

# The extensions of revolution-bump mapping

## ABSTRACT

*Creating 3D computer-generated surfaces has long been a difficult challenge in computer graphics, particularly when portraying massive landscapes with extremely detailed surfaces in real-time. Despite significant advances in computer vision in recent years, there is still a great demand for improved realism and the capacity to edit computer-generated 3D surfaces in real-time. We propose three scalable and faster algorithms for creating extended, beveled, and chamfered patterns using only two textures and a simple shape box. The proposed techniques produce visually pleasing results in real-time while retaining optimal rendering performance and without increasing the mesh density of the shape box.*

## KEY WORDS

Per-pixel revolution mapping, image-based modeling, rendering technique, bump mapping, ray-tracing

Anouar Ragragui <sup>1</sup>   
Adnane Ouazzani  
Chahdi <sup>2</sup>   
Akram Halli <sup>3</sup>   
Khalid Satori <sup>2</sup>  
Hicham El Moubtahij <sup>4</sup> 

<sup>1</sup> Abdelmalek Essaadi University, LSA Laboratory National School of Applied Sciences (ENSAH), Tetouan, Morocco

<sup>2</sup> Sidi Mohamed Ben Abdellah University, LISAC Laboratory Faculty of Science Dhar El Mahraz, Fez, Morocco

<sup>3</sup> Moulay-Ismaïl University, OMEGA-LERES Laboratory FSJES-UMI, Meknes, Morocco

<sup>4</sup> Ibn Zohr University, Modeling, Systems, and Technologies of Information Team, High School of Technology, Agadir, Morocco

Corresponding author:

Anouar Ragragui

e-mail:

[a.ragragui@uae.ac.ma](mailto:a.ragragui@uae.ac.ma)

First received: 5.10.2021.

Revised: 18.11.2021.

Accepted: 23.11.2021.

## Introduction

Real-time rendering using traditional methods is still hampered by a large amount of graphics primitives (polygons and vertices) that the graphics card must calculate. This has an impact on the interaction of 3D scenes, particularly those with complex 3D objects. Image-based rendering techniques (IBMR) gained traction as an alternative to traditional polygon-based rendering approaches because they can show 3D surfaces in real-time at a low-performance cost while

avoiding mesh densification. These approaches use textures to store a collection of geometry-related data that will be retrieved during the ray-tracing algorithm stage, hence bypassing the computation of geometry.

Our research focuses on revolution-bump mapping, a technique that falls under the category of image-based modeling and rendering approaches (Halli et al., 2010; Ragragui et al., 2018). Because it uses a simple shape box and two textures. The first texture is used to produce the revolution surface, while the second is utilized to provide

---

the microrelief effect. We were able to greatly refine this approach such that it could generate extended, beveled, and chamfered revolution objects while preserving the features on their surfaces. Our approaches take advantage of the fact that the surface revolution is modified without the need to recalculate the texture (shape map). On the other hand, we can control in real-time the effect of extension, bevel, and chamfer parameters.

## Related Work

Texture mapping is a method of adding realism to a computer-generated 3D shape introduced in (Catmull, 1974; Heckbert, 1986). This technique is the simplest and oldest of image-based techniques. It aims to find the relation between the texture elements defined in the two-dimensional texture space and the surface defined in a three-dimensional space. In effect, it is a process that takes a surface and changes its appearance at each location using an image, function, or other data set. It should be noted that this technique was extended by Blinn and Newell (Blinn & Newell, 1976).

In 1978, Blinn introduced, in the article (Blinn, 1978), a method to achieve what is called bump mapping. The latter simulates the wrinkles of a surface without the need to modify the geometry of the 3D model. The normal of a given surface is perturbed according to the partial derivatives of the applied texture, called the height map. This texture is a simple grayscale image, which can be seen as an elevation map. The perturbed normal is then used instead of the original normal when shading the surface according to the Blinn-Phong model (Blinn, 1977). This method changes the appearance of wrinkles and micro-reliefs seen on the surface of 3D models. An improvement of the bump mapping is the normal mapping (Percy, Airey & Cabral, 1997). Its principle is to use a texture to save not the variation, but the coordinates of the normal for each fragment and then use them in the shading model. The principle is to perform the calculations directly in the tangent space. This space is defined for each face of the mesh and keeps the normal unchanged.

Displacement mapping is the first method to use the height map to add detail to a surface (Cook, 1984; Lee, Moreton & Hoppe, 2000). It masked almost all of the defects in the bump mapping and the normal mapping, as the surface geometry is completely changed instead of just disturbing the normal. This approach is based on the subdivision and the displacement of the sub-polygons of the surface along the normal to the vertices based on the distances obtained from the displacement map. The result is a more realistic rendering where the displaced geometry is also visible in the silhouette. The subdivision of a base surface considerably increases the number of graphic primitives (vertices and

polygons) that the graphic map must manage, and this influences the execution time in the rendering stage. From there, research is focused on alternative rendering methods based on mesh simplification and ray tracing algorithms. The techniques presented in (Gumhold & Hüttner, 1999; Doggett & Hirche, 2000) took a different approach, their methods truly modified the geometry in such a way as to minimize the number of triangles rendered as a function of viewpoint. They proposed to use an adaptive subdivision for the displacement map to limit the number of polygons generated.

Per-pixel displacement mapping is an interesting improvement of the per-vertex displacement mapping technique introduced in (Patterson, Hoggar & Logie, 1991), the strong point of this approach is that it increases the realism of the surface without densifying the mesh. This method aims at solving the bottleneck caused by the very large number of graphics primitives sent by the vertex displacement mapping to the graphics processor (polygons, 3D points, normals, texture coordinates...).

Parallax mapping not only disrupts the normals but also changes the texture coordinates used, without changing the geometry of the object (Kaneko et al., 2001). Its principle is quite simple, it aims to shift the texture coordinates to perform an approximate search for the point of intersection between the view radius, expressed in tangent space, and the relief stored in the height map. To overcome the problems of simple parallax mapping, Welsh introduced parallax mapping with offset limitation (Welsh, 2004). A better improvement of the parallax mapping is called the Steep Parallax Mapping. It consists in assuming that the surface is always flat, but its normal vector can be arbitrary (McGuire & McGuire, 2005).

