Volume 0 (1981), Number 0 pp. 1–7 COMPUTER GRAPHICS *forum*

Resampling Strategies for Deforming MLS Surfaces

J. P. Gois and G. C. Buscaglia

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo, São Carlos, Brazil

Abstract

*Moving-Least-Squares (MLS) Surfaces undergoing large deformations need periodic regeneration of the point set (*point-set resampling*) so as to keep the point-set density quasi-uniform. Previous work by the authors dealt with algebraic MLS surfaces, and proposed a resampling strategy based on defining the new points at the intersections of the MLS surface with a suitable set of rays. That strategy has very low memory requirements and is easy to parallelize. In this article new resampling strategies with reduced CPU-time cost are explored. The basic idea is to choose as set of rays the lines of a regular, Cartesian grid, and to fully exploit this grid: as data structure for search queries, as spatial structure for traversing the surface in a marching-cubes-like algorithm, and also as approximation grid for an interpolated version of the MLS surface. It is shown that in this way a very simple and compact resampling technique is obtained, which cuts the resampling cost by half with affordable memory requirements.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling G.1.2 [Numerical Analysis]: Approximation G.2.3 [Numerical Analysis]: Applications

1. Introduction

Point-based surfaces undergoing large deformations need periodic regeneration of the point set (*point-set resampling*). In fact, resampling must ensure that the density of points remains quasi-uniform, so as to guarantee reasonable results [WSS08, AK04]. An effective resampling algorithm should be endowed with the following attributes: (a) Systematically produce quasi-uniform point sets; (b) do not depend on any tunable parameter; and (c) be efficient in terms of CPU time and memory requirements.

In this article we address moving-least-squares (MLS) surfaces, for which there already exist resampling techniques oriented towards rendering [ABCO*03, GBMP04, GGG08], simplification [PGK02] or refinement [GBP05] applications. However, none of the existing techniques exhibits the three attributes discussed above when applied to the modeling of surfaces undergoing large deformations. Recently, Gois et al. [GNNB08] introduced a robust variant of algebraic MLS surfaces, together with a resampling strategy based on intersecting the surface with a suitably chosen set of rays. Though they proposed to take as set of rays the lines of a background cartesian grids, no use whatsoever was made of the background grid to accelerate the algorithm. This had the advantages of minimizing the memory requirements and making parallelization easy, at the expense of not optimizing the CPU time cost.

New resampling strategies with reduced CPU-time cost are explored here. The basic idea is to choose as set of rays the lines of a regular, Cartesian grid, and to fully exploit this grid: as data structure for search queries, as spatial structure for traversing the surface in a marching-cubes-like algorithm [Blo94], and also as approximation grid for an interpolated version of the MLS surface. We eliminate the need of kd-trees, ray-spheres intersections and other geometrical calculations needed in the ray-tracing algorithm. Further, exploiting the grid structure and the proposed traversing method, we are able to inhibit the creation of point clusters, thus avoiding to detect and remove them. This results in a simple and compact resampling technique, which cuts the cost by half with respect to the ray-tracing algorithm, keeping the memory requirements affordable. Another version is also introduced, based on the interpolation of the implicit MLS function, that further reduces the CPU time cost.

We illustrate the proposed techniques with some quite

stringent 2D and 3D examples, and assess their accuracy in the well-known single vortex flow test. The results show the convenience of the optimized, grid-based algorithm, since it saves CPU time without compromising the accuracy. On the other hand, the interpolation-based version is shown to significantly increase the error, making it attractive only if CPU-time requirements are critical.

2. MLS Surface Definitions

Let $\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$ be a set of quasi-uniformly-spaced points, and let $\{q_k\}$ $(k = 1, \ldots, 5)$ be the so-called algebraic–sphere polynomial basis (i.e.; $q_1(\mathbf{x}) = 1$, $q_2(\mathbf{x}) = x_1$, $q_3(\mathbf{x}) = x_2$, $q_4(\mathbf{x}) = x_3$, $q_5(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2$). We introduce the function

$$f(\underline{\alpha}, \mathbf{x}) = \sum_{k=1}^{5} \alpha_k \, q_k(\mathbf{x}) \tag{1}$$

where $\underline{\alpha} = (\alpha_1, \ldots, \alpha_5) \in \mathbb{R}^5$. The *Algebraic-Moving-Least-Squares* (AMLS) Surface $\mathcal{S}$ is defined as the zero-set of the (implicit) function

$$F(\mathbf{x}) = f(\underline{\gamma}(x), x), \tag{2}$$

where $\underline{\gamma}(x) = (\gamma_1(x), \ldots, \gamma_5(x)$ is obtained as follows:

Let us consider a continuous non-negative localization function:

$$\psi(s) = \begin{cases} (1-s^2)^4 & \text{if } s < 1 \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

and an influence radius Δ. A set of weight functions is defined as $w_i(\mathbf{x}) = \psi\left(\frac{\|\mathbf{x}-\mathbf{p}_i\|}{\Delta}\right)$. The following two-steps complete the definition:

Step 1: *For each* $\mathbf{p}_i \in \mathcal{P}$, *compute the* pseudo-normal vectors $\mathbf{N}_i$ *as the normal to the best-fit sphere at* $\mathbf{p}_i$. Let

$$\begin{aligned} Q &= \left\{ \underline{\alpha} \in \mathbb{R}^5 : \|\nabla_{\mathbf{x}} f(\underline{\alpha}, \mathbf{x})\| = 1 \;\; \forall \mathbf{x} \text{ s.t. } f(\underline{\alpha}, \mathbf{x}) = 0 \right\} \\ &= \left\{ \underline{\alpha} \in \mathbb{R}^5 : \alpha_2^2 + \alpha_3^2 + \alpha_4^2 - 4\alpha_1\alpha_5 = 1 \right\} \end{aligned} \tag{4}$$

For each point $\mathbf{p}_i \in \mathcal{P}$, compute $\underline{\alpha}_{(i)} \in Q$ by solving the constrained minimization problem

$$\underline{\alpha}_{(i)} = \arg\min_{\underline{\beta} \in Q} \sum_{j=1}^{m} w_j(\mathbf{p}_i) |f(\underline{\beta}, \mathbf{p}_j)|^2, \tag{5}$$

which reduces to a generalized eigenvalue problem in $\mathbb{R}^5$ [GNNB08, GG07]. The pseudo-normal $\mathbf{N}_i$ at $\mathbf{p}_i$ are given by

$$\mathbf{N}_i = \nabla_{\mathbf{x}} f(\underline{\alpha}_{(i)}, \mathbf{p}_i) / \|\nabla_{\mathbf{x}} f(\underline{\alpha}_{(i)}, \mathbf{p}_i)\|. \tag{6}$$

Step 2: *Computing the AMLS surface* $\mathcal{S}(\mathcal{P})$. It only remains to compute $\underline{\gamma}(\mathbf{x})$ for all $\mathbf{x}$, which is done by solving unconstrained minimization problem:

$$\begin{aligned} \underline{\gamma}(\mathbf{x}) \;=\; & \lim_{K\to\infty} \arg\min_{\underline{\beta} \in \mathbb{R}^5} \left(\sum_{i=1}^{m} w_i(\mathbf{x}) |f(\underline{\beta}, (\mathbf{p}_i))|^2 + \right. \\ & \left. + \; K \sum_{i=1}^{m} w_i(\mathbf{x}) \|\nabla_{\mathbf{x}} f(\underline{\beta}, \mathbf{p}_i) - \mathbf{N}_i\|^2 \right), \end{aligned} \tag{7}$$

All necessary details can be found in [GNNB08].

3. Point-set Resampling

As previously stated, the resampling technique consists of defining the new points at the intersections of $\mathcal{S}(\mathcal{P})$ with a Cartesian grid of spacing h. The details of the proposed implementation are as explained below.

3.1. Storing and querying points

The underlying grid automatically defines a simple hashing operator $\mathcal{H} : \mathbf{p}_m \in \mathcal{P} \mapsto \mathcal{H}(\mathbf{p}_m) = (i, j, k)$, as depicted in Fig. 1. Since more than one point can be mapped to the same grid vertex, chaining by linked lists is applied. The name of the corresponding method is `Hash()`.

