Per-Pixel Extrusion Mapping

Akram Halli, Abderrahim Saaidi, Khalid Satori and Hamid Tairi
LIIAN Laboratory, Faculty of Science Dhar El Mahraz, Fez, Morocco

Summary
Extruded shapes and patterns are widely used by graphic designers to increase the visual richness and realism of 3D models and virtual environments in general. However, the traditional approach that uses polygonal mesh is inappropriate for real-time rendering, in which, just a limited number of graphic primitives can be processed. Hence the interest of using per-pixel approaches. In this paper, we introduce a new image-based technique for rendering extruded details and shapes. We use a single RGBA texture in which we store a binary shape, its Euclidean Distance Transform (EDT), and the two components of the EDT gradient. The rendering algorithm is based on a ray-tracing like procedure, performed in texture space. The use of the EDT allows skipping empty space and thus, minimizes the number of ray-tracing steps. Per-pixel extrusion mapping produces very convincing results, and runs at interactive frame rates.

Key words:
Real-Time Rendering, Image-Based Modeling and Rendering, Ray-Tracing in GPU, Mesostructures, Extrusion, Euclidean Distance Transform.

1. Introduction

Rendering complex scenes with very detailed surfaces remains one of the major problems for real-time rendering. The traditional method consists in using high definition meshes, having a very large number of vertices and triangles. Therefore, a complex scene may include a million of such primitives, making its processing impossible for an interactive rendering. To overcome this problem, graphics designers create less detailed scenes having far fewer objects. However, even with techniques like texture mapping and bump mapping used to enhance the realism of such scenes, most surfaces look flat with polygonized silhouettes.

The architecture of the new generation of graphics cards, which includes a programmable pipeline, allowed introducing an alternative to polygon-based rendering. Indeed, Image-Based Modeling and Rendering (IBMR) uses images to store geometry data, which will be recreated with a ray-tracing like algorithm, running in parallel on the vertex and pixel shader units. Thus, IBMR avoids processing a large number of graphics primitives. In addition, IBMR is performed in image space, so only visible pixels are processed; unlike polygon based rendering that uses algorithms such as visibility, culling, or level of details, which consume significant resources themselves.

Several IBMR techniques were introduced for adding micro-details to simplified meshes [1]. These techniques use 2D grayscale images called heightfield maps, and in some cases, 3D maps. However, most of these techniques are inappropriate for rendering extruded details, whereas others are not yet adapted to an interactive rendering, such as algorithms based on shell space or 3D maps, which need considerable calculation and memory resources.

In this paper, we introduce a new technique for rendering extruded shapes and details. We use a single RGBA texture (the shape map) to store all the necessary data. First, we encode the shape as a 2D binary image in the alpha channel. Then, we compute its Euclidean Distance Transform (EDT) which is stored in the blue channel. Finally, in the red and green channels, we store the unit gradient components of the EDT. The resulting texture is then mapped on a polygonal mesh using 2D texture coordinates.

At the rendering stage, we use a per-pixel ray-tracing procedure to find the intersection point between the viewing ray and the surface generated by the extrusion of the binary shape. The EDT is used for space leaping, which ensures a fast convergence towards the intersection point. Then, the surface normal at the intersection point is calculated in term of the unit gradient components.

Per-pixel extrusion mapping is rendered interactively on current graphics cards. It is the fastest technique for rendering extruded shapes and patterns. It also gives the best results, and does not suffer from depth limitation or grazing angle artifacts.

2. Related Work

The idea of using textures to simulate mesostructures was first introduced by Blinn [2]. Bump mapping simulates the surface details using a normal perturbation. The texture coordinates and the geometry remain unchanged. This technique is very fast but can't produce self-occlusions, shadows and silhouettes.

