







# Revolution-bump mapping with texture function adjustment according to the geometry of the revolved object

## ABSTRACT

*Nowadays, 3D computer graphics are firmly anchored in our daily lives, extending across a multitude of distinct fields. Although each field follows its specific objectives, two major objectives are taken into consideration: realism and rendering speed. This is why image-based rendering (IBMR) techniques, such as revolution mapping, are gaining interest. Revolution-bump mapping is an image-based rendering that allows the creation of 3D objects in their entirety and without using polygonal meshes. The objective of the study presented in this paper is to improve the revolution-bump mapping technique as well as its extensions while ensuring that the application of textures on revolved surfaces is realized adequately. This development will allow the creation of pre-existing revolve models, while maintaining the essential rendering speed requirements for real-time rendering.*

## KEY WORDS

Computer graphics, Revolution mapping, Image-based modeling and rendering, Bump mapping, Ray-tracing

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

<sup>1</sup> Abdelmalek Essaadi University, National School of Applied Sciences Al Hoceima (ENSAH), SOVIA Research Team, Tetouan, Morocco

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

<sup>3</sup> Ibn Zohr University, High School of Technology, Agadir, Morocco

<sup>4</sup> Moulay Ismail University, Faculty of Law, Economics, and Social Sciences (FSJES), OMEGA-LERES Laboratory, Meknes, Morocco

*Corresponding author:*  
Anouar Ragragui  
e-mail:  
[a.ragragui@uae.ac.ma](mailto:a.ragragui@uae.ac.ma)

First received: 21.11.2023.

Revised: 23.4.2024.

Accepted: 21.6.2024.

## Introduction

Based on traditional methods, real-time rendering still suffers from the number of vertices and polygons that the graphics cards need to handle, which affects the interactivity of rendering complex 3D scenes.

Furthermore, the appearance of the microreliefs constituted a problem in real-time rendering due to their diminutive size to be created by a mesh that requires a series of decomposition into a set of triangles, moreover, they represent a lot of details which make them difficult to simulate for shading functions and this because of

---

visual poverty. So, it is often necessary to make sacrifices by decreasing the number of polygons constituting the 3D scene so that it can maintain a reasonable display rate and consequently reduce the rendering quality.

The reason that pushed us to move towards alternative methods to polygonal mesh, namely image-based rendering and more precisely revolution mapping because it uses only a single RGBA texture to create 3D objects.

Revolution-bump mapping is a technique that combines two different approaches: revolution mapping and bump mapping (Ragragui et al., 2018). Revolution mapping is based on the use of a binary form stored in a 2D texture, which we call a shape map. During the rendering stage, the model to be revolved is represented by only the pixels with zero values. This method is based on the ray tracing algorithm to find the intersection point of the viewing ray and the revolved surface by using an empty space which is calculated using the Euclidean Distance Transform (EDT) computed from the binary form (Danielsson, 1980; Fabbri et al., 2008; Gustavson & Strand, 2011). On the other hand, bump mapping consists of adding more realism to 3D objects by simulating micro-reliefs during the shading phase. It uses a displacement map to disrupt the normals associated with the 3D surface to produce a microrelief effect.

Unfortunately, revolution mapping faces a recurring problem of poor texturing of revolved surfaces. Indeed, it has gaps in the ability to adequately apply textures to these surfaces due to the use of inappropriate texturing functions. This article aims to present an innovative solution that relies on the configuration of the revolution object to select the appropriate texturing function. Indeed, the two types of textures used for the revolved object, namely the color texture and the displacement map, prove insufficient to ensure optimal texturing of the surface of the 3D object, whether in terms of colorimetry or microreliefs.

As can be seen in Figure 1, the direct application of the texturing function presents problems in adapting texture to the shape of 3D objects. Looking at the object at the top of Figure 1a, it can be seen that spherical projection does not guarantee appropriate texturing, whereas the object at the bottom is adequately textured. On the other hand, in Figure 1b, the cylindrical projection guarantees perfect texturing of the object at the top, but the object at the bottom does not benefit from satisfactory texturing.

In this study, we propose an approach that uses the specific geometry of the revolved object to guide the choice of texturing method. This approach aims to solve the problems inherent in texturing the surfaces of a revolution, considering the coloring and rendering requirements of microreliefs.

By determining the texturing method based on the shape and characteristics of the revolved object, we aim to overcome the limitations of revolution bump mapping and enable the creation of 3D renderings that respect both visual realism and detail accuracy.

## Related Work

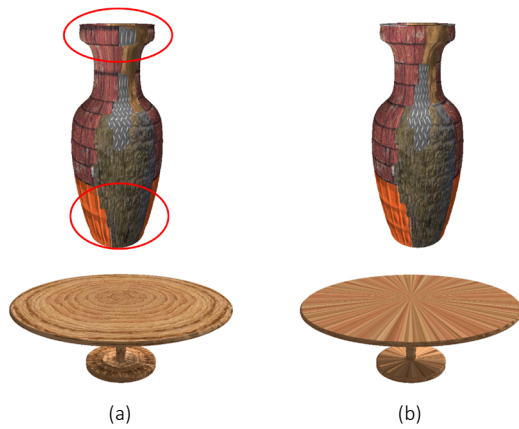
One of the most popular techniques for real-time rendering is texture mapping. It allows to add realism to a computer-generated 3D object (Catmull, 1974; Heckbert, 1986; Blinn & Newell, 1976). Another use of texture mapping is presented by the authors Lim et al. (2023) and Kao, Chen & Ueng (2023). It is the oldest and simplest of the image-based techniques. Its goal is to determine the relationship between texture elements defined in 2D space and surfaces defined in 3D space. Bump mapping was introduced by Blinn (1978). This method reproduces microrelief on 3D surfaces without changing their geometry. A displacement value is calculated based on the partial derivatives of the applied texture, which is used to perturb the normal of a given surface.

