

Per-pixel displacement mapping using cone tracing with correct silhouette

ABSTRACT

Per-pixel displacement mapping is a texture mapping technique that adds the microrelief effect to 3D surfaces without increasing the density of their corresponding meshes. This technique relies on ray tracing algorithms to find the intersection point between the viewing ray and the microrelief stored in a 2D texture called a depth map. This intersection makes it possible to determine the corresponding pixel to produce an illusion of surface displacement instead of a real one. Cone tracing is one of the per-pixel displacement mapping techniques for real-time rendering that relies on the encoding of the empty space around each pixel of the depth map. During the preprocessing stage, this space is encoded in the form of top-opened cones and then stored in a 2D texture, and during the rendering stage, it is used to converge more quickly to the intersection point. Cone tracing technique produces satisfactory results in the case of flat surfaces, but when it comes to curved surfaces, it does not support the silhouette at the edges of the 3D mesh, that is to say, the relief merges with the surface of the object, and in this case, it will not be rendered correctly. To overcome this limitation, we have presented two new cone tracing algorithms that allow taking into consideration the curvature of the 3D surface to determine the fragments belonging to the silhouette. These two algorithms are based on a quadratic approximation of the object geometry at each vertex of the 3D mesh. The main objective of this paper is to achieve a texture mapping with a realistic appearance and at a low cost so that the rendered objects will have real and complex details that are visible on their entire surface and without modifying their geometry. Based on the ray-tracing algorithm, our contribution can be useful for current graphics card generation, since the programmable units and the frameworks associated with the new graphics cards integrate today the technology of ray tracing.


KEY WORDS

Real-time rendering, texture mapping, per-pixel displacement mapping, ray-tracing, cone tracing, silhouette correction, quadratic approximation

Adnane Ouazzani

Chahdi¹ 

Anouar Ragragui¹ 

Akram Halli² 

Khalid Satori¹

¹Sidi Mohamed Ben Abdellah University, Faculty of Science Dhar EL Mahraz LISAC Laboratory, Fez, Morocco

²Moulay-Ismaïl University FSJES-UMI OMEGA-LERES Laboratory Meknes, Morocco

Corresponding author:

Adnane Ouazzani Chahdi

e-mail:

adnaneouazzanichahdi@gmail.com

First received: 20.6.2021.

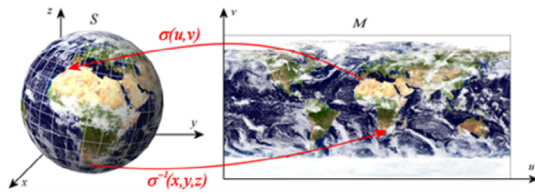
Revised: 22.9.2021.

Accepted: 24.9.2021.

Introduction

Per-pixel displacement mapping (Patterson, Hoggart & Logie, 1991) is a technique based on texture mapping (Catmull, 1974; Blinn & Newell, 1976). It is inspired at the same time by bump mapping (Blinn, 1978; Peercy, Airey & Cabral, 1997) that proceeds on pixels of the microreliefs

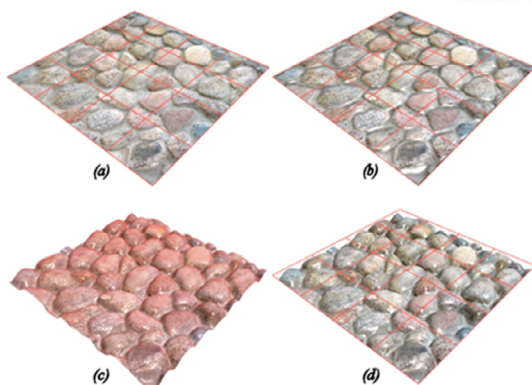
texture, and the displacement mapping (Cook, 1984) that proceeds on the vertex of the 3D mesh. Texture mapping associates a two-dimensional image with a three-dimensional surface using a function called parameterization. This function maps each vertex (x, y, z) of the mesh surface, with a pair of coordinates (s, t) representing a pixel of the texture (Figure 1).



» **Figure 1:** Parameterization $\sigma(u, v)$ and inverse parameterization $\sigma^{-1}(x, y, z)$ matching a surface S of R^3 with the domain M of R^2

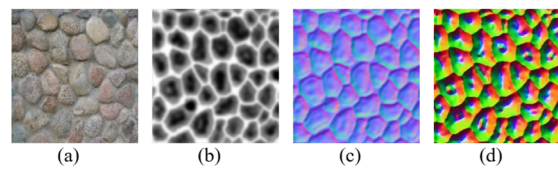
Texture mapping does not produce any microrelief effects and the colors of the pixels representing the object in the scene always remain the same regardless of the lighting conditions (Figure 2a). To solve this problem, Blinn introduced bump mapping (Blinn, 1978) based on the disturbance of the surface normals in the function of a depth map (Figure 3b). The disruption of normals produces an illusion of small displacements and produces a microrelief effect (Figure 2b). The displacement mapping uses the depth map in another way. It consists of displacing the vertices of the surface according to the values stored in the depth map. For this, the mesh must be subdivided so that it adapts to the texture resolution (depth map), which generates a lot of graphic primitives (vertices and polygons) to be processed (Figure 2c). The main goal of per-pixel displacement mapping is to have the same rendering as displacement mapping but without increasing the density of the base mesh (Figure 2d). It consists of reducing the number of graphics primitives while retaining the overall visual quality of the scene.

Figure 2 shows the difference between texture mapping techniques. As shown in the figure, the mesh density is the same in images (a), (b), and (d) but per-pixel displacement mapping allows rendering very detailed geometry. And compared to displacement mapping (c), per-pixel displacement mapping produces the same result but at a very low cost.



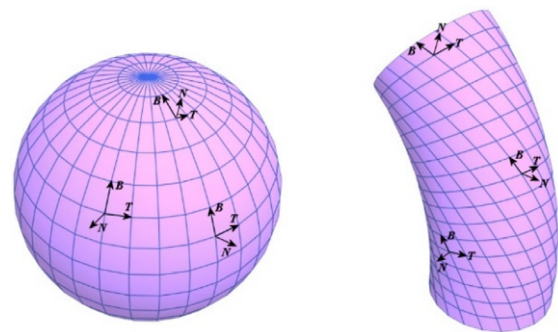
» **Figure 2:** Comparison of the different texture mapping techniques (Halli et al., 2008). (a) Texture mapping. (b) Bump mapping. (c) Displacement mapping. (d) Per-pixel displacement mapping

Per-pixel displacement mapping is based on three main elements: the displacement map, the tangent space, and the ray-tracing algorithm. The displacement map (Figure 3) is a two-dimensional image whose pixels are not used to store colors, but geometrical data (i.e. depths and normals). In the α channel, we store the depths associated with the microrelief mapped on the 3D surface. The other three channels: red, green, and blue, are used to store the three x , y , and z components of the normal, which are calculated from the depths. Since the z component of the normal can be retrieved as a function of the two others, the blue channel can be released to store other data used by certain techniques such as cone tracing. In this case, the displacement map can be named according to this technique (i.e. the Cones Map as shown in Figure 3d).



» **Figure 3:** (a) The corresponding texture. (b) The depth map. (c) The components x , y , and z of the normal. (d) A cones map that stores the depths in the α channel, the x and y components of the normal are stored in the red and green channels, and the blue channel stores the cones' radius

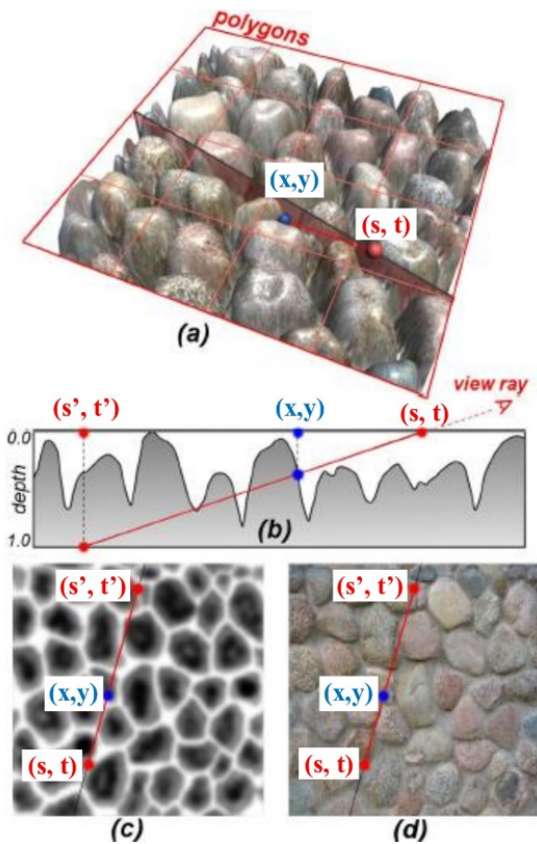
As shown in figure 4, the tangent space is a local space associated with each vertex constituting the 3D mesh (Peercy, Airey & Cabral, 1997). It is calculated using the normal to the vertex and the associated texture coordinates. The viewing ray vector and the light vector must be expressed in this space.



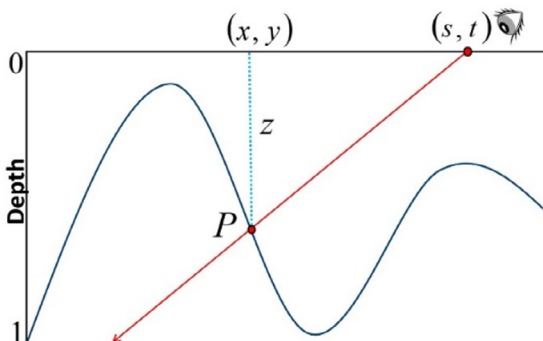
» **Figure 4:** Tangent space. It is a local space constituted by three vectors: the normal, binormal, and tangent associated with each vertex of the 3D mesh

Ray tracing is an algorithm that searches the intersection of the viewing ray and the microrelief stored in the displacement map (Figure 5 and Figure 6). This search is performed in texture space for each pixel resulting from the 3D mesh. The main problem of per-pixel displacement mapping is to find the first intersection point.

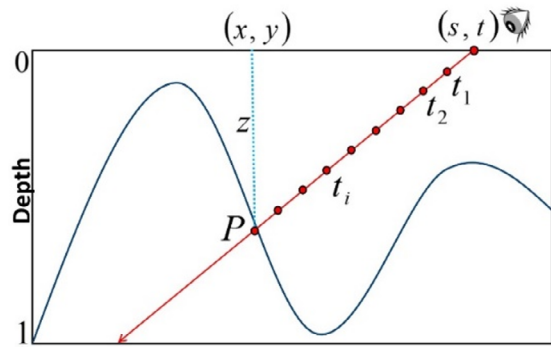
As shown in figures 5 and 6, the first intersection is represented by the point P along the viewing ray V. So, to have an illusion of relief displacement, the fragment (s, t) will be textured using the texel (x, y) instead of (s, t).



» **Figure 5:** Ray tracing in the depth map (Halli et al., 2008). (a) 3D view of the ray tracing. (b) The relief's slice including the viewing ray. (c) The depth map. (d) The corresponding texture. (s,t) is the starting point. The main problem of per-pixel displacement mapping is to find the first intersection (x,y) of the viewing ray and the microrelief. So, the fragment (s,t) will be textured using the texel (x,y) instead of (s,t)



» **Figure 6:** Ray tracing in the depth map. The first intersection point between the viewing ray and the microrelief is the P(x, y, z)



» **Figure 7:** Search for the intersection with iterations (Linear search). The number of iterations and the size of the displacement step must be defined in advance. The t_i parameter represents the sum of the step size at iteration i

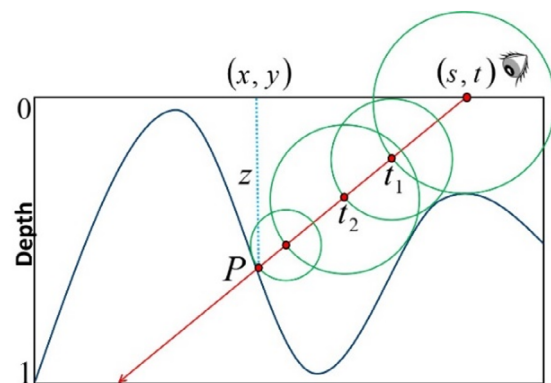
As shown in figures 5 and 6, the point P can be expressed as follows:

$$P = Vt \quad (1)$$

The speed constraint does not allow an exact search, which makes it necessary to find a point as close as possible to the intersection point. The different approaches to performing this search are presented in (Szirmay-Kalos & Umenhoffer, 2008). To better locate the first intersection, the number of iterations is predefined in advance. The size of the displacement step is defined according to the technique used, which can be constant (Figure 7) or variable (Figure 8). During the search for the intersection, we use the following general formulas:

$$t_{i+1} = t_i + \text{step} \quad (2)$$

$$P_{i+1} = vt_{i+1} \quad (3)$$



» **Figure 8:** Search for the intersection with iterations and with an encoding of the empty space (Sphere tracing). The number of iterations must be defined in advance and the size of the displacement step is calculated as a function of the sphere parameters encoded in a pre-processing stage. The t_i parameter represents the sum of the step size at iteration i .

Where v is the normalized viewing ray vector expressed in the texture space having a normalized depth (i.e. $v/v_z=1$) involving $v_z=1$.

These two formulas are used together to determine the point P_{i+1} . At iteration $i+1$, the parameter t_{i+1} makes it possible to determine the position of the point P along with vector v , and the step parameter makes it possible to determine its displacement, the value of which is calculated according to the used technique.

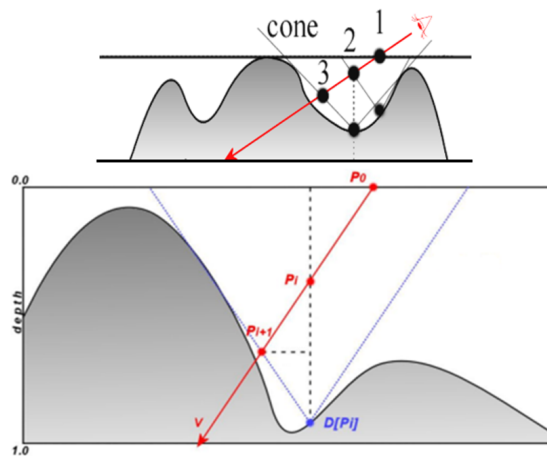
We note that the viewing ray is reversed during the search for the intersection. It means that we start from the eye (camera) towards the microrelief.

Contribution

Per-pixel displacement mapping suffers from three main problems, namely, the time to compute the displacement map, the search of the intersection between the viewing ray and the microrelief stored in the displacement map, and the treatment of the silhouette.

The preprocessing is the phase in which we calculate a displacement map for each texture. The computing speed depends on the used algorithm. An improvement has been proposed in (Halli et al., 2008) which consists of using linear algorithms instead of quadratic ones and which has considerably increased the computing speed.

To find the intersection point, we use ray-tracing algorithms. The best technique in this sense is cone tracing (Figure 9).

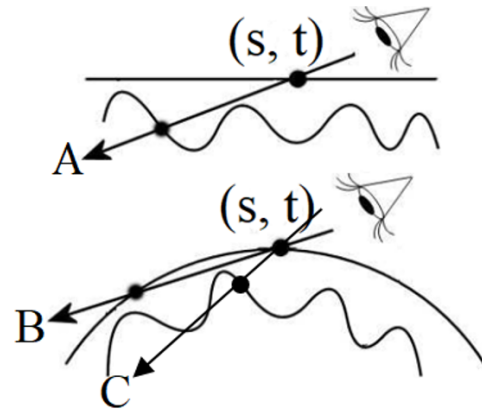


» **Figure 9:** Cone tracing on the displacement map. At each iteration, the next position P_{i+1} of the viewing ray v is calculated as a function of the current position P_i and the cone parameters which are the height $D[P_i]$ and its radius.

This technique relies on the encoding of the empty space to converge more quickly. This space is stored in a texture

called a cones map (Figure 3d). Improvements concerning the search of the intersection (ray-tracing algorithm) have been proposed in (Halli et al., 2008; Ouazzani Chahdi et al., 2017; Ouazzani Chahdi et al., 2018).

Despite the improvements proposed in (Halli et al., 2008; Ouazzani Chahdi et al., 2017; Ouazzani Chahdi et al., 2018), the silhouette problem persists. The silhouette is visible at the edges of the 3D object (Figure 10, Figure 11). To explain this problem, we have Figure 10 which shows a flat and curved surface. For flat surfaces, the viewing ray always pierces the reliefs (Ray A), that is to say, there we will always have an intersection. But for curved surfaces, sometimes the viewing ray does not pierce the relief (Ray B), that is to say, there is no intersection. Then, the ray tracing algorithm will not find an intersection point or it will find an erroneous one, and in this case, the pixel must be discarded to be able to have a correct treatment of the silhouette.

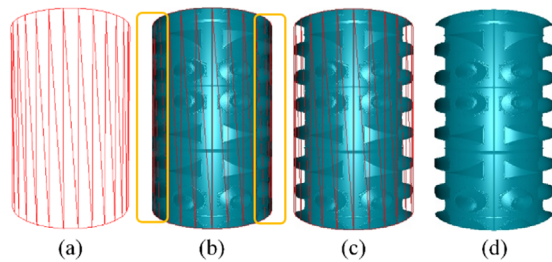


» **Figure 10:** Highlighting the silhouette problem. (top) a flat surface always allows us to have an intersection with the relief. (bottom) with a curved surface, the viewing ray can leave the surface without piercing the relief. The pixel (s,t) in this case belongs to the silhouette

The main contribution proposed in this paper compared to the improvements proposed in (Halli et al., 2008; Ouazzani Chahdi et al., 2017; Ragragui et al., 2017; Ouazzani Chahdi et al., 2018; Ragragui et al., 2018a; Ragragui et al., 2018b; Ragragui et al., 2020) is the resolution of the silhouette problem.

Indeed, as shown in Figure 11b, the cone tracing technique does not support the treatment of the silhouette. Based on this observation, this paper presents two new cone-tracing algorithms based on a quadratic approximation at each vertex of the 3D mesh (Jean, 2002; Oliveira & Policarpo, 2005). The first algorithm consists of rectifying the viewing ray after each new displacement (after the cone tracing), and the second one consists of rectifying the cone before each new displacement (before the cone tracing).

Figure 11 shows a comparison of a cylinder rendered with the cone tracing technique without and with correction of the silhouette and by highlighting the polygonal mesh. We notice that the silhouette is not visible on the 3D object when rendered without the correct silhouette (Figure 11b); this problem is surmounted by exploiting the parameters of the quadratic surface to take into account the curvature of the 3D object. Figure 11c and Figure 11d show the same 3D model rendered by our approach that provides a correct rendering.



» **Figure 11:** *Rendering of a cylinder with cone tracing techniques. (a) Basic mesh using 56 triangles. (b) Without the correct silhouette (original technique). (c), (d) With correct silhouette (proposed technique). We notice in (b) that the surface of the cylinder and the reliefs are confused. This problem has been solved by our approach in (c, d)*

Contrary to the displacement mapping, the silhouette is generated without needing to change its geometry and by using a minimal number of triangles (Figure 11a). Figure 12 shows an example of a scene that can be realized by the contribution of this paper.



» **Figure 12:** *Rendering of a scene by the relaxed cone mapping with the correct silhouette*

The main advantage of our contribution is that it can be integrated into the new graphic card units. Indeed, the programmable pipeline model for ray tracing has been introduced in (Parker et al., 2013). Currently, the frameworks and the programmable units associated with the new graphics cards integrate a programmable GPU-accelerated ray-tracing that provides a simple, recursive, and flexible pipeline for accelerating ray-tracing algorithms.

Related Works

The displacement mapping was introduced in (Cook, 1984). It consists of displacing each vertex of the 3D mesh according to the normal with a value given by the height map. To have better rendering quality, the basic mesh must be subdivided into sub-polygons to be adapted to the height map resolution, which leads to a very high number of primitives processed by the graphic cards.

Contrary to the displacement mapping, which changes the geometry (Cook, 1984), the bump mapping occurs only at the level of the shading (Blinn, 1978; Peercy, Airey & Cabral, 1997). The latter being a function of the normals, the disruption of this one will cause a microrelief illusion. So instead of creating the displacement surface, just calculate its normal and use it in a shading formula to simulate the surface details. When it comes to miniature reliefs, this technique produces a satisfactory rendering, but it is limited for shading and self-occlusion. For a large elevation of reliefs, the per-pixel displacement mapping has been proposed in (Patterson, Hoggar & Logie, 1991). For shading, the use of a horizon map has been introduced in (Max, 1988; Sloan & Cohen, 2000).

To avoid the calculation of the normal during the rendering stage, a normal mapping has been introduced in (Peercy, Airey & Cabral, 1997) that consists of storing the normals of the microrelief in a texture called: normal map. For real-time rendering, several implementations have been proposed in (Ernst et al., 1998; Kilgard, 2000; Sung Kim, Hyun Lee & Ho Park, 2001; Lee et al., 2007).

Parallax mapping is an extension of the bump mapping (Kaneko et al., 2001; Welsh, 2004; McGuire & McGuire, 2005; Premecz, 2006). This technique performs an approximate search for the intersection between the viewing ray and the relief contained in the displacement map. This point is defined by the intersection of the viewing ray and the horizontal line, which passes through the height of the relief at the current point. The main advantage of this technique is the addition of the parallax effect. However, it is limited to irregular microrelief. Improvements were introduced in (Brawley & Tatarchuk, 2004; Tatarchuk, 2006) to manage the shading correctly.

Relief mapping introduced in (Policarpo, Oliveira & Comba, 2005; Policarpo & Oliveira, 2006) is based

on relief texture mapping (Oliveira, 2000; Oliveira, Bishop & McAllister, 2000). This technique calculates the intersection point by two stages, in the first one, it determines the interval where the first intersection is located, and in the second one, the intersection point is refined using a binary search.

The binary search does not take into account the depths of the microrelief. To overcome this problem, a linear search coupled with a secant one makes it possible to converge even more quickly by using the depths of the microrelief (Brawley & Tatarchuk, 2004; Yerex & Jagersand, 2004; Tatarchuk, 2006). An improvement of the relief mapping technique presented in (Ouazzani Chahdi et al., 2018) consists of choosing the number of iterations dynamically according to the relief's depth.

To converge rapidly towards the first intersection point, Donnelly introduced the notion of coding a conservative space in the sphere tracing technique (Donnelly, 2005). This is the first method that is based on the calculation of the empty space to converge quickly to the first intersection. This space is calculated during the preprocessing stage and during the rendering stage, a sphere tracing allows each iteration to approach significantly the first intersection with the relief.

The cylinder tracing was introduced in (Baboud & Decoret, 2006a). The preprocessing stage of this technique defines for each pixel of the depth map, a radius of a cylinder inside which, no viewing ray can pierce the relief more than once. During the search for the intersection, this radius allows moving forward without the risk of skipping the first intersection. The second step is to perform a binary search between the last two positions.

The cone tracing technique introduced in (Paglieroni & Petersen, 1994; Dummer, 2006; Policarpo & Oliveira, 2007) proposed to calculate the empty space as a form of top-opened cones using 2D texture. The technique has been proposed in two versions, the conservative technique (Paglieroni & Petersen, 1994; Dummer, 2006) and the relaxed one (Policarpo & Oliveira, 2007). Both versions were subsequently improved in (Halli et al., 2008). These improvements consist firstly of using linear algorithms $O(n)$ instead of quadratic ones $O(n^2)$ to compute the conservative and the relaxed cone. Secondly, calculating and storing the cones' radius instead of the cones' ratios thereby having cone angles to the order of $\pi/2$ rather than $\pi/4$, and finally extending the technique to support the non-square texture using elliptical rectification of cones during the rendering stage.

The third version of the cone has been proposed in (Ouazzani Chahdi et al., 2017), it consists of using a hybrid cone which is located between the conservative and the relaxed one so that the cone tracing pierces the relief only once and without the need for binary research.

Another way to calculate the empty space around a texel is to use a dilatation and an erosion map (Kolb & Rezk-Salama, 2005). These two maps are calculated from the depth map and allow having at each texel a secure region. The successive intersections of the viewing ray with these regions make it possible to converge to the intersection point.

Pyramidal displacement mapping introduced in (Oh, Ki & Lee, 2006; Tevs, Ihrke & Seidel, 2008) makes it possible to create a pyramidal structure of the depths by calculating in each time a map that is four times smaller than the previous one and taking the maximum of the depth of each group of four pixels. The intersection point between the viewing ray and the depths is obtained by the successive intersections with the horizontal lines representing the maximum depth of each level of the pyramid.

Per-pixel extrusion mapping consists of extruding the 3D models according to a binary form stored in a 2D texture without perturbing the basic mesh (Halli et al., 2009). The empty space is calculated by using the Euclidean Distance Transform EDT described in (Danielsson, 1980) and stored in a 2D texture called distance map, the normals of the extruded form are calculated from this later. The binary form, the distance map, and the normals are stored in a 2D texture called a shape map. Improvements were proposed to correct the intersection point between the viewing ray and the extruded form and to extend the extrusion algorithm for creating the outline extruded surfaces (Ragragui et al., 2017).

The algorithms of extrusion and revolution have been combined with a shape box to create extruded and revolved 3D objects without polygonal meshes (Halli et al., 2010). The two algorithms are based on the shape map. The extrusion consists of lifting the 2D binary form stored in the shape map, on the other hand, the revolution uses this one to create a revolved object around a revolution axis. For the texturing of revolved objects, we use one of the two projections, cylindrical or spherical. A rectification concerning these two types of projection has been proposed in (Ragragui, et al., 2018b).

3D objects created by extrusion or by revolution do not present any microrelief effect, that is to say, they are textured by the classic texture mapping technique. To solve this problem, two improvements have been proposed, one for extrusion (Ragragui et al., 2020) and the other for revolution (Ragragui et al., 2018a), which consists of making a combination with the bump mapping technique.

To manage the silhouette, four approaches have been proposed. The silhouette of an object is visible on the edges of the associated 3D mesh.

The first solution is to use a local representation of the 3D surface at each vertex. Two local representations

have been proposed. The first consists of using a quadratic approximation represented by two parameters (Jean, 2002; Oliveira & Policarpo, 2005). And the second consists of using a local space for each vertex (Chen & Chang, 2008; Na & Jung, 2008). The two representations are calculated and associated with each vertex during the preprocessing stage. During the rendering stage, the solution adopted makes it possible to determine the fragments belonging to the silhouette.

Shell mapping proceeds to the extrusion of each triangle of the mesh according to the normals of its three vertices (Hirche et al., 2004). The extrusion gives a prism constituted by eight triangles that will have to be included in the rendering stage. To avoid some discontinuity defects related to the bilinear interpolation, the prism is subdivided into three tetrahedrons using an algorithm described in (Shirley & Tuchman, 1990). The use of barycentric coordinates introduced in (Porumbescu et al., 2005) makes it possible to define a relation between each 3D point contained in the prism, and a single texel in the 3D displacement map. The use of semi-transparent 3D textures allows supporting some more advanced functionalities (Dufort, Leblanc & Poulin, 2005). A smoothing function coupled with the patches of coons makes it possible to eliminate strongly the distortions, and thus produces very satisfactory results (Jeschke, Mantler & Wimmer, 2007).

View-dependent displacement mapping (Wang et al., 2003; Wang et al., 2004) consists of calculating, for each viewing ray, the distance between each point of a polygon and the displacement surface. To be able to manage the silhouette, the curvature of the base surface must also be taken into consideration. A five-dimensional is thus defined to store the texture coordinates, the spherical coordinates of the viewing ray, and the curvature index of the surface along the viewing ray. This function represents a large amount of data, for this reason, it is compressed and stored as a 3D texture.

Image-based modeling and rendering techniques (IBMR) allow creating entire 3D objects without polygonal meshes based on per-pixel displacement mapping (Oliveira, Bishop & McAllister, 2000; Yerec & Jagersand, 2004; Baboud & Décoret, 2006a; Baboud & Décoret, 2006b; Policarpo & Oliveira, 2006; Ritsche, 2006; Toledo, Lévy & Levy, 2008; Toledo, Wang & Lévy, 2008; Halli et al., 2010; Ragragui et al., 2017; Ragragui et al., 2018a; Ragragui, et al., 2018b; Ragragui et al., 2020). Despite the diversity of the objects that can be created using these techniques, the silhouette problem persists. Once this problem is resolved and seen that modern graphics cards integrate and implement ray tracing algorithms, these techniques represent a better alternative to displacement mapping for creating 3D objects.

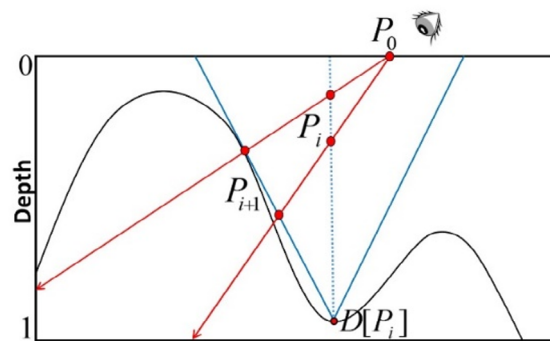
Cone tracing technique

In the pre-processing stage, the cone tracing technique calculates the empty space around each pixel of the depth map as a top-opened cone and stores its radius in a displacement map (Halli et al., 2008) (i.g. alpha channel). Then we use this space during the search for the intersection to converge quickly. This technique has been presented in two versions, the conservative technique and the relaxed one. In the first one, the cone is defined so that the cone tracing does not pierce the relief (Figure 13), and in the second one (Figure 14), the cone is defined so that the cone tracing cannot pierce the relief more than once, and then, the intersection point is refined using a binary search.

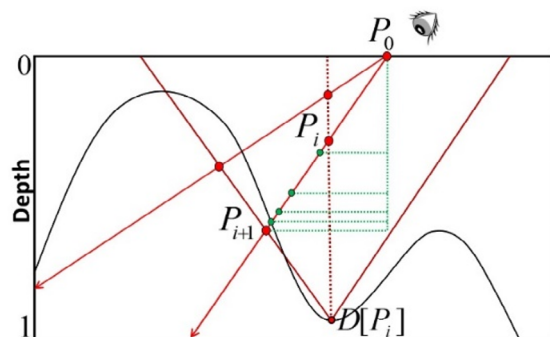
In both cases, the next t_{i+1} parameter is given by:

$$t_{i+1} = t_i + \frac{\text{cone_radius}(D[P_i] - P_{iz})}{D[P_i] \|v_{xy}\| + \text{cone_radius}} \quad (4)$$

The next point P_{i+1} is computed with the formula (3), where $D[P_i]$ is the depth at point P_i in this case, it represents the height of the cone.

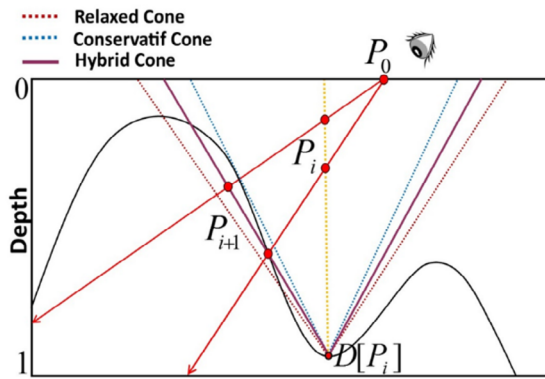


» **Figure 13:** Ray tracing in the cones map (cross-section). In each iteration, the following position P_{i+1} of the viewing ray is calculated according to the current position P_i and the value of the t_{i+1} parameter



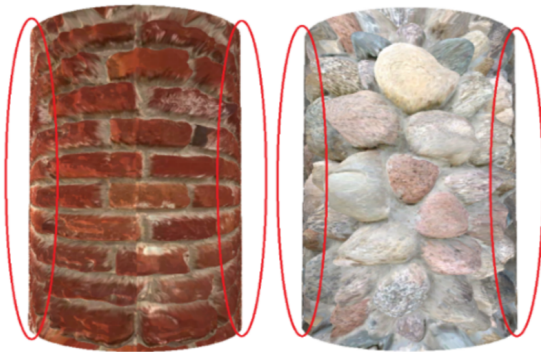
» **Figure 14:** The binary search phase with the relaxed cone tracing. It is made between the last position P_{i+1} and the starting position P_0 of the viewing ray

The third version of the cone has been proposed in (Ouazzani Chahdi et al., 2017), which is about the hybrid cone (Figure 15). The principle of this contribution is to use a cone that is located between the conservative cone and the relaxed one so that the cone tracing pierces the relief only once and without the need for binary research. This contribution further improves rendering quality and increases rendering speed.



» **Figure 15:** The hybrid cone is located between the conservative and the relaxed one (Ouazzani Chahdi et al., 2017)

Cone tracing (in its three versions) remains effective for real-time rendering on flat surfaces, but when it is about of the curved surfaces, the silhouette is not visible at the edges of the rendered objects (Figure 16).



» **Figure 16:** Rendering of a cylinder with the original cone-tracing techniques. The silhouette is not visible at the edges of the 3D objects. That is to say, the reliefs elevations coincide with the surface of the cylinder

To solve this problem, we propose to use the quadratic approximation approach to exploit its parameters in the cone-tracing phase to determine the silhouette fragment. The proposed contribution is based on the originals cone tracing techniques (Halli et al., 2008) and the quadratic approximation approach (Jean, 2002; Oliveira & Policarpo, 2005).

Quadratic approximation

The quadratic approximation was used with the relief mapping technique in (Oliveira & Policarpo, 2005), it consists of calculating an approximate quadratic surface for each vertex of the 3D mesh during a preprocessing stage, and in the rendering stage, this surface is used to adapt the ray-tracing process so that it takes into consideration the form of the mesh geometry.

The approximate quadratic surface is represented by two parameters a and b so that:

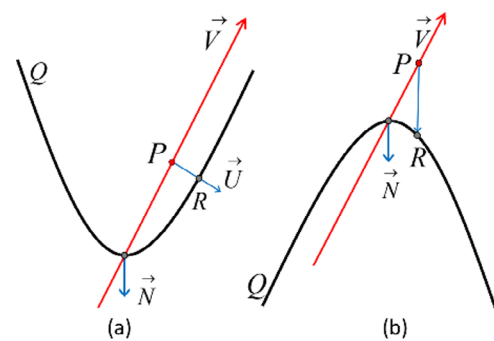
$$z = ax^2 + by^2 \quad (5)$$

where (x, y, z) are the coordinates of the processed vertex.

These parameters are calculated by using the quadrics (Jean, 2002): let E be the set of the triangles sharing a vertex $m_k(x_k, y_k, z_k)$, and let $M = \{m_1, m_2, \dots, m_n\}$ the set of the vertices in E . All the vertices in M are expressed in the tangent space associated with m_k . Given $M' = \{m'_1, m'_2, \dots, m'_n\}$, where $m'_i = (x'_i, y'_i, z'_i) = (x_i - x_k, y_i - y_k, z_i - z_k)$, the coefficients a and b are obtained by solving the following system $Ax = b$:

$$\begin{pmatrix} x_1'^2 & y_1'^2 \\ x_2'^2 & y_2'^2 \\ \vdots & \vdots \\ x_n'^2 & y_n'^2 \end{pmatrix} \times \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} z_1' \\ z_2' \\ \vdots \\ z_n' \end{pmatrix} \quad (6)$$

During the rendering stage, coefficients a and b will be interpolated for each pixel and then used to calculate the distance between the viewing ray and the quadratic surface. We have two cases as shown in Figure 17.



» **Figure 17:** Cross-section of two quadratic surfaces. On the left surface (a), the viewing ray is inside the quadric, and on the right surface (b), the viewing ray is outside. In both cases, the distance between the viewing ray and the quadric Q is given by the PR segment

V is the viewing ray and lets R be a point belonging to the quadric Q , U is the unit vector perpendicular to V at the point P .

In the first case (V inside the quadric, see Figure 17a); R is obtained by translating P by d units along the vector U:

$$R = P + Ud \quad (7)$$

The distance between the point P and the quadric is simply d , which can be obtained by substituting the coordinates of R in the equation of the quadric:

$$\begin{aligned} aR_x^2 + bR_y^2 - R_z &= 0 \\ a(P_x + dU_x)^2 + b(P_y + dU_y)^2 - (P_z + dU_z) &= 0 \end{aligned} \quad (8)$$

The solution of this equation gives:

$$d = \frac{-B + \sqrt{\Delta}}{2A} = \frac{-B + \sqrt{B^2 - 4AC}}{2A} \quad (9)$$

With $\Delta > 0$ and $\begin{cases} A = aU_x^2 + bU_y^2 \\ B = 2aP_xU_x + 2bP_yU_y - U_z \\ C = aP_x^2 + bP_y^2 - P_z \end{cases}$

In the second case (V outside the quadric, see Figure 17b), where $\Delta < 0$, the viewing ray is outside the quadric, in this case, the distance d is:

$$d = P_z - (aP_x^2 + bP_y^2) = P_z - q \quad (10)$$

We denote by q the quadric ($aP_x^2 + bP_y^2$) associated with the parameters a and b .

The texture space is planar, and in the rendering stage, the approximate surface calculated at each vertex of the 3D mesh during the preprocessing stage is used so that this space can be adapted to the 3D object geometry. In reality, the texture space remains always planar, and during the search for the intersection, the viewing ray is rectified to correct the position of the point P_{i+1} using the characteristics of the approximate surface. We denote by v and u respectively the vector V and U expressed in the texture space.

In the first case (Figure 17a), the next point P_{i+1} is corrected by:

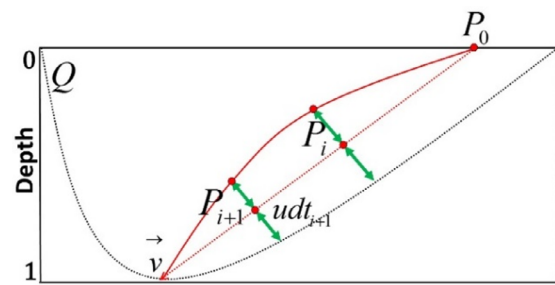
$$P_{i+1} = (v + ud)t_{i+1} = (v + w)t_{i+1} \quad (11)$$

Moreover, in the second case (Figure 17b), the next point P_{i+1} is corrected by:

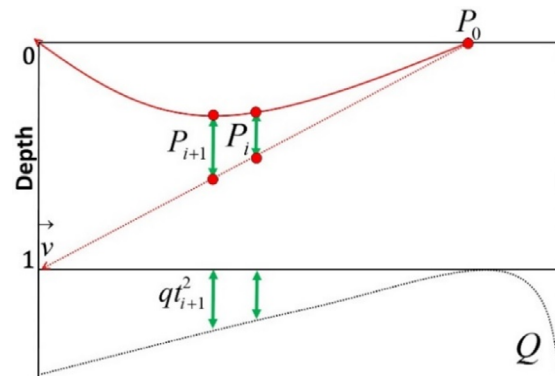
$$P_{i+1} = (v_x t_{i+1}, v_y t_{i+1}, v_z t_{i+1} - q t^2) \quad (12)$$

Since the depth of v is normalized (v/v_z), so, in the first case, the distance d must be divided by v_z , and in the second case, the quadric q must be divided by v_z^2 , and this before normalizing the depth of v .

Figure 18 and Figure 19 show the general appearance of the viewing ray during the search for the intersection.

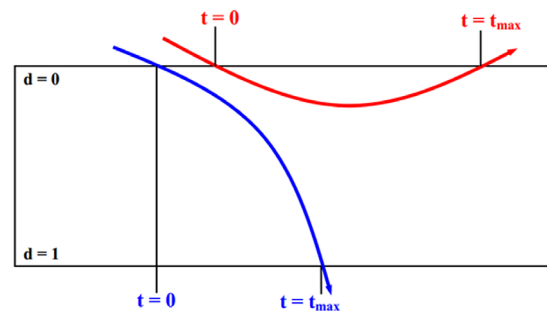


» **Figure 18:** The viewing ray is inside the quadric. At each iteration, we approach the quadratic surface



» **Figure 19:** The viewing ray is outside the quadric. At each iteration, we move away from the quadratic surface

During the linear search, the relief mapping technique (Policarpo, Oliveira & Comba, 2005) chooses the t parameter in the interval $[0, 1]$. This search is optimized in (Oliveira & Policarpo, 2005) by choosing this one in the interval $[0, t_{max}]$, with t_{max} is the smallest $t > 0$ such that the distance from the viewing ray to the quadric is equal to 0 or 1 (Figure 20).



» **Figure 20:** A ray that hits depth 1 ($d = 1$) in the texture space has reached the bottom of the depth field characterizing an intersection (the blue ray). On the other hand, a ray that returns to the depth 0 ($d = 0$) can be safely discarded as belonging to the silhouette (the red ray)

To find the most accurate value, t_{max} must be calculated by substituting (P_x, P_y, P_z) by $(V_x t, V_y t, V_z t)$ and setting $d=0$ and $d=1$ respectively in both equations (8) and (10), then solve for t . Algorithms 1 and 2 implement this optimization in both cases.

Algorithm 1: tMax1

```

Input: V, U, (a,b) | Output: tmax
Begin
  A ← a*V.x*V.x + a*V.y*V.y
  B ← 2*a*V.x*U.x + 2*b*V.y*U.y - V.z
  C ← a*U.x*U.x + b*U.y*U.y - U.z
  D ← B*B - 4*A*C
  If D > 0 Then
    tmax ← (B - Sqrt(D)) / (-2*A)
  EndIf
  D ← V.z/A
  If D > 0 Then
    tmax ← Min(tmax, D)
  EndIf
  tmax ← Abs(tmax)
End

```

Algorithm 2: tMax2

```

Input: V, q | Output: tmax
Begin
  D ← V.z*V.z - 4*q
  If D > 0 Then
    tmax ← (-V.z + Sqrt(D)) / (-2*q)
  EndIf
  D ← V.z/q
  If D > 0 Then
    tmax ← Min(tmax, D)
  EndIf
  tmax ← Abs(tmax)
End

```

During the search for the intersection, the parameter t_{i+1} is calculated by:

$$t_{i+1} = t_i + t_{max}/steps \quad (13)$$

At the end of the linear search, we check whether the value of the t parameter is greater than t_{max} , if this is the case, the pixel must be discarded, else the intersection point is refined with a binary search.

The combination of relief mapping with quadratic approximation produces satisfactory results, but when the depth scale is large enough or when the viewing ray shaves the surface, defects become visible as shown in Figure 21.



» **Figure 21:** The defects are visible in the parts where the viewing ray shaves the surface

Cone tracing with correct silhouette

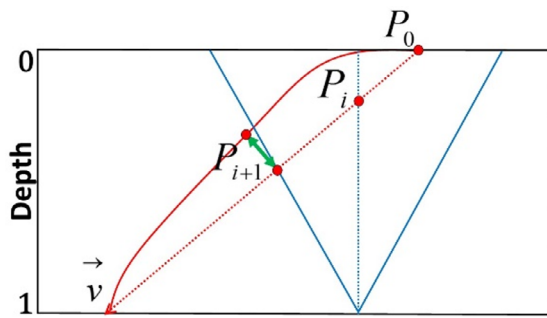
As mentioned above, the original cone tracing technique produces satisfactory results when it is about flat surfaces (Halli et al., 2008), but for the curved surfaces, the silhouette didn't render correctly at the edges of the 3D object. To correct the silhouette problem, we will use the parameters of the quadratic surface to adapt the cone tracing process so that it takes into account the characteristics of the 3D surface. For this, we opted two solutions. The first one uses a rectification of the viewing ray after each new displacement along the viewing ray and the second one uses a rectification of the cone before each new displacement along the viewing ray. Algorithm 7 of this section presents an implementation of the new cone-tracing techniques in both cases.

Rectification of the viewing ray

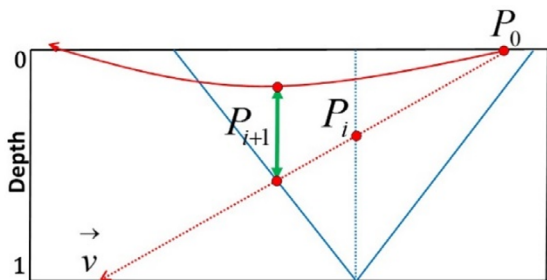
This rectification consists of adapting the displacements along the viewing ray so that they take into account the forms of the quadratic surfaces presented in Figure 17. In both cases, the t_{i+1} parameter is calculated with the formula (4).

In the first case (Figure 17a), the point P_{i+1} approaches the quadratic surface, and if there is an intersection, we converge quickly to the depth value 1 (Figure 22). This rectification is realized with the formula (11).

In the second case (Figure 17b), the point P_{i+1} moves away from the quadratic surface, that is to say, that we move away from the depth value 1. And if there is no intersection, we converge quickly to the depth value 0 (Figure 23). The rectification is realized with the formula (12).



» **Figure 22:** The viewing ray is inside the quadric, at each iteration, the point P_{i+1} is rectified according to the value du_{i+1}



» **Figure 23:** The viewing ray is outside the quadric. At each iteration, the depth of the point P_{i+1} is rectified according to the value $-qt_{i+1}^2$

Algorithms 3 and 4 implement this rectification. Figure 24 shows a comparison between a sphere rendered without and with silhouette correction and Figure 25 shows a torus rendered with this approach and by highlighting the polygonal meshes.

Algorithm 3: CurvedTransformeRay1

```

Input: p0, v, w, C, tmax | Output: t
Begin
  t ← 0
  For i=1 To STEPS And t <= tmax Do
    p ← p0 + (v + w)*t
    radius ← C[p.x,p.y].blue
    depth ← C[p.x,p.y].alpha
    t ← t + (radius * Max(depth - p.z,
      0) / (radius +
      depth*Length(v.xy)))
  EndFor
End

```

Algorithm 4: CurvedTransformeRay2

```

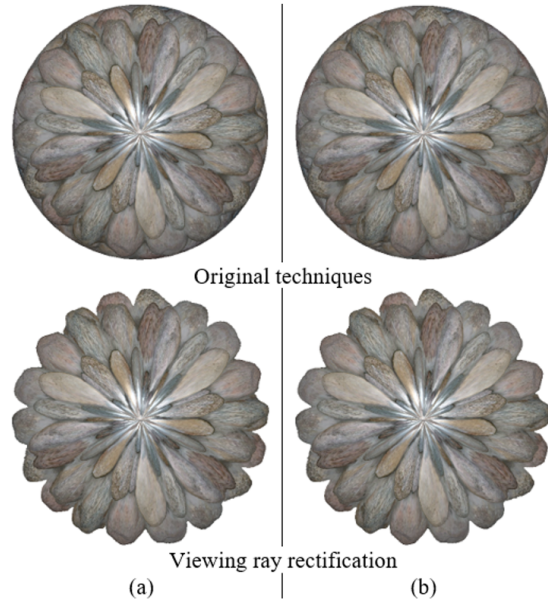
Input: p0, v, q, C, tmax | Output: t
Begin
  t ← 0
  For i=1 To STEPS And t <= tmax Do

```

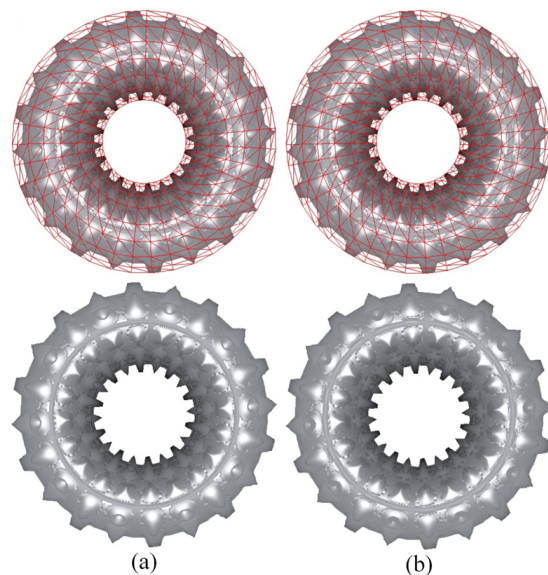
```

p ← p0 + v*t
p.z ← p.z - t*t*q
radius ← C[p.x,p.y].blue
depth ← C[p.x,p.y].alpha
t ← t + (radius * Max(depth - p.z,
  0) / (radius + depth*Length(v.xy)))
EndFor
End

```



» **Figure 24:** Comparison of a sphere rendered by the original cone tracing techniques and by using the viewing ray rectification. (a) Conservative technique. (b) Relaxed technique



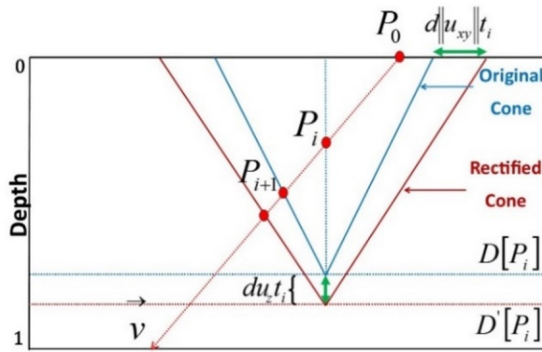
» **Figure 25:** Rendering of a torus by using the viewing ray rectification approach and by highlighting the polygonal meshes. (a) Conservative technique. (b) Relaxed technique

The change of the camera position does not influence the rectification process or the rendering quality since the rectification is realized in real-time and with each movement of the camera, we will have a new image rendered with a new rectification.

Rectification of the cone

The approximate surface has two forms, concave and convex (Figure 17). Instead of adapting the texture space to these forms, the cone is rectified so that it is influenced by the characteristics of the approximate surface. This rectification is realized before the cone tracing on its parameters, namely the radius and the height (cone depth).

In the first case (Figure 17a), the cone must be enlarged so that it approaches the quadratic surface. The rectification consists of increasing the values of the cones parameters stored in the displacement map by using the distance d and the vector u (Figure 26).



» **Figure 26:** The quadratic surface approaches the point P_{i+1} , so we move forward rapidly towards the intersection point

The depths increase, which implies the increase of the cones' depths (heights). The new cone depth is:

$$D'[P] = D[P] + du_{xy}t \quad (14)$$

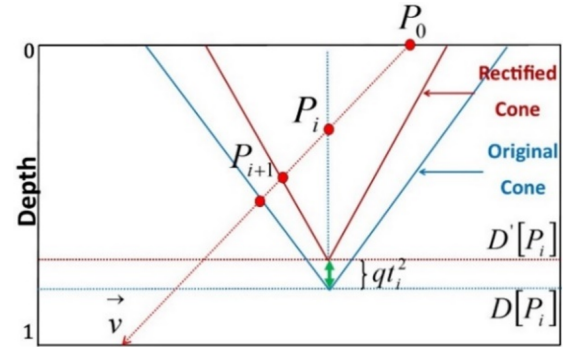
For the cone radius, we have:

$$cone_radius' = cone_radius + d|u_{xy}|t \quad (15)$$

Figure 26 shows that the displacement along the viewing ray with the rectified cone is faster than the base one because the displacement step increases. At each iteration, the cone rectification advances the P_{i+1} point along the viewing ray, and we converge more quickly in the case of an intersection.

In the second case (Figure 17b); the cone decreases so that it moves away from the quadratic surface. Indeed, the cone rectification consists of reducing

the values of its parameters stored in the displacement map by using the quadric q (Figure 27).



» **Figure 27:** The quadratic surface moves away from the point P_{i+1} , where the pixel belongs to the silhouette, we will don't have an intersection, so the pixel will be discarded

The depths decrease, which implies the decrease of cones depths (heights), so the new cone depth is:

$$D'[P] = D[P] - t^2q \quad (16)$$

we have:

$$cone_ratio = \frac{cone_radius}{D[P]} = \frac{cone_radius'}{D'[P]}$$

therefore, the new cone radius is given by:

$$cone_radius' = \frac{cone_radius}{D[P]} \times D'[P] \quad (17)$$

Figure 27 shows that the displacement along the viewing ray with the rectified cone is slower than the base one because the displacement step decreases. At each iteration, the cone rectification moves back the point P_{i+1} along the viewing ray, and we diverge in the case where there is no intersection.

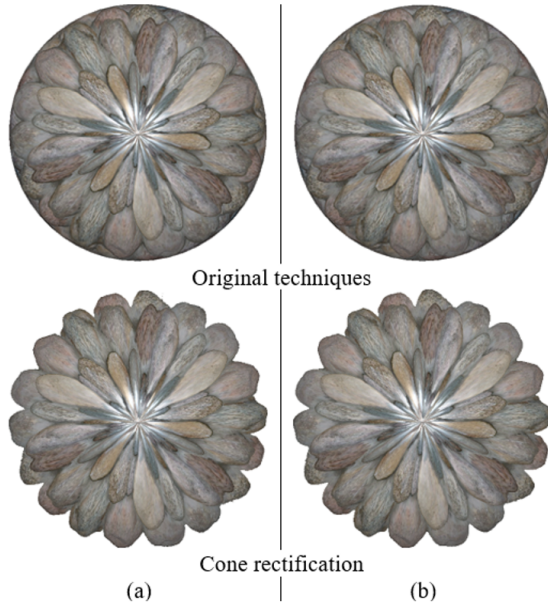
In both cases, the t_{i+1} parameter is calculated by:

$$t_{i+1} = t_i + \frac{cone_radius'(D'[P_i] - P_z)}{D'[P_i]||v_{xy}|| + cone_radius'} \quad (18)$$

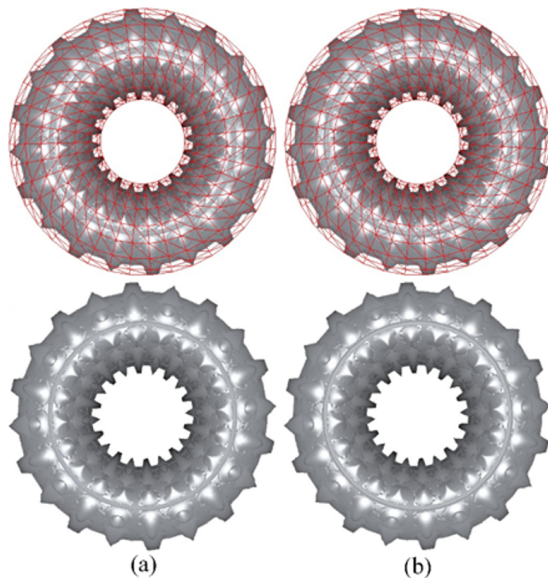
The next point P_{i+1} is computed with the formula (3). Algorithms 5 and 6 implement this rectification. Figure 28 shows a comparison between a sphere rendered without and with silhouette correction and Figure 29 shows a torus rendered with this approach and by highlighting the polygonal meshes.

Cone rectification does not depend on the depth map; it depends only on the quadratic parameters associated with the 3D surface. Also, the cones map is not attached to the base geometry onto which it is

mapped, because the rectification process is realized in real-time. This makes it possible to use the same rectification process and the same texture in real-time on different 3D objects, it means that the rectification process and the cones map are independent of the surface on which they will be used (Figure 30).



» **Figure 28:** Comparison of a sphere rendered by the original cone tracing techniques and by using the cone rectification. (a) Conservative technique. (b) Relaxed technique



» **Figure 29:** Rendering of a torus by using the cone rectification approach and by highlighting the polygonal meshes. (a) Conservative technique. (b) Relaxed technique

Algorithm 5: CurvedTransformeCone1

Input: p_0, v, w, C, t_{max} | Output: t

```

Begin
  p ← p0
  t ← 0
  For i=1 To STEPS And t ≤ tmax Do
    radius1 ← C[p.x,p.y].blue
    depth1 ← C[p.x,p.y].alpha
    radius2 ← radius1 + t*Length(w.xy)
    depth2 ← depth1 + t*w.z
    t ← t + (radius2 * Max(depth2 - p.z,
      0)/(radius2 +
    depth2*Length(v.xy))
    p ← p0 + v*t
  EndFor
End

```

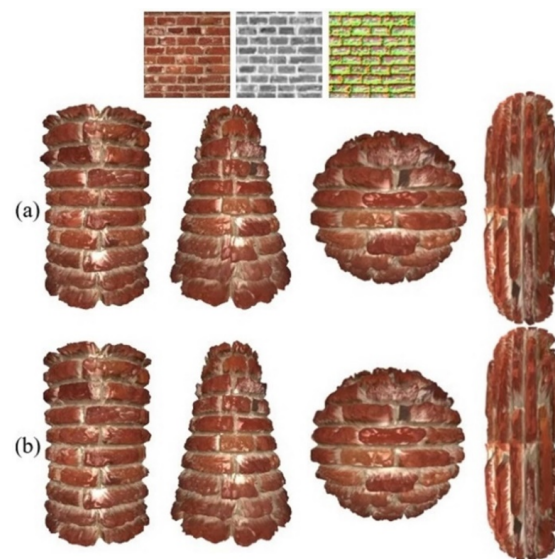
Algorithm 6: CurvedTransformeCone2

Input: p_0, v, q, C, t_{max} | Output: t

```

begin
  p ← p0 , t ← 0
  For i=1 To STEPS And t ≤ tmax Do
    radius1 ← C[p.x,p.y].blue
    depth1 ← C[p.x,p.y].alpha
    depth2 ← depth1 - t*t*q
    radius2 ← (radius1/depth1)*depth2
    t ← t + (radius2 * Max(depth2-
      p.z,0)/(radius2 +
    depth2*Length(v.xy))
    p ← p0 + v*t
  EndFor
End

```



» **Figure 30:** Rendering of several 3D objects in real-time with the same cone rectification process and with the same texture. (a) Conservative technique. (b) Relaxed technique

The curved cone tracing algorithm

In this subsection, we present the implementation of the cone tracing algorithm with silhouette correction in the two approaches.

In the rendering stage, the search for the intersection is performed in the texture space, but the calculations related to the quadratic approximation are performed in the tangent space where t is equal to 1 so that the quadratic distance is computed as the viewing ray progresses.

Algorithm 7: CurvedConeTracing

Input: (s, t) , T , V , U , (S_x, S_y, S_z) , C , R , (a, b)

Output: p

Begin

```
p0 ← (T.x*s, T.y*t, 0)
v ← Normalize(V/(Sx, Sy, Sz))
v.z ← -v.z
vz ← v.z
v ← v/v.z
vR ← v*(Length(v.xy)/
(Sqrt(v.x*v.x+R*R*v.y*v.y)))
A ← a*U.x*U.x + b*U.y*U.y
B ← 2*a*V.x*U.x + 2*b*V.y*U.y - U.z
C ← a*V.x*V.x + b*V.y*V.y - V.z
D ← B*B - 4*A*C
If D > 0 Then
    tmax ← tMax1(V, U, a, b)
    u ← Normalize(U/(Sx, Sy, Sz))
    w ← ((B - Sqrt(D))/-2*A)*u/vz
    t←CurvedTransformer-
ay1(p0, vR, w, C, tmax)
    //t←CurvedTransformer-
Cone1(p0, vR, w, C, tmax)
Else
    q ← a*V.x*V.x + b*V.y*V.y
    q ← Sign(q) * Max(Abs(q), 0.001)
    tmax ← tMax2(V, q)
    q ← (q/Sz)/(vz*vz)
    t←CurvedTransformer-
ay2(p0, vR, q, C, tmax)
    //t←CurvedTransformer-
Cone2(p0, vR, q, C, tmax)
EndIf
If t > tmax Then
    Discard
Else
    p ← p0 + v*t
```

```
EndIf
// Binary search (only in the
case of relaxed cones)
v ← (v*p.z)/2 // initial step size
p ← p0 + v // starting point
For i=1 To STEPS Do
    depth ← C[p.x,p.y].alpha
    v ← v/2
    If p.z < depth then
        p ← p + v
    Else
        p ← p - v
    EndIf
EndFor
End
```

Results and discussion

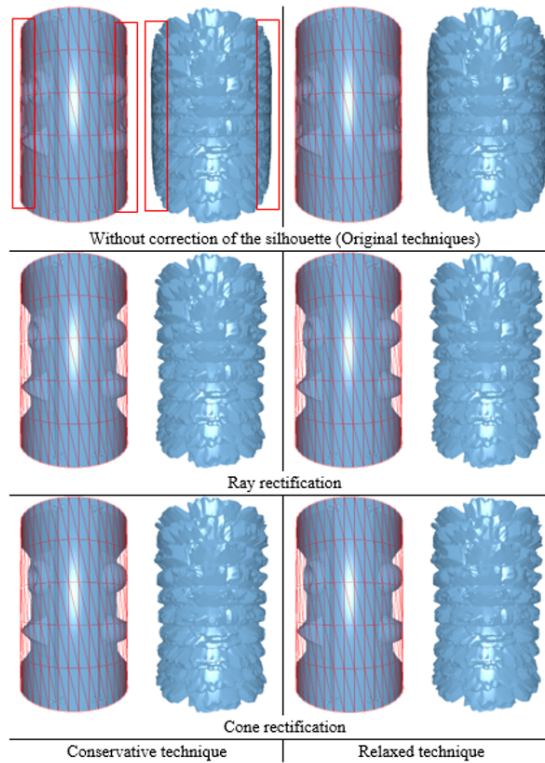
We have implemented the pre-processing part of the techniques discussed in this paper in C++. For rendering, we have exploited the programmable units of the GPU, namely Vertex Shader and Fragment Shader using OpenGL/GLSL. The figures are obtained using a Core-i7-4510U-2GH-4CPUs architecture with 8GB of RAM and GeForce-GT-840M with 4GB of memory.

In this paper, we have implemented different techniques of per-pixel displacement mapping, namely, conservative cone tracing, relaxed cone tracing, and relief mapping, to make a comparison with the proposed improvements. In our implementation, we have attached the magnification/minification method for the two camera positions (near and far) to the displacement map (cones map) and the color map using the same resolution (256×256, 512×512, 1024×1024, and 2048×2048).

The images of the figures are rendered with 25 linear steps and 5 binary steps. For the texture resolution, we have used 512×512. Figure 24 and Figure 28 show comparisons of a sphere rendered without and with silhouette correction using the two rectifications, namely the viewing ray rectification and the cone rectification. The images of Figure 25 are rendered with the viewing ray rectification approach and those of Figure 29 are rendered with the cone rectification approach. Figure 30 shows several 3D objects rendered with cone rectification using the same texture.

Figure 31 shows a comparison between the original cone tracing techniques and the addition of the viewing ray rectification and cone rectification. The figure shows clearly that the original techniques suffer from the silhouette problem. That is to say, during the search for the intersection point, the technique considers

that each processed 3D surface is flat, and does not take into consideration its curvature. Then, the reliefs at the edges of the 3D object are confused with its surface, which produces an incorrect rendering. The two proposed rectifications solved this problem.

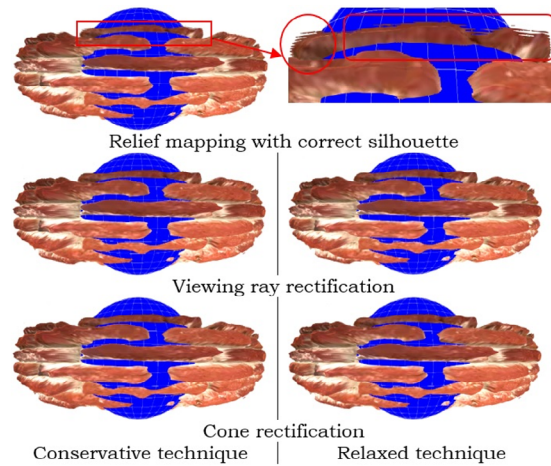


» **Figure 31:** Comparison of a cylinder rendered by the cone tracing techniques. In the case of the original techniques (Halli et al., 2008), the reliefs near the silhouette are not rendered correctly, that is to say, the reliefs are confused with the surface of the cylinder. On the other hand, with the help of the viewing ray rectification and the cone rectification, the silhouette is visible at the edges of the 3D objects

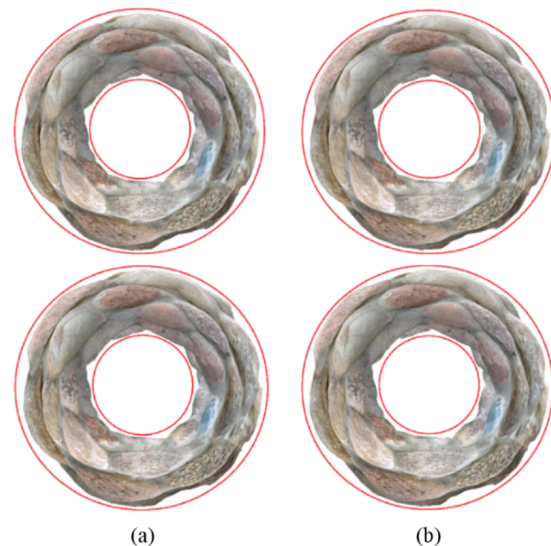
Concerning the comparison between relief mapping with correct silhouette (Oliveira & Policarpo, 2005) and cone tracing combined with the proposed rectifications, we have found that the major problem is related to the grazing angles. The combination of cone tracing with the quadratic approximation solves this problem.

Figure 32 shows the disappearance of the artifacts at the grazing angles in the images rendered by our rectifications. The same problem persists in the case of the interpenetration of 3D objects.

The images qualities of Figure 24, Figure 25, Figure 28, and Figure 29 are close, but in some cases, where the viewing ray or the depth scale is changed, small differences become visible as shown in Figure 33 and Figure 34. The figures show a comparison of a torus rendered with the approaches proposed in this paper.



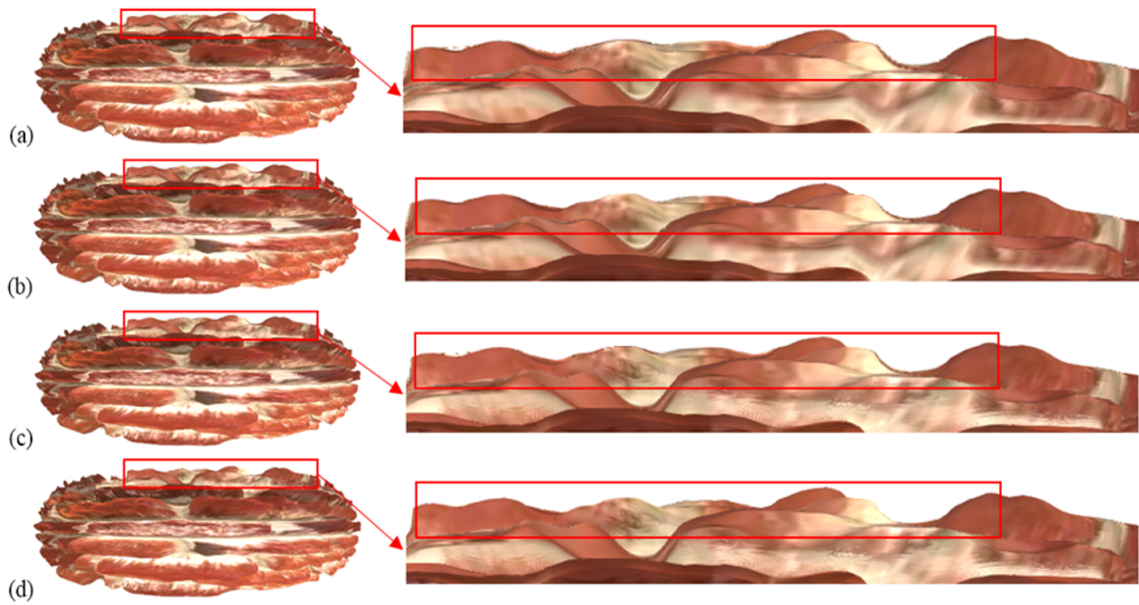
» **Figure 32:** The interpenetration of two 3D objects, a sphere, and a torus. We observe that the rendering done by the relief mapping with the correct silhouette present always the same problem related to the grazing angles, this problem is solved by our proposed rectifications as shown in the figure



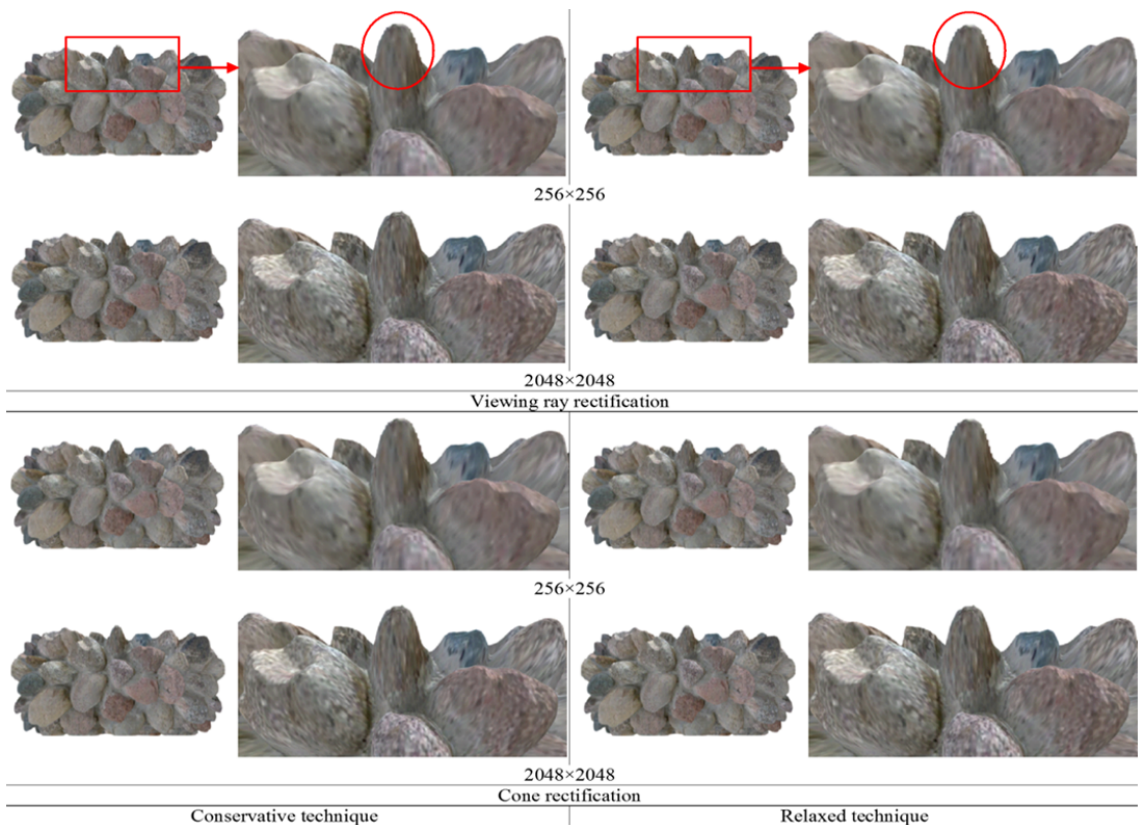
» **Figure 33:** Rendering of a torus by the approaches proposed in this paper with a depth scale equal to 0.4. (a) Conservative technique. (b) Relaxed technique. The approach by rectification of the viewing ray at the top and the approach by rectification of the cone at the bottom. Rendering differences are visible on the edges

The qualities of the images rendered by the two approaches (i.e. viewing-ray rectification and cone rectification) seem identical. Minimal differences can be observed when we use a minimal number of steps (i.e. linear steps ≤ 25 , binary steps ≤ 5), but when the number of steps is greater, the qualities of the rendering images become closer.

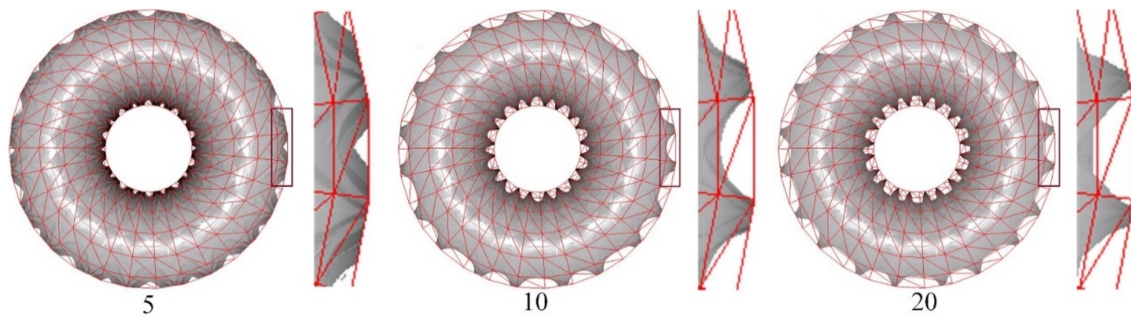
Figure 35 shows a cylinder rendered with the different approaches discussed in this paper using two resolutions



» **Figure 34:** Rendering of a torus by the four approaches of cone tracing with a depth scale equal to 0.6. (a) Conservative cone with viewing ray rectification. (b) Conservative cone with cone rectification. (c) Relaxed cone with viewing ray rectification. (d) Relaxed cone with cone rectification. We observe minimal differences at the edges between the different rendered images.



» **Figure 35:** Rendering of a cylinder with a texture resolution of 256x256 and 2048x2048 (depth map and color map have the same resolution). We observe that the resolution of the textures plays a very important role in the quality of the rendered images. The images rendered with a low resolution of texture present some aliasings which are corrected by using a texture with a high resolution



» **Figure 36:** Rendering of a torus by the cone tracing with correct silhouette using a different number of linear steps (5, 10, and 20), We observe that the appearance degree of the silhouette depends on the number of the steps. The image rendered with 20 steps presents the best correct silhouettes.

of the texture: 256×256 and 2048×2048 and two camera positions (far and near). The figure shows that the texture resolution plays a very important role in the quality of rendered images; a high resolution of texture allows having better quality and thus avoiding the aliasing problems.

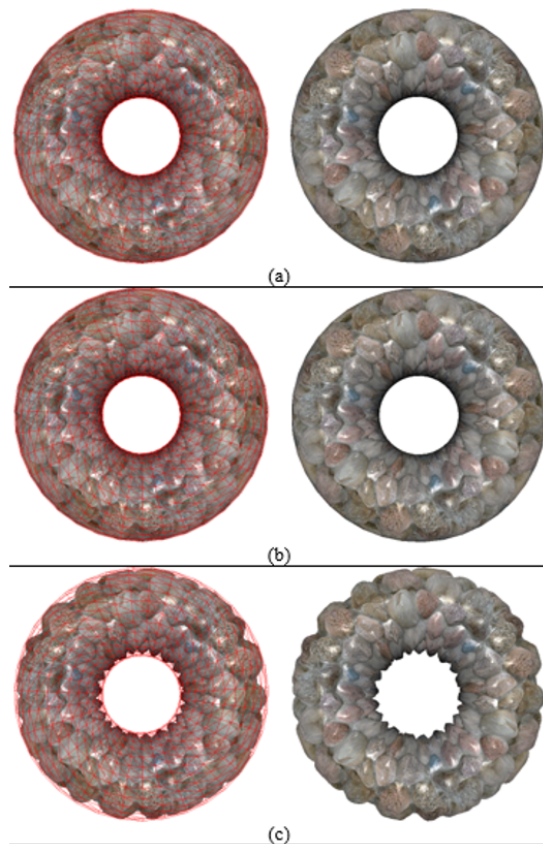
Figure 36 shows a torus rendered with the cone tracing with the correct silhouette using different steps number and by highlighting the polygonal meshes. We observe that the number of steps plays a very important role in the appearance of the microreliefs and the correct silhouette.

The proposal of the hybrid cone in (Ouazzani Chahdi et al., 2017) and the dynamic relief mapping in (Ouazzani Chahdi et al., 2018) made it possible to improve the rendering quality for flat surfaces. But when it is about curved surfaces, the silhouette is not treated correctly. Indeed, figure 37 shows the difference between these last two techniques and the proposed contributions.

Figure 38 shows the difference between a sphere and a cylinder which are rendered by revolution-bump mapping (Figure 38a), extrusion-bump mapping (Figure 38b), and cone tracing with a correct silhouette (Figure 38c).

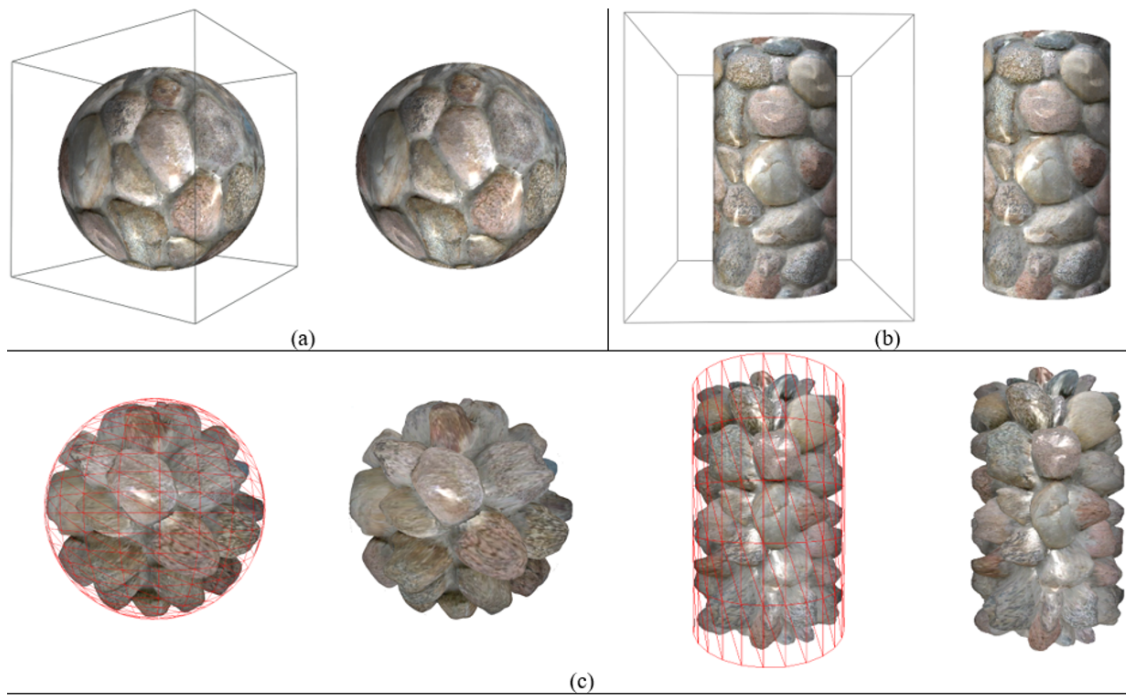
Revolution and extrusion are based on a shape box and the combination of the bump mapping allows adding a microrelief effect and does not create real displacements of the reliefs. Using a new ray-tracing algorithm, the revolution creates a 3D object around an axis of revolution based on a 2D form, and the extrusion extrudes this form upwards.

The objects created by these two techniques are not represented by any parametric surface which allows giving information on its curvature for each pixel, moreover, at the extrusion or revolution phase, the curvature of the extruded or the revolved form is not taken into account, and in this case, the treatment of the silhouette will be limited.



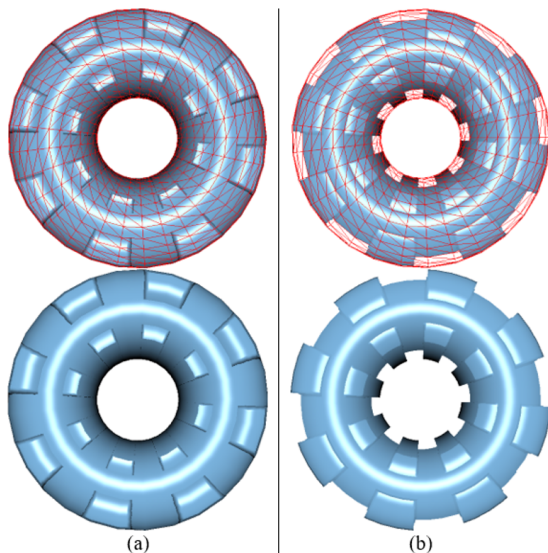
» **Figure 37:** Comparison of a torus rendered by the three techniques of cone tracing. (a) Hybrid cone (Ouazzani Chahdi et al., 2017). (b) Dynamic relief mapping (Ouazzani Chahdi et al., 2018). (c) Cone tracing with the correct silhouette. The problem concerning the silhouette is located at the edges of the torus rendered by the two techniques (a) and (b). This problem is solved with the help of the two proposed rectifications

In the case of per-pixel extrusion mapping, a new ray-tracing algorithm has been introduced in (Halli et al., 2009). Its advantage is the acceleration of the search for the intersection point, but the disadvantage is that it only deals with extrusion and does not take into account the silhouette treatment (Figure



» **Figure 38:** Rendering of a cylinder and a sphere. (a) Revolution with bump mapping (Ragragui et al., 2018a; Ragragui, et al., 2018b). (b) Extrusion with bump mapping (Ragragui et al., 2017; Ragragui et al., 2020). (c) Cone tracing with the correct silhouette. Revolved or extruded objects are rendered using a shape box and the bump mapping allows just a microrelief effect (simulation of small displacements). The objects created by cone tracing are rendered using a polygonal mesh and the reliefs are displaced without modifying the mesh geometry, moreover, the proposed cone rectifications make the silhouette visible

39a). The advantage of the algorithms proposed in this paper is that they are suitable for relief or extrusion and solve the silhouette problem (Figure 39b).



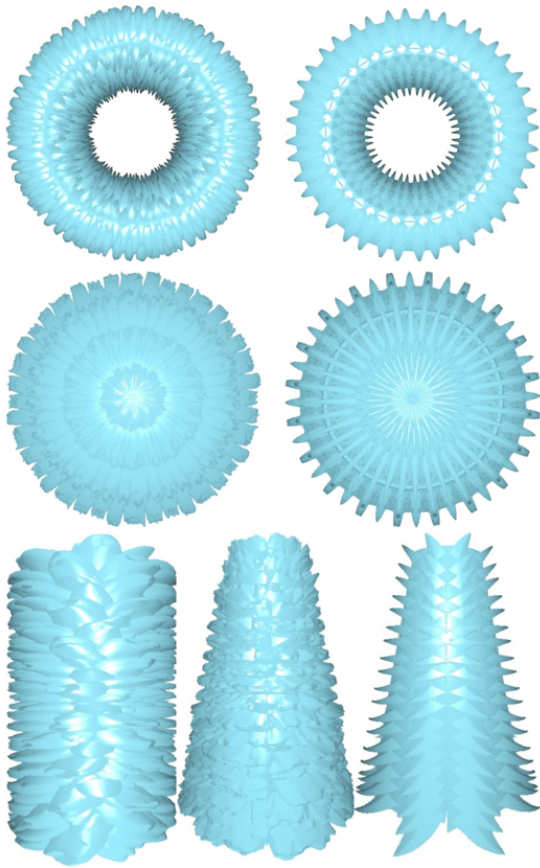
» **Figure 39:** Comparison of a torus rendered by two techniques. (a) Per-pixel extrusion mapping (Halli et al., 2009). (b) Cone tracing with the correct silhouette. Both techniques use a basic polygonal mesh but the silhouette problem is corrected only in the torus rendered by our approach

Figure 40 shows some extra examples with simple high-frequency displacements and no color texture. The 3D objects are rendered by the cone tracing technique with the correct silhouette and by using different depth maps and different depth values.

Figure 41 shows a vase rendered by the cone-tracing technique with the correct silhouette and by highlighting the basic polygonal mesh. The 3D object is rendered with different depth maps and different depth values. In the different images of the figure, we notice that the silhouette is corrected whatever the depth map used.

Generally, the combination of the quadratic approximation with the cone-tracing technique produces satisfactory results, but in some cases, this combination produces holes as shown in Figure 42. This problem is due to the use of the quadratic approximation for the local representation of the surface at each vertex. Because sometimes, the viewing ray pierces the relief in the object space and leaves it in the texture space. This problem has been also mentioned in (Jeschke, Mantler & Wimmer, 2007; Chen & Chang, 2008; Na & Jung, 2008).

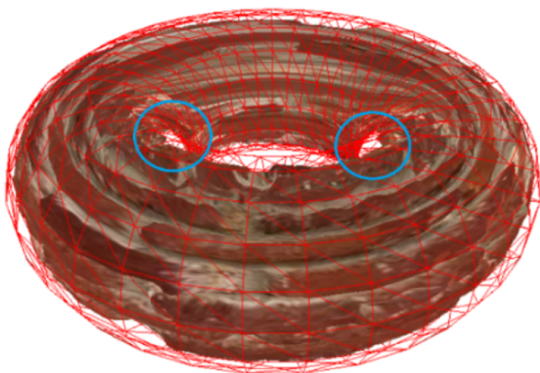
To surmount this problem, the quadratic approximation can be replaced by a local space at each vertex of the 3D mesh (Chen & Chang, 2008; Na & Jung, 2008).



» **Figure 40:** *Rendering of different 3D objects by the cone tracing techniques with correct silhouette and by using simple high-frequency displacements and no color texture*



» **Figure 41:** *Rendering of a vase with the cone tracing technique with correct silhouette by highlighting the basic polygonal mesh and by using different depth maps and different depth values*



» **Figure 42:** *Distortions and holes are due to the quadratic approximation.*

Indeed, this space makes it possible to give an idea of the curvature of the surface at each vertex, suddenly, its exploitation makes it possible to adapt the cone tracing so it takes into account the curvature of the surface.

To compare the rendering speed (Frames Per Second), we used high-resolution textures 1024×1024 and 2048×2048, 35 linear steps, 10 binary steps, and

a depth scale equal to 1. Table 1 shows the difference between the approaches discussed in this paper.

The table shows also the views on which the speed calculation is performed. It is clear that the approach by cone rectification is the fastest and relief mapping with correct silhouette remains always less fast compared to the proposed rectifications released during the cone tracing phase. It is also noted that the relaxed technique is always slower than the conservative one because its speed is slowed down by the binary search.

As shown in Table 1 and Table 2, the rendering speed decreases by a means of 37FPS for the viewing ray rectification and by 24FPS for the cone rectification compared to the originals cone tracing techniques. This slowdown is due to the processing concerning the correction of the silhouette. The stability of the images' quality rendered by per-pixel displacement mapping is influenced by the camera position and especially by the viewing angle, indeed, the ray-tracing algorithm uses the camera position to determine the viewing ray along which the searching for the intersection is performed.

Table 1

Comparison of the rendering speed FPS (Frames Per Second) between the discussed approaches with a torus. The approach by the cone rectification is the fastest.




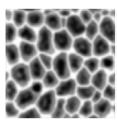
Screen 1000×600	Texture	Resolution	Conservative Technique			Relaxed Technique			Relief Mapping With Correct Silhouette
			Viewing ray rectification	Cone rectification	Without rectification	Viewing ray rectification	Cone rectification	Without rectification	
		1024 ²	210	225	227	198	210	226	202
		2048 ²	115	127	140	107	117	141	105
		1024 ²	230	258	280	211	227	262	197
		2048 ²	140	148	190	130	137	168	117
		Average	173	189	209	161	172	199	155

Table 2

The average FPS (Frames Per Second) number of the decrease in the rendering speed of the two proposed rectifications compared to the original cone tracing techniques.

	Viewing ray Rectification	Cone Rectification
Original Conservative Technique	36	20
Original Relaxed Technique	38	27

In the case of grazing viewing-angles, it is necessary to ensure that the silhouette is rendered correctly and that the iteration number of the ray-tracing algorithm is optimal. One of the solutions to have a stable quality is to determine the iteration number dynamically according to the viewing angle (Ouazzani Chahdi et al., 2018).

One must finally note that the improvements proposed in this article preserve all properties and characteristics of the cone-tracing technique with the latest improvements (Halli et al., 2008).

Conclusion

In this article, we have presented two new cone-tracing algorithms by combining the cone-tracing process with the quadratic approximation. This approximation consists of representing the 3D surface by approximate parameters at each vertex constituting the corresponding mesh.

During the cone-tracing phase, the first algorithm consists of using the parameters of the quadratic surface to

rectify the viewing ray. This rectification makes it possible to know whether the viewing ray pierces or leaves the relief and it is realized after each new displacement along the viewing ray. The second algorithm uses these parameters to rectify the cones parameters (i.e. depth and radius) to be influenced by the characteristics of the approximate surface. This rectification makes it possible to know whether the displacements along the viewing ray make it possible to have or not an intersection and it is realized before each new displacement.

In some cases, the qualities of the rendered images of the two approaches remain almost identical, except for minimal differences. However, when it is about the rendering speed (Frames Per Second), the approach by cone rectification remains the fastest. Also, the rendering quality of the microreliefs and the silhouette depends on the texture resolution and the number of steps of the cone-tracing algorithms. The choice between the conservative and the relaxed technique, the texture resolution, and the number of steps can be made according to the criterion of rendering quality/rendering speed.

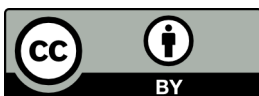
The advantage of the cone tracing techniques is that they can be combined with any silhouette correction approach with several possible improvements. Indeed, the processing of the silhouette depends on the ray-tracing algorithm, and how the fragments, belonging to the silhouette, are determined. So, a good coupling makes it possible to have a better algorithm of ray tracing and which supports the treatment of the silhouette. Another advantage of cone tracing is that it is possible to integrate it into the pipeline of new graphics cards since they integrate today the ray-tracing technology.

References

- Baboud, L. & Decoret, X. (2006a) Rendering geometry with relief textures. In: *Proceedings of the 2006 Conference on Graphics Interface, GI '06, 7-9 June 2006, Quebec, Canada*. pp. 195–201.
- Baboud, L. & Décoret, X. (2006b) 'Realistic Water Volumes in Real-Time', In: *Proceedings of the Second Eurographics Conference on Natural Phenomena, NPH'06, 5 September, 2006, Vienna, Austria*. Goslar, Eurographics Association. pp. 25-32.
- 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
- Brawley, Z. & Tatarchuk, N. (2004) Parallax Occlusion Mapping: Self Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing. In: Engel, W. (ed.) *ShaderX3: Advanced Rendering with DirectX and OpenGL*. Hingham, Massachusetts, Charles River Media, pp. 135-154.
- Catmull, E. E. (1974) *A subdivision algorithm for computer display of curved surfaces*. PhD thesis. The University of Utah.
- Chen, Y. C. & Chang, C. F. (2008) A prism-free method for silhouette rendering in inverse displacement mapping. *Computer Graphics Forum*. 27 (7), 1929–1936. Available from: doi: 10.1111/j.1467-8659.2008.01341.x
- 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
- 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.
- Dufort, J., Leblanc, L. & Poulin, P. (2005) Interactive Rendering of Meso-structure Surface Details using Semi-transparent 3D Textures. In: *Vision Modeling and Visualization, 16-18 November 2005, Erlangen, Germany*. Amsterdam, IOS Press. pp. 399-406.
- 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]
- Ernst, I., Ruesseler, H., Schulz, H. & Wittig, O. (1998) Gouraud bump mapping. In: *Euro98: 1998 Eurographics/SIGGRAPH on Graphics Hardware, 31 August 1998, Lisbon, Portugal*. New York, Association for Computing Machinery. pp. 47-54. Available from: doi: 10.1145/285305.285311
- 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
- Hirche, J., Ehlert, A., Guthe, S. & Doggett, M. (2004) Hardware accelerated per-pixel displacement mapping. *Graphics Interface*, 153–160.
- 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
- Jeschke, S., Mantler, S. & Wimmer, M. (2007) Interactive Smooth and Curved Shell Mapping. In: *Rendering Techniques 2007: Eurographics Symposium on Rendering, EGSR'07, 25-27 June 2007, Grenoble, France*. Aire-la-Ville, Eurographics Association. pp. 351-360
- 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.
- Kilgard, M. J. (2000) A Practical and Robust Bump-mapping Technique for Today's GPUs. In: *Game Developers Conference, GDC 2000, 9-13 March, San Jose, California*. pp. 1–39.
- Kolb, A. & Rezk-Salama, C. (2005) 'Efficient Empty Space Skipping for Per-Pixel Displacement Mapping. In: *Vision Modeling and Visualization, 16-18 November 2005, Erlangen, Germany*. Amsterdam, IOS Press.
- Lee, S. G., Park, W. C., Lee, W. J., Yang, S. B. & Han, T. D. (2007) An effective bump mapping hardware architecture using polar coordinate system. *Journal of Information Science and Engineering*. 23 (2), 569–588.
- Max, N. L. (1988) Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer*. 4 (2), 109–117. Available from: doi: 10.1007/BF01905562
- McGuire, M. & McGuire, M. (2005) Steep Parallax Mapping. In: *I3D 2005 Posters Session, ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games, 3-6 April 2005, Washington D.C., Washington*. Available from: <http://casual-effects.com/research/McGuire2005Parallax/mcguire-steep-parallax-poster.pdf> [Accessed 20th October 2021]
- Na, K.-G. & Jung, M.-R. (2008) Curved Ray-Casting for Displacement Mapping in the GPU. In: *Advances in Multimedia Modeling, 14th International Multimedia Modeling Conference - MMM 2008, 9-11 January 2008, Kyoto, Japan*. Berlin,

- Springer, Berlin, Heidelberg. pp. 348–357. Available from: doi: 10.1007/978-3-540-77409-9_33
- Oh, K., Ki, H. & Lee, C. H. (2006) Pyramidal displacement mapping: A GPU based artifacts-free ray tracing through an image pyramid. In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST 2006, 1-3 November 2006, Limassol, Cyprus*. New York, Association for Computing Machinery. pp. 75–82. Available from: doi: 10.1145/1180495.1180511
- Oliveira, M. M. (2000) *Relief Texture Mapping*. PhD thesis. University of North Carolina.
- 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., Rragragui, A., Halli, A. & Satori, K. (2017) Per-pixel displacement mapping using hybrid cone approach. In: *2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP), 22-24 May 2017, Fez, Moorocco*. New York, IEEE. pp. 1–4. Available from: doi: 10.1109/ATSIP.2017.8075577
- Ouazzani Chahdi, A., Rragragui, 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/ISACV.2018.8354053
- Pagliaroni, D. W. & Petersen, S. M. (1994) Height Distributional Distance Transform Methods for Height Field Ray Tracing. *ACM Transactions on Graphics*. 13 (4), 376–399. Available from: doi: 10.1145/195826.197312
- Parker, S. G., Friedrich, H., Luebke, D., Morley, K., Bigler, K., Hoberock, J., McAllister, D., Robison, A., Dietrich, A., Humphreys, G., McGuire, M. & Stich, M. (2013) GPU ray tracing. *Communications of the ACM*. 56 (5), 93. Available from: doi: 10.1145/2447976.2447997
- 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
- Porumbescu, S. D., Budge, B., Feng, L. & Joy, K. I. (2005) Shell maps. *ACM Transactions on Graphics*. 24 (3), 626–633. Available from: doi: 10.1145/1073204.1073239
- Premecz, M. (2006) Iterative Parallax Mapping with Slope Information. In: *10th Central European Seminar on Computer Graphics, CESC 2006, 24-25 April 2006, Castá-Papiernicka, Slovakia*. Austrian Computer Society.
- Rragragui, A., Ouazzani Chahdi, A., Halli, A. & Satori, K. (2017) Per-Pixel Extrusion Mapping: The correction of the intersection point between the extrusion geometry and the viewing ray. In: *2017 Intelligent Systems and Computer Vision, ISCV 2017, 17-19 April 2017, Fez, Morocco*. New York, IEEE. Available from: doi: 10.1109/ISACV.2017.8054957
- Rragragui, A., Ouazzani Chahdi, A., Halli, A. & Satori, K. (2018a) Per-pixel revolution mapping with rectification of the texture projection. In: *2018 International Conference on Intelligent Systems and Computer Vision, ISCV 2018, 2-4 April 2019, Fez, Morocco*. New York, IEEE. Available from: doi: 10.1109/ISACV.2018.8354056
- Rragragui, A., Ouazzani Chahdi, A., Halli, A. & Satori, K. (2018b) Revolution mapping with bump mapping support. *Graphical Models*. 100, 1–11. Available from: doi: 10.1016/j.gmod.2018.09.001
- Rragragui, 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
- Ritsche, N. (2006) Real-time shell space rendering of volumetric geometry. In: *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia, GRAPHITE '06, 29 November - 2 December 2006*. New York, Association for Computing Machinery. p. 265. Available from: doi: 10.1145/1174429.1174477
- Shirley, P. & Tuchman, A. (1990) A polygonal approximation to direct scalar volume rendering. In: Péroche, B. & Rushmeier, H. (eds.) *Proceedings of the 1990 Workshop on Volume Visualization, VVS 1990, 10-11 December 1990, San Diego, California*. New York, Association for Computing Machinery, pp. 63–70. Available from: doi: 10.1145/99307.99322
- Sloan, P.-P. J. & Cohen, M. F. (2000) Interactive Horizon Mapping. In: *Eurographics Workshop on Rendering Techniques, EGSR 2000, 26-28 June 2000, Brno*,

- Czech Republic. Vienna, Austria, Springer. pp. 281–286. Available from: doi: 10.1007/978-3-7091-6303-0_25
- Sung Kim, J., Hyun Lee, J. & Ho Park, K. (2001) A fast and efficient bump mapping algorithm by angular perturbation. *Computers & Graphics*. 25 (3), 401–407. Available from: doi: 10.1016/S0097-8493(01)00064-4
- Szirmay-Kalos, L. & Umenhoffer, T. (2008) Displacement Mapping on the GPU — State of the Art. *Computer Graphics Forum*. 27 (6), 1567–1592. Available from: doi: 10.1111/J.1467-8659.2007.01108.X
- Tatarchuk, N. (2006) Dynamic parallax occlusion mapping with approximate soft shadows. *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. 63-69. Available from: doi: 10.1145/1111411.1111423
- Tevs, A., Ihrke, I. & Seidel, H. P. (2008) Maximum mip-maps for fast, accurate, and scalable dynamic height field rendering. In: *Proceedings of the Symposium on Interactive 3D Graphics and Games, I3D08, 15-17 February 2008, New York, New York*. New York, Association for Computing Machinery. pp. 183–190. Available from: doi: 10.1145/1342250.1342279
- Toledo, R., Lévy, B. & Levy, B. (2008) Visualization of Industrial Structures with Implicit GPU Primitives. In: *4th International Symposium on Visual Computing, ISVC08, 1-3 December 2008, Las Vegas, Nevada*. Berlin, Springer-Verlag. pp. 139-150. Available from: doi: 10.1007/978-3-540-89639-5_14
- Toledo, R., Wang, B. & Lévy, B. (2008) Geometry Textures and Applications. *Computer Graphics Forum*. 27 (8), 2053–2065. Available from: doi: 10.1111/J.1467-8659.2008.01185.X
- Wang, L., Wang, X., Tong, X., Lin, S., Hu, S., Guo, B. & Shum, H. Y. (2003) View-dependent displacement mapping. *ACM Transactions on Graphics*. 22 (3). Available from: doi: 10.1145/1201775.882272
- Wang, X., Tong, X., Lin, S., Hu, S., Guo, B. & Shum, H. Y. (2004) Generalized Displacement Maps. In: *Proceedings of the 15th Eurographics Workshop on Rendering Techniques, 21-23 June, Norköping, Sweden*. 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*. Available from: http://page.mi.fu-berlin.de/block/htw-lehre/wise2015_2016/bel_und_rend/skripte/welsh2004.pdf [Accessed 20th October 2021]
- Yerex, K. & Jagersand, M. (2004) Displacement Mapping with Ray-casting in Hardware. In: *ACM Siggraph 2004 Sketches, SIGGRAPH '04, 8-12 August 2004, Los Angeles, California*. New York, Association for Computing Machinery. p. 2000. Available from: doi: 10.1145/1186223.1186410



© 2021 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/>).