Manuscript received March 5, 2009
Manuscript revised March 20, 2009

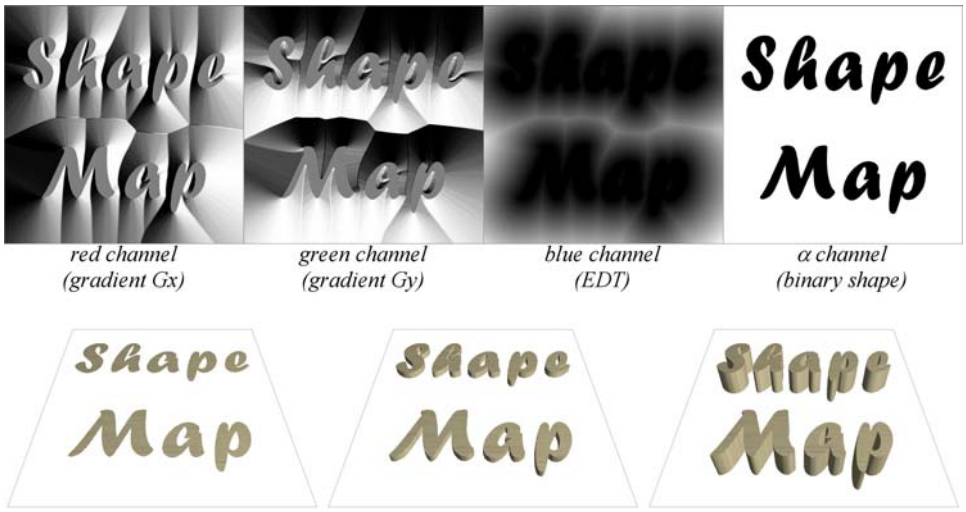

Fig. 1. The shape map represented as an RGBA texture. *Top*: The binary shape is stored in the alpha channel, its Euclidean distance transform is stored in the blue channel, while the red and green channels are used to encode the components of the EDT gradient. *Bottom*: Per-pixel extrusion rendering of a plane mapped with the shape map shown in top using different depths.

Parallax mapping [3] optionally with offset limiting [4] or with slope information [5] uses textures augmented with per-texel depth. In this approach the texture coordinates along the view ray direction are shifted according to the depth values. This technique produces nice results at a very low cost but it is only appropriate for irregular bumps.

Displacement mapping [6] subdivides original geometry into a large number of micro-polygons which are displaced perpendicularly according to a 2D height map. This technique produces correct self-occlusions, shadows and silhouettes. Unfortunately, displacement mapping is unsuitable for real time rendering due to the huge number of micro polygons to be rendered. To overcome this problem, Patterson et Al [7] introduced the idea of using per-Pixel displacement mapping.

Relief mapping [8], performs an image space search to find the intersection point between the viewing ray and a 2D depth map. It begins with a linear search using regular intervals followed by a binary search to refine the intersection point. This technique correctly handles self-occlusions, shadows and inter-penetrations. However some artefacts become visible at grazing angles, especially with thin structures. Similar approaches to relief mapping were presented in [9], [10], [11] and [12].

Instead of using unsafe approach, Donnelly [13] uses sphere tracing. He creates a 3D distance map that gives a measure of the distance between points in space and the displaced surface. In the same way, Dummer [14] uses top-opened cones as empty space bounding volumes stored in a 2D cone step map. This idea was first introduced by Paglieroni and Petersen [15] for height field ray tracing. Other conservative approaches were proposed like the use of pyramidal structure [16], [17].

Baboud et Al [18] introduced the use of relaxed pre-computed volumes allowing safe binary search. They used a safety radius allowing cylinder tracing. In the same way, Policarpo and Oliveira [19] relaxed the rule of the conservative cones. Techniques based on cone tracing have been improved in [20], particularly the speed of the preprocessing algorithms.

To resolve the problem of the flat silhouettes, Oliveira and Policarpo use a quadratic function to approximate curved surfaces [21]. Hirche et al [22] extrude the triangles of the base mesh along their respective normal directions, and then the resulting prisms are rendered by casting rays inside and intersecting them with the displaced surface. The prisms are subdivided into three tetrahedrons if the main mesh is not planar.

View-Dependent Displacement Mapping [23] and Generalized Displacement Mapping [24] do a pre-process shell space ray tracing and store the result as a five-dimensional function that can be queried during rendering time. These methods produce very nice results but require significant pre-processing and storage to operate.

3. Preprocessing Extrusion Maps

The extrusion map is an RGBA texture, which stores the data used by per-pixel extrusion algorithm (see Figure 1). The alpha channel encodes a profile shape as a binary image or, more precisely, a grey scale image in which only pixels above a certain threshold are considered part of the

Fig. 2. Calculation of the unit gradient. *Left*: Underline of the EDT level curves. *Center*: Unit gradient of the binary shape (red and green channels). *Right*: Unit gradient of the EDT.

shape. The blue channel stores the Euclidean distance transform of the shape channel. As for the red and green channels, they are used to store the two components of the normalized gradient of the EDT channel.

3.1 The Euclidean Distance Transform

In order to calculate the intersection point between the viewing ray and the extruded geometry, we use the Euclidean distance transform applied to the binary shape channel. Unlike some arbitrary relief rendering techniques described in [13] and [25], which use 3D textures to store the distance transform and thus, require a significant amount of pre-processing and storage, per-pixel extrusion mapping use only 2D textures, and the distance transform requires just one channel.

The EDT is defined as follow: From an n by m binary image B made from an object O and its background O', an Euclidean distance transformation makes an n by m output image D, in which the value of any pixel p is the Euclidean distance from this pixel to the object O, i.e. the distance to the nearest pixel of O:

$$D(p) = \min\left\{\|p - q\|,\ \mathrm{q} \in \mathrm{O}\right\}$$

The use of the EDT for space leaping during ray-tracing, instead of intervals with fixed sampling, allows faster convergence, and avoids missing intersections at grazing angles. The EDT computation is based on the 4SED algorithm described in [26]. This linear time algorithm has the advantage of giving a satisfactory and fast approximation of the distance transform.

3.2 Unit Gradient

The shading stage, which comes after the intersection search, needs the normal to the geometry at the intersection point. Usually, we should calculate the normal in term of the gradient of the binary shape. However, in some cases, the intersection point will not be reached exactly, so it will not be close to the shape outline where the gradient is non-zero. This problem will also arise for outward extrusion, in which, the shape outline will never be reached. To avoid this problem, we calculate the unit gradient of the distance transform instead of the binary shape. Because the level curves of the EDT represent isocontours of the shape outline on all parts of the texture (see Figure 2). The gradient can be calculated in parallel with the EDT, since the calculation of the latter implicitly relies on the gradient. However, we prefer the use of a gradient filter in order to reduce aliasing artifacts.

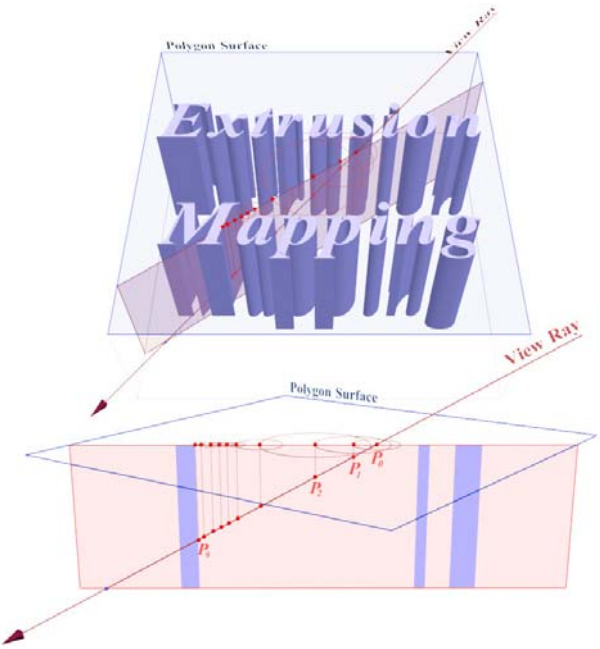

Fig. 3. View ray intersection with the extruded geometry generated from the extrusion map. The use of the minimal distance (circle radius) as ray tracing steps allows skipping safely empty space, and thus converging quickly towards the intersection point. The bottom image underlines the slice including the view ray.

4. Rendering Per-Pixel Extrusion

Per-pixel extrusion mapping consists of creating a virtual geometry under the polygons of a given mesh. This geometry represents the extruded surface, obtained from the shape map mapped over the polygons using the traditional 2D texture coordinates. For each fragment, we have to calculate the intersection point between the extrusion geometry and the viewing ray. This procedure is performed in the texture space (see Figure 3).

The transformation of the view-ray and light vectors, from modeling space to texture space, is performed per-vertex. Then, these vectors are interpolated across each fragment. The depth of the extrusion is set by the user and can vary in real-time.

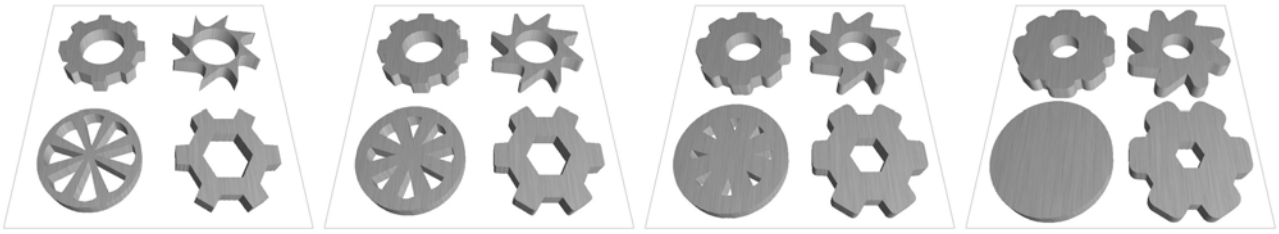

Fig. 4. Changing the value of the outward parameter in real-time allows smoothing the extruded shapes.

4.1 Intersection Point

Let (u,v) be the texture coordinates of the current fragment, and p_0 the search starting point with coordinates $(x_0,y_0,z_0)=(u,v,0)$. And let V the viewing direction obtained as the normalized vector from the viewpoint to the 3D p_0's position. We control the extrusion depth by dividing the z-component of the view ray by a scale parameter (i.e. $V_z \leftarrow V_z/\text{DEPTH\_SCALE}$). In order to converge towards the first intersection point p_j, we use the distance fields to make safe steps along the view ray (see Figure 3).

At each step, the distance d_i, between the current position p_i and the extruded shape, is recovered from the blue channel of the shape map at texture coordinates (x_i,y_i). Then, the next view ray position p_{i+1} is easily obtained with the following formula:

$$p_{i+1} = p_i + \frac{d_i}{\left\| V_{xy} \right\|} V \tag{1}$$

If the shape map is not square, we use the elliptical rectification described in [20]. Thus, we have to multiply the distance d_i by the following correction factor:

$$\frac{\left\| V_{xy} \right\|}{\sqrt{V_x^2 - \rho^2 V_y^2}}$$

with $\rho=height/width$. The equation (1) becomes:

$$p_{i+1} = p_i \frac{d_i}{\sqrt{V_x^2 + \rho^2 V_y^2}} \cdot V$$

Since the rectification depends only on the viewing ray V, we can use the following optimization:

$$p_{i+1} = p_i + d_i \cdot V_\rho \tag{2}$$

with:

$$V_\rho = \frac{1}{\sqrt{V_x^2 + \rho^2 V_y^2}} \cdot V$$

After the last step p_j, the current fragment will be discarded (i.e. no intersection found) if $z_j>1$.

4.2 Normal at the Intersection Point.

In order to compute the final output color, we have to calculate the normal to the extruded surface at the intersection point $p_j=(x_j,y_j,z_j)$. First, we retrieve the components of the unit gradient (Gx_j,Gy_j) from the extrusion map at position (x_j,y_j). Then, we remap Gx_j and Gy_j to a range of [-1,1]. Since the normal is perpendicular to the extrusion geometry, its z component is equal to zero. Thus, the normal is defined by:

$$N_j = \left(Gx_j, Gy_j, 0\right)$$

except at the polygon surface (i.e. $p_j = p_0$), where the polygon normal will be used instead.

4.3 Outward Extrusion Mapping

As the level curves of the Euclidean distance transform represent extended shapes of the original one, we can use them to create an outward extrusion. To this end, we replace the distance d_i in the formula (2) by the distance to the outward shape:

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

where e is the parameter controlling the extension width in real-time.

The outward extrusion mapping algorithm is slightly slower, but it could be very useful for smoothing the main shape as illustrated in Figure 4.

Table 1 - Preprocessing times (in seconds) of different shape maps.

Extrusion Map		*EDT*	*Gradient*	*Total*
Text	512 x 512	0.094	0.093	**0.187**
Shapes	512 x 512	0.094	0.109	**0.203**
Text	1024 x 1024	0.390	0.438	**0.828**
Shapes	1024 x 1024	0.359	0.454	**0.813**

Table 2 - Rendering speeds of per-pixel extrusion mapping and some per-pixel displacement mapping techniques in FPS (Frames Per Second). Note that the extrusion mapping and the outward extrusion mapping are clearly faster.

	Screen 800 x 600		*Screen 1024 x 768*	
	512^2	*1024^2*	*512^2*	*1024^2*
Extrusion Mapping	**105**	**95**	**98**	**74**
Outward Extrusion Mapping	**97**	**65**	**80**	**61**
Relief Mapping	78	38	66	37
Parallax Occlusion mapping	69	35	56	35
Cone Step Mapping	72	47	49	36

5. Results and Discussion

We implemented the different algorithms described in this paper in C++ using OpenGL and its high level shading language GLSL. The measurements were made on a PENTIUM IV 3GHz and a GeForce 7600GS Graphics card.

Table 1 shows the pre-processing times of different shape maps. We note that the complete calculation (EDT + gradient) is performed very quickly (a few tenths of a second). This is an important advantage for interactive applications which must constantly update the shape map.

Table 2 gives the rendering speed of the basic and outward per-pixel extrusion mapping, as well as some reference techniques. The total number of iteration to find the intersection of a given ray is 30 steps. As expected, the basic and the outward extrusion mapping are sharply faster. The other techniques are nevertheless rendered with an interactive framerate. The images shown in Figure 5 are screen shots taken during the test, in which, we can notice that the shape box occupies the major part of the screen. It should be noted that the difference in speed between the various techniques is not uniform. It may depend on several parameters like the shape map, its dimensions, the depth, or the number of tiles.

Figure 5 shows a comparison between the extrusion mapping and other per-pixel displacement mapping techniques. We can notice that these techniques are clearly not adapted for rendering extrusions. Especially for important depths and for very fine details. In addition, they are much slower than the extrusion mapping.

Figure 6 shows several models created by per-pixel extrusion mapping. We can point out the variety of shapes and patterns that can be created using this technique. The number of graphics primitives which can be avoided is very important, and depends naturally on the complexity of the main shape.

6. Conclusion

In this paper we have introduced a new approach for rendering extruded shapes and patterns. The rendering algorithm is based on a per-pixel ray casting procedure, and uses just a single RGBA texture for space leaping. This shape map stores the Euclidean distance transform of a binary image and the unit gradient of the EDT. The proposed technique runs at interactive frame rates, and produces very convincing rendering, unlike similar approaches, which suffer from depth limitation or grazing angles artifacts.

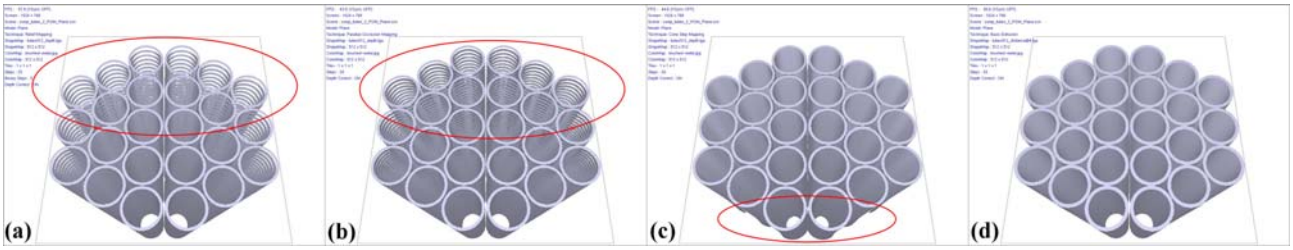

Fig. 5. Comparison between per-pixel displacement mapping techniques and per-pixel extrusion mapping. (a) Relief Mapping (57 fps). (b) Parallax Occlusion Mapping (43 fps). (c) Cone Step Mapping (44 fps). (d) Per-pixel Extrusion Mapping (80 fps). The shape map size is 512x512, the screen resolution is 1024x768, and the total steps number is 30. We can clearly note that the traditional methods are not suitable for shapes extrusion. Moreover, they are much slower than the extrusion mapping.

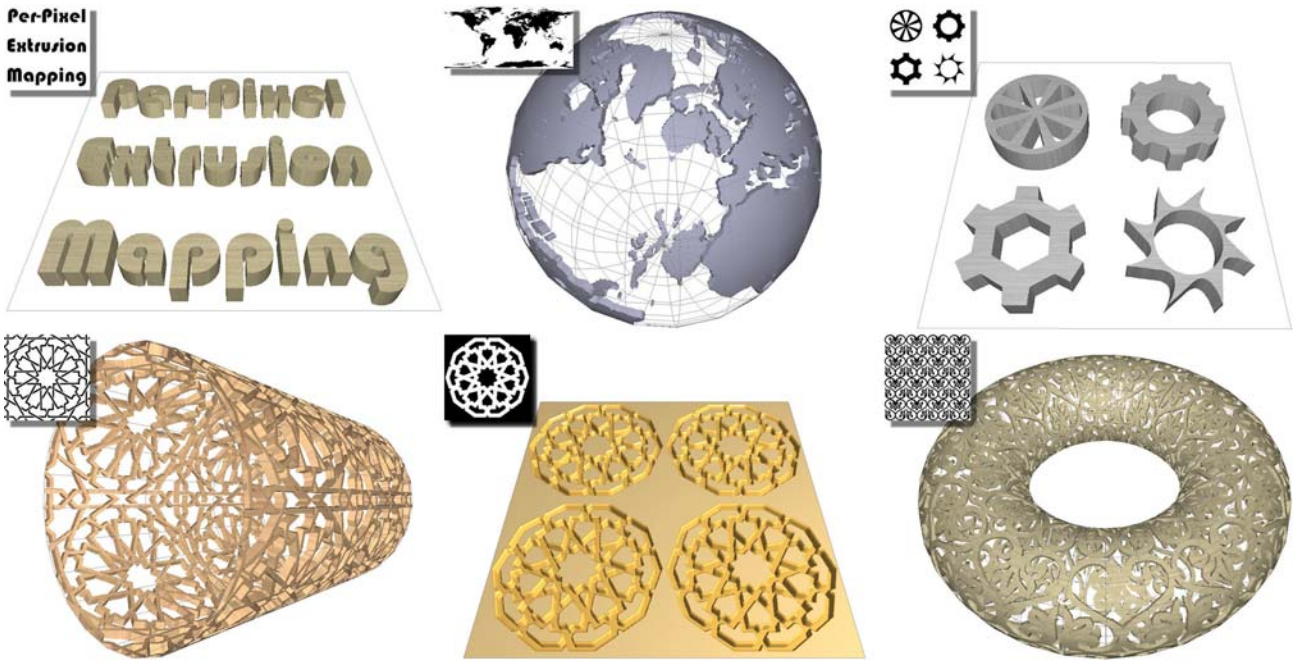

Fig. 6. Samples of models rendered using per-pixel extrusion mapping. These low polygonal meshes, mapped with extruded shapes and patterns, avoid creating and rendering a large number of vertices and polygons.

References

[1] L. Szirmay-Kalos and T. Umenhoffer, "Displacement Mapping on the GPU – State of the Art, Computer Graphics Forum", Vol 27 (6), pp. 1567-1592, 2008

[2] J. F. Blinn. "Simulation of wrinkled surfaces", Proc. of Siggraph 1978, ACMPress, pp. 286–292, 1978.

[3] T. Kaneko, T. Takahei, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda, and S. Tachi, "Detailed shape representation with parallax mapping", Proc. of the ICAT 2001, pp. 205–208, 2001.

[4] T. Welsh. "Parallax mapping with offset limiting: A per-pixel approximation of uneven surfaces", Infiscape Corp., 2004.

[5] M. Premecz, "Iterative parallax mapping with slope information", Central European Seminar on Computer Graphics CESCG'06, Available online at http://www.cescg.org/CESCG-2006/papers/ TUBudapest-Premecz-Matyas.pdf, 2006.

[6] R. L. Cook. Shade trees, Proc. of Siggraph 1984, ACMPress, pp. 23–231, 1984.

[7] J. W. Patterson, S. G. Hoggar, and J. R. Logie, "Inverse displacement mapping", Computer Graphics Forum, Vol. 10, no 2, pp. 129–139. 1991.

[8] F. Policarpo, M.M. Oliveira, and J. L. D. Comba, "Real-time relief mapping on arbitrary polygonal surfaces", Proc. of I3D'05, ACMPress, pp. 155–162, 2005.

[9] M. McGuire, "Steep parallax mapping", I3D'05 Poster, 2005.