Parallax mapping (Kaneko et al., 2001) is a technique similar to bump mapping, but based on different principles. It allows to significantly increase the detail of a textured surface, even if this detail is an illusion. It aims at displacing the texture coordinates to find approximately the intersection at which the height map's relief and the viewing ray, given in tangent space, cross. Further improvements are presented by the authors Welsh (2004) and McGuire & McGuire (2005).

To add detail, the height map's values are used in the displacement mapping (Cook, 1984; Lee, Moreton & Hoppe, 2000). It was able to hide almost all the defects of bump mapping by completely changing the surface geometry instead of just perturbing the normals. This approach is based on dividing the 3D surface into sub-polygons and displacing them along their normals using distances extracted from a displacement map. The method used is different from that described in the papers by Gumhold & Hüttner (1999) and Doggett & Hirche (2000), which significantly modified the geometry to reduce the number of triangles produced as a function of viewpoint. To reduce the polygon generation, they proposed the adoption of an adaptive subdivision based on the displacement map.

View-dependent displacement mapping is a technique suitable for real-time rendering. It relies on preprocessing to compute a texture (Wang et al., 2003). The goal is to move the surface by performing calculations at the texel level, thus optimizing performance. A significant improvement of this method is presented in Wang & Dana (2005), where the use of a compression method is introduced to meet high memory requirements.

Another interesting extension is discussed in Wang et al. (2004), which aims to generalize the displacement map approach under the name "GDM" (Generalized Displacement Maps).



» **Figure 1:** Illustration of texturing function problem of revolved objects. (a) The spherical projection (b) The cylindrical projection

The ray tracing method aims to use the viewing ray to find the intersection point, sometimes referred to as linear search. It has only been used in the context of parallax mapping in the research by McGuire & McGuire (2005), and as a first step in the papers of the authors Policarpo, Oliveira & Comba (2005) and Tatarchuk & Natalya (2006). Unfortunately, relying solely on linear search results in stair-step artifacts, unless very narrow intervals are used. This issue was resolved by the method suggested in Tatarchuk & Natalya (2006), which involved combining a secant step with a fine linear search (Wen, 2023; Yang & Jia, 2023; Wu et al., 2024; Zellmann et al., 2022). Unlike ray tracing, which follows light rays deterministically, path tracing uses a probabilistic approach to calculate light paths in the scene; This is a technique that uses the principle of ray tracing but in a different way (Chen, Chen & Yu, 2023; Wald & Parker, 2022; Wald, Jaroš & Zellmann, 2023).

The combination of secant and linear searches offers a solution for improving ray tracing (Yerex & Jagersand, 2004). A further improvement to this technique is introduced in Risser, Shah & Pattanaik (2006), where the secant method is repeated several times to accurately determine the intersection point.

The key advantage of per-pixel displacement mapping is its ability to enhance the reality of surfaces without adding complexity to the mesh structure, making it an exciting evolution of the per-vertex displacement mapping method previously described in Patterson, Hoggar & Logie (1991). This technique overcomes the bottleneck caused by the significant number of graphics primitives sent to the graphics processor as part of vertex displacement mapping.

Per-pixel displacement mapping relies on ray-tracing technology to accurately determine the texture coordinates for each pixel with respect to the displacement map.

Surfaces of revolution are commonly used in various sectors such as engineering, architecture and 3D modelling, as they can be used to generate refined and complex shapes (Li & Li, 2022). Several techniques are based on this approach, including extrusion mapping and revolution mapping, as discussed in Halli et al. (2009) and Halli et al. (2010).

Both methods use shape maps to create 3D surfaces. To increase the realism of the generated surfaces, various improvements have been made to both approaches, as shown in Ragragui et al. (2020), Ragragui et al. (2018), Ragragui et al. (2022) and Chahdi et al. (2021b).

To quickly converge to the intersection point, sphere tracing uses spheres to encode the empty space (Hart, 1996). It was subsequently adapted for the intersection point by using ray-tracing and the height map (Donnelly, 2005). Further improvements were presented in Fabbri et al. (2008) and Gustavson & Strand (2011), which aim at perfecting the algorithm for calculating distance maps.

Cone tracing determines the empty space around each pixel of the depth map in the pre-processing stage as an open cone at the top, and then stores its ratio in a cone map. During the rendering stage, the cone map is used to accelerate the convergence to the intersection point of the viewing ray and the surface, so that there is no chance of missing it. This approach comes in two flavors: a relaxed version (Policarpo & Oliveira, 2007) and a conservative version (Dummer, 2006). Both variants of cone mapping have been extended in Halli et al. (2008) and Chahdi et al. (2021a).

Due to its ability to speed up convergence to the point where the viewing ray intersects with the relief, relief mapping is one of the most popular methods for real-time rendering (Policarpo, Oliveira & Comba, 2005; Policarpo & Oliveira, 2006; Chahdi et al., 2018). This technique is an evolution of the relief texture mapping method introduced in Oliveira, Bishop & McAllister (2000).

Shadow mapping is a widely used technique that provides satisfactory results and is characterized by the fact that is relatively easy to implement, as suggested in Wang et al. (2003), Policarpo, Oliveira & Comba (2005) and Wang et al. (2017).

The idea behind shadow mapping is quite simple: it is based on the principle that the scene is illuminated according to the viewpoint of the light source.

A technique for interactive deformation and collision with reliefs was presented in Nykl, Mourning & Chelberg (2014). This technique can be seamlessly combined with existing relief rendering techniques, including parallax mapping, relief mapping, as well as applications using displacement mapping.

