Modeling Shapes for Pattern Recognition: A Simple Low-Cost Spline-based Approach

Radu-Daniel VATAVU[1], Stefan-Gheorghe PENTIUC[1], Laurent GRISONI[2] , Christophe CHAILLOU[2]

[1] *"Stefan cel Mare" University of Suceava*
str.Universitatii nr.13, RO-720229 Suceava
vatavu@eed.usv.ro, pentiuc@eed.usv.ro
[2] *Laboratoire d'Informatique Fondamentale de Lille, Villeneuve d'Ascq 59650, France*
laurent.grisoni@lifl.fr, christophe.chaillou@lifl.fr

Abstract—**We present a simple procedure for modeling shapes and trajectories of points using cubic polynomial splines. The procedure may prove useful for researchers working in the field of pattern recognition that are in the search of a simple functional representation for shapes and which are not particularly interested in diving into the high-theoretical aspects of more complex representations. The use of splines brings in a few advantages with regards to data dimensionality, speed and accuracy of processing, with minimal effort required for the implementation part. We describe several algorithms for data reduction, spline creation and query for which we provide pseudo code procedures in order to demonstrate the ease of implementation. We equally provide measurements on the approximation error and rate of data reduction[1].**

Index Terms—**spline, cubic polynomial, pattern recognition, algorithms, shape modeling, curves, approximation error, Haussdorff distance**

I. INTRODUCTION

The main purpose of this paper is to present researchers working in the field of pattern recognition with a simple and low-cost procedure for modeling shapes and raw trajectories of points in the plane. Although the process is very simple, the advantages are multiple: reduction in the dimensionality of data being stored, smoother shapes and increased accuracy of processing due to re-sampling at custom resolutions.

Although there are thorough procedures that allow for modeling shapes using splines (see B-Splines) or wavelets [5, 6] with accurate and precise control, their understanding and use in practice may prove hard to achieve for novices or they may demand additional attention in order to understand their underlying full potential (or equally may lead to unwanted side effects due to misuse). We present a simple solution for modeling general shapes or trajectories of points that is easy to implement and gives good results in practice. We equally provide pseudo code of the various algorithms as well as a discussion on the approximation error and data reduction rates.

II. PRE-PROCESSING DATA

The original (sometimes raw) data given as a series of points, as it was acquired or computed from various sources, may be subjected to acquisition errors, computation

---

truncation errors or may contain too many points than needed. We start by pre-processing the original data using a simple filtering algorithm inspired from the Douglas-Peucker poly-line reduction algorithm [1]. The raw data may be looked at as a series of r two-dimensional points sampled at equal intervals of time:

$$p = \{p(0), p(T), p(2 \cdot T) \mathbf{K} \, p((r-1) \cdot T)\} \qquad (1)$$

where T is the sampling interval in time and $p_i = p(i \cdot T) = (x_i, y_i) \in R^2$ represents the ith two-dimensional sampled point.

An initial simple filter that may be applied on the raw data consists in removing all the points that are too close to their neighbors than a given threshold eps. The principle is illustrated in Figure 1 while the pseudo code procedure DATA-REDUCTION-1 is listed under the appendix. The complexity of the algorithm is $O(r)$ where r is the length of the points array.
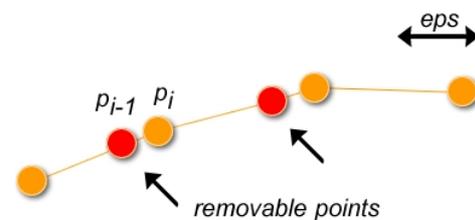

Figure 1. Data Reduction, 1st filter: points that are closer than a given threshold *eps* with respect to their neighbors may be removed.
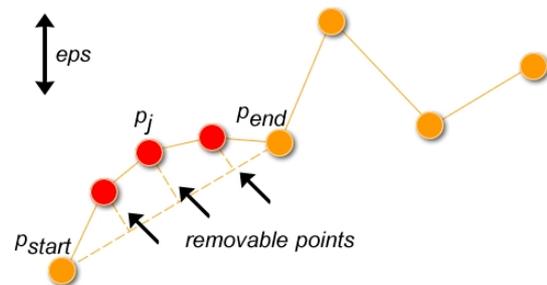

Figure 2. Data Reduction, 2nd filter: points that are closer than a given threshold value *eps* from the line segment between their previous and next neighbors may be removed.

We may further consider a second filter for which we see as superfluous hence removable all the points p_j of the raw

data trajectory for which the Euclidean distance to the line segment having as extremities the previous and the next points p_{j-1} and p_{j+1} is less than a given error margin *eps*. Even more, we can further enhance the filter by removing all the points p_j that lie between two points p_{start} and p_{end} with *start* $<$ *j* $<$ *end*, *start, j, end* $\in$ *{0..r-1}* if the average Euclidean distance from all the points p_j to the line segment having as extremities p_{start} and p_{end} is less than *eps*. The principle is illustrated in Figure 2 while the data reduction algorithm is given as a pseudo code procedure DATA-REDUCTION-2 under the appendix. The complexity of the algorithm is $O(r^2)$ where *r* is the length of the points array. A faster version with several enhancements running in $O(r \cdot log(r))$ is described in [1].