Binary search is a method specifically designed to converge quickly to the point of intersection to ensure fast rendering. This technique assumes that the actual intersection point is either at the top or bottom of the relief depth value retrieved from the displacement map (Policarpo, Oliveira & Comba, 2005).

Ray tracing uses the viewing ray to determine the point of intersection. It is also called linear search. It has been used alone in parallax mapping (McGuire & McGuire, 2005), as well as in (Policarpo, Oliveira & Comba, 2005) and (Tatarchuk & Natalya, 2006) as a first step.

Relief mapping is among the popular methods of real-time rendering since it quickly converges to the point of intersection that lies between the view ray and the relief (Policarpo, Oliveira & Comba, 2005; Policarpo & Oliveira, 2006). This is an extension of another technique called relief texture mapping (Oliveira, Bishop & McAllister, 2000). While this method will function and render satisfactorily in most cases, it may fail in some special situations. This problem will be solved

in (Oliveira & Policarpo, 2005), whose principle is to reinforce each vertex of the polygonal model with two coefficients representing a quadric surface that is locally close to the geometry of the object (Jean, 2002). This quadratic surface is used to produce correct renderings of objects and their silhouettes. Another improvement of relief mapping is proposed in (Ouazzani Chahdi et al., 2018) which is called dynamic relief mapping.

Sphere tracing uses spheres to converge most quickly to the intersection point. It was first introduced in (Hart, 1996) and then it was used for height map search using ray tracing (Donnelly, 2005). It is based on two main elements, namely the distance map and an iterative algorithm, whose goal is to find the first point of intersection between the view radius and the relief. Further improvements are presented in (Fabbri et al., 2008; Gustavson & Strand, 2011), whose objective is to improve the algorithm for computing the distance map.

Cone tracing is a technique that calculates the empty space, during the pre-processing phase, around each pixel of the depth map as an open cone at the top and then stores its ratio in a texture called a cone map. Subsequently, the cone map is used when searching for the intersection to converge more quickly on the intersection point without the risk of avoiding it. They exist in two versions, the first is a conservative technique (Dummer, 2006) and the second is a relaxed technique (Policarpo & Oliveira, 2007) coupled with a binary search. These two versions have been improved in (Halli et al., 2008). These improvements consist in using  $O(n)$  linear algorithms instead of  $O(n^2)$  quadratic algorithms to compute the conservative cone and the relaxed cone. Then calculate and store the radius of the cones instead of their ratio to have cone angles on the order of  $\pi/2$  rather than  $\pi/4$ . Finally extend the technique to support non-square textures by using elliptical cone rectification during the rendering step.

Halli et al. introduced a new image-based approach for rendering revolved surfaces (Halli et al., 2009; Halli et al., 2010). Indeed, revolution mapping and extrusion mapping are based on a single RGBA texture that stores all the related data to the geometry. Then, the resulting texture is mapped on a shell box using 3D texture coordinates. This technique allows rendering full models and limits considerably the number of graphic primitives constituting the complex scenes. Further improvements of these techniques are presented in (Ragragui et al., 2018; Ragragui et al., 2020; Ouazzani Chahdi et al., 2021).

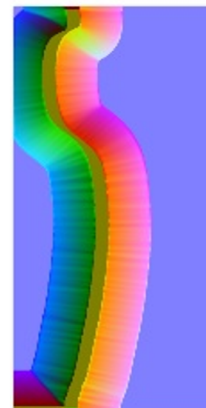
## Revolution-bump mapping

The revolution-bump mapping uses a 2D binary form stored in the shape map, where only the zero-valued pixels constitute the base shape of the pattern that will

be revolved during the rendering stage. This technique is based on ray tracing, and uses the Euclidean Distance Transform (EDT) computed from the base shape to skip the empty space and speed up the search for the intersection point between the viewing ray and the generated surface (Danielsson, 1980; Fabbri et al., 2008; Gustavson & Strand, 2011). It also relies on bump mapping to add realism to 3D objects by adding the illusion of microreliefs. The revolution-bump mapping makes it possible to considerably limit the number of graphic primitives constituting the complex scenes.

## Revolution mapping

Per-pixel revolution mapping is an image-based modeling and rendering technique. It consists of generating very convincing surfaces of revolution without polygonization effect which are displayed interactively. The principle is to generate virtual surfaces using only a shape map that contains the geometry data of the basic form (Figure 1). This geometry represents the revolution of the basic form plated on the shape box using the texture coordinates (Figure 2). These coordinates are calculated using cylindrical or spherical projection. The revolution mapping is based on three main elements: the shape map, the ray tracing algorithm, and the shape box.

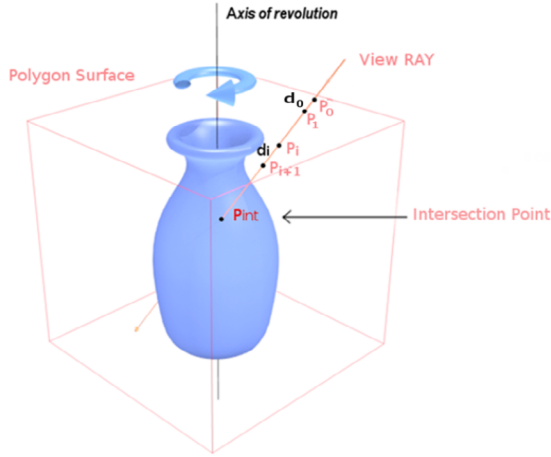


» **Figure 1:** *The shape map that will be sent to the graphic card*

**Shape map** is an RGBA texture that contains the data needed for the revolution mapping algorithms (Figure 1). The alpha channel is used to store the basic form represented by a binary image. The blue channel is used to store the distance map. Finally, the red and green channels contain respectively the x and y components of the gradient which will be used to determine the coordinates of the normal.

**Ray tracing** algorithm consists of searching for the intersection of the viewing ray with the revolved surface by using the distances stored in the shape map (see Figure 2). Let  $(u, v)$  be the coordinates of the current pixel and  $p_o$  be the starting point of the search with the coordi-

nates  $(x_o, y_o, z_o) = (u, v, O)$ , and let  $\vec{V}$  be the normalized view direction determined by going from the viewpoint to the starting point  $p_o$ , all of which is expressed in the texture space associated with the current pixel.



» **Figure 2:** Ray tracing process associated with revolution mapping. At each iteration, a circle tracing is performed to converge quickly and without the risk of skipping the first intersection.

