the distance of the object from the surface. Thus the ratio of these two distances will always be less than 1, if we take the object distance as the numerator, and the distance of the light source as the denominator. In order to meet the requirement to produce a darker shade of shadow if the object and the light source are closer, then the resulting ratio should be deducted from the Eq. (1) (given in the Eq. (1) and following pseudo code).

$$k = 1 - \frac{\text{distObj}}{\text{distLightSource}}, \tag{1}$$

```
functioncalculateShade(object,
lightSource)
begin
distObj =
calculateDistance(objCenter,
shadCenter);
distLS = calculateDistance(lsCenter,
shadCenter);
koefShade = 1 - distObj/distLS;
shade = koefShade*255;
color = new RBGColor(shade, shade,
shade);
return color;
end
```

3.2 Calculating the shadow for the circle

In the case when the circuit placed parallel to the surface is illuminated by the light source, only the umbra is present. There are two considered situations regarding the position of light source: when it is just above the circle ($x = x_p = 0$) (Fig.1a) and when it is to the left or the right (Fig. 1b).

Two important points for rendering shadows are the center of the shadow and the size of radius of the shadow. The shadow in this case is also a circuit, and coordinates of its center could be found using the equation of a line which passes through the two points: the center of the light source and the center of the shadowed object.

In the first case, the coordinates of the center of the shadow are the coordinates of beginning point (0, 0, 0). In the second situation, the formula for obtaining the center of the shadow is given in Eq. (2).

$$x_s = \frac{-d \cdot x_i}{y_i - d}, \tag{2}$$

$$r_1 = \frac{r \cdot y_i}{y_i - d}. \tag{3}$$

When $d = 0$, x_s equals zero, which means that no shadow appears if the object is on the surface and the light source is above it.

Calculating the length of the radius of the shadow is done using the theorems on similarity of triangles. Fig. 1 shows the relevant triangles for calculating the increasing factor for original radius. The equation for calculating the radius of the shadow is given in Eq. (3). From this formula it can be seen that the length of the radius depends on r, y_i, and d.

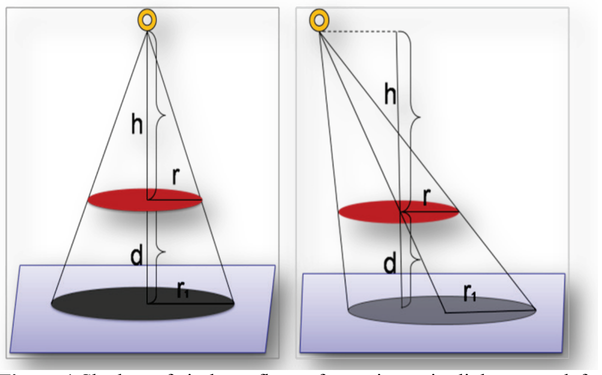

Figure 1 Shadow of circle on flat surface using point light source left $x_i = x_p = 0$; right $x_i \neq x_p$

Implementation of these equations is given by pseudo code that consists of three functions, calculateShadowRadius, calculateShadowX, drawCircleShadow, and their invocation in the case of using two light sources.

```
functioncalculateShadowRadius(object,
lightSource)
begin
dist = calculateDistance(objCenter,
lsCenter);
shadowRadius =
(circleRadius*lightSourceY)/(lightSou
rceY-dist);
return shadowRadius;
end

functioncalculateShadowX(object,
lightSource)
begin
```

```
dist = calculateDistance(objCenter,
lsCenter);
shadowX = (-
dist*lightSourceX)/(lightSourceY-
dist);
return shadowX;
end

function drawCircleShadow(object,
lightSource)
begin
radius =
calculateShadowRadius(object,
lightSource);
shadowX = calculateShadowX(object,
lightSource);
shadowY = 0;
shade = calculateShade(object,
lightSource);
shadowPostition = new Point(shadowX,
shadowY);
drawCircle(shadowPosition, radius,
shade);
end

drawCircle;

drawCircleShadow(object,
lightSource1);

drawCircleShadow(object,
lightSource2);
```

As it can be seen from last two lines of the code, we can draw as many shadows as light sources are present, by calling a method e.g. drawCircleShadow (object, lightSourceN).

3.3 Calculating the shadow for the triangle

Like in the previous section, we can compute the points for the shadow of the isosceles triangle in the case when its base is parallel to the X axis. Then the characteristic points will be collinear, hence its shadow will be a line.

It is important to notice the similarity when computing the shadows for the seemingly different forms. Significant points to draw the triangle shadow in this case are those where the beam of light that passes through the vertices of the triangle cuts the surface. Here we can use mitigating circumstance that the equation of the shadow of the circle can facilitate obtaining of these points.

A connection between the base of a triangle and its center, and diameter of a circle and its center is that in both cases the shadow is a line.

We can consider the half of the basis as a circle radius to find the center of the triangle shadow and the length of the shadow basis Eq. (4) and (5).

Now the problem is simplified to discovering the triangle peak, and as a common point of two equal edges will be calculated according to Eq. (6), (7) and (8).

In the case where the base of the triangle is parallel to the Z axis, relevant points are not collinear, and the shadow is not a line but a triangle. Three essential points are: $P_1(0, 0, z_{s1})$, $P_2(0, 0, z_{s2})$, $P_3(x_{s3}, 0, 0)$, where z_{s1} and z_{s2} are calculated similar to x_{s1} and x_{s2}. There are two situations considered in regard of the position of light source: when it is just above the triangle ($x = x_p = 0$) and when it is the left or right in Fig. 2.

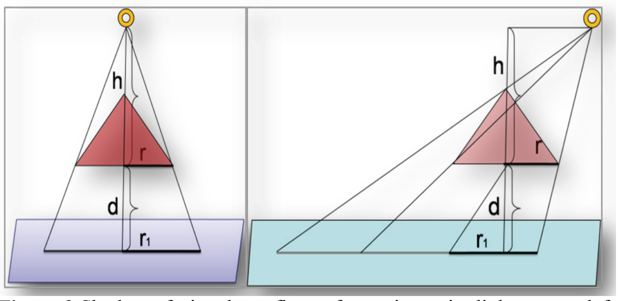

Figure 2 Shadow of triangle on flat surface using point light source: left $x_i = x_p = 0$; right $x_i \neq x_p$

If a is base length, where $r = a/2$, the coordinates of important points for drawing shadow of a triangle are: $P_1(x_{s1}, 0, 0)$, $P_2(x_{s2}, 0, 0)$, $P_3(x_{s3}, 0, 0)$.

$$a_1 = \frac{a \cdot y_i}{y_i - d}, \tag{4}$$

$$x_s = \frac{-d \cdot x_i}{y_i - d}, \tag{5}$$

$$x_{s1} = x_s - \frac{a_1}{2}, \tag{6}$$

$$x_{s2} = x_s + \frac{a_1}{2}, \tag{7}$$

$$x_{s3} = \frac{x_i \cdot (h_p + d)}{d + h_p - y_i}. \tag{8}$$

The following pseudo code is used for drawing triangle shadow.

```
function drawTriangleShadow(object,
lightSource)
begin
shadowBias =
calculateShadowRadius(object,
lightSource);
shadowX = calculateShadowX(object,
lightSource);
shadowY = 0;
shadowX1 = shadowX - shadowBias/2;
shadowX2 = shadowX + shadowBias/2;
dist = calculateDistance(objCenter,
lsCenter);
shadowX3 =
((height+dist)*lightSourceX)/(height
+dist-lightSourceY);
shade = calculateShade(object,
lightSource);
drawTriangle(shadowX1, shadowX2,
shadowX3, shade);
end
```

3.4 Calculating the shadow for the sphere

When creating a shadow of the sphere, we can use the fact that it is generated by rotating a circle around the x axis. As shown in Fig. 3, the shadow of the sphere is also a circle, and for its calculation in this case, we can use the previous equations, given above in the circle section (shown in Fig. 3).

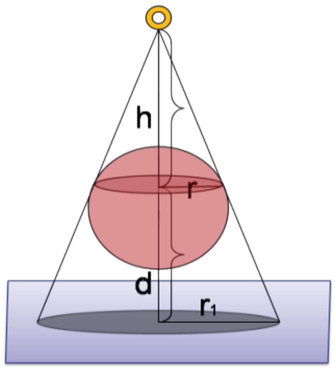

Figure 3 Shadow of the sphere on flat surface using point light source

Following pseudo code shows how the circle shadowing method could be improved and reused for drawing the shadow of the sphere.

```
function drawSphereShadow(object,
lightSource)
begin
objectC =
findCrossSectionCircleByTangents();
drawCircleShadow(objectC,
lightSource);
end
```

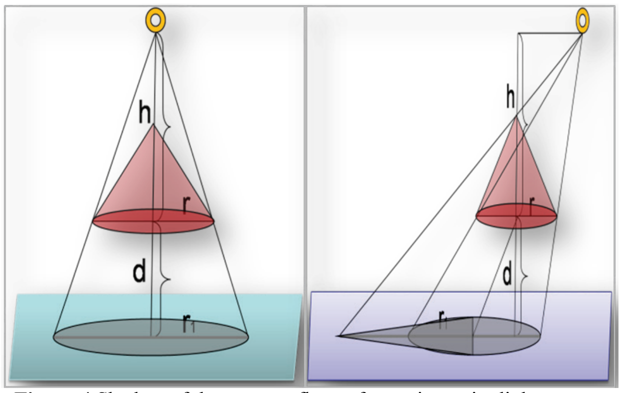

Figure 4 Shadow of the cone on flat surface using point light source: left $x_i = x_p = 0$; right $x_i \neq x_p$

3.5 Calculating the shadow for the cone

When creating a shadow of the cone, we can use the fact that it is generated by rotating a triangle, whose base is, in this case, parallel to the *Z* axis. Three characteristic pieces of information are necessary for shadowing: projection of top of cone, base center projection and the radius of the shadow. The shadow is a union of the bias shadow (circle) and the cross section triangle shadow (triangle). If the light source is just above the cone, the shadow consists of the circle only (Fig. 4). Calculations for the shadow of a circle are already given above, and the triangle part of the shadow is determined by $P_1(0, 0, z_{s1})$, $P_2(0, 0, z_{s2})$, $P_3(x_{s3}, 0, 0)$, where z_{s1} and z_{s2} are calculated similar to x_{s1} and x_{s2}.

The following pseudo code shows how the circle shadowing method could be improved and reused for drawing the shadow of the sphere.

```
function drawConeShadow(object,
lightSource)
begin
objectT = findCrossSectionTriangle();
drawTriangleShadow(objectT,
lightSource);
drawCircleShadow(object,
lightSource);
end
```

4 Experiments - shadowing with multiple lights with the capability of moving

In order to confirm the validity of the given formula and pseudo codes and to check the speed of drawing real time shadows, a simple C++ application that uses OpenGL is created. The application takes into account the above limitations. The application has the following features:

- changing of shadowed object (keys K(circle), L(sphere), T(triangle), U(cone));
- changing the position of camera (keys Y (increasing *y* coordinate), X (increasing *x* coordinate), C (increasing *z* coordinate), A (decreasing y coordinate), S (decreasing *x* coordinate), D (decreasing z coordinate));
- changing the position of the chosen point light source (keys V (increasing y coordinate), B (decreasing y coordinate), N (increasing *x* coordinate), M (decreasing *x* coordinate));
- changing the chosen point light source (G – light source 1, H – light source 2).

The results are given with following screenshots (Fig. 5. and Fig. 6):

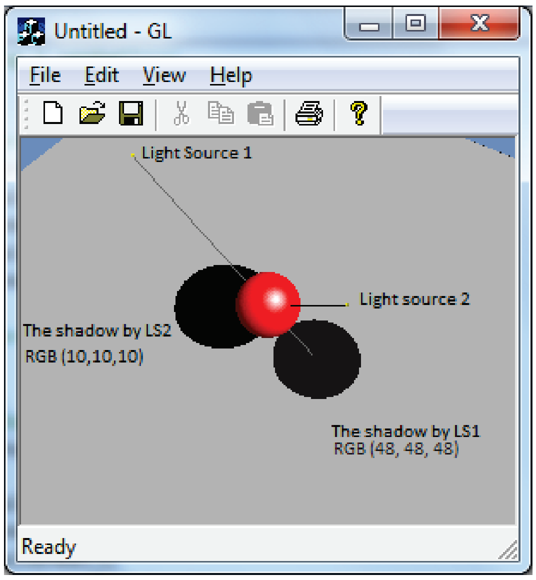

Figure 5 Shading the sphere in OpenGL by two point light sources

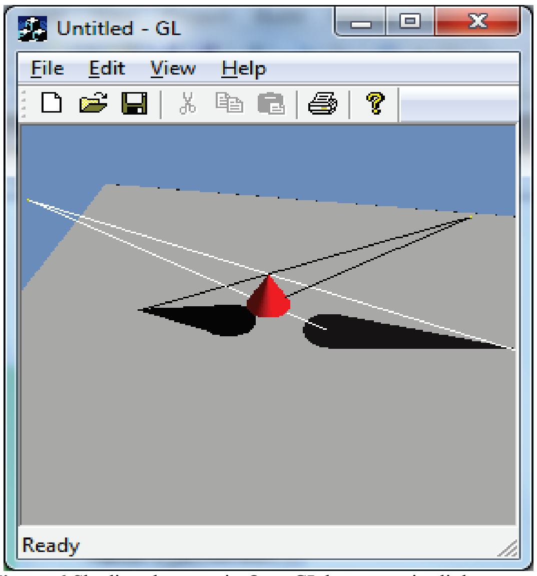

Figure 6 Shading the cone in OpenGL by two point light sources

Also, we made a comparison (observing the rendering time) of our method for generating shadows with two most often used methods which are described in second chapter (projected shadows and ray tracing). The results are given in Fig. 7.

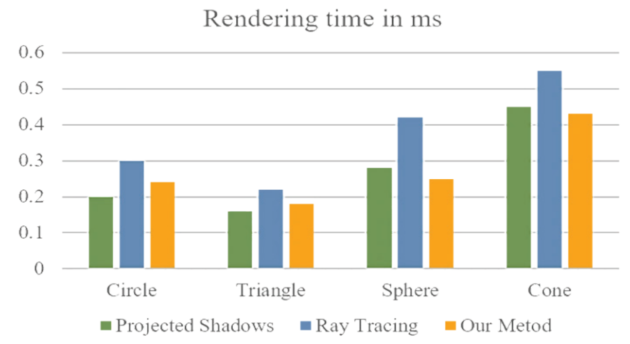

Figure 7 The comparison of methods

This comparison was implemented in the environment of VS 2010 and on a PC with Intel Core2Duo 2.27 GHz CPU and 4 GB RAM.

What can be deduced by observing the graph is that our method is faster than the ray tracing method, but a little slower compared to projected shadows. Since the shadows generated by our method are more realistic, this advantage in speed of method of projected shadows is negligible.

When it comes to more complex structures, our method shows its advantages, because the shadows are generated much faster compared to the other two methods. This is because, in our calculations of shadows, we used the fact that complex objects are made by rotation of primitives, as described above.

5 Conclusions and future work

The fact is that techniques for forming shadows for various objects need not be independently perceived. That should be used to facilitate the calculation of the position of a shadow for objects on the scene.

When forming the shadow of a compound object, it can be considered as a union of the shadows of its components. Rendering the scene would be tough and delayed if we projected all points of the objects on it, for the purpose of shading.

For that reason, it is sufficient to find the lowest number of specific points of a shadow, thus the shadow looks real, and its sketching will be easy and cheap.

If there is a method for drawing a shadow on the position of light source, then a number of light sources can be incremented; all of them could change their position, without making the scene more complex for shading.

In further research, the attention will be focused on discovering methods for shading the residual primitives that are frequently used. Other cases will be studied, such as the occurrence when a shaded item can move, when there is more than one shaded object in the scene at the same time, or when the surface is not flat.

After analyzing all the circumstances in which the shaded object can occur, the result could be a united and effective method of constructing real OpenGL shadows.

Acknowledgements

This paper was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia within the projects TR35026, III44006 and III44007.

6 References

[1] Kolivand, H.; Sunar, M. S.; Jusoh, N. M.; Olufemi, F. Real-Time Shadow Using a Combination of Stencil and the Z-Buffer. // International Journal of Multimedia & Its Applications. 3, 3(2011), pp. 27-38. https://doi.org/10.5121/ijma.2011.3303
[2] McReynolds, T.; Blythe, D. Advanced Graphics Programming Using OpenGL. Elsevier, San Francisco, 2005.
[3] Hu, X.; Qi, Y.; Shen, X. A. Real-Time Anti-Aliasing Shadow Algorithm Based on Shadow Maps. // Pattern Recognition CCPR / Beijing, 2008, pp. 1-5.
[4] McCool, M. D. Shadow Volume Reconstruction from Depth Maps. // ACM Transactions on Graphics, Canada N2L. 19, 1(2000), pp. 1-25. https://doi.org/10.1145/343002.343006
[5] Angel, E.; Shreiner, D. Teaching a Shader-Based Introduction to Computer Graphics. // IEEE Computer Graphics and Applications. 31, 2(2011), pp. 9-13. https://doi.org/10.1109/MCG.2011.27
Haller, M.; Drab, M.; Hartmann, W. A real-time shadow approach for an augmented reality application using shadow volumes. // Proceedings of the ACM symposium on Virtual reality software and technology VRST'03 / Osaka, 2003, pp. 56-65. https://doi.org/10.1145/1008653.1008665
[6] Akenine-Möller, T.; Haines, E.; Hoffman, N. Real-Time Rendering. 3rd ed. CRC Press, Massachusetts, 2008.
[7] Fernando, R. GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics. Addison-Wesley Professional, London, 2004.
[8] Fernandez, S.; Bala, K.; Fernando R.; Greenberg, D. Adaptive Shadow Maps. // Proceedings of the 28th annual conference on Computer graphics and interactive techniques / New York, 2001, pp. 387-390.
[9] Dachsbacher, C.; Drettakis, G.; Stamminger, M. Perspective Shadow Maps. // ACM Transactions on Graphics (TOG). 21, 3(2003), pp. 557-562. https://doi.org/10.1145/566570.566616

[10] Martin, T.; Tan, T. Anti-aliasing and Continuity with Trapezoidal Shadow Maps. // Eurographics Symposium on Rendering / Norköping, 2004, pp. 1-9.

[11] Parallel-Split Shadow Maps on Programmable GPUs. // GPU Gems 3 / Zhang, Fan, Hanqiu Sun, and Oskari Nyman. Boston : Addison-Wesley Professional, 2007. pp. 203-237.

[12] Scherzer, D. Light Space Perspective Shadow Maps. // Eurographics Symposium on Rendering / Norköping, 2004, pp. 143-151.

[13] Sukthankar, R.; Cham, T. J.; Sukthankar, G. Dynamic shadow elimination for multi-projector displays. // Proceedings of Computer Vision and Pattern Recognition / Kauai, 2001, pp. II-151-158.
Raskar, R.; Brown, M.; Yang, R.; Chen, W.; Welch, G.; Towles, H.; Seales, B.; Fuchs, H. Multi-projector displays using camera-based registration. // Proceedings of the conference on Visualization / San Francisco, 1999, pp. 161-522. https://doi.org/10.1109/VISUAL.1999.809883

[14] Jaynes, C.; Webb, S.; Steele, R. M.; Brown, M.; Seales, W. B. Dynamic shadow removal from front projection displays. // Proceedings of the conference on Visualization / San Diego, 2001, pp.175-182.
https://doi.org/10.1109/VISUAL.2001.964509

[15] Pljaskovic, A.; Avdic, Dz.; Spalevic, P.; Rancic, D. Sphere and Cone Composite Realtime Shadows in OpenGL. // Information and Communication Technology Forum / Sarajevo, 2013, pp. 133-137.

[16] Cham, T. J.; Rehg, J. M.; Sukthankar, R.; Sukthankar, G. Shadow elimination and occluder light suppression for multi-projector displays. // Computer Vision and Pattern Recognition, IEEE Computer Society Conference / Madison, 2003, pp. II-513.

[17] Kessenich, J.; Baldwin, D.; Rost, R. The OpenGL Shading Language. Version, The Khronos Group Inc, http://www.cse.chalmers.se/edu/year/2010/course/TDA361/GLSLangSpec.Full.1.30.08.pdf (07.08.2008)

[18] Shadow mapping for hemispherical and omnidirectional light sources. // Advances in Modelling, Animation and Rendering / Brabec, Stefan, Thomas Annen, and Hans-Peter Seidel, London : Springer, 2002, pp. 397-407.

[19] Liu, N.; Pang, M. A Survey of Shadow Rendering Algorithms: Projection Shadows and Shadow Volumes. // Second International Workshop on Computer Science and Engineering / Qingdao, 2009, pp. 488-492.
https://doi.org/10.1109/WCSE.2009.716

[20] Wyman, C.; Hansen, C. D. Penumbra maps: Approximate soft shadows in real-time. // Proceedings of Eurographics Symposium on Rendering/ Leuven, 2003, pp. 202-207.

Authors' addresses

Dženan Avdić
State University of Novi Pazar,
Vuka Karadžića bb,
36300 Novi Pazar, Serbia
dzavdic@np.ac.rs

Dejan Rančić
Faculty of Electrical Engineering,
Aleksandra Medvedeva 14,
18000 Nis, Serbia
dejan.rancic@elfak.ni.ac.rs

Petar Spalević
Faculty of Technical Sciences,
Kneza Miloša 7
38220 Kosovska Mitrovica, Serbia
petar.spalevic@pr.ac.rs

Aldina Avdić
State University of Novi Pazar,
Vuka Karadžića bb,
36300 Novi Pazar, Serbia
apljaskovic@np.ac.rs

Edin Dolićanin
State University of Novi Pazar,
Vuka Karadžića bb,
36300 Novi Pazar, Serbia
edolicanin@np.ac.rs