The opposite operation for data reduction is data enhancement. After the original trajectory has been reduced by consecutively running the two filters, it is sometimes useful to insert a number of *N* linearly interpolated points between any two consecutive points p_i and p_{i+1}. The reason for this is that further modeling of the reduced points using splines leads to a much finer and smoother result but this step is not mandatory. Usually, good results are obtained for *N=2* or *N=3* points. Figure 3 illustrates the insertion principle while the linear interpolation pseudo code procedure DATA-ENHANCEMENT-1 is given under the appendix. The complexity of the algorithm is $O(N \cdot r)$ where *r* is the length of the points array but it may be considered as $O(r)$ due to the fact that *N* takes small values such as *1, 2 or 3*. The decision to insert additional points may be further constrained by the actual distance between the two consecutive points p_i and p_{i+1} with respect to a given threshold value *eps*.
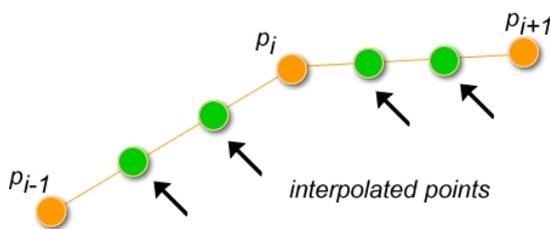

Figure 3. Data enhancement: linear interpolated points are inserted in the reduced trajectory.

III. CATMULL-ROM SPLINES

Splines represent a powerful mathematical tool for the representation of curves through the means of control points at given timestamps (knots) and blending functions that allow computation of the curve points [2].

Catmull-Rom splines are a family of cubic interpolating splines defined such that the tangent t_i at each control point p_i is calculated using the previous and next points on the spline, p_{i-1} and p_{i+1}:

$$t_i = r_i \cdot (p_i - p_{i-1}) + (1 - r_i) \cdot (p_{i+1} - p_i) \tag{2}$$

The spline is completely defined by its control points p_i and their associated tangents t_i as Figure 4 illustrates. The *i*th segment of the spline is defined between the control points p_i and p_{i+1} as cubic polynomial:

$$l(u) = \sum_{k=0}^{3} c_k \cdot u^k \tag{3}$$

where *u* is a local parameter that varies between [0..1]. The coefficients c_k, *k=0..3*, are computed for each segment using end continuity conditions:

$$\begin{cases} p_i & = l(0) & = c_0 \\ p_{i+1} & = l(1) & = c_0 + c_1 + c_2 + c_3 \\ t_i & = l'(0) & = c_1 \\ t_{i+1} & = l'(1) & = c_1 + 2 \cdot c_2 + 3 \cdot c_3 \end{cases} \tag{4}$$

The above system may be put under the concise form:

$$C = G \times P \tag{5}$$

where *C* is the coefficients vector $C=[c_0\ c_1\ c_2\ c_3]^T$, *P* is the control points vector $P=[p_{i-1}\ p_i\ p_{i+1}\ p_{i+2}]^T$ and *G* is the geometry matrix given by:

$$G = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -r_i & 2r_i - 1 & 1 - r_i & 0 \\ 2r_i & -4r_i + r_{i+1} - 1 & 2r_i - 2r_{i+1} + 2 & r_{i+1} - 1 \\ -r_i & 2r_i - r_{i+1} + 1 & 2r_{i+1} - r_i - 2 & 1 - r_{i+1} \end{bmatrix} \tag{6}$$

The parameters r_i are known as tensions and affect how sharply the curve bends at the interpolated control point p_i. A common value is *0.5* however r_i may be determined by the length of the adjacent segments $p_{i-1} p_i$ and $p_i p_{i+1}$:

$$r_i = \frac{|p_i p_{i+1}|}{|p_{i-1} p_i| + |p_i p_{i+1}|} \tag{7}$$

where |·| denotes the length of a segment. This choice of the tension values allows for each segment $p_i p_{i+1}$ to be pondered inversely proportional to its length when considered for the computation of the tangents. The result that is obtained in the end is a much smoother curve shape.