[10] Z. Brawley, N. Tatarchuk, "Parallax Occlusion Mapping: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing", ShaderX3, 2004.

[11] N. Tatarchuk, "Dynamic parallax occlusion mapping with approximate soft shadows", SI3D'06, pp. 63-69, 2006.

[12] F. Policarpo, and M.M. Oliveira, "Relief Mapping of Non-Height-Field Surface Details", Proc. of the 2006 Symposium on Interactive 3D Graphics and Games, pp. 55–62. 2006.

[13] W. Donnelly, "Per-pixel displacement mapping with distance functions"; GPUGems2, Addison-Wesley, 2004.

[14] J. Dummer, "Cone Step Mapping: An Iterative Ray-Heightfield Intersection Algorithm", Technical Report, Available online at: http://www.lonesock.net, 2006

[15] D.W. Paglieroni and S. M. Petersen, "Height Distributional Distance Transform Methods for Height Field Ray Tracing", ACM Transactions on Graphics, Vol. 13(4), pp. 376-399, 1994.

[16] K. Oh, H. Ki, and C.H. Lee, "Pyramidal displacement mapping: A GPU-based artifacts-free ray tracing through an image pyramid", ACM Symposium on Virtual Reality Software and Technology (VRST'06), pp..75–82, 2006.

[17] A. Tevs, I. Ihrke, and H.-P. Seidel, "Maximum Mipmaps for Fast, Accurate, and Scalable Dynamic Height Field Rendering", Proc. Symp. Interactive 3D Graphics and Games, pp. 183-190, 2008.

[18] L. Baboud, and X. Décoret, "Rendering Geometry with Relief Textures", Proc. of Graphics Interface 2006, vol 137, pp. 195-201, 2006.

[19] F. Policarpo and M.M. Oliveira, "Relaxed Cone Stepping for Relief Mapping", GPU Gems 3, pp. 409-428, 2007.

[20] A. Halli, A. Saaidi, K. Satori, and H. Tairi, "Per-Pixel Displacement Mapping Using Cone Tracing", International Review on Computers and Software (I.Re.Co.S), Vol. 3(5), September 2008.

[21] M. M. Oliveira, F. Policarpo, "An Efficient Representation for Surface Details", UFRGS technical report RP-351, 2005

[22] J. Hirche, A. Ehlert, S. Guthe, and M. Doggett, "Hardware accelerated per-pixel displacement mapping", Proc. of Graphics Interface 2004, vol. 62, pp. 153–158, 2004.

[23] L. Wang, X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H. Y. Shum, "View-dependent displacement mapping", ACM Trans. Graphics, vol. 22, no. 3, pp. 334–339, 2003.

[24] X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H. Y. Shum. "Generalized displacement maps", Proc. of the Eurographics Symposium on Rendering, pp. 227-234, 2004.

[25] N. Ritsche, "Real-time shell space rendering of volumetric geometry". In Proceedings of GRAPHITE'06, pp. 265-274, 2006

[26] P.E. Danielsson, "Euclidean Distance Mapping", Computer Graphics and Image Processing, Vol. 14, pp. 227–248, 1980.

Akram Halli received the bachelor's and master's degrees from USMBA-Fez University in 2002 and 2004 respectively. He is currently working toward the PhD degree in the LIIAN Laboratory (Laboratoire d'Informatique, Imagerie et Analyse Numérique) at USMBA-Fez University. His current research interests include real-time rendering, Image-based rendering and virtual reality.

Abderrahim Saaidi received the bachelor's and master's degrees from USMBA-Fez University in 1997 and 2004 respectively. He is currently working toward the PhD degree in the LIIAN Laboratory at USMBA-Fez University. His current research interests include camera self calibration, 3D reconstruction and real-time rendering.

Khalid Satori received the PhD degree from the National Institute for the Applied Sciences INSA at Lyon in 1993. He is currently a professor of computer science at USMBA-Fez University. His is the director of the LIIAN Laboratory. His research interests include real-time rendering, Image-based rendering, virtual reality, biomedical signal, camera self calibration and 3D reconstruction.

Hamid Tairi received the PhD degree from USMBA-Fez University in 2001. He is currently a professor of computer science at USMBA-Fez University. His is also a member of the LIIAN Laboratory. His research interests include image processing, biomedical signal, Image-based rendering, visual tracking for robotic control and 3D reconstruction.