A neighbor-query of a point $\mathbf{p}_m$, mapped to $\mathcal{H}(\mathbf{p}_m) = (i, j, k)$ consists of finding all points $\mathbf{p}_\ell \in \mathcal{P}$ such that $w_\ell(\mathbf{p}_m) \neq 0$. These points are efficiently computed as the image, by the inverse mapping $\mathcal{H}^{-1}$, of the neighboring vertices $[i-d, i+d) \times [j-d, j+d) \times [k-d, k+d)$, where d is an integer such that $d\,h > \Delta$. In general, we choose $d = 3$.

There are three methods which require neighbor-queries: The method `Update_Normals()`, which computes pseudo-normals for each point in $\mathcal{P}$ (as described in Section 2); the method `Update_Function()`, which computes the values of F at grid vertices in a neighboorhood of $\mathcal{P}$ (this neighborhood is described in Fig.1(b), it consists of 12 vertices in 2D and of 44 in 3D); finally, the method `Resample()`, which creates new points on grid edges. This last method is detailed in the next section.

3.2. Marching-cubes-like point creation

The intersections of the grid lines with the surface are computed by traversing the edges of the underlying grid in a marching-cubes-like way but, instead of creating polygons, just the points are created, making the case table unnecessary. The algorithm for the two-dimensional case reads:

3.2.1. Method: `Resample()`

```
For each point p ∈ P
  (I,J) = H(p);                                  /*1*/
  For each grid vertex (i,j) ∈ [I-1,I+3) x
    [J-1,J+3)                                    /*2*/
```

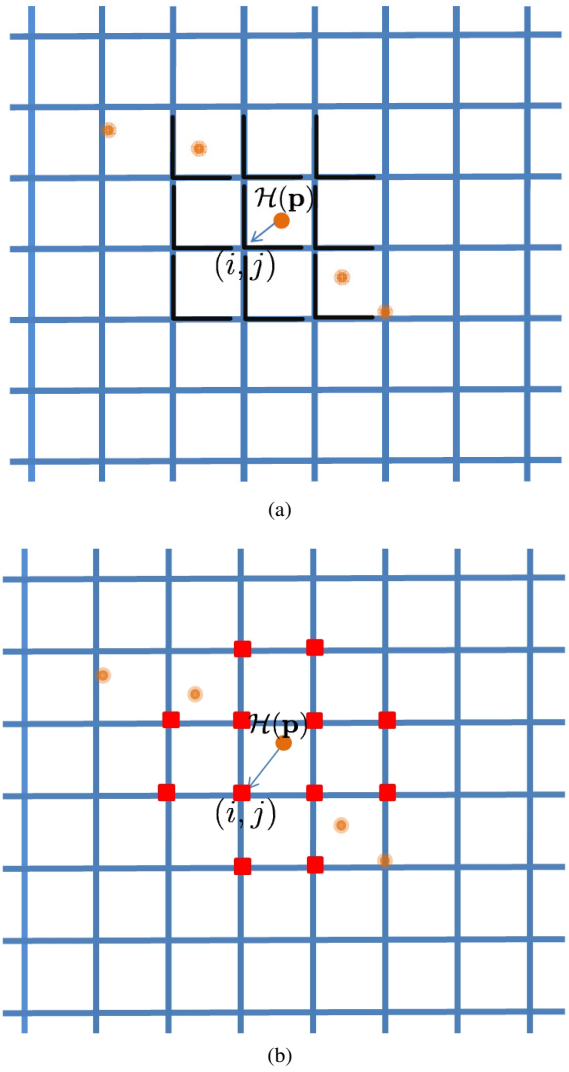


Figure 1: *Two-dimensional case of hashing operator $\mathcal{H}$. (a) `Resample()`: new points are created on pair of black edges. These "L"-shapes are traversed from bottom to top and from left to right. (b) `Update_Function()`: red vertices are those where MLS function is stored, to safe processing time.*

```
If (i,j) is not VISITED;                    /*3*/
  Set (i,j) as VISITED;                     /*4*/
  If MLS surface S(P) crosses edge
    ((i,j)(i+1,j));                         /*5*/
    Compute q as intersection be-
      tween
    S(P) and ((i,j)(i+1,j));                /*6*/
    s = H(q);                               /*7*/
    If ||s-(i,j)|| < ||s-(i+1,j)||          /*8*/
      Then ri = i;                          /*9*/
      Else ri = i+1;                        /*10*/
    If (ri,j) does not HAVECLOSE-
      POINT                                 /*11*/
      Insert q in the new list of
        points;                             /*12*/
      Set (ri,j) as HAVECLOSEPOINT;
        /*13*/
If S(P) crosses edge ((i,j)(i,j+1))
  /*14*/
  Proceed as previously, but
    considering the
  edge (i,j)(i,j+1);                        /*15*/
```

Above, we denote by $\left(\overline{(i,j)(i+1,j)}\right)$ the edge with vertices (i,j) and $(i+1,j)$. Fig. 1(a) depicts those edges that are traversed from a point $\mathbf{p}$. These edges are organized as a set of pairs sharing a common vertex; i.e. the "L"-shapes shown in black in the figure. These L-shapes are traversed from bottom to top and from left to right, but since several points can share the same grid vertex (i,j) as neighbor, the flag `VISITED` is assigned to (i,j) to preclude the same L-shape to be analyzed more than once. Also, the flag `HAVECLOSEPOINT` is used to avoid creating points very close from one another, thus eliminating the need of removing point clusters. The 3D algorithm is analogous.

4. MLS Surfaces under Deformations

Let $\mathcal{P}_0$ be the initial point set and let $\mathcal{S}$ be the operator that, for any point set, assigns the AMLS surface defined by it. We thus have an initial surface $S_0 = \mathcal{S}(\mathcal{P}_0)$.

Our aim is to deform the surface S_0 according to some known deformation history $\varphi_{s\to t} : \mathbb{R}^n \to \mathbb{R}^n$, continuous with respect to the parameters s and t, which can be thought of as initial and final instants of time. To be precise, a point $x \in \mathbb{R}^n$, as it deforms from time s to time t, goes from x to $\varphi_{s\to t}(x)$. The deformed surface S_t is, in terms of $\varphi_{s\to t}$, defined as

$$S_t = \varphi_{0\to t}(S_0). \tag{8}$$

Though S_t is well-defined by (8), it is not feasible to compute it from its definition (one should map each and every $x \in S_0$!). Instead, the initial point set is mapped to its deformed position, $\mathcal{P}_t = \varphi_{0\to t}(\mathcal{P}_0)$, and used to define an approximation of S_t by

$$\tilde{S}_t = \mathcal{S}(\mathcal{P}_t) = \mathcal{S}\left(\varphi_{0\to t}(\mathcal{P}_0)\right). \tag{9}$$

The surface $\tilde{S}_t$ is only approximate because the operators $\mathcal{S}$ and $\varphi_{0\to t}$ do not commute. The approximation worsens as the point set $\mathcal{P}_t$ deforms and becomes unevenly distributed.

The basic idea to tackle this difficulty is to periodically replace the point set $\mathcal{P}_t$ with a new point set obtained by resampling the surface $\mathcal{S}(\mathcal{P}_t)$.

To clarify ideas, let $\mathcal{R}$ be the resampling operator defined in Section 3. Notice that the deformation history from 0 to t can be split, for some $0 < \tau < t$, as

$$\varphi_{0\to t}(x) = \varphi_{\tau\to t}\left(\varphi_{0\to\tau}(x)\right) \tag{10}$$

We can thus define a new approximation of S_t, in which the point set is deformed from 0 to τ, then the surface is resampled, and the new point set is deformed from τ to t. In terms of the operators defined above this reads

$$\tilde{\tilde{S}}_t = \mathcal{S}\left(\varphi_{\tau\to t}\left(\mathcal{R}\left(\mathcal{S}(\varphi_{0\to\tau}(\mathcal{P}_0))\right)\right)\right) \tag{11}$$

This can be performed periodically with step τ, leading to the following algorithm:

```
For each deformation φ_{mτ→(m+1)τ} (m = 1,2,...)
  Update_Normals()
  Update_Function()
  Resample()
  Deform_Points()
  Hash()
End For
```