The system (5) has a unique solution if $r_i \neq 0$ and $r_{i+1} \neq 1$, conditions that are fulfilled due to the fact that $0 < r_i < 1$ as may be noted from equation (7). The algorithms for computing the tension values, tangents and for generating the Catmull-Rom spline from a series of points COMPUTE-TENSIONS, COMPUTE-TANGENTS and GENERATE-CATMULL-ROM-SPLINE are given in pseudo code under the appendix section.
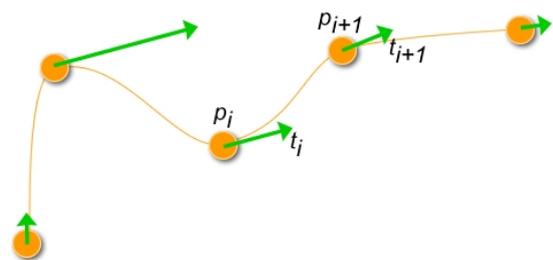

Figure 4. A Catmull-Rom spline is defined by a number of control points p_i and the tangent at each point t_i.

Creating a Catmull-Rom spline from a series of *r* points *P* is achieved with a complexity of *O(r)* as the *r* points are only traversed three times: in order to compute tensions *O(r)*, tangents *O(r)*, and finally to generate the spline coefficients *O(r)*. Computing the point on the spline at a given moment *T* (querying the spline) is done in *O(1)* time as presented in the pseudo code procedure EVALUATE-SPLINE listed under the appendix although even faster evaluation methods exist [3].

Catmull-Rom splines present several interesting properties [4]:

i. They pass through all of the control points p_i which are (usually) located at a time interval of 1 time unit away one from another.

ii. They have local control which means that modifying one control point only affects the part of the curve near that control point. This can be seen from the system (4) where the coefficients of the cubic spline segment *i* only depend on the points p_{i-1}, p_i, p_{i+1} and p_{i+2}.

iii. Catmull-Rom splines are continuous functions of class C^1, i.e. there are no discontinuities in the tangent direction or magnitude. They are not C^2 continuous however: the second derivative is linearly interpolated within each segment causing the curvature to vary linearly over the length of the segment.

iv. They present affine invariance which means that an affine change in the coordinates system does not affect the geometry of the curve. Practically, the curve shape rests the same whether it is rotated, scaled or translated. This is an important feature to have for pattern recognition problems that require affine invariance recognition: the invariance is acquired right from the very beginning as it comes from the representation form.

v. Catmull-Rom splines are low-order cubic polynomial functions which makes them easy to compute.

IV. SHAPES AS SPLINES FOR PATTERN RECOGNITION

Representing shapes or trajectories of points as splines brings in several advantages:

i. First of all, the dimensionality of data is considerably reduced. For example, the shapes illustrated in Figure 5 are downsized as follows: flower (365 points downsized to 62), heart (241 to 34), butterfly (317 to 31) and rectangle (59 to 13) which translates into an average reduction rate of (356 + 241 + 317 + 59) / (62 + 34 + 31 + 13) = 973 / 140 = 6.95 or 7:1 ratio for the four cases.

ii. Second, we benefit of a continuous C^1 class function for representing shapes which allows the application of results directly from differential geometry. This in turn provides more accurate results for local or global shape parameters (such as tangents for example) instead of values that would have been otherwise computed using approximation methods.

iii. The interpolating property of the spline may attenuate small execution mistakes or small acquisition errors, by providing in the end smooth curves.

iv. Splines may be sampled at any given resolution, as fine as desired as Figure 6 illustrates. The samplings at different resolutions, coarser or finer, may be needed by further discrete computations. This useful property derives directly from the fact that our representation takes the form of a continuous function and hence we are allowed to select any sampling step *TStep* in the spline total time-length interval *[0..T]*. Equally important, the sampling may be done uniformly, at equally spaced intervals, or non-uniformly which is a definite improvement on the original series of raw points.

v. As Catmull-Rom splines are affine invariant, the shapes representations present as well translation, rotation and size invariance which is a definitely plus for any classifier algorithm that is comparing or matching shapes.
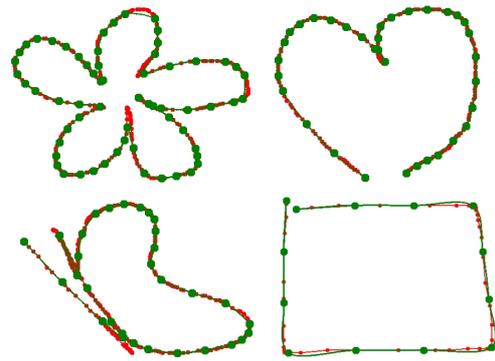

Figure 5. Original shapes (red) and splines (green) over-imposed. Control points of the splines are highlighted. Average data reduction for the four cases: 7 points to 1.
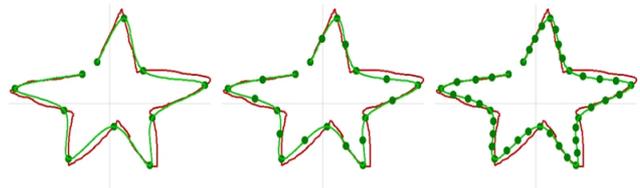

Figure 6. Splines may be sampled at any resolution desired, fine or coarse, to accommