

ADNANE OUZZANI CHAHDI  
ANOUAR RAGRAGUI  
AKRAM HALLI  
KHALID SATORI

## PER-PIXEL EXTRUSION MAPPING WITH CORRECT SILHOUETTE

**Abstract** *Per-pixel extrusion mapping consists of creating a virtual geometry that is stored in a texture over a polygon model without increasing its density. There are four types of extrusion mapping; namely, basic extrusion, outward extrusion, beveled extrusion, and chamfered extrusion. These different techniques produce satisfactory results in the case of plane surfaces; however, when it is about curved surfaces, a silhouette is not visible at the edges of the extruded forms on the 3D surface geometry, as they not take the curvatures of the 3D meshes into account. In this paper, we present an improvement that consists of using curved ray-tracing to correct the silhouette problem by combining per-pixel extrusion-mapping techniques with a quadratic approximation that is computed at each vertex of a 3D mesh.*

**Keywords** image-based rendering, real-time rendering, texture mapping, per-pixel extrusion mapping, ray-tracing, silhouette

**Citation** Computer Science 22(3) 2021: 407–432

**Copyright** © 2021 Author(s). This is an open access publication, which can be used, distributed and reproduced in any medium according to the Creative Commons CC-BY 4.0 License.

## 1. Introduction

Per-pixel extrusion mapping [14, 15, 35, 38] is an image-based modeling and rendering (IBMR) technique that is inspired by per-pixel displacement mapping. Image-based modeling and rendering techniques are based on textures to store geometry-related data. This geometry is subsequently rendered by using a simple ray-tracing algorithm that runs in the programmable units of a graphics card.

Contrary to the conventional techniques of per-pixel displacement mapping that are based on grayscale images that represent any relief, extrusion mapping uses a 2D binary image where only the zero-values pixels constitute the basic form of the pattern to be extruded (Figure 4a).

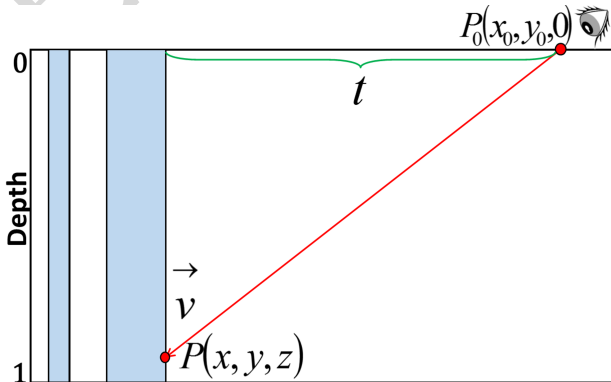
In order to find the intersection point between the viewing ray and the extruded form, this technique uses a distance map that is calculated from the basic shape that is stored in a binary image to converge more quickly.

We notate the viewing ray that is expressed in a texture space as  $v$  (as shown in Figure 1); the P point along viewing ray  $v$  is expressed by the following:

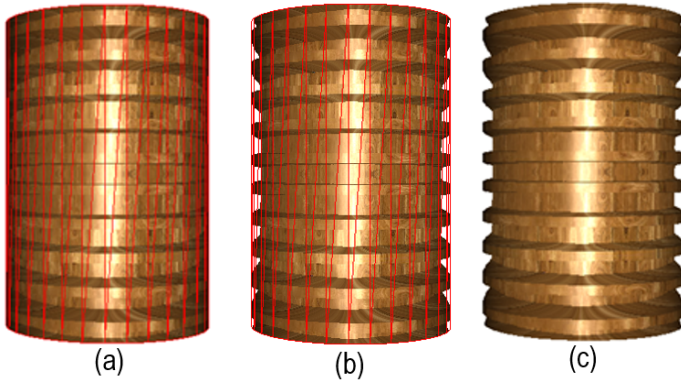
$$P = vt. \quad (1)$$

The problem with this technique is that it does not support a silhouette at the edges of a 3D mesh, as it does not take the curvatures of the mesh into account. Indeed, Figure 2 shows a comparison of a cylinder that is rendered with the extrusion-mapping technique with and without the correction of the silhouette and by highlighting the polygonal mesh.

We notice that a silhouette is not visible at the edges of a 3D object when rendered with basic extrusion mapping (Figure 2a). This problem is surmounted by using a curved ray-tracing algorithm for searching for the intersection point; Figures 2b and 2c show the same 3D model after being rendered by our approach. One can see that it provides a correct silhouette at the edges of the 3D object.



**Figure 1.** First intersection point between viewing ray and pattern is  $P(x, y, z)$ . Search for first intersection consists of finding value of  $t$  parameter.



**Figure 2.** Comparison of cylinder rendered with extrusion-mapping technique without and with silhouette correction and by highlighting polygonal mesh: (a) basic extrusion mapping; (b) and (c) correction of silhouette by using curved ray-tracing algorithm. We observe that silhouette problem is corrected by our approach.

Based on this observation, this paper presents a new extrusion-mapping algorithm based on a quadratic approximation at each vertex of a 3D mesh [26,30].

The proposed solution consists of curving the viewing rays during the search for the intersection point by using the parameters of the quadratic surface. This allows the ray-tracing algorithm to take the curvatures of the 3D meshes into account and determine which parts belong to the silhouette. The algorithm is presented for the different types of extrusion mapping.

## 2. Related works

Contrary to the displacement mapping [8] (which modifies the geometry), the bump-mapping technique [3, 29] intervenes only at the shading phase. Since the latter is a function of the normal, the disturbance of the normals will lead to the illusion of a microrelief. Thus, it is enough to disturb the normal to the surface and to use it in a shading formula.

Per-pixel displacement mapping [28] allows very fine details to be added to a polygon-based mesh without increasing its density, which is contrary to displacement mapping [8]. This allows for displaying very detailed models while avoiding the bottleneck that can be caused by a very large number of vertices and polygons.

An extension of bump mapping [3, 29], parallax mapping [4, 18, 21, 34, 41, 45] allows us to take the parallax effect into account (a distortion of the texture). This technique performs an approximate search for the intersection between the viewing direction and the relief that is 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.

Relief mapping [6, 31, 32] is an extension of another technique called relief texture mapping [24, 25]. This consists of starting a search from the current pixel and advancing with regular steps in the viewing direction. This operation is repeated as long as the current position is above the relief. Then, the intersection point is refined with a binary search.

The binary search does not take the depths of the microrelief into account during the search. To overcome this problem, a linear search coupled with a secant search used in [4, 12, 41, 46] makes it possible to converge even more quickly using the depths of the microrelief during the search.

Sphere tracing is the first method that is based on precomputing the empty space to quickly converge to the first intersection point. Introduced in [10], this technique creates a 3D map from a 2D depth map, where each 3D texel receives a value of 1 if it belongs to the relief and 0 if not; it then calculates its Euclidean distance transform, which gives the minimum distance to the relief for each texel. During the search for the intersection, a sphere's tracing allows each iteration to be significantly closer to the first intersection with the relief.

A dilatation map and erosion map were introduced in [20] to define the empty space. These two maps are calculated from a depth map and allow us to have a secure region (empty space) in each texel. The successive intersections of the viewing ray with these regions make it possible to converge to the intersection point with the microrelief.

concerning the sphere tracing [1], the pre-processing step consists of defining a radius of a cylinder for each pixel of a depth map within which no viewing direction can pierce the relief more than once. During the search phase of the intersection, this radius makes it possible to advance without the risk of jumping over the first intersection. The second step is to perform a binary search between the last two positions.