At each iteration, the minimum distance  $d_i$  between the point  $p_i$  and the revolved object is extracted from the blue channel of the shape map and a circle tracing is performed to advance to the intersection point without the risk of skipping the first intersection. The next point  $p_{i+1}$  is determined by the formula:

$$p_{i+1} = p_i + d_i \cdot \vec{V}_p \quad (1)$$

To retrieve the distance  $d_i$  between the current position and the base shape that is stored in the alpha channel of the shape map, the revolution algorithm uses the coordinates  $(s_i, t_i)$  (Figure 3). Using the following formula:

$$(s_i, t_i) = (\|\vec{R}_i\|, z_i) \quad (2)$$

With :

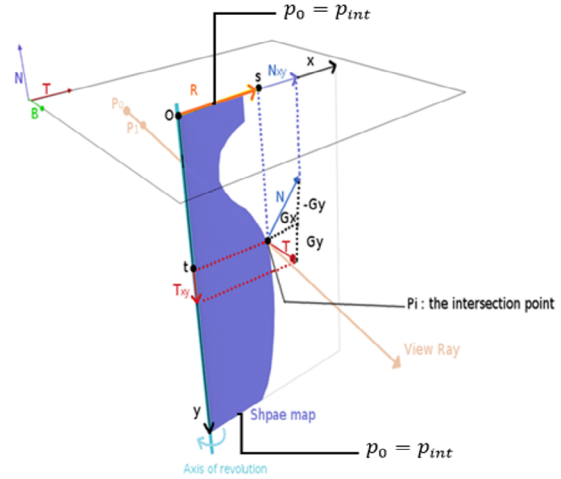
$$\vec{R}_i = (x_i - O_x, y_i - O_y)$$

From Figure 3, the normal is obtained from the components  $(G_{int_x}, G_{int_y})$  of the gradient unit stored in the red and green channels of the shape map:

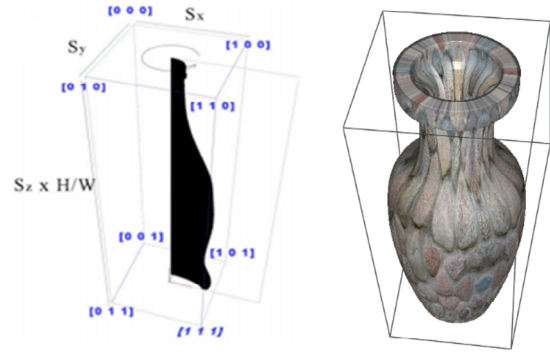
$$\vec{N}_{int} = \left( G_{int_x} \frac{R_{int_x}}{\|\vec{R}_{int}\|}, G_{int_y} \frac{R_{int_y}}{\|\vec{R}_{int}\|}, -G_{int_z} \right) \quad (3)$$

The shape box will be created from the shape map, as shown in Figure 4.a.  $W$  and  $H$  are the dimensions of the shape map. As well as  $S$  is a real-time editable vector, representing the scale of the shape box in the scene frame. The values in blue are the texture coordinates.

From figure 4.b we notice that the model is displayed in its entirety with a correct rendering of the silhouette.



» **Figure 3:** The calculation of the normal and tangent at the intersection point.



» **Figure 4:** Creating the shape box. (a) Shape box that corresponds to the revolved object where the shape is placed vertically on the surface. (b) Rendering of a revolved object using the shape box.

## 2. Calculation of the perturbed normal

The problem of revolution mapping is that the revolved object is created without any microrelief effect, so it does not take into account their realism. From this observation, the solution is to combine bump mapping with revolution mapping, which is called revolution-bump mapping (Ragragui et al., 2018).

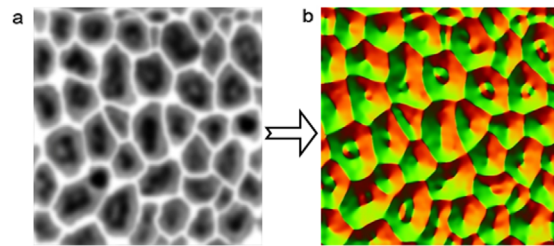
Revolved surfaces are created without any polygonal meshes and do not possess any parametric equation. Therefore, we must compute the tangent space associated with each intersection point. Therefore, the perturbed normal  $\vec{N}'(u, v)$  is calculated from the sum of the normal vector  $\vec{N}(u, v)$  and the displacement vector  $\vec{D}(u, v)$  according to the following equation:

$$\vec{N}'(u, v) = \frac{\vec{N}(u, v) + \vec{D}(u, v)}{\|\vec{N}(u, v) + \vec{D}(u, v)\|} \quad (4)$$

With:

$$\vec{D}(u, v) = \left( \frac{a \cdot \partial H(u, v)}{\partial u} (\vec{N} \wedge \vec{B}) - \frac{a \cdot \partial H(u, v)}{\partial v} (\vec{N} \wedge \vec{T}) \right) \quad (5)$$

The parameter  $a$  is a factor that can be controlled in real-time and is used to control the depth scale of the microrelief. The partial derivatives  $\partial H(u, v)$  are computed along  $u$  and  $v$  of the height map  $H(u, v)$  in the preprocessing stage. Then they are saved in the red and green channels of a texture that we call the depth map (Figure 5).



» **Figure 5:** The calculation of the partial derivatives of the height map. (a) The height map. (b) The depth map.

This map will be sent to the graphic card during the rendering stage. Finally, the tangent can be deduced graphically using Figure 3:

$$\vec{T}_{int} = \left( Gy_{int} \frac{Rint_y}{\|\vec{R}_{int}\|}, -Gy_{int} \frac{Rint_x}{\|\vec{R}_{int}\|}, -Gx_{int} \right) \quad (6)$$

While the vector  $\vec{B}_{int}$  is obtained by the vector product of the two vectors  $\vec{N}_{int}$  and  $\vec{T}_{int}$  using the formula:

$$\vec{B}_{int} = \vec{T}_{int} \wedge \vec{N}_{int} \quad (7)$$

## The extensions of revolution-bump mapping

### Extended revolution-bump mapping

Since the level lines of the distance map are extended forms of the base form, we can use them to create a surface of outside revolution. To do this, we simply replace the distance in the revolution-bump algorithm with the distance to the extended shape. Formula (1) becomes:

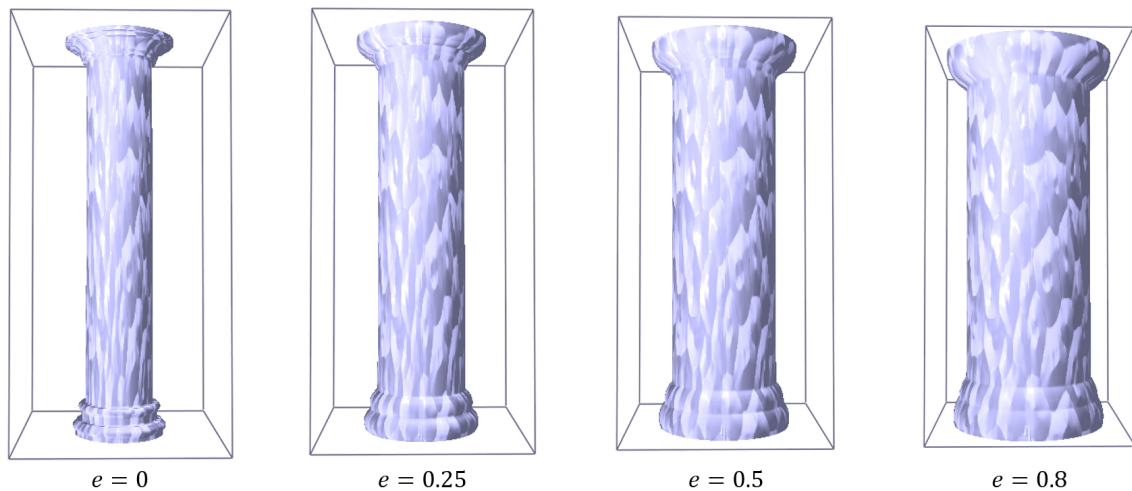
$$p_{i+1} = p_i + \max(0, d_i - e) \cdot \vec{V}_p \quad (8)$$

Where  $e$  is a parameter that can be modified in real-time, allow modulating the extension effect.

Even though the extended revolution-bump mapping is slightly slower than the revolution-bump mapping, it can be very useful for smoothing shapes as shown in Figure 6. It should be noted that for the extended revolution-bump mapping, the normal vector  $\vec{N}_{int}$  and the tangent vector  $\vec{T}_{int}$  remain the same as for the revolution-bump mapping, since the level lines of the distance map are extended forms of the base form (Figure 7).

## 2. Beveled revolution-bump mapping

The bevel consists of creating an outwardly extended revolution that varies as a function of depth. We start by searching for the intersection based on the distance to the original form by using formula (8) (because we are combining the bevel with the extended revolution-bump mapping). During this search, it



» **Figure 6:** Rendering of an object using the extended revolution-bump mapping. These images are taken in real-time by changing only the value of the parameter  $e$ .

is necessary to check if the current position is inside the geometry by using the following difference:

$$\Delta d_i = \max(0, d_i - e) - b \cdot z_i \quad (9)$$

Where  $b$  is a parameter to control the effect of the bevel and  $\Delta d_i$  denotes the width of the bevel at  $z_i$ . The intersection point is determined according to the following system:

$$p_{i+1} = \begin{cases} p_i + \max(0, d_i - e) \vec{V}_p & \text{If } \Delta d_i > 0 \\ p_i + d_i \cdot \vec{V}_p & \text{otherwise} \end{cases} \quad (10)$$



» **Figure 7:** Illustration of an example of the distance map that is stored in the blue channel of the shape map.

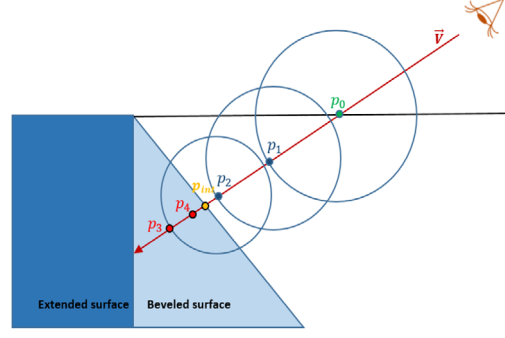
If  $\Delta d_i$  is positive, we calculate the next position  $p_{i+1}$  using equation (10), because the point  $p_i$  in the current step is always outside the geometry. If  $\Delta d_i$  is negative, we get the point which is inside the beveled surface like the point  $p_3$  in Figure 8. Then we proceed to the binary refinement by successively dividing the last distance  $\max(0, d_{i-1} - e)$  by 2 to converge to the intersection point  $p_{int}$  of the geometry with the viewing ray  $\vec{V}$  (Figure 8).

In contrast to the extended revolution-bump mapping, the  $z$  coordinates of the normal and  $y$  coordinates of the tangent must be changed. However, they remain uniform and are obtained by rotation of the gradient (Figure 9). The coordinates of the normal in this case become:

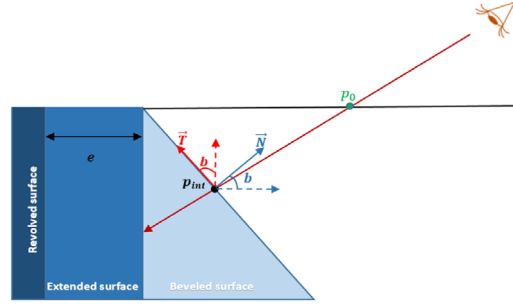
$$\vec{N}_{int} = \left( G_{int_x} \frac{R_{int_x}}{\|\vec{R}_{int}\|}, G_{int_x} \frac{R_{int_y}}{\|\vec{R}_{int}\|}, -b \cdot G_{int_y} \right) \quad (11)$$

And the coordinates of the tangent are :

$$\vec{T}_{int} = \left( G_{y_{int}} \frac{R_{int_y}}{\|\vec{R}_{int}\|}, -b \cdot G_{y_{int}} \frac{R_{int_x}}{\|\vec{R}_{int}\|}, -G_{x_{int}} \right) \quad (12)$$



» **Figure 8:** The intersection of the viewing ray with the beveled surface.



» **Figure 9:** Diagram for the calculation of the normal vector and the tangent vector in the case of the beveled surface.

These two vectors, normal and tangent, must then be normalized before the perturbation of the normal  $\vec{N}_{int}$  by using formulas (4) and (5). As illustrated in the Figure 10, this enhancement enables for real-time modification of the revolution surface.

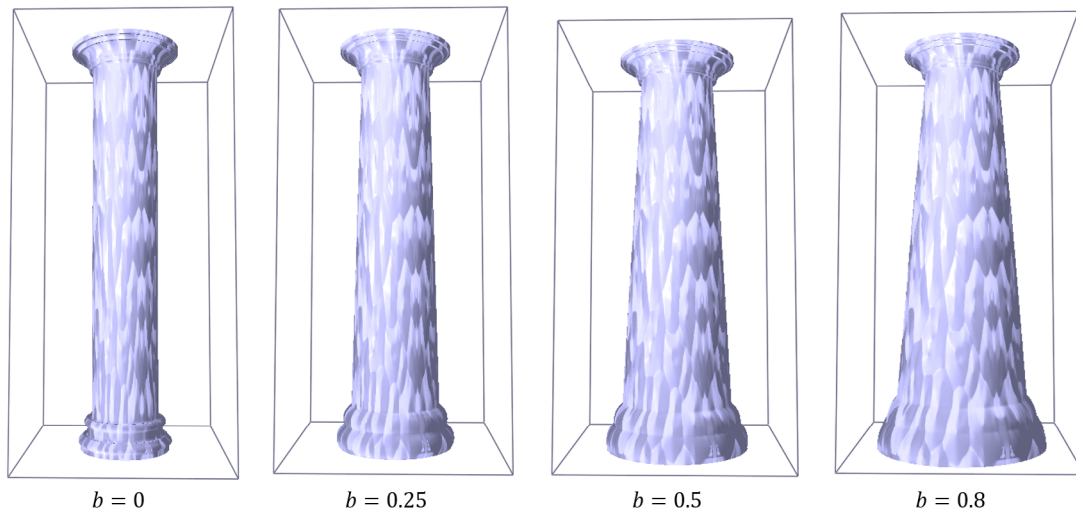
## Chamfered revolution-bump mapping

The revolution-bump mapping with chamfer consists of limiting the effect of the bevel to a certain depth value to have edges with chamfer (Figure 11). To do this, simply replace formula (9), which defines the test above/below the beveled revolution-bump mapping, by the following formula:

$$\Delta d_i = \max(0, d_i - e) - b \cdot \min(z_i, c) \quad (13)$$

With  $c$  is a parameter that allows modulating the chamfer effect. The modification of the  $z$ -coordinate of the normal must also be limited to the depth  $c$ . Note that the search for the intersection point is done according to the formula (10). For the revolution-bump mapping with chamfer, the normal and tangent will be equivalent to formulas (11) and (12) respectively if its depth is greater than the value of  $c$  in formula (13). Otherwise, these vectors are equivalent to one of the extended revolution-bump mapping.





» **Figure 10:** Rendering of an object using the beveled revolution-bump mapping. These images are taken in real-time by changing the value of the **b** parameter.

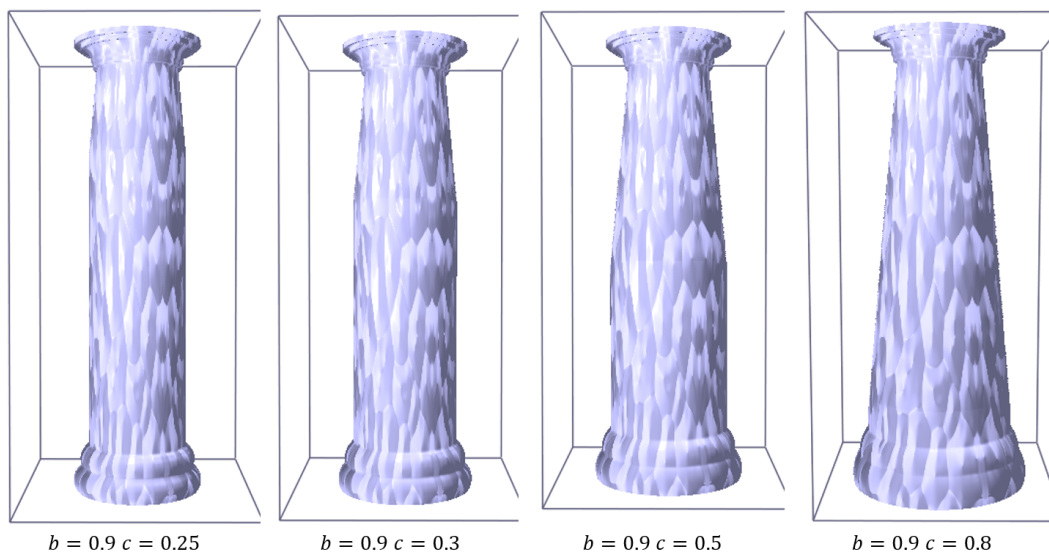
## Results and Discussion

To measure the rendering speed of the different techniques discussed in this paper, we have implemented the preprocessing stage of the algorithm that will compute the shape map and the displacement map in C++. After that, at the rendering stage, we have exploited the two programmable units of the graphics card (GPU), namely the Vertex Shaders and the Fragment Shaders using OpenGL associated with its parallel processing programming language GLSL. The measurements and figures presented in this section were performed using an Intel Core i7-3612QM 2.10GHz CPU architecture with 8GB of RAM and a GeForce GT 630M graphics card with 1024Mb of memory. Note that before starting the rendering, we send to the graphics card the shape map and

the displacement map created during the preprocessing stage as well as the coordinates of the shape box.

The images of the figures used to compare performance are rendered with textures resolution equal to or greater than 512×512, and with the microrelief depth scale parameter  $a=1$ . In addition, these images are screenshots taken during the test, and that the shape box occupies most of the screen. Note that the total number of iterations for the intersection search is 20. For the last two techniques, namely the beveled revolution-bump mapping and chamfered revolution-bump mapping, we set the total number of iterations to 10 for the binary search.