5. Results

We first examine the improvements brought by the proposed resampling strategy to the modeling of curves and surfaces under large deformations. The first example consists of deforming a circle of unit radius into a four-leaf clover. The deformation consists of several steps, as shown in Fig. 2(a), and is defined by repositioning the control points (as shown in the figure) using the method proposed by Schaefer et al. [SMW06]. Part (b) of the figure shows the evolution of the MLS curve defined by applying the deformation to a point set consisting of 250 evenly spaced points on the initial circle (shown in blue). Notice that, because of our choice of polynomial basis, the initial MLS curve exactly coincides with the circle. If no resampling is applied and $\Delta = 0.0075$ is kept fixed, the MLS curve associated to the deformed point set is only well defined until the stage shown in the second graph of Fig. 2(b). After that stage, the points on the stem are too far apart (third graph) and the MLS curve breaks down. One can nevertheless go on deforming the point set so as to take it to the final deformed position. In the rightmost graph of Fig. 2(b) we show the final point set and the pseudo-normals obtained with the `Update_Normals()` method. Their illogical magnitudes and orientations are clear evidences of the breakdown of the MLS curve.

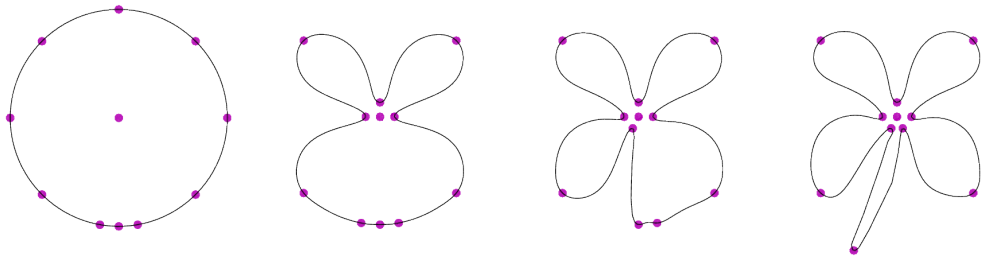
In Fig. 2(c), on the other hand, we show the results obtained with the resampling algorithm, using a grid that keeps the distance between points close to the initial one. In this case, the MLS curve remains well defined throughout the deformation, and the final curve is a good approximation of the exact final shape (compare to the rightmost graph in part (a) of the figure). The pseudo-normals are also plotted, showing that they are consistent with the deformed curve.

We also examined the ability of the proposed algorithm to model three-dimensional deformations. In Fig. 3 we show two representative examples, both of them using a 250^3 grid. The first one corresponds to exactly the same deformation as above, but now applied to a sphere. The point set contains 10,000 points initially, and ends up with 18,000 points, which is a quite small quantity for modeling complex 3D shapes. The second one corresponds to a model of Homer Simpson, which is deformed so as to render it "elf-like". The initial point set has 48,000 points, which reduces to 20,000 points in the last stage. Both examples show that the resampling algorithm allows us to accomplish severe deformations without any user intervention, and without the appearance of numerical artifacts.

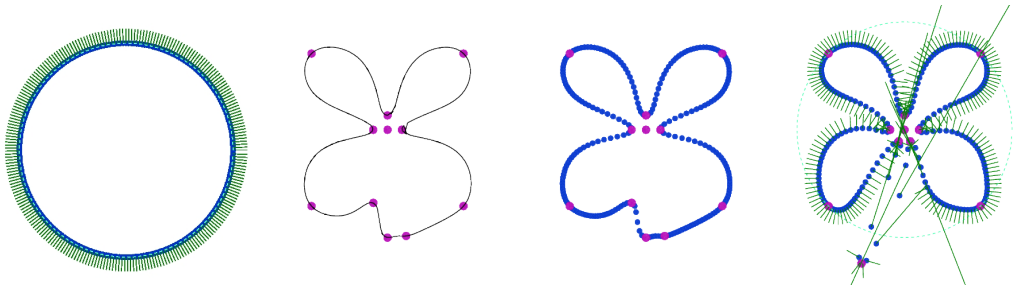
Having shown the benefitial effects of resampling on the modeling of deformations of point-set surfaces, let us now turn to a quantitative assessment of the newly-developed, marching-cubes-like algorithm. For this purpose, we consider the *single vortex flow test* already considered in the previous article [GNNB08]. The initial geometry is a circle centered at $(0.5, 0.75)$ with radius $r = 0.15$ inside a square unit domain. The deformation is defined by its associated velocity field ($\mathbf{v}(x,t) = \partial_s \varphi_{t\to t+s}(x)|_{s=0}$)

$$\mathbf{v}(x_1,x_2,t) = T(t)\Big(\sin^2(\pi x_1)\sin(\pi x_2)\cos(\pi x_2), \\ \sin(\pi x_1)\cos(\pi x_1)\sin^2(\pi x_2)\Big) \tag{12}$$