The conservative [27, 32] and relaxed [1] cone techniques propose storing the empty space in the form of top-opened cones using 2D textures. Some improvements of cone-tracing techniques were subsequently proposed in [13]. A third version of a cone was proposed in [5].

Introduced in [19, 23, 42], pyramidal displacement mapping makes it possible to create a pyramidal structure of the depths by calculating a map each time 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 is obtained by the successive intersections, with the horizontal lines representing the maximum depth of each level of the pyramid.

More advanced and general techniques were presented in [2], allowing us to efficiently render details expressed as height fields, for instance, using safety shapes.

Per-pixel extrusion mapping [14, 35] consists of extruding 3D models according to a binary form stored in a 2D texture without perturbing the basic mesh. The empty space is calculated by using the Euclidean distance transform (EDT) described in [9]

and stored in a 2D texture called a distance map; the normals of the extruded form are calculated from this map.

Per-pixel extrusion and revolution mapping were combined with a shape box to create a 3D objects without a polygonal mesh [15, 36–38].

With the exception of displacement mapping, the techniques presented thus far do not manage silhouettes. The silhouette of an object is visible on the edges of its associated 3D mesh. Four approaches have been proposed.

Introduced in [43], view-dependent displacement mapping takes a precalculation approach. The main idea is to calculate the distance between each point of a polygon and the surface of displacement for each viewing direction. A five-dimensional function is defined to store the result. This technique has been generalized in [44] in order to manage 3D depth maps and limit some distortion.

One of the best solutions for silhouette support is to extrude a base mesh to create an area that will hold the microrelief to be mapped on the surface (shell mapping) [11, 16, 33, 39]. This shell is obtained by extruding each triangle of the mesh following the normals of its three vertices. In order to avoid some discontinuity defects that are related to the bilinear interpolation, the prism is subdivided into three tetrahedrons using an algorithm that is described in [40]. The division of the prism into tetrahedrons is not the only solution to reduce distortions. Indeed, the smoothing function coupled with the patches of coons introduced in [17] makes it possible to strongly eliminate the distortions and, thus, produce very satisfactory results.

In order to be able to represent the silhouette fragments, the quadratic approximation relies on the local approximation of the curvature at each point of a surface [26]. This approximation is based on quadrics [30]. The approximate surface is used in the rendering stage to discard the silhouette fragments.

An improvement consists of replacing the quadratic approximation by the tangent space [7, 22] to represent the object geometry locally at each vertex. This space is computed and stored as a texture in a pre-processing stage, then it is used during the search for the intersection to have a piece-wise linear approximation of a curved viewing ray in the texture space.

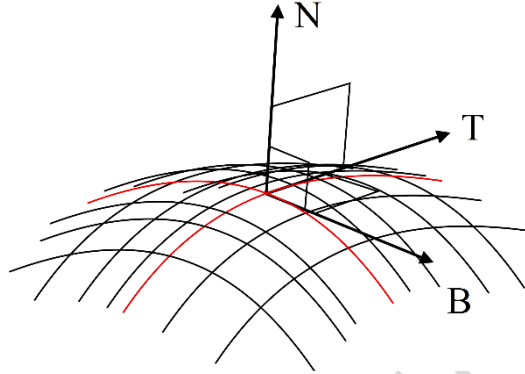
### 3. Per-pixel extrusion mapping

Per-pixel extrusion mapping [14, 15, 35] is based on three main elements: the tangent space, the shape map, and the ray-tracing algorithm.

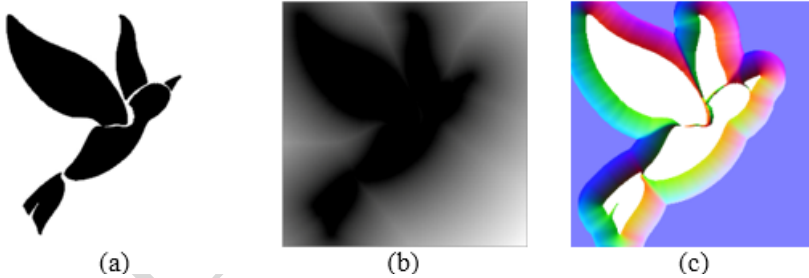
The tangent space [29] is a local space for each vertex that constitutes a 3D mesh (Figure 3). This space is calculated as a function of the normal to the vertex and the associated texture coordinates. The viewing ray vector and light vector must be expressed in this space.

A shape map is an RGBA texture that contains the needed data for the extrusion-mapping algorithms (Figure 4). The alpha channel serves to store the basic shape that is represented by a binary image, where only the zero-value pixels are considered

to be part of the form to be extruded. The blue channel is used to store a distance map that is calculated from the alpha channel. Finally, the red and green channels contain the  $x$  and  $y$  components, respectively, of the normal that is calculated from the distance map.



**Figure 3.** Tangent space. This is local space constituted by three vectors (normal, binormal, and tangent) associated with each vertex of 3D mesh.



**Figure 4.** Illustration of shape map: (a) basic form that will be extruded on polygonal model; (b) distance map calculated from basic form; (c) shape map that stores basic form in  $\alpha$  channel, Euclidean distance transform in blue channel, and  $x$  and  $y$  components of normal are stored in red and green channels, respectively.

Ray-tracing is an algorithm that searches for the intersection of a viewing ray with the extruded form stored in the shape map (Figure 5). To find the next  $P_{i+1}$  point, we use the following formulas:

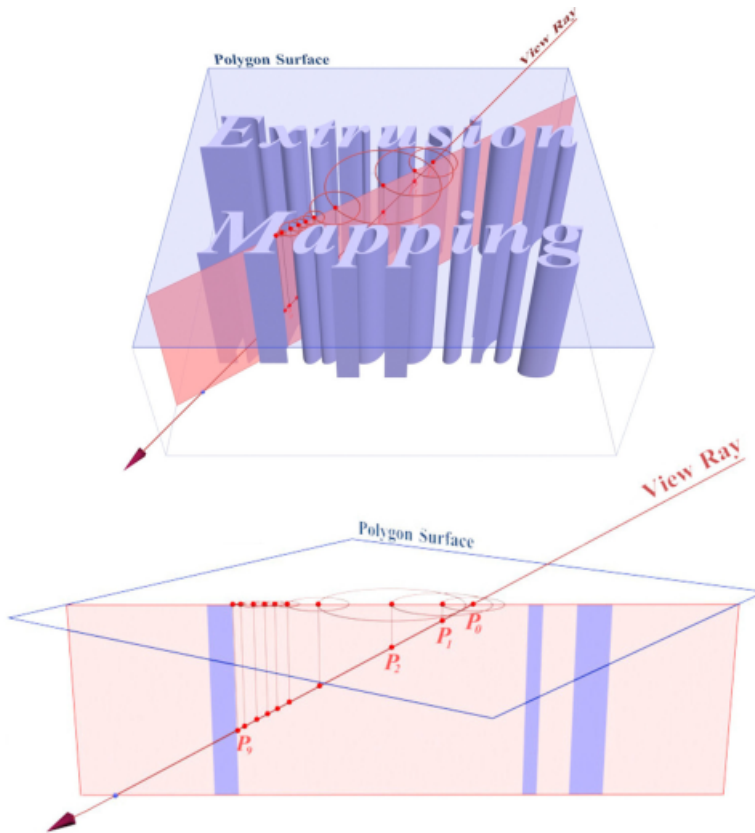
$$t_{i+1} = t_i + D(P_i) \quad (2)$$

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

where  $t_i$  is the sum of the distances at iteration  $i$ , and  $D(P_i)$  represents the distance at the  $(x_i, y_i)$  coordinate that separates the  $P_i$  point from the extruded form, which is calculated using the following formula:

$$D(P_i) = \tau EDT(P_i), \quad (4)$$

where  $EDT(P_i)$  represents the Euclidean distance transform that is extracted from the shape map at coordinates  $(x_i, y_i)$ , and  $\tau$  represents the scale factor [14,15]. Figure 5 shows the process of searching for the intersection.