## Bump mapping

Rendering microrelief was one of the main issues, particularly in real-time rendering. Blinn proposed bump mapping as a microrelief simulation technique (Blinn, 1978). As shown in Figure 2, this method involves adjusting the 3D surface's normals to create the appearance of microrelief. The principle is quite simple: it consists of displacing the normals to a surface to induce variations in shading, thus giving the illusion of relief without modifying the basic geometry of the 3D object. More concretely, it is based on the partial derivatives computed from a microrelief saved as a monochrome image known as a height map, which entails displacing the normal interpolated during the rasterization stage. Disturbed normals are calculated using the following formula:

$$\vec{N}(u,v) = \vec{N}(u,v) + \frac{\left( \frac{\partial H(u,v)}{\partial u} \left( \vec{N} \wedge \frac{\partial \sigma(u,v)}{\partial v} \right) - \frac{\partial H(u,v)}{\partial v} \left( \vec{N} \wedge \frac{\partial \sigma(u,v)}{\partial u} \right) \right)}{\|\vec{N}\|} \quad (1)$$

By using the height map  $H(u, v)$  saved as a 2D grayscale image, we can calculate the disturbed normal  $\vec{N}(u,v)$  for each pixel by using the formula (1).



» **Figure 2:** Comparison of a teapot rendered using texture mapping (top) and bump mapping (bottom)

## Revolution-bump mapping

Revolution mapping is a method for generating highly convincing 3D objects without resorting to polygonal meshes and presenting them interactively. The underlying concept is to use a shape map, which holds the geometric information of the basic form, to build virtual surfaces.

Figure 3 illustrates a geometric representation of the base form's revolution, positioned on a box (shape box), and highlights the process of searching the intersection point.

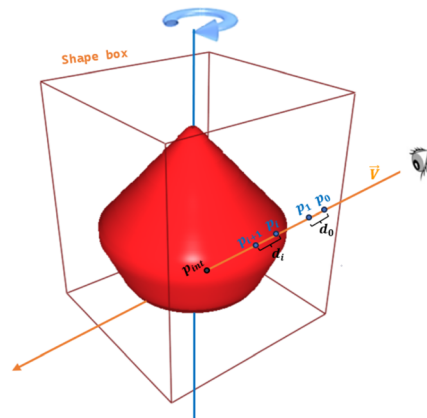
There are four essential components in the revolution mapping algorithm. Firstly, the shape map and displacement map are essential for precisely defining the geometry and displacement variations (microreliefs) associated with the basic form. Secondly, the ray-tracing algorithm plays a crucial role in visual creation by calculating the interactions between the viewing ray and the revolution surface. The last other components are the shading process that adds lighting and shadow effects, contributing to the realism of the final 3D object, and the shape box that provides a reference element for positioning the basic form and managing intersections.

## Shape and displacement map

These two maps are obtained during the pre-processing phase. They contain the essential information for generating the surface of revolution (shape map) and for adding realism (displacement map).

**Shape map:** The data for the revolution mapping algorithm is contained in this map, which is an RGBA texture (Figure 4e). The alpha channel saves a binary image that represents the basic form (Figure 4a). The distance map is kept in the blue channel (Figure 4b).

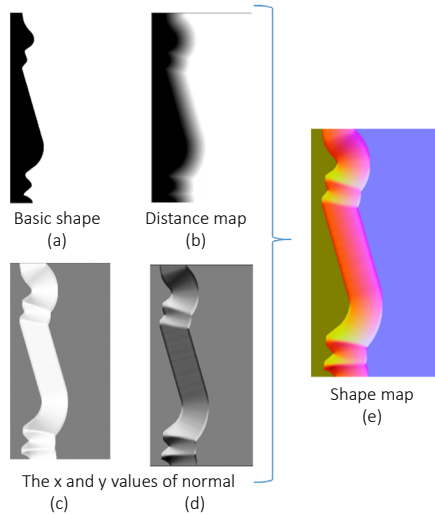
Finally, the red and green channels contain the gradient values along x and y, which are used to calculate the normal coordinates (Figure 4c,d).



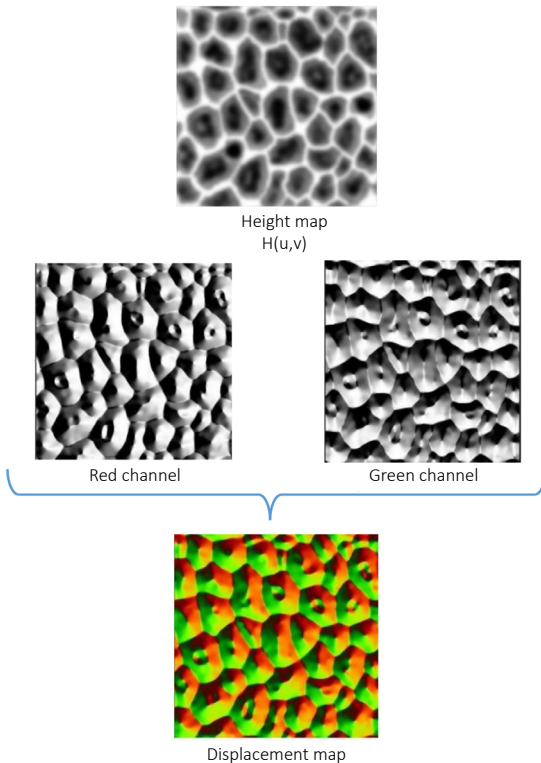
» **Figure 3:** The shape of a piece of jewelry placed at the center of a box and the process of finding the point of intersection  $p_{int}$

**Displacement map:** the partial derivatives  $\partial H(u,v)$  are calculated as a function of  $u$  and  $v$  from the height map  $H(u,v)$  (Figure 5a), then saved in the red and green channels (Figure 5b and c) of a 2D texture called the displacement map (Figure 5d).

Before starting the rendering phase, the two maps; the shape and the displacement map; are transferred to the graphics card.



» **Figure 4:** Example of different data. (a) The alpha channel. (b) The blue channel. (c) Red and (d) green channels. (e) Shape map stores all this data



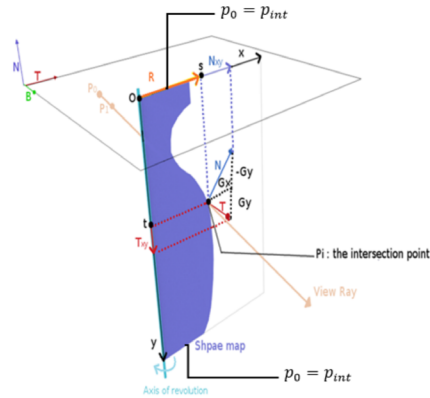
» **Figure 5:** Example of displacement map  $\partial H(u,v)$  for which we store the partial derivatives of  $H(u,v)$

## Ray tracing

Finding the intersection point is the first step in the revolution mapping algorithm. For this, the technique is

based on ray tracing, whose goal is to use the distances  $d$  recorded in the shape map to locate the intersection of the viewing ray and the revolved surface. The current pixel has coordinates  $(u,v)$ , and the start point  $p_0$  of the search has coordinates  $(x_0, y_0, z_0) = (u, v, 0)$ . The normalized viewing ray is determined from the viewing point to the starting point  $p_0$ . The blue channel (figure 4b) is used, at each point  $p_i$  to extract the distance  $d_i$ . The point  $p_{i+1}$  is calculated by the formula:

$$p_{i+1} = p_i + d_i \cdot \vec{V}_p \quad (2)$$



» **Figure 6:** At the intersection point  $p_{int}$  the process for computing the tangent and normal vectors

To access the shape map and retrieve the distance  $d$  between the current point and the form, the revolution algorithm uses the coordinates  $(s, t)$  (Figure 6). Using the equation below:

$$(s, t) = (\|\vec{R}_i\|, z_i) \text{ with: } \vec{R}_i = (x_i - O_x, y_i - O_y) \quad (3)$$

## Shading

For each intersection point found by the ray tracing algorithm, the next step is to identify the tangent space (TBN). This space will enable shading of the pixel correctly using the following equation:

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

With

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

According to Figure 6, the normal is constituted by the components  $G_{int_x}$  and  $G_{int_y}$  of the gradient unit:

$$\vec{N}_{int} = \left( G_{int_x} \frac{R_{int_x}}{\|\vec{R}_{int}\|}, G_{int_y} \frac{R_{int_y}}{\|\vec{R}_{int}\|}, -G_{int_z} \right) \quad (6)$$

The tangent can be determined by the following equation:

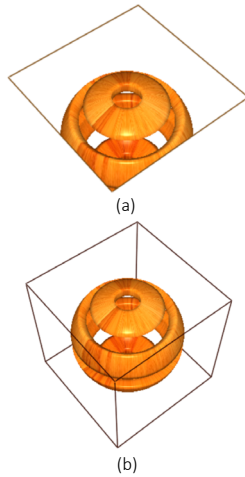
$$\vec{T}_{int} = \left( G_{int_y} \frac{R_{int_y}}{\|\vec{R}_{int}\|}, -G_{int_y} \frac{R_{int_x}}{\|\vec{R}_{int}\|}, -G_{int_y} \right) \quad (7)$$

The vector  $\vec{B}_{int}$  is calculated by using the formula:

$$\vec{B}_{int} = \vec{T}_{int} \wedge \vec{N}_{int} \quad (8)$$

## Shape box

Applying the shape map to a simple polygonal mesh structure, such as a plane as shown in Figure 7a, will not completely render the 3D object created by revolution mapping. This is because the geometry is only visible from the textured surface of the polygons. To create and display complete 3D objects regardless of the viewing ray direction, this technique implements a solution that consists of enveloping the virtual volume in revolution with a shell space made of a single box. This envelope is called a shape box (Figure 7b).



» **Figure 7:** Highlighting the problem associated with applying the shape map to a simple plan (a) This problem is solved by using a shape box (b)

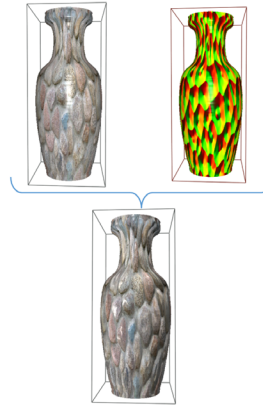
## Texturing function

In the context of our contribution, our goal is closely related to the precise determination of the color of a revolved object.

To achieve this, we rely on two fundamental elements: the color map and the displacement map. These elements are crucial for each intersection point  $p_{int}$ .

The process we've developed consists of several key steps. First, it is imperative to accurately calculate the texture coordinates, which we denote by  $(\xi_{int}, \eta_{int})$ . These coordinates are of paramount importance because they are used to access the information contained in the color map and the displacement map.

These maps meticulously wrap around the revolved object (Figure 8).



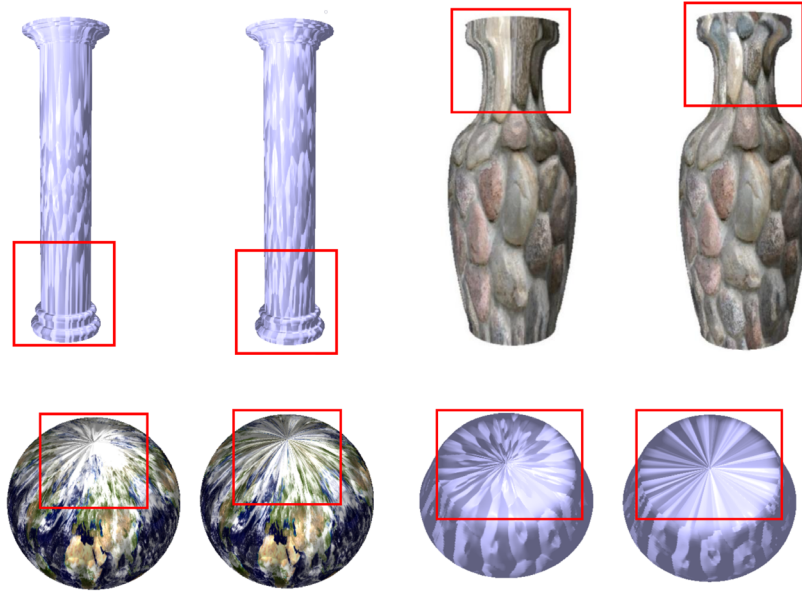
» **Figure 8:** To properly texture the revolved object (below), we need to apply the color map to the object and the partial derivative map, using the appropriate projection for the object

By accessing the color map, we can extract the information needed to determine the specific color of each intersection point. In addition, by consulting the displacement map, we have the elements we need to evaluate displacement variations as a function of position. This is crucial for our purpose, as it allows us to retrieve the desired values related to the color and displacement properties of the 3D object.

Direct use of the coordinates  $(x_{int}, y_{int}, z_{int})$  allows only simple mapping with planar projection, which is well suited for a flat surface, but not for a revolved surface. The latter requires cylindrical or spherical projection, as shown in Figure 9. This distinction becomes clear when we look at Figure 9, where we can easily see that the use of a spherical or cylindrical projection must be adjusted according to the specific geometry of the revolved object. This adjustment is crucial to avoid visual errors, which are marked in Figure 9.

This problem also arises when revolution mapping is extended to variations such as revolution-bump mapping, extended, beveled, or chamfered mapping. Our fundamental goal is to give revolved objects a realistic texture that reflects their shape and appearance. To achieve this, we propose a two-step methodology. In the pre-processing stage, we propose to associate the appropriate projection type directly with the revolved object by integrating it with the name of the shape map. This approach greatly simplifies the management of different projections and ensures that each object receives the correct texture according to its unique geometry.

However, the key step is the rendering phase. This is where the information previously encoded in the name of the shape map comes into play.



» **Figure 9:** Rendering of the various 3D objects created in real-time by revolution-bump mapping, using spherical projection (left) and cylindrical projection (right). The red outline shows the texturing defects on the revolution surfaces

During the rendering stage, we extract this information to select the projection that perfectly matches the revolved object that we want to render.

By doing so, we succeed in transcending the constraints of simple projections and offering realistic textures, thus contributing to the visual quality and verisimilitude of revolution-generated objects, as shown in Figure 9.

To accurately ascertain the texture coordinates, and given that the texturing process must be applied to each replica, it is imperative to consider only the fractional portion of the intersection point coordinates:

$$\left( x'_i, y'_i, z'_i \right) = \left( x_i, y_i, z_i \right) - \left( E(x_i), E(y_i), E(z_i) \right) \quad (9)$$

### Cylindrical projection

For cylindrical projection mapping, we use the cylindrical coordinates of the intersection point  $p_{int}$ , expressed with respect to an axis centered on the shape box:

$$\begin{cases} x_{int} - 0,5 = r \cos \varphi \\ y_{int} - 0,5 = r \sin \varphi \end{cases} \quad (10)$$

With:

$$r = \sqrt{(x_{int} - 0,5)^2 + (y_{int} - 0,5)^2} \quad (11)$$

We deduce the value of  $\varphi$ :

$$\varphi = \text{atan2}(y_{int} - 0,5, x_{int} - 0,5) \quad (12)$$

The texture coordinate  $\eta_{int}$  is equal to  $z'_{int}$ . As for  $\xi_{int}$ , it changes from 0 to 1 when  $\varphi$  changes from  $\pi$  to  $-\pi$ . We then have:

$$\left( \xi_{int}, \eta_{int} \right) = \left( \frac{\pi - \varphi}{2\pi}, z_{int} \right) \quad (13)$$

For revolution-bump mapping, use  $(O_x, O_y)$  instead of  $(0.5, 0.5)$  if the axis of revolution is not centered on the shape box.

### Spherical projection

Based on the intersection point's spherical coordinates, the spherical projection mapping is produced, expressed with respect to a reference frame placed at the centre of the shape box.

$$\begin{cases} x_{int} - 0,5 = r \sin \theta \cos \varphi \\ y_{int} - 0,5 = r \sin \theta \sin \varphi \\ z_{int} - 0,5 = r \cos \theta \end{cases} \quad (14)$$

With:

$$r = \sqrt{(x_{int} - 0,5)^2 + (y_{int} - 0,5)^2 + (z_{int} - 0,5)^2} \quad (15)$$

We deduce the values of  $\varphi$  and  $\theta$ :

$$\begin{cases} \varphi = \text{atan2}(y_{int} - 0,5, x_{int} - 0,5) \\ \theta = \arccos \frac{z_{int} - 0,5}{r} \end{cases} \quad (16)$$

$\xi_{int}$  varies in  $[0,1]$  when  $\varphi$  goes from  $\pi$  to  $-\pi$ , and  $\eta_{int}$  varies in  $[0,1]$  when  $\theta$  goes from  $\pi$  to 0. Then we have:

$$\left( \xi_{int}, \eta_{int} \right) = \left( \frac{\pi - \varphi}{2\pi}, \frac{\pi - \theta}{\pi} \right) \quad (17)$$

## Results and discussion

In the pre-processing stage, we implemented the algorithms using the C++ language to produce the displacement map and the shape map. During the rendering stage, we exploited OpenGL and its parallel processing language, GLSL, to create the vertex and fragment shaders.