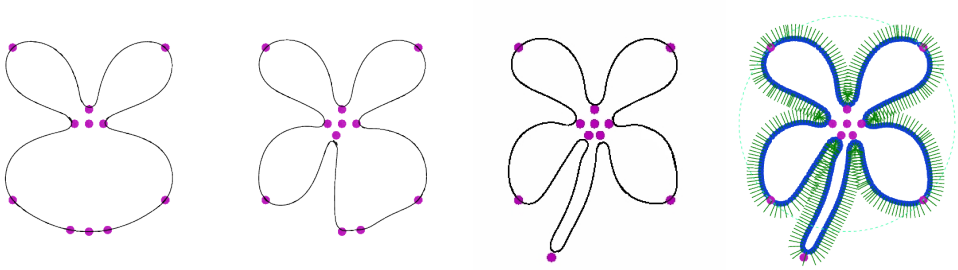
where $T(t) = 2\cos\left(\pi\frac{t}{8}\right)$. At $t = 8$ the exact surface S_t coincides with the initial circle S_0, as illustrated in Fig. 4. We resample the surface every 0.01 time units, leading to a total of 800 resampling operations. A comparison of the ray-tracing algorithm with the marching-cubes-like algorithm is reported in Table 1. The total time (in seconds) for a single processor of a QuadCore 3.0 GHz is 96.57 (512^2 grid) and 48.73 (256^2 grid) for the original ray-tracing algorithm. These CPU times are reduced by 60% with the marching-cubes-like algorithm, to 39.95 and 20.69, respectively. Most of the time of the ray-tracing algorithm (85%) is spent in calculating the intersections to get the new point set. In the new algorithm, this is accomplished by calculating the values of the implicit function at the vertices (Update Function in the table) so as to detect edges in which change of sign occurs, together with the resampling step that computes the actual intersections. These two steps of the new algorithm take 57% of the CPU time, and in them concentrates most of the gain with respect to the ray-tracing algorithm, with a reduction from 70.5 seconds to 22.6 seconds on the 512^2 grid. Some gain is also obtained in the step of calculating the pseudo-normals, coming from the convenience of the hash-



(a) Exact deformation



(b) Approximate (MLS) deformation without resampling



(c) Approximate (MLS) deformation with 120 resampling operations along the process

Figure 2: *Deforming a circle into a four-leaf clover, with a deformation defined by the control points (in purple). (a) Exact deformed shapes. (b) MLS curve corresponding to the deformed point set (on the left) at the last stage in which the curve is well defined. (c) Same as (b), but with resampling turned on (120 resampling operations between initial and final shapes).*

ing technique based on the background grid with respect to the general kd-tree used in the ray-tracing version. Finally, the convenience of inhibiting the creation of points that are too close, as proposed in method `Resample()`, leads to additional CPU-time savings, since less points are deformed and no need of removing excess points is needed. This saves about 4 seconds on the 512^2 grid.

We believe the performances of Table 1 to be quite satisfactory, since they tell us essentially that a relatively complex 2D shape of about 1,000 points can be resampled in just 24 milliseconds. Similar tests in 3D yield an estimate of about 15 seconds for a complex 3D shape defined by 50,000 points.

It is possible to further reduce the CPU-time by not computing the intersection exactly. Instead, the new points can be created at those edges where the implicit function changes sign by linearly interpolating the values at the grid points. This leads to a redefinition of the AMLS surface, reducing its accuracy (as will be shown below), but reduces the total time to just 24 seconds in the 512^2 grid. This means just 15 milliseconds to resample a 2D curve of 1,000 points.

It is however essential to numerically assess the accuracy both of the new algorithm and of its interpolated variant. As discussed above, the marching-cubes-like algorithm does not generate exactly the same points as the ray-tracing one, implying that the approximate deformed surfaces they generate are not coincident. Further, the interpolated version of the proposed algorithm explicitly modifies the MLS surface by redefining it as the zero level-set of the interpolant of the implicit function. To quantify the difference between the different approximations, we measure the error of the final shape $\tilde{\tilde{S}}_{t=8}$ with respect to the exact one S_0 (i.e., the unit cir-

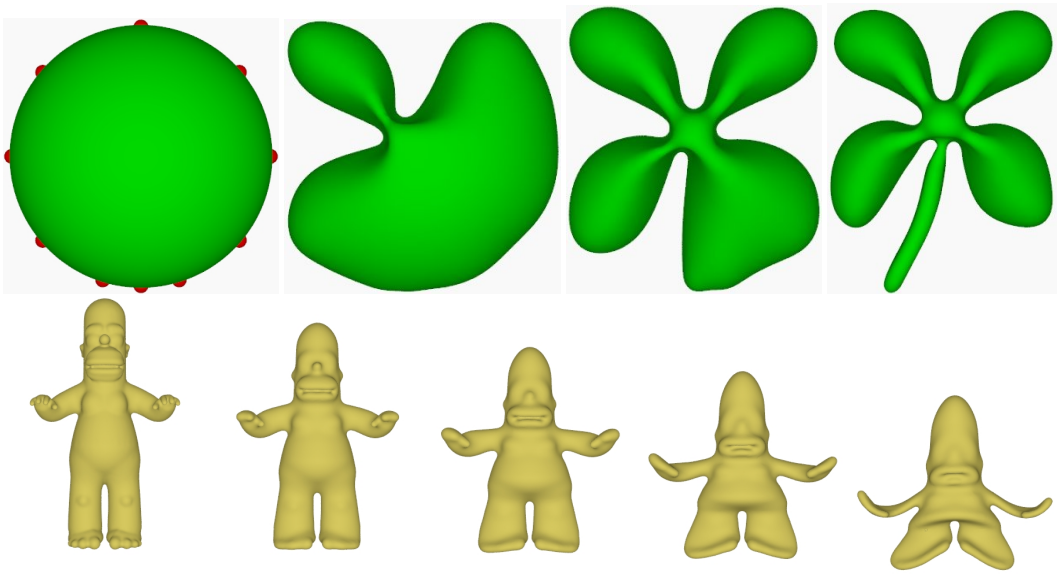


Figure 3: *Three-dimensional examples. The top sequence corresponds to the same deformation of Fig. 2, here applied to a sphere. The bottom sequence corresponds to the deformation of Homer Simpson into an elf-like creature.*

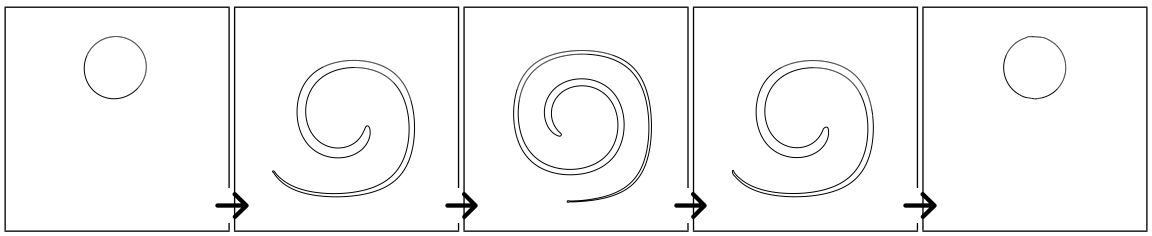


Figure 4: *Scheme of single vortex flow deformation at* $t = 0$ *(initial),* 2, 4 *(maximun torsion),* 6 *and* 8 *(final curve) time units. Results of the proposed method corresponding to* 512^2*-grid.*

cle). This error is defined as the area between $\tilde{\tilde{S}}_{t=8}$ and S_0 (this is denoted by L_1-error by Gois et al. [GNNB08] and other authors).

Table 2 reports the errors obtained with the new algorithm, both in its non-interpolated and interpolated versions, together with those obtained with the original ray-tracing algorithm. The comparison is made for meshes ranging from 128^2 to 1024^2. It is observed that the new algorithm does not exhibit any significant loss of accuracy with respect to the original ray-tracing one. In fact, the error of both algorithms converge to zero at a rate of about $h^{2.7}$, h being the grid spacing. The accuracy of the interpolated version, on the other hand, is significantly poorer. In fact, its error is of order h^2, requiring a 1024^2-grid to attain the same accuracy as the non-interpolated algorithm on a 256^2-grid.

6. Conclusion

We have explored the benefits of exploiting a background Cartesian grid to reduce CPU-time cost of resampling MLS surfaces under deformation. A new algorithm was proposed which traverses the MLS surface in a marching-cubes-like way and cuts the cost by half. It was also shown that further savings can be obtained by replacing the implicit function by its interpolant. An accuracy assessment was then conducted, which suggests that the non-interpolated version should be preferred in general, unless CPU-time restrictions are critical.

Suggested improvement of the proposed method includes the development of point-set quality to automatically trigger resampling, and the use of the hierarchical run-length encoding of the grid data [HNB*06] to enable the use of very large 3D grids.

References

[ABCO*03] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9*, 1 (2003), 3–15.

Table 1: *CPU time of ray-tracing and marching-cubes-like algorithms for the single vortex flow test*

Original	**512**		**256**	
Method	Sec	%	Sec	%
Deform	4.59	2.55	2.35	2.55
Remove	1.04	0.58	0.53	0.58
Rebuild Tree	0.19	0.11	0.07	0.08
Intersections	70.51	85.52	35.9	86.05
Comp. Normals	20.24	11.25	9.88	10.74
Total	**96.57**	100.00	**48.73**	100.00

Proposed	**512**		**256**	
Method	Sec	%	Sec	%
Deform	1.80	4.51	1.12	5.41
Update F	6.55	16.40	3.20	15.47
Resample	16.11	40.33	8.86	42.82
Hash	0.48	1.20	0.17	0.82
Comp. Normals	15.01	37.57	7.34	35.48
Total	**39.95**	100	**20.69**	100

Table 2: *Single vortex flow test: Errors obtained with the proposed technique and with its interpolated version, as compared to those of the original ray-tracing technique. NC denotes that the error was not computed and F denotes that the MLS surface failed*

Grid	Original	Proposed	Interpolated
128	0.00730	0.00643	F
256	0.00107	0.00102	0.02326
512	0.00015	0.00015	0.00573
1024	NC	0.00003	0.00138

[AK04] AMENTA N., KIL Y. J.: The domain of a point set surfaces. In *Eurographics Symposium on Point-based Graphics* (2004), pp. 139–147.

[Blo94] BLOOMENTHAL J.: An implicit surface polygonizer. In *Graphics Gems IV*, Heckbert P., (Ed.). Academic Press, Boston, 1994, pp. 324–349.

[GBMP04] GUENNEBAUD G., BARTHE L., MATHIAS, PAULIN: Dynamic surfel set refinement for high quality rendering. *Computer & Graphics 28*, 6 (2004), 827–838.

[GBP05] GUENNEBAUD G., BARTHE L., PAULIN M.: Interpolatory refinement for real-time processing of point-based geometry. *Comp. Graphics Forum 24*, 3 (2005), 657–667.

[GG07] GUENNEBAUD G., GROSS M.: Algebraic point set surfaces. In *SIGGRAPH* (2007), ACM, p. 23.

[GGG08] GUENNEBAUD G., GERMANN M., GROSS M.: Dynamic sampling and rendering of algebraic point set surfaces. In *Eurographics* (2008).

[GNNB08] GOIS J. P., NAKANO A., NONATO L. G., BUSCAGLIA L. C.: Front tracking with moving-least-squares surfaces. *Journal of Computational Physics 227*, 22 (2008), 9643–9669.

[HNB*06] HOUSTON B., NIELSEN M. B., BATTY C., NILSSON O., MUSETH K.: Hierarchical rle level set: A compact and versatile deformable surface representation. *ACM Trans. Graph. 25*, 1 (2006), 151–175.

[PGK02] PAULY M., GROSS M., KOBBELT L. P.: Efficient simplification of point-sampled surfaces. In *VIS'02: Proceedings of the conference on Visualization* (2002).

[SMW06] SCHAEFER S., MCPHAIL T., WARREN J.: Image deformation using moving least squares. *ACM Trans. Graph. 25*, 3 (2006), 533–540.

[WSS08] WANG H., SCHEIDEGGER C., SILVA C.: Optimal bandwidth selection for mls surfaces. In *Shape Modeling and Applications* (2008), pp. 111–120.