**Figure 5.** Ray-tracing process associated with per-pixel extrusion mapping [14,15]. At each iteration, circle tracing is performed to converge more quickly and without risk of jumping first intersection.

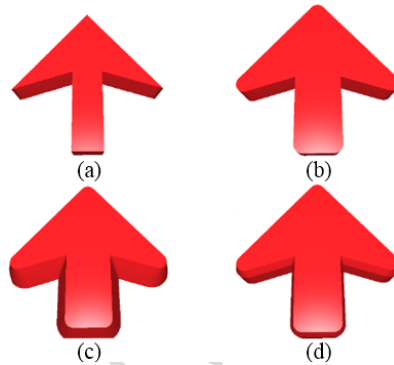
The use of the  $EDT$  for space leaping during ray-tracing allows for faster convergence and avoids missing intersections at grazing angles. Since the distance fields will usually be stored in the 8-bit integer format channel, we must remap the distances to a range of  $[0, 1]$  in order to have an optimal  $EDT$  distribution. Thus, we modulate

the distances in term of the maximum value of the EDT and a scale value  $f$  that is used in the pre-processing stage to control the distance distribution, so the scale factor is calculated by the following formula:

$$\tau = f \max(EDT)/W, \quad (5)$$

where  $W$  is the width of the shape map.

Other versions of the per-pixel extrusion mapping are presented in [15]. Namely, beveled extrusion mapping (Figure 6c), which consists of creating an outward extrusion to outside that varies as a function of the depth scale, and chamfered extrusion mapping (Figure 6d), which consists of limiting the bevel effect to a certain depth scale (between 0 and 1) in order to have edges with a chamfer.



**Figure 6.** Different types of extrusion-mapping techniques: (a) basic extrusion; (b) outward extrusion; (c) beveled extrusion; (d) chamfered extrusion.

About beveled extrusion mapping [15], we start by searching for the intersection with the original form of the distance using Formula (3). During this search, it is necessary to test whether the current position is within the geometry using the following difference [15]:

$$D_b(P_i) = D(P_i) - bz_i, \quad (6)$$

where  $z_i$  represents the depth component of the  $P_i(x_i, y_i, z_i)$  point, and  $b$  is a parameter that is used to control the bevel range during the rendering stage [15].

If difference  $D_b(P_i) > 0$ , we continue the search; otherwise, the found intersection point is refined with a binary search by checking the current position if it is inside the geometry with the same difference  $D_b(P_i)$ .

About chamfered extrusion mapping [15], we begin by performing a search of the intersection with the beveled extrusion. The found intersection point will be the retained point if its depth value is less than a value  $c$  that controls the chamfer effect; otherwise, we perform a second search using the following formula [15]:

$$P_{i+1} = P_i + \max(0, \tau EDT(P_i) - bc)v = P_i + D_c(P_i)v. \quad (7)$$



In the case of outward extrusion mapping [14,15], the term  $\tau EDT(P_i)$  is replaced by  $(\tau EDT(P_i) - e)$ , where  $e$  is a parameter that can be modified in real time, allowing us to modulate the extension effect (Figures 6b, 6c, and 6d).

In this article, the depths and the search for the intersection are bounded on interval  $[0, 1]$ ; so, the depth of the viewing ray is normalized  $(v/v_z)$ .

#### 4. Quadratic approximation

A quadratic approximation was used with the relief-mapping technique in a technical report [26]. This consists of calculating an approximate quadratic surface for each vertex of a 3D mesh in a pre-processing stage; in the rendering stage, this surface is used to adapt the ray-tracing process so that it takes the form of the mesh geometry into consideration.

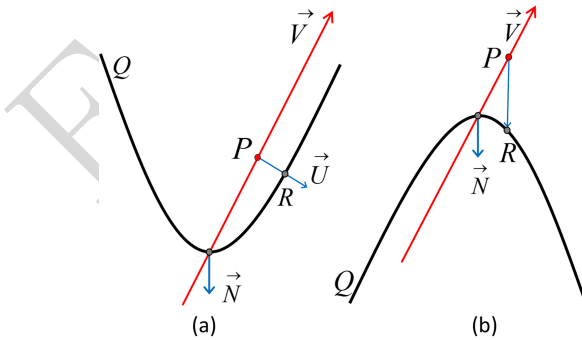
The approximate quadratic surface is represented by two parameters ( $a$  and  $b$ ) so that:

$$z = ax^2 + by^2, \quad (8)$$

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

These parameters are calculated in the tangent space that is associated with vertex  $(x, y, z)$  using the quadrics [26, 30].

During the rendering stage, coefficients  $a$  and  $b$  will be interpolated for each pixel and then used to calculate the distance  $d$  between the viewing ray and the quadratic surface. This distance is calculated in the tangent space that is associated with each vertex; so, we have two cases (as shown in Figure 7).



**Figure 7.** Cross section of two quadratic surfaces: in left surface (a), viewing ray is inside quadric; in right surface (b), viewing ray is outside. In both cases,  $PR$  segment gives distances between viewing ray and quadric  $Q$ .

Let  $R$  be a point that belongs to the quadric  $Q$ ,  $U$  be the unit vector that is perpendicular to  $V$  at the  $P$  point, and  $V$  be the viewing ray expressed in the tangent space.

In the first case (Figure 7a),  $V$  is inside the quadric, so  $R$  is obtained by translating the  $P$  by  $d$  units along  $U$ :

$$R = P + Ud. \tag{9}$$

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

$$aR_x^2 + bR_y^2 - R_z = 0 \iff a(P_x + U_x d)^2 + b(P_y + U_y d)^2 - (P_z + U_z d) = 0. \tag{10}$$

The solution of this equation gives the following:

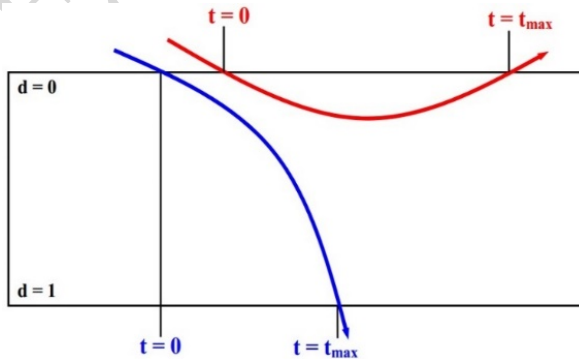
$$d = \frac{-B + \sqrt{\Delta}}{2A} = \frac{-B + \sqrt{B^2 - 4AC}}{2A}, \tag{11}$$

with  $\Delta > 0$  and  $\left\{ \begin{array}{l} A = aU_x^2 + bU_y^2 \\ B = 2aP_x U_x + 2bP_y U_y - U_z \\ C = aP_x^2 + bP_y^2 - P_z \end{array} \right.$

In the second case (Figure 7b),  $V$  is outside the quadric; in this case, the distance  $d$  is given by the following:

$$d = P_z - (aP_x^2 + bP_y^2). \tag{12}$$

During the linear search, the relief-mapping technique [32] chooses the  $t$  parameter in interval  $[0, 1]$ . This search is optimized in [26] by choosing  $t$  interval  $[0, t_{max}]$ , with  $t_{max}$  being the smallest  $t > 0$  such that the distance from the viewing ray to the quadric is equal to 0 or 1 (Figure 8).



**Figure 8.** Ray that hits depth 1 ( $d = 1$ ) in texture space has reached bottom of depth field characterizing an intersection (blue ray). On the other hand, ray that returns to depth 0 ( $d = 0$ ) can be safely discarded as belonging to silhouette (red ray).

To find the most accurate value,  $t_{max}$  must be calculated by substituting  $(P_x, P_y, P_z)$  with  $(V_x t, V_y t, V_z t)$  and setting  $d = 0$  and  $d = 1$ , respectively, in both Equations (10) and (12); then, solving for  $t$ . Algorithms 1 and 2 implement this optimization in both cases.

---

**Algorithm 1: tMaxInside**


---

```

input :  $V, U$ 
output:  $t_{max}$ 
1 begin
2    $A \leftarrow aV_x^2 + aV_y^2;$ 
3    $B \leftarrow 2aV_x U_x + 2bV_y U_y - V_z;$ 
4    $C \leftarrow aU_x^2 + bU_y^2 - U_z;$ 
5    $D \leftarrow B^2 - 4AC;$ 
6   if  $D > 0$  then
7      $t_{max} \leftarrow (B - \sqrt{D}) / -2A;$ 
8   end
9    $D \leftarrow V_z / A;$ 
10  if  $D > 0$  then
11     $t_{max} \leftarrow \min(t_{max}, D);$ 
12  end
13   $t_{max} \leftarrow |t_{max}|;$ 
14 end

```

---



---

**Algorithm 2: tMaxOutside**


---

```

input :  $V, q$ 
output:  $t_{max}$ 
1 begin
2    $D \leftarrow V_z^2 - 4q;$ 
3   if  $D > 0$  then
4      $t_{max} \leftarrow (-V_z + \sqrt{D}) / -2q;$ 
5   end
6    $D \leftarrow V_z / q;$ 
7   if  $D > 0$  then
8      $t_{max} \leftarrow \min(t_{max}, D);$ 
9   end
10   $t_{max} \leftarrow |t_{max}|;$ 
11 end

```

---

## 5. Per-pixel extrusion mapping with correct silhouette

The texture space is planar; in the rendering stage, the approximate surface that is calculated at each vertex of the 3D mesh during the pre-processing stage is used so that this space can be adapted to the 3D object geometry. In reality, the texture space always remains planar; during the search for the intersection, the viewing ray

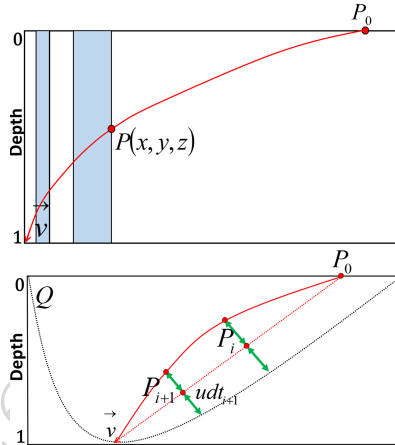
is rectified to correct the position of the  $P_{i+1}$  point by using the characteristics of the approximate surface.

In the first case (Figure 7a), the next  $P_{i+1}$  point is given by the following:

$$P_{i+1} = (v + ud)t_{i+1} = (v + w)t_{i+1}, \quad (13)$$

where  $u$  is the normalized  $U$  vector that is expressed in the texture space. We denote the quadratic vector as  $w$ .

Figure 9 shows the general appearance of the viewing ray during the ray-tracing phase. The figure shows that the  $P_{i+1}$  point approaches the quadratic surface at each iteration; that is to say, we move forward to depth value 1. In the case of an intersection, we quickly converge to the intersection point. Algorithms 3 and 4 implement this rectification in the cases of the basic and beveled extrusions, respectively.



**Figure 9.** Viewing ray is inside quadric. At each iteration, we approach quadratic surface by making rectification of viewing ray according to value  $udt_{i+1}$ .

---

**Algorithm 3:** InsideRayIntersect

---

```

input :  $P_0, v, w$ 
output:  $t$ 
1 begin
2    $t \leftarrow 0$ ;
3    $P \leftarrow P_0$ ;
4   for  $i \leftarrow 1$  to  $STEPS$  and  $P_z < 1$  do
5      $t \leftarrow t + D(P)$ ;
6      $P \leftarrow P_0 + (v + w)t$ ;
7   end
8 end

```

---

---

**Algorithm 4:** InsideBeveledRayIntersect

---

```

input :  $P_0, v, w$ 
output:  $t$ 
1 begin
2    $t \leftarrow 0;$ 
3    $P \leftarrow P_0;$ 
4   for  $i \leftarrow 1$  to  $STEPS$  and  $P_z < 1$  do
5     if  $D_b(P) > 0$  then
6        $t \leftarrow t + D(P);$ 
7        $P \leftarrow P_0 + (v + w)t;$ 
8     end
9   end
10 end

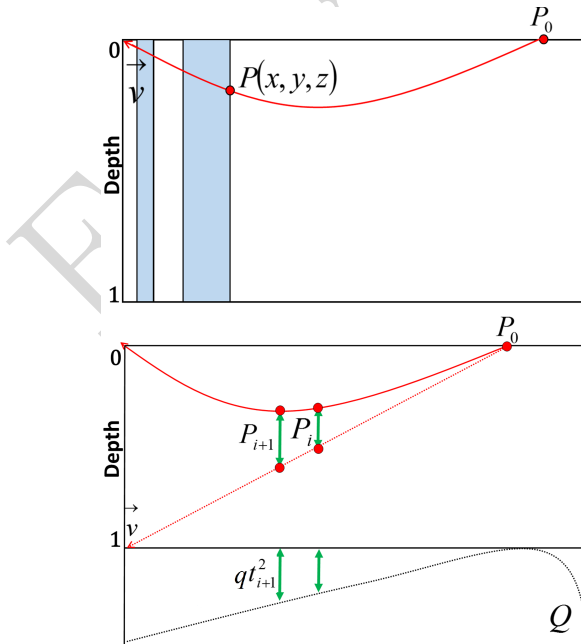
```

---

In the second case (Figure 7b), the next  $P_{i+1}$  point is given by the following:

$$P_{i+1} = (v_x t_{i+1}, v_y t_{i+1}, v_z t_{i+1} - q t_{i+1}^2), \tag{14}$$

where  $q$  is the quadric ( $av_x^2 + bv_y^2$ ) that is associated with parameters  $a$  and  $b$  that are expressed in the texture space.



**Figure 10.** Viewing ray is outside quadric. At each iteration, we move away from quadratic surface by making depth rectification of viewing ray with value  $-qt^2$ .

Figure 10 shows the general appearance of the viewing ray during the ray-tracing phase. The figure shows that the  $P_{i+1}$  point moves away from the quadratic surface at each iteration; that is to say, we move away from depth value 1. In the case where there is no intersection, we quickly converge to depth value 0. Algorithms 5 and 6 implement this rectification in the cases of the basic and beveled extrusions, respectively.

---

**Algorithm 5:** OutsideRayIntersect
 

---

```

input :  $P_0, v, q$ 
output:  $t$ 
1 begin
2    $t \leftarrow 0;$ 
3    $P \leftarrow P_0;$ 
4   for  $i \leftarrow 1$  to  $STEPS$  and  $P_z < 1$  do
5      $t \leftarrow t + D(P);$ 
6      $P \leftarrow P_0 + vt;$ 
7      $P_z \leftarrow P_z - t^2q;$ 
8   end
9 end

```

---



---

**Algorithm 6:** OutsideBeveledRayIntersect
 

---

```

input :  $P_0, v, q$ 
output:  $t$ 
1 begin
2    $t \leftarrow 0;$ 
3    $P \leftarrow P_0;$ 
4   for  $i \leftarrow 1$  to  $STEPS$  and  $P_z < 1$  do
5     if  $D_b(P) > 0$  then
6        $t \leftarrow t + D(P);$ 
7        $P \leftarrow P_0 + vt;$ 
8        $P_z \leftarrow P_z - t^2q;$ 
9     end
10  end
11 end

```

---

Since the depth of  $v$  is normalized ( $v/v_z$ ), distance  $d$  must be divided by  $v_z$  in the first case (Figure 7a) and quadric  $q$  must be divided by  $v_z^2$  in the second case. These must occur before normalizing the depth of  $v$ .

The decision regarding the silhouette fragments is realized after the search for the intersection. To optimize this search, we can check whether the value of the  $t$  parameter is greater than  $t_{max}$  at each iteration. In such a case, the search is interrupted.

In the case of the beveled extrusion mapping, the found intersection point is refined with a binary search (during the search, it is necessary to test whether the current position is inside the geometry by using Formula (6), and in the case of an extrusion with a chamfer, we continue the search after the binary search with Formula (7). Algorithms 7 and 8 give the appropriate implementations.

---

**Algorithm 7:** BinarySearch
 

---

```

input :  $P, b$ 
output:  $P$ 
1 begin
2    $distance \leftarrow D(P);$ 
3   for  $i \leftarrow 1$  to  $STEPS$  do
4      $distance \leftarrow distance/2;$ 
5     if  $D_b(P) > 0$  then
6        $P \leftarrow P + v \times distance;$ 
7     else
8        $P \leftarrow P - v \times distance;$ 
9     end
10  end
11 end

```

---



---

**Algorithm 8:** ChamferedIntersection
 

---

```

input :  $P, c$ 
output:  $P$ 
1 begin
2   if  $P_z > c$  then
3     for  $i \leftarrow 1$  to  $STEPS$  do
4        $P \leftarrow P + D_c(P)v$ 
5     end
6   end
7 end

```

---

In the rendering stage, the search for the intersection is performed in the texture space. However, the calculations that are 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 9 implements the per-pixel extrusion mapping with the correct silhouette in its different types.

**Algorithm 9:** CurvedExtrusionMapping

**input :**  $SM, R, T, (x_0, y_0), (S_x, S_y, S_z), (a, b), \tau, e, b, c, V, U,$

isBevel(if there is a beveled extrusion), isChamfer(if there is a chamfered extrusion)

**output:**  $P$

```

1 begin
2    $P_0 \leftarrow (x_0T_x, y_0T_y, 0);$ 
3    $v \leftarrow \|V/(S_x, S_y, S_z)\|;$ 
4    $v_z \leftarrow -v_z;$ 
5    $z \leftarrow v_z;$ 
6    $v \leftarrow v/v_z;$ 
7    $v_R \leftarrow v/\sqrt{v_x^2 + R^2v_y^2};$ 
8    $A \leftarrow aU_x^2 + bU_y^2;$ 
9    $B \leftarrow 2aV_xU_x + 2bV_yU_y - U_z;$ 
10   $C \leftarrow aV_x^2 + bV_y^2 - V_z;$ 
11   $D \leftarrow B^2 - 4AC;$ 
12  if  $D > 0$  then
13     $t_{max} \leftarrow tMaxInside(V, U);$ 
14     $u \leftarrow \|U/(S_x, S_y, S_z)\|;$ 
15     $d \leftarrow ((B - \sqrt{D}) / -2A) / z;$ 
16     $w \leftarrow du;$ 
17    if isBevel or isChamfer then
18       $t \leftarrow InsideBeveledRayIntersect(P_0, v_R, w)$ 
19    else
20       $t \leftarrow InsideRayIntersect(P_0, v_R, w)$ 
21    end
22  else
23     $q \leftarrow aV_x^2 + bV_y^2;$ 
24     $t_{max} \leftarrow tMaxOutside(V, q);$ 
25     $q \leftarrow (q/S_z)/(z^2);$ 
26    if isBevel or isChamfer then
27       $t \leftarrow OutsideBeveledRayIntersect(P_0, v_R, q)$ 
28    else
29       $t \leftarrow OutsideRayIntersect(P_0, v_R, q)$ 
30    end
31  end
32  if  $t > t_{max}$  then
33     $discard;$ 
34  else
35     $P \leftarrow P_0 + v_Rt;$ 
36  end
37  if isBevel or isChamfer then
38     $P \leftarrow BinarySearch(P, b);$ 
39    if isChamfer then
40       $P \leftarrow ChamferedIntersection(P, c);$ 
41    end
42  end
43 end

```



We note the following:

$P_0$ : Starting point;

$P$ : Intersection point;

$(x_0, y_0)$ : Current pixel;

$T$ : Repetition along  $x$  and  $y$ ;

$(S_x, S_y, S_z)$ : Texture space;

$SM$ : Shape Map;

$R$ : Width/Height of Shape Map  $SM$ ;

$V$ : Viewing ray expressed in tangent space;

$v$ : Viewing ray expressed in texture space;

$U$ : Vector perpendicular to  $V$ ;

$u$ :  $U$  vector expressed in texture space;

$w$ : Quadratic vector expressed in texture space;

$(a, b)$ : Quadratic parameters;

$d$ : Quadratic distance;

$q$ : Quadric;

$\tau$ : Scale factor;

$e$ : Extension parameter;

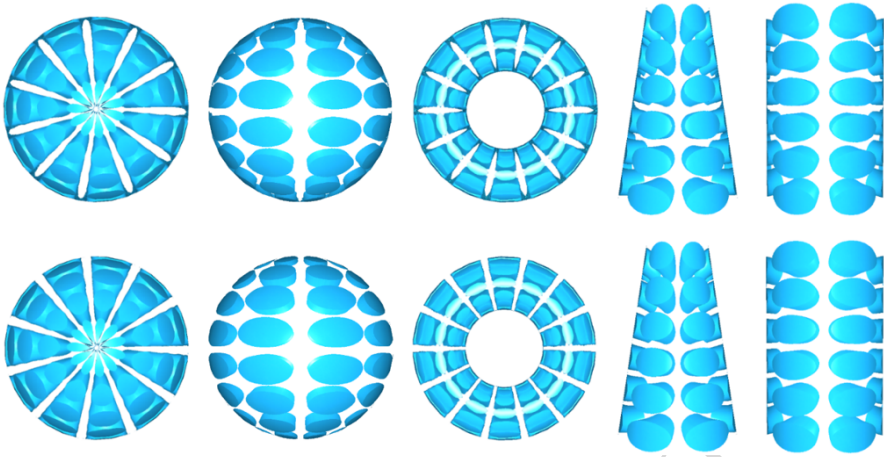
$b$ : Bevel parameter;

$c$ : Chamfer parameter.

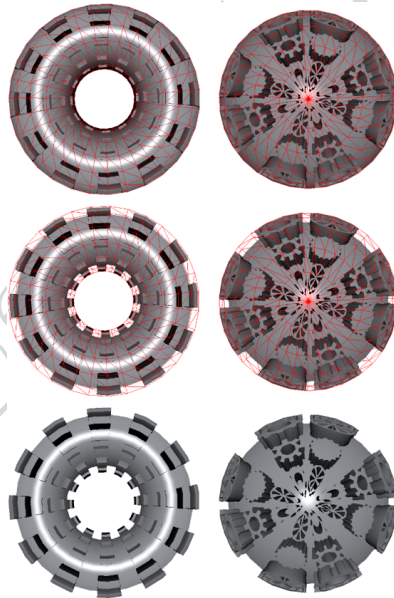
## 6. Results and discussions

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 were obtained using a Core-i7-4510U-2GH-4CPUs architecture with 8 GB of RAM and GeForce-GT-840M with 4 GB of memory.

The curved ray-tracing algorithm does not concern the geometry of the 3D object at the silhouette parts. Its main purpose is to eliminate the non-visible parts of the silhouette. Indeed, Figures 11 and 12 show the difference between those objects that are rendered with the extrusion-mapping technique without and with the correct silhouette using different depth scales. We notice that the rendering differences are visible at the edges of the 3D objects. In the images that are rendered by the curved ray-tracing algorithm, the silhouette is visible on the parts where the viewing ray shaves the 3D surface.

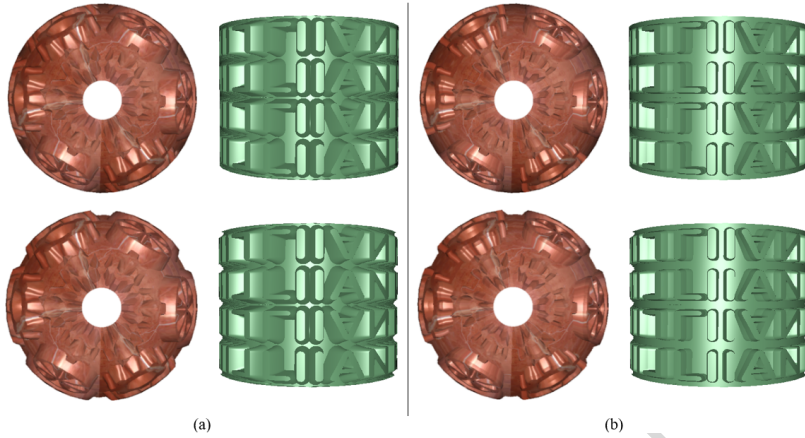


**Figure 11.** Comparison of renderings of 3D objects without and with silhouette correction: top – basic extrusion mapping; bottom – same objects rendered with correct silhouette by using curved ray-tracing algorithm.



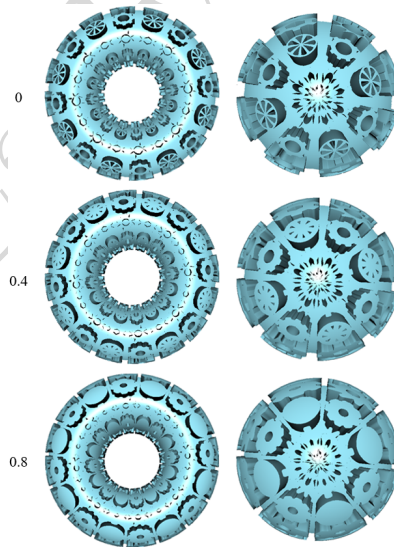
**Figure 12.** Comparison of renderings of extrusion-mapping technique without and with correct silhouette and by highlighting polygonal mesh. We note that silhouette is clearly visible on edges of 3D objects.

The proposed rectification works equally well in the cases of beveled and chamfered extrusions. Figure 13 shows that the bevel and chamfer effects are not affected by this correction; only the fragments belonging to the silhouette are concerned.



**Figure 13.** Beveled (a) and chamfered (b) extrusion mapping without correction and with correction of silhouette. Effects of bevel and chamfer are not affected by rectification process; only fragments belonging to silhouette are concerned.

The same also applies in the case of outward extrusion. Figure 14 shows the extension effect on the 3D objects by gradually increasing the value of the extension. Outward extrusion can be combined with any type of extrusion while preserving the same silhouette-rendering quality of the curved ray-tracing algorithm.

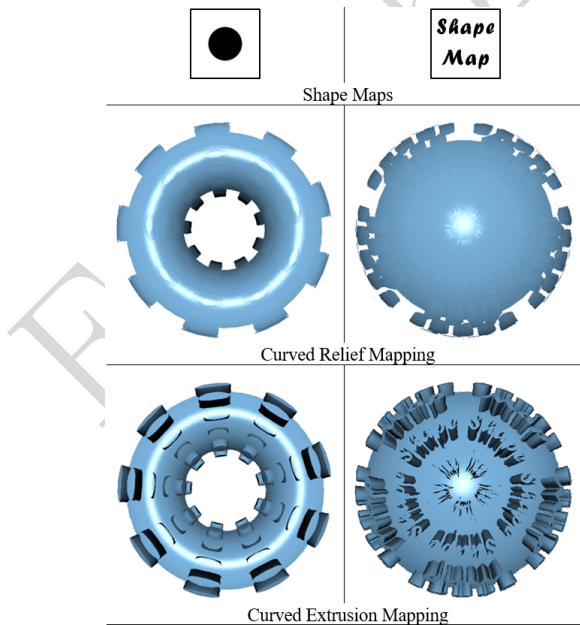


**Figure 14.** Tor and sphere rendered using outward extrusion mapping with silhouette correction. Extension parameter increases from top to bottom (0, 0.4, and 0.8). When value extension parameter is zero, we find basic extrusion mapping. Extension effect is not affected by rectification process; only fragments belonging to silhouette are concerned.

The proposed rectification does not depend on the shape map; it depends only on the quadratic parameters that are associated with the 3D surface. In addition, the shape map is not attached to the base geometry onto which it is mapped, as 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 (Figures 11 and 14). This means that the rectification process and the shape map are independent of the surface on which they will be used.

In addition, a change in the camera position does not influence the rectification process nor the rendering quality, as the rectification is realized in real time and we will have a new image rendered with a new rectification with each movement of the camera.

To compare the rendering quality between the curved relief mapping [26] and the curved extrusion mapping, we used two 3D objects (a sphere and a torus) with two different shape maps and with the same number of repetitions along  $x$  and  $y$  (Figure 15). Figure 15 shows that the curved relief mapping is not well-adapted for the case of extrusion. On the other hand, the curved extrusion mapping allows us to produce 3D objects with a satisfactory quality and with the correct support of the silhouette.



**Figure 15.** Comparison between curved relief mapping and curved extrusion mapping. Images rendered with good quality are those rendered by curved extrusion mapping.

To compare the rendering speeds between the different techniques discussed in this paper, we used a scene of  $800 \times 600$ , a torus with 2,028 triangles, and three



different shape maps that have a resolution of  $1024 \times 1024$ . Concerning the iteration numbers of the ray-tracing algorithm, we used 25 linear steps and 5 binary steps. We used 0.08 for the bevel parameter and 0.6 for the chamfer parameter.

Table 1 shows the difference in the rendering speeds; it also shows the shape map used in each case (with its respective repetitions along  $x$  and  $y$ ) and the views on which the calculations were made where the 3D objects occupy each entire scene.

From Table 1, we can observe that the different extrusion-mapping techniques are faster when compared to the curved relief mapping. In the case of the curved extrusion mapping (the case that interests us), we have considerable means differences of 42, 26, and 18 FPS for the basic, beveled, and chamfered extrusions, respectively.

**Table 1**

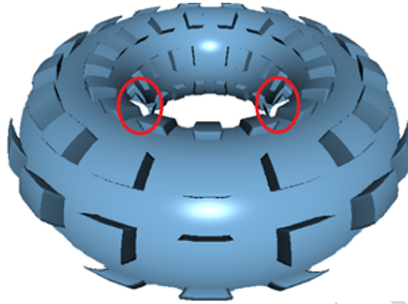
Comparison of rendering speed FPS (frames per second) among different techniques discussed in this paper. We found that curved relief mapping is slowest. We also found that silhouette treatment degraded rendering speeds of curved extrusion-mapping techniques when compared to extrusion-mapping techniques. This degradation is in order of 8 FPS.

Screen $800 \times 600$ Shape Map with $(T_x, T_y)$	Extrusion Mapping			Curved Extrusion Mapping			Curved Relief Mapping
	Basic	Beveled	Chamfered	Basic	Beveled	Chamfered	
<b>Shape Map</b> (6 6)	200	184	176	192	176	168	136
 (10,10)	160	144	136	152	136	128	128
 (12,10)	152	136	128	144	128	120	96

We also note that the rendering speeds of the curved extrusion-mapping techniques have been decreased when compared to the extrusion-mapping techniques by a means of 8 FPS. The degradation is minimal; this is due to the calculations that are related to the treatment of the silhouette that is based on the quadratic approximation.

Outward extrusion can be combined with any other extrusion-mapping technique without affecting the ray-tracing process. Figure 6 shows that we added the extension effect in the other three types of extrusion (basic, beveled, and chamfered). This is the same for the silhouette correction.

The combination of the per-pixel extrusion-mapping techniques and the quadratic approximation produce very satisfactory results at a low cost. In some cases, this combination produces distortions and holes (as shown in Figure 16). This problem is due to the use of the quadratic approximation for the local representation of the surface at each vertex, as the viewing ray occasionally pierces the relief in the object space and leaves it in the texture space. This problem was also mentioned in [7,17,22].



**Figure 16.** Distortions and holes due to use of quadratic approximation.

The proposed combination does not affect the complexity of the ray-tracing algorithm. Indeed, the instructions that are related to the quadratic approximation have a linear complexity of  $O(n)$ ; so, the complexity of the resulting algorithm remains always linear.

## 7. Conclusion and perspectives

In this paper, we have presented a curved ray-tracing algorithm for the basic, outward, beveled, and chamfered extrusion-mapping techniques by combining a ray-tracing algorithm with a quadratic approximation. This approximation consists of representing a 3D surface by approximate parameters at each vertex that constitutes a corresponding mesh.

The proposed rectification consists of adapting the displacements along the viewing ray so that it takes the forms of the quadratic surfaces into account. During the curved ray-tracing phase, the algorithm uses the parameters of the quadratic surface in order to rectify the viewing ray. This rectification makes it possible to know whether the viewing ray pierces or leaves the extruded form and if it is realized after each new displacement along the viewing ray.

The proposed rectification allows us to produce images at a very high speed and with a satisfactory quality. However, due to the use of the quadratic approximation, the viewing ray sometimes pierces the relief in the object space and leaves it in the texture space. This leads to the appearance of holes and distortions.

Possible improvements can be made regarding the complexity of the algorithm by providing an optimization and the way the surface curvature is represented at each vertex in order to increase the rendering quality.

## References

- [1] Baboud L., Décoret X.: Rendering Geometry with Relief Textures. In: *Proceedings of Graphics Interface 2006*, p. 195–201, GI '06, Canadian Information Processing Society, CAN, 2006.
- [2] Baboud L., Eisemann E., Seidel H.: Precomputed Safety Shapes for Efficient and Accurate Height-Field Rendering, *IEEE Transactions on Visualization and Computer Graphics*, vol. 18(11), pp. 1811–1823, 2012. doi: 10.1109/TVCG.2011.281.
- [3] Blinn J.F.: Simulation of Wrinkled Surfaces, *SIGGRAPH Comput Graph*, vol. 12(3), pp. 286—292, 1978. doi: 10.1145/965139.507101.
- [4] Brawley Z., Tatarchuk N.: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing, *ShaderX3: Advanced Rendering with DirectX and OpenGL (ShaderX Series)*, pp. 135–154, 2004.
- [5] Chahdi A.O., Halli A., Ragragui A., Satori K.: Per-pixel displacement mapping using hybrid cone approach. In: *2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, pp. 1–4, 2017. doi: 10.1109/ATSIP.2017.8075577.
- [6] Chahdi A.O., Ragragui A., Halli A., Satori K.: Dynamic relief mapping1. In: *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, pp. 1–6, 2018. doi: 10.1109/ISACV.2018.8354053.
- [7] Chen Y.C., Chang C.F.: A Prism-Free Method for Silhouette Rendering in Inverse Displacement Mapping, *Computer Graphics Forum*, 2008. doi: 10.1111/j.1467-8659.2008.01341.x.
- [8] Cook R.L.: Shade Trees. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, p. 223–231, SIGGRAPH '84, Association for Computing Machinery, New York, NY, USA, 1984. doi: 10.1145/800031.808602.
- [9] Danielsson P.E.: Euclidean distance mapping, *Computer Graphics and Image Processing*, vol. 14(3), pp. 227 – 248, 1980. doi: 10.1016/0146-664X(80)90054-4.
- [10] Donnelly W.: Per-Pixel Displacement Mapping with Distance Functions. In: *GPU Gems 2*, pp. 123–136, Addison-Wesley, 2005.
- [11] Dufort J.F., Leblanc L., Poulin P.: Interactive Rendering of Meso-structure Surface Details using Semi-transparent 3D Textures. In: *Proceedings of Vision, Modeling, and Visualization 2005*, pp. 399–406, 2005.
- [12] Eric R., Musawir S., Sumanta P.: Interval Mapping. In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, I3D '06*, Association for Computing Machinery, 2006.