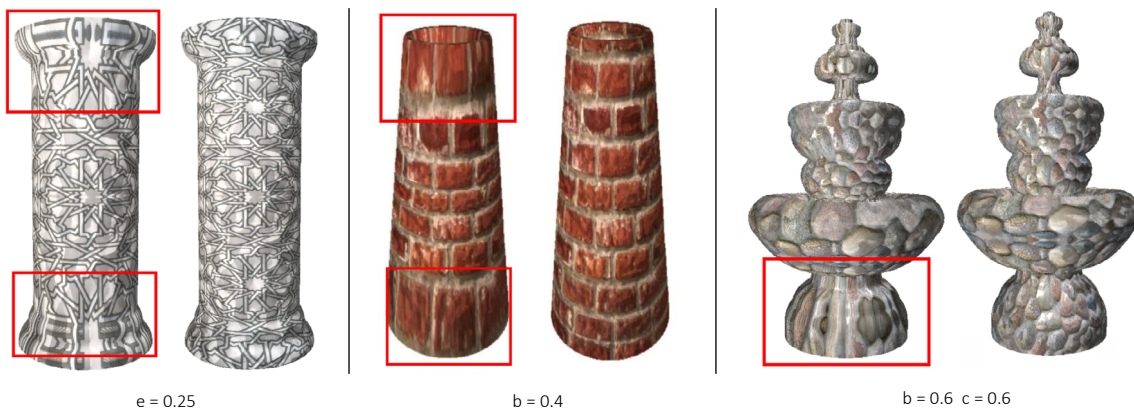
During the rendering stage, in addition to the coordinates associated with the shape box, the two maps created during the preprocessing stage are transmitted to the graphics card. The analyses and illustrations were executed using an 8CPU Intel Core i7-11657G7 architecture at 2.80GHz, equipped with 8GB of RAM, and a GeForce MX330 graphics card with 4GB of dedicated memory. In this paper, the images provided were taken during a test in which the box occupies a significant part of the screen. We would like to note that the total number of iterations to find the intersection is 20, except for the beveled revolution mapping (35 iterations, of which 10 are for binary refinement), as well as the revolution mapping with chamfer (20 iterations, of which 35 are for the chamfer phase).

As shown in Figure 10, our approach correctly textures objects created by the revolution-bump mapping extensions: outward, beveled, and chamfered revolution-bump mapping. Our contribution perfectly adapts to the geometry of the rendered object.

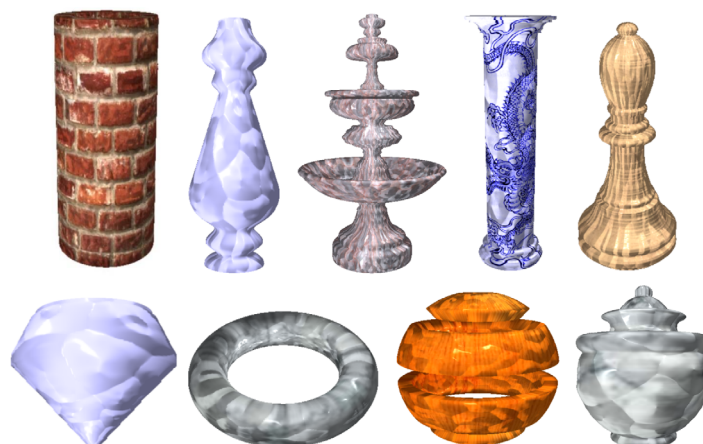
Figure 11 shows additional examples of well-textured surfaces, using either cylindrical or spherical projection, depending on the object's geometry. These 3D objects are generated with revolution-bump mapping, using various shape and displacement maps. We can emphasize how many different kinds of objects can be made with this technique, as well as how many graphical primitives (polygons and vertices) can be avoided.

Figure 12 shows 3D objects rendered using revolution bump mapping with spherical projection (Figure 12a) and cylindrical projection (Figure 12b) by applying repetition to the texture and displacement map.

It should be noted that the improvement presented in this paper also applies to the other revolution mapping techniques, i.e. revolution with mirror and repetition (see Figure 13).

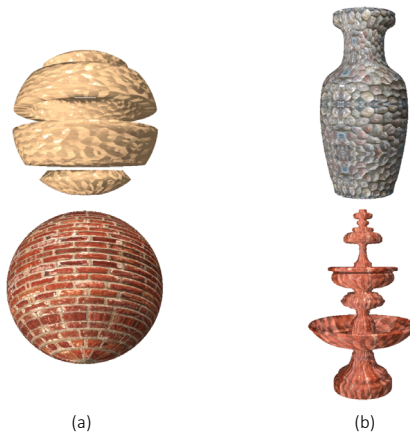


» **Figure 10:** Extension of revolution-bump mapping. (a) Outward, (b) Beveled, and (c) Chamfered revolution-bump mapping. For each method, the spherical projection is on the left and the cylindrical one is on the right

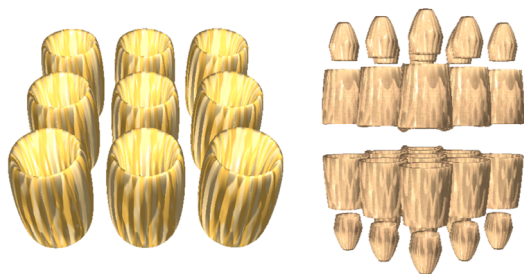


» **Figure 11:** Various objects are rendered using revolution-bump mapping by adjusting the texturing function to the rendered surface. (Top) Cylindrical projection. (Bottom) Spherical projection