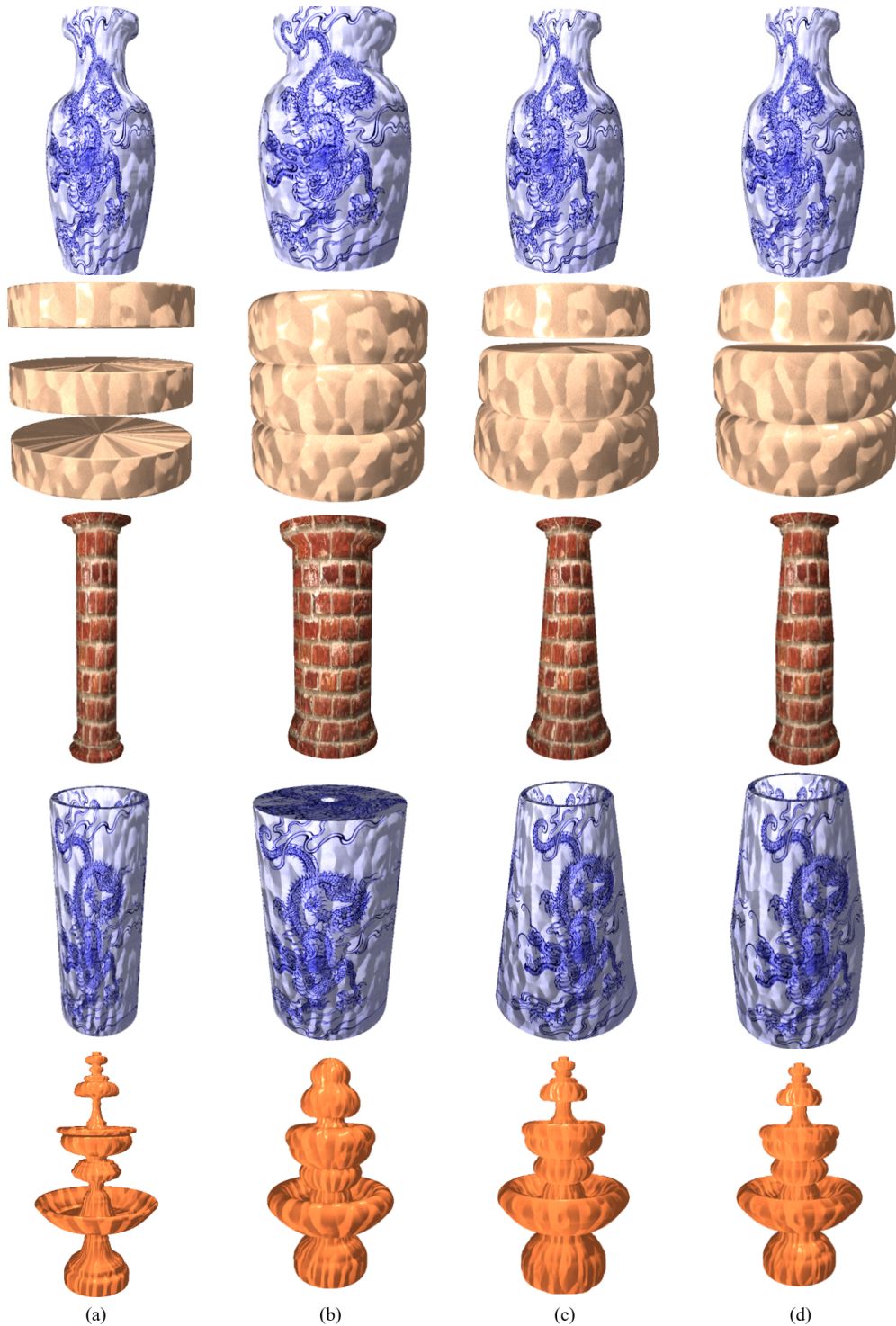
Figures 6, 10, and 11 show the techniques discussed in this paper. We notice that the images rendered by the



» **Figure 11:** Rendering of an object using chamfered revolution-bump mapping. The images are taken in real-time by setting the value of **b** to 0.9 and changing the value of the parameter **c**.

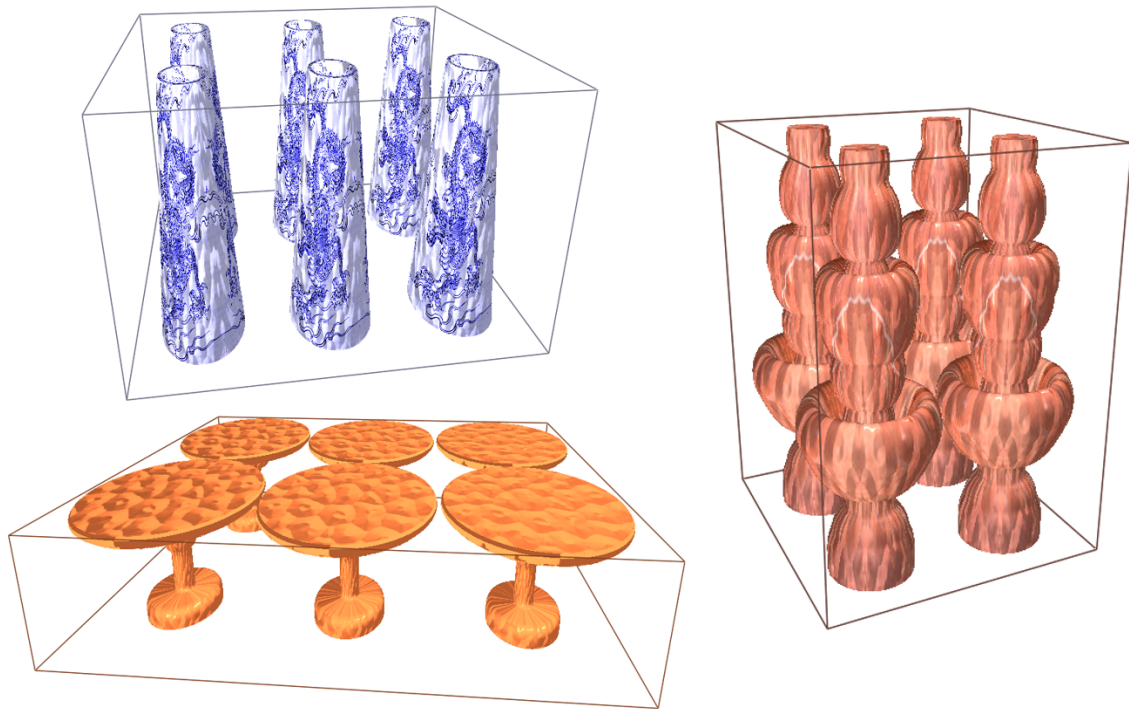
approaches proposed in this paper present realistic surfaces of revolution and that we can change them in real-time. Note also that the models created by these techniques present microrelief on their surfaces. Figure 12 shows a comparison between some models created by the different techniques discussed in this paper. These objects are obtained from a low-density mesh (shape box), on which a shape map and a displacement map

have been plated. We can notice the variety of objects that can be created by these techniques and the change of the surface in real-time that we can control as well as the important number of graphical primitives (vertices and polygons) that can be avoided. The approaches discussed in this paper allow rendering revolved objects with a microrelief effect and without mesh densification.