- [13] Halli A., Saaidi A., Satori K., Tairi H.: Per-Pixel Displacement Mapping Using Cone Tracing, *International Review on Computers and Software*, vol. 3(5), 2008.
- [14] Halli A., Saaidi A., Satori K., Tairi H.: Per-Pixel Extrusion Mapping, *IJC-SNS International Journal of Computer Science and Network Security*, vol. 9(3), pp. 118–124, 2009.
- [15] Halli A., Saaidi A., Satori K., Tairi H.: Extrusion and Revolution Mapping, *ACM Trans Graph*, vol. 29(5), 2010. doi: 10.1145/1857907.1857908.
- [16] Hirche J., Ehler A., Guthe S., Doggett M.: Hardware Accelerated Per-Pixel Displacement Mapping. In: *Proceedings of Graphics Interface 2004*, pp. 153–158, GI '04, Canadian Human-Computer Communications Society, Waterloo, CAN, 2004.
- [17] Jeschke S., Mantler S., Wimmer M.: Interactive Smooth and Curved Shell Mapping. In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, p. 351–360, EGSR'07, Eurographics Association, Goslar, DEU, 2007.
- [18] Kaneko T., Takahei T., Inami M., Kawakami N., Yanagida Y., Maeda T., Tachi S.: Detailed Shape Representation With Parallax Mapping. In: *Proceedings of the ICAT 2001*, pp. 205–208, 2001.
- [19] Ki H., Oh K.: *Accurate Per-Pixel Displacement Mapping using a Pyramidal Structure*, Tech. rep., 2007. <http://ki-h.com/archive/KiH-TA07-IPDM.pdf>.
- [20] Kolb A., Rezk-Salama C.: Efficient Empty Space Skipping for Per-Pixel Displacement Mapping. In: *Proceedings of Vision, Modeling and Visualization (VMV 05)*, pp. 407–414, 2005.
- [21] McGuire M., McGuire M.: Steep Parallax Mapping, *I3D 2005 Poster; Brown Web Report*, 2005. <https://casual-effects.com/research/McGuire2005Parallax/index.html>. I3D 2005 Poster.
- [22] Na K.G., Jung M.R.: Curved Ray-Casting for Displacement Mapping in the GPU. In: *Proceedings of the 14th International Conference on Advances in Multimedia Modeling*, p. 348–357, MMM'08, Springer-Verlag, Berlin, Heidelberg, 2008.
- [23] Oh K., Ki H., Lee C.H.: 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*, p. 75–82, VRST '06, Association for Computing Machinery, New York, NY, USA, 2006. doi: 10.1145/1180495.1180511.
- [24] Oliveira M.M.: *Relief Texture Mapping*, Ph.D. thesis, University of North Carolina, 2000.
- [25] Oliveira M.M., Bishop G., McAllister D.: Relief Texture Mapping. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, p. 359–368, SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., USA, 2000. doi: 10.1145/344779.344947.
- [26] Oliveira M.M., Policarpo F.: *An Efficient Representation for Surface Details*, Tech. rep., Instituto de Informática UFRGS, 2005.



- [27] Paglieroni D.W., Petersen S.M.: Height Distributional Distance Transform Methods for Height Field Ray Tracing, *ACM Trans Graph*, vol. 13(4), pp. 376—399, 1994. doi: 10.1145/195826.197312.
- [28] Patterson J., Hoggar S., Logie J.: Inverse Displacement Mapping, *Computer Graphics Forum*, 1991. doi: 10.1111/1467-8659.1020129.
- [29] Peercy M., Airey J., Cabral B.: Efficient Bump Mapping Hardware. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 303—306, SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., USA, 1997. doi: 10.1145/258734.258873.
- [30] Petitjean S.: A Survey of Methods for Recovering Quadrics in Triangle Meshes, *ACM Comput Surv*, vol. 34(2), pp. 211—262, 2002. doi: 10.1145/508352.508354.
- [31] Policarpo F., Oliveira M.M.: Relief Mapping of Non-Height-Field Surface Details. In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, p. 55–62, I3D '06, Association for Computing Machinery, New York, NY, USA, 2006. doi: 10.1145/1111411.1111422.
- [32] Policarpo F., Oliveira M.M., Comba J.a.L.D.: Real-Time Relief Mapping on Arbitrary Polygonal Surfaces. In: *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, p. 155–162, I3D '05, Association for Computing Machinery, New York, NY, USA, 2005. doi: 10.1145/1053427.1053453.
- [33] Porumbescu S.D., Budge B., Feng L., Joy K.I.: Shell Maps, *ACM Transactions on Graphics*, vol. 24(3), pp. 626–633, 2005.
- [34] Premecz M.: Iterative Parallax Mapping with Slope Information, *Central European Seminar on Computer Graphics (CESCG 06)*, 2006. [www.cescg.org/CESCG-2006/papers/TUBudapest-Premecz-Matyas.pdf](http://www.cescg.org/CESCG-2006/papers/TUBudapest-Premecz-Matyas.pdf).
- [35] Ragragui A., Chahdi A.O., Halli A., Satori K.: 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)*, pp. 1–6, 2017. doi: 10.1109/ISACV.2017.8054957.
- [36] Ragragui A., Chahdi A.O., Halli A., Satori K.: Per-pixel revolution mapping with rectification of the texture projection. In: *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, pp. 1–6, 2018. doi: 10.1109/ISACV.2018.8354056.
- [37] Ragragui A., Ouazzani Chahdi A., Halli A., Satori K.: Revolution mapping with bump mapping support, *Graphical Models*, vol. 100, pp. 1–11, 2018. doi: 10.1016/j.gmod.2018.09.001.
- [38] Ragragui A., Ouazzani Chahdi A., Halli A., Satori K.: Image-based extrusion with realistic surface wrinkles, *Journal of Computational Design and Engineering*, vol. 7(1), pp. 30–43, 2020.

- [39] Ritsche N.: 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*, p. 265–274, GRAPHITE '06, Association for Computing Machinery, New York, NY, USA, 2006. doi: 10.1145/1174429.1174477.
- [40] Shirley P., Tuchman A.: A Polygonal Approximation to Direct Scalar Volume Rendering, *SIGGRAPH Comput Graph*, vol. 24(5), pp. 63—70, 1990. doi: 10.1145/99308.99322.
- [41] Tatarchuk N.: Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows. In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, p. 63–69, I3D '06, Association for Computing Machinery, New York, NY, USA, 2006. doi: 10.1145/1111411.1111423.
- [42] Tevs A., Ihrke I., Seidel H.P.: Maximum Mipmaps for Fast, Accurate, and Scalable Dynamic Height Field Rendering, *Proc of Interactive 3D Graphics and Games, I3D08*, pp. 183–190, 2008.
- [43] Wang L., Wang X., Tong X., Lin S., Hu S., Guo B., Shum H.Y.: View-Dependent Displacement Mapping, *ACM Trans Graph*, vol. 22(3), p. 334–339, 2003. doi: 10.1145/882262.882272.
- [44] Wang X., Tong X., Lin S., Hu S., Guo B., Shum H.Y.: Generalized Displacement Maps. In: *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques*, p. 227–233, EGSR'04, Eurographics Association, Goslar, DEU, 2004.
- [45] Welsh T.: *Parallax Mapping with Offset Limiting*, Tech. rep., Infiscape Corporation, 2004.
- [46] Yerex K., Jagersand M.: Displacement Mapping with Ray-Casting in Hardware. In: *ACM SIGGRAPH 2004 Sketches*, p. 149, SIGGRAPH '04, Association for Computing Machinery, New York, NY, USA, 2004. doi: 10.1145/1186223.1186410.

## Affiliations

### Adnane Ouazzani Chahdi

Sidi Mohamed Ben Abdellah University, Faculty of Science Dhar EL Mahraz, LISAC Laboratory, Fez, Morocco, adnaneouazzanichahdi@gmail.com

### Anouar Ragraoui

Sidi Mohamed Ben Abdellah University, Faculty of Science Dhar EL Mahraz, LISAC Laboratory, Fez, Morocco, anouar.ragraoui@usmba.ac.ma

### Akram Halli

Moulay-Ismaïl University IA Laboratory Meknes, Morocco, akramhalli@yahoo.fr

### Khalid Satori

Sidi Mohamed Ben Abdellah University, Faculty of Science Dhar EL Mahraz, LISAC Laboratory, Fez, Morocco, khalidsatori@gmail.com

**Received:** ???

**Revised:** ???

**Accepted:** ???