» **Figure 12:** Various objects are rendered using revolution-bump mapping with adjustment of the texturing function and application of repetition to the texture and displacement map. (a) Spherical projection. (b) Cylindrical projection



» **Figure 13:** Various objects are rendered using revolution-bump mapping with adjustment of the texturing function and application of repetition to the texture and displacement map

Note that our contribution has no impact on rendering speed. There is not much difference between revolution-bump mapping and revolution-bump mapping with texture function adjustment according to the geometry of the revolved object. Our improvement fully preserves all the features and properties of revolution-bump mapping and its extensions. In addition, we found that this improvement has no impact on the complexity of the raytracing algorithm since the revolution mapping algorithms always have linear complexity in  $O(n)$ . As a result, the complexity of the algorithm is always linear. It's also worth noting that the memory used by the shape map does not exceed 10 MB, while the partial derivatives map does not exceed 4 MB. This ensures that the memory of the graphics card is not saturated, thus guaranteeing the continuous interactivity of the 3D scene.

## Conclusion

This research presents an enhancement of revolution-bump mapping and its extensions. Our contribution makes it possible to adjust the texture of the revolution surface in real-time while enriching the visual quality of

the rendered objects. This improvement avoids overloading the graphics pipeline that can result from processing a large number of polygons and vertices. It focuses on determining the displacement and color values at each intersection point using the displacement map and color map.

This requires a careful determination of the texture coordinates and an adaptation of the texturing function to the geometry of the revolved object. For this, we have included the type of projection in the name of the shape map which will be appropriate to the object to be generated during the rendering phase. Our contribution relies only on the use of two different textures, a box, and the tangent space at each intersection point.

The first texture, or shape map, is used to generate the revolved surface, and the second texture, or displacement map, is used to integrate the microrelief effects. However, in the rendering stage, we adapt the rendering of the 3D object by using the appropriate texturing function. The improvements we propose are in line with the two goals we set out in the introduction: to meet rendering speed requirements and to present revolution models in a particularly convincing way.

## Funding

The research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

## References

- 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
- Blinn, J. F. (1978) Simulation of wrinkled surfaces. *ACM SIGGRAPH Computer Graphics*. 12 (3), 286–292. Available from : doi: 10.1145/965139.507101
- Catmull, E. E. (1974) *A subdivision algorithm for computer display of curved surfaces*. PhD thesis. The University of Utah.
- Chahdi, A. O., Ragragui, A., Halli, A. & Satori, K. (2018) Dynamic relief mapping<sup>1</sup>. In: *2018 International Conference on Intelligent Systems and Computer Vision, ISCV, 2-4 April 2018, Fez, Morocco*. Piscataway, IEEE. pp. 1-6. Available from: doi: 10.1109/ISACV.2018.8354053
- Chahdi, A. O., Ragragui, A., Halli, A. & Satori, K. (2021a) Per-pixel displacement mapping using cone tracing with correct silhouette. *Journal of Graphic Engineering and Design*. 12 (4), 39–61. Available from: doi: 10.24867/JGED-2021-4-039
- Chahdi, A. O., Ragragui, A., Halli, A. & Satori, K. (2021b) Per-Pixel Extrusion Mapping With Correct Silhouette.