» **Figure 12:** Some objects were created using the techniques discussed in this paper. (a) Revolution-bump mapping. (b) Extended revolution-bump mapping. (c) Beveled revolution-bump mapping. (d) Chamfered revolution-bump mapping.






» **Figure 13:** Models rendered using extended, beveled and chamfered revolution-bump mapping with repetition.

Figure 13 shows that the approaches presented in this paper can be also applied for the technique of revolution with repetition.

Table 1 shows a comparison of the rendering speed of the extended revolution-bump mapping with different values of the parameter  $e$  that extends the revolution surface, as well as the view on which the speed calculation is performed. We notice that the rendering speed of our approach decreases. This slowdown is due to the treatment concerning the enlargement of the generated surface, which is quite normal.

**Table 1**

Comparison of rendering speed in frames per second using the extended revolution-bump technique with different values of parameter  $e$ .

Model	Screen shot	$e = 0,25$	$e = 0,5$	$e = 0,8$
	800 x 600	180	170	160
	1366x706	150	145	130

Concerning the beveled revolution-bump mapping, Table 2 shows that the speed increases even if we increase the depth scale parameter  $b$ . This is because every time the value of  $b$  increases, the search interval of the intersection decreases, hence the number of iterations decreases.

**Table 2**

Comparison of the rendering speed in frames per second of the beveled revolution-bump mapping using different values of the parameter  $b$ .



Model	Screen shot	$b = 0,25$	$b = 0,5$	$b = 0,8$
	800 x 600	200	210	230
	1366x706	190	195	200

Table 3 shows a comparison of the rendering speed of the chamfered revolution-bump mapping by setting the value of  $b = 0.8$  and by varying the values of the parameter  $c$ . It can be seen that the speed increases when we increase the value of the parameter  $c$ . This increase is because we reduced the search interval of the intersection.

**Table 3**

Comparison of rendering speed in frames per second using different values of  $c$  and setting the value  $b = 0.8$  using the chamfered revolution-bump technique.

Model	Screen shot	$c = 0,25$	$c = 0,5$	$c = 0,7$
	800 x 600	195	209	216
	1366x706	174	176	182

## Conclusion

In this paper, we presented three new algorithms of the revolution-bump mapping technique that allow the creation of extended, beveled, and chamfered objects.

The proposed algorithms allow real-time control while maintaining the interactivity and visual richness of the created objects. The proposed algorithms allow real-time control while maintaining the interactivity and visual richness of the created objects. In addition, they avoid saturating the graphics pipeline, which can be caused by processing a very large number of vertices and polygons.

Extended, beveled, and chamfered revolution-bump mapping represent an interesting solution capable of providing control of the revolution surface and appreciable rendering quality. These techniques derive their advantages from the fact that they bypass the mesh densification because they use only a simple box, the tangent space associated with each intersection point, and two textures. The first texture is used to generate the surface of revolution while the second one is used to add the microrelief effect. The proposed improvements respect two objectives, namely the required rendering speed and the display of the revolution models in a very convincing way.

## References

Blinn, J. F. (1977) Models of light reflection for computer synthesized pictures. In: *Proceedings of the 4th annual conference on Computer graphics and interactive techniques, SIGGRAPH'77, 20-22 July 1977, San Jose, California*. New York, Association for Computing Machinery. pp. 192–198. Available from: doi: 10.1145/563858.563893

- Blinn, J. F. (1978) Simulation of wrinkled surfaces. *ACM SIGGRAPH Computer Graphics*. 12 (3), 286–292. Available from: doi: 10.1145/965139.507101
- Blinn, J. F. & Newell, M. E. (1976) Texture and reflection in computer generated images. *ACM SIGGRAPH Computer Graphics*. 10 (2), 266–266. Available from: doi: 10.1145/965143.563322
- Catmull, E. E. (1974) *A subdivision algorithm for computer display of curved surfaces*. PhD thesis. The University of Utah.
- Cook, R. L. (1984) SHADE TREES. In: *Proceedings of the 11th annual conference on Computer graphics and interactive techniques, SIGGRAPH'84, 23-27 July 1984, Minneapolis, Minnesota*. New York, Association for Computing Machinery. pp. 223–231.
- Danielsson, P. E. (1980) Euclidean distance mapping. *Computer Graphics and Image Processing*. 14 (3), 227–248. Available from: doi: 10.1016/0146-664X(80)90054-4
- Doggett, M. & Hirche, J. (2000) Adaptive view dependent tessellation of displacement maps. In: *Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware, HWWS'00, 21-22 August 2000, Interlaken, Switzerland*. New York, Association for Computing Machinery. pp. 59–66. Available from: doi: 10.1145/346876.348220
- Donnelly, W. (2005) Per-Pixel Displacement Mapping with Distance Functions. In: Pharr, M. (ed.) *GPU Gems 2: Programming Techniques For High-Performance Graphics And General-Purpose Computation*. London, Addison-Wesley Professional, pp. 123–137.
- Dummer, J. (2006) *Cone step mapping: An iterative ray-heightfield intersection algorithm*. Available from: <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Cone+Step+Mapping:+An+iterative+ray-heightfield+intersection+algorithm#0> [Accessed: 20th October 2021]
- Fabbri, R., Costa, L. F., Torelli, J. C., & Bruno, O. (2008) 2D Euclidean distance transform algorithms. *ACM Computing Surveys*. 40 (1), 1–44. Available from: doi: 10.1145/1322432.1322434
- Gumhold, S. & Hüttner, T. (1999) Multiresolution rendering with displacement mapping. In: *1999 SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware, HWWS'99, 8-9 August 1999, Los Angeles, California*. New York, Association for Computer Machinery. pp. 55–66. Available from: doi: 10.1145/311534.311578
- Gustavson, S. & Strand, R. (2011) Anti-aliased Euclidean distance transform. *Pattern Recognition Letters*. 32 (2), 252–257. Available from: doi: 10.1016/j.patrec.2010.08.010
- Halli, A., Saaidi, A., Satori, K. & Tairi, H. (2008) Per-Pixel Displacement Mapping Using Cone Tracing. *International Review on Computers and Software*. 3 (3), 1–11.
- Halli, A., Saaidi, A., Satori, K. & Tairi, H. (2009) Per-Pixel Extrusion Mapping. *IJCSNS International Journal of Computer Science and Network Security*. 9 (3), 118–124.