- Computer Science*. 22 (3), 407–432. Available from: doi: 10.7494/csci.2021.22.3.3337
- Chen, J., Chen, L. & Yu, Z. (2023) Accelerating path tracing rendering with Multi-GPU in Blender cycles. In: *25th International Conference on Advanced Communication Technology, ICACT, 19–22 February 2023, Pyeongchang, Republic of Korea*. Piscataway, IEEE. pp. 314–318. Available from: doi: 10.23919/ICACT56868.2023.10079514
- Cook, R. L. (1984) Shade trees. *ACM SIGGRAPH Computer Graphics*. 18 (3), 223–231. Available from: doi: 10.1145/964965.808602
- 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. and Randima, F. (eds.) *GPU Gems 2*. Boston, Addison-Wesley Professional, pp. 123–137.
- Dummer, J. (2006) *Cone step mapping: An iterative ray-heightfield intersection algorithm*. Available from: <https://www.scribd.com/document/57896129/Cone-Step-Mapping> [Accessed 20th September 2024].
- Fabrizi, R., Costa, L. D. F., Torelli, J. C. & Bruno, O. M. (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: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware, HWWS'99, 8–9 August 1999, Los Angeles, California*. New York, Association for Computing 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. *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
- 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 11th International Conference on Artificial Reality and Telexistence, ICAT 2001, 5–7 December 2001, Tokyo, Japan*. pp. 205–208.
- Kao, Y. C., Chen, W. H. & Ueng, S. K. (2023) Texture Mapping for Voxel Models Using SOM. In: *Proceedings - 2023 6th International Symposium on Computer, Consumer and Control, IS3C, 30 June – 3 July 2023, Taichung, Taiwan*. Piscataway, IEEE. pp. 99–102. Available from: doi: 10.1109/IS3C57901.2023.00035
- 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. pp. 85–94. Available from: doi: 10.1145/344779.344829
- Li, H. & Li, M. (2022) Constant Winding Angle Curve on Revolution Surface and its Application. *CAD Computer Aided Design*. 144. Available from: doi: 10.1016/j.cad.2021.103160
- Lim, A. X. W., Ng, L. H. X., Griffin, C., Kryer, N. & Baghernezhad, F. (2023) Reverse Projection: Real-Time Local Space Texture Mapping. In: *Proceedings – ACM SIGGRAPH 2023 Posters, SIGGRAPH'23, 6–10 August 2023, Los Angeles, California*. New York, Association for Computing Machinery. Available from: doi: 10.1145/3588028.3603653
- McGuire, M. & McGuire, M. (2005) *Steep Parallax Mapping*. Available from: <https://casual-effects.com/research/McGuire2005Parallax/index.html> [Accessed 20th September 2024].
- Nykl, S., Mourning, C. & Chelberg, D. (2014) Interactive mesostructures with volumetric collisions. *IEEE Transactions on Visualization and Computer Graphics*. 20 (7), 970–982. Available from: doi: 10.1109/TVCG.2014.2317700
- Oliveira, M. M., Bishop, G. & McAllister, D. (2000) Relief texture mapping. 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. pp. 359–368. Available from: doi: 10.1145/344779.344947
- 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
- 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, I3D'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*. Boston, 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: Gross, M. (ed.) *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
- Ragragui, 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
- Ragragui, 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
- Ragragui, A., Ouazzani Chahdi, A., Halli, A., Satori, K. & El Moubtahij, H. (2022) The extensions of revolution-bump mapping. *Journal of Graphic Engineering and Design*. 13 (1), 21–31. Available from: doi: 10.24867/JGED-2022-1-021
- Risser, E., Shah, M. A. & Pattanaik, S. (2006) *Interval Mapping*. Available from: <https://www.semanticscholar.org/paper/Interval-Mapping-Risser-Shah/d16d9da41ec53b604e15976b0615ad3993c67ed-c#citing-papers> [Accessed 20th September 2024].
- 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. pp. 81-112. Available from: doi: 10.1145/1185657.1185830
- Wald, I. & Parker, S. G. (2022) Data Parallel Path Tracing with Object Hierarchies. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*. 5 (3). Available from: doi: 10.1145/3543861
- Wald, I., Jaroš, M. & Zellmann, S. (2023) Data Parallel Multi-GPU Path Tracing using Ray Queue Cycling. *Computer Graphics Forum*. 42 (8). Available from: doi: 10.1111/CGF.14873
- Wang, J. & Dana, K. J. (2005) Compression of View Dependent Displacement Maps. In: Chantler, M. and Drbohlav, O. (eds.) *Proceedings of the 4th International Workshop on Texture Analysis and Synthesis, Texture 2005, 21 October 2005, Beijing, China*. pp. 143–148.
- Wang, L., Wang, X., Tong, X., Lin, S., Hu, S., Guo, B. & Shum, H.-Y. (2003) View-dependent displacement mapping. In: *ACM SIGGRAPH 2003 Papers, SIGGRAPH'03, 27-31 July 2003, San Diego, California*. New York, Association for Computing Machinery. pp. 334-339. Available from: doi: 10.1145/1201775.882272
- Wang, L., Zhang, W., Li, N., Zhang, B. & Popescu, V. (2017) Intermediate shadow maps for interactive many-light rendering. *The Visual Computer: International Journal of Computer Graphics*. 34 (10), 1415-1426. Available from: doi: 10.1007/s00371-017-1449-7
- Wang, X., Tong, X., Lin, S., Hu, S., Guo, B. & Shum, H.-Y. (2004) Generalized displacement maps. In: *Proceedings of the Fifteenth Eurographics conference on Rendering Techniques, EGSR04, 21-23 June 2004, Norrköping, Sweden*. Goslar, The Eurographics Association. pp. 227–233. Available from: doi: 10.2312/egwr/egsr04/227-233
- Welsh, T. (2004) *Parallax mapping with offset limiting: A per-pixel approximation of uneven surfaces*. Infiscape Corporation. Available from: [https://page.mi.fu-berlin.de/block/htw-lehre/wise2015\\_2016/bel\\_und\\_rend/skripte/welsh2004.pdf](https://page.mi.fu-berlin.de/block/htw-lehre/wise2015_2016/bel_und_rend/skripte/welsh2004.pdf) [Accessed 20th September 2024].
- Wen, H. (2023) A Novel Ray Tracing Method Based on Unity Scriptable Render Pipeline and DirectX Raytracing. In: *2023 15th International Conference on Computer Research and Development, ICCRD, 10-12 January 2023, Hangzhou, China*. Piscataway, IEEE. pp. 156–160. Available from: doi: 10.1109/ICCRD56364.2023.10079997
- Wu, C., Xia, Y., Xu, Z., Liu, L., Tang, X., Chen, Q. & Xu, F. (2024) Mathematical modelling for high precision ray tracing in optical design. *Applied Mathematical Modelling*. 128, 103–122. Available from: doi: 10.1016/j.apm.2024.01.012
- Yang, M. & Jia, J. (2023) Implementation and Optimization of Hardware-Universal Ray-tracing Underlying Algorithm Based on GPU Programming. In: *2023 6th International Conference on Artificial Intelligence and Big Data, ICAIBD, 26-29 May 2023, Chengdu, China*. Piscataway, IEEE. pp. 171-178. Available from: doi: 10.1109/ICAIBD57115.2023.10206260
- Yerex, K. & Jagersand, M. (2004) Displacement Mapping with Ray-casting in Hardware. In: Barzel, R. (ed.) *ACM Siggraph 2004 Sketches, SIGGRAPH'04, 8-12 August 2004, Los Angeles, California*. New York, Association for Computing Machinery. p. 149. Available from: doi: 10.1145/1186223.1186410
- Zellmann, S., Wald, I., Barbosa, J., Dermici, S., Sahistan, A. & Güdükbay, U. (2022) Hybrid Image-/Data-Parallel Rendering Using Island Parallelism. In: *Proceedings - 2022 IEEE 12th Symposium on Large Data Analysis and Visualization, LDAV, 16 October 2022, Oklahoma City, Oklahoma*. Piscataway, IEEE. Available from: doi: 10.1109/LDAV57265.2022.9966396



© 2025 Authors. Published by the University of Novi Sad, Faculty of Technical Sciences, Department of Graphic Engineering and Design. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution license 3.0 Serbia (<http://creativecommons.org/licenses/by/3.0/rs/>).