- Halli, A., Saaidi, A. Satori, K. & Tairi, H. (2010) Extrusion and revolution mapping. *ACM Transactions on Graphics*. 29 (5), 1–14. Available from: doi: 10.1145/1857907.1857908
- Hart, J. C. (1996) Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *Visual Computer*. 12 (10), 527–545. Available from: doi: 10.1007/s003710050084
- Heckbert, P. S. (1986) Survey of Texture Mapping. *IEEE Computer Graphics and Applications*. 6 (11), 56–67. Available from: doi: 10.1109/MCG.1986.276672
- Jean, S. P. (2002) A Survey of Methods for Recovering Quadrics in Triangle Meshes. *ACM Computing Surveys*. 34 (2), 211–262. Available from: doi: 10.1145/508352.508354
- Kaneko, T., Takahei, T., Inami, M., Kawakami, N., Yanagida, Y., Maeda, T. & Tachi, S. (2001) Detailed Shape Representation with Parallax Mapping. In: *Proceedings of the ICAT 2001, 5-7 December 2001, Tokyo, Japan*. pp. 205–208.
- Lee, A., Moreton, H. & Hoppe, H. (2000) Displaced subdivision surfaces. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques, SIGGRAPH '00, 23-28 July 2000, New Orleans, Louisiana*. New York, ACM Press/Addison-Wesley Publishing Co. pp. 85-94. Available from: doi: 10.1145/344779.344829
- McGuire, M. & McGuire, M. (2005) *Steep Parallax Mapping*. Available from: <https://casual-effects.com/research/McGuire2005Parallax/index.html> [Accessed 20th October 2021]
- Oliveira, M. M., Bishop, G. & McAllister, D. (2000) Relief texture mapping. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques, SIGGRAPH '00, 23-28 July 2000, New Orleans, Louisiana*. New York, ACM Press/Addison-Wesley Publishing Co. pp. 359–368. doi: 10.1145/344779.344947
- Oliveira, M. M. & Policarpo, F. (2005) *An Efficient Representation for Surface Details*. 55 (51), 1–8.
- Ouazzani Chahdi, A., Rraguai, A., Halli, A. & Satori, K. (2018) Dynamic relief mapping. In: *2018 International Conference on Intelligent Systems and Computer Vision (ISCV), 2-4 April 2018, Fez, Morocco*. New York, IEEE. pp. 1–6. Available from: doi: 10.1109/ISCV.2018.8354053
- Ouazzani Chahdi, A., Rraguai, A., Halli, A. & Satori, K. (2021) Per-Pixel Extrusion Mapping with Correct Silhouette. *Computer Science*. 22 (3), 407-432. Available from: doi: 10.7494/csci.2021.22.3.3337
- Patterson, J. W., Hoggar, S. G. & Logie, J. R. (1991) Inverse Displacement Mapping. *Computer Graphics Forum*. 10 (2), 129–139. Available from: doi: 10.1111/1467-8659.1020129
- Peercy, M., Airey, J. & Cabral, B. (1997) Efficient bump mapping hardware. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques, SIGGRAPH '97, 3-8 August 1997, Los Angeles, California*. New York, ACM Press/Addison-Wesley Publishing Co. pp. 303–306. Available from: doi: 10.1145/258734.258873
- Policarpo, F. & Oliveira, M. M. (2006) Relief mapping of non-height-field surface details. In: *Proceedings of the 2006 symposium on Interactive 3D graphics and games, SI3D '06, 14-17 March 2006, Redwood City, California*. New York, Association for Computing Machinery. pp. 55-62. Available from: doi: 10.1145/1111411.1111422
- Policarpo, F. & Oliveira, M. M. (2007) Relaxed cone stepping for relief mapping. In: Nguyen, H. (ed.) *GPU Gems 3*. London, Addison-Wesley Professional, pp. 409–428.
- Policarpo, F., Oliveira, M. M. & Comba, J. L. D. (2005) Real-time relief mapping on arbitrary polygonal surfaces. In: *ACM SIGGRAPH 2005 Papers, SIGGRAPH '05, 31 July - 4 August 2005, Los Angeles, California*. New York, Association for Computing Machinery, p. 935. Available from: doi: 10.1145/1186822.1073292
- Rraguai, A., Ouazzani Chahdi, A., Halli, A. & Satori, K. (2018) Revolution mapping with bump mapping support. *Graphical Models*. 100, 1–11. Available from: doi: 10.1016/j.gmod.2018.09.001
- Rraguai, A., Ouazzani Chahdi, A., Halli, A. & Satori, K. (2020) Image-based extrusion with realistic surface wrinkles. *Journal of Computational Design and Engineering*. 7 (1), 30–43. Available from: doi: 10.1093/jcde/qwaa004
- Tatarchuk, N. & Natalya (2006) Practical parallax occlusion mapping with approximate soft shadows for detailed surface rendering. In: *ACM SIGGRAPH 2006 Courses, SIGGRAPH'06, 30 July - 3 August 2006, Boston, Massachusetts*. New York, Association for Computing Machinery. p. 81. Available from: doi: 10.1145/1185657.1185830
- Welsh, T. & Infiscape Corporation (2004) *Parallax mapping with offset limiting: A per-pixel approximation of uneven surfaces*. Available from: <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Parallax+Mapping+with+Offset+Limiting+:+A+Per+?+Pixel+Approximation+of+Uneven+Surfaces#0> [Accessed 20th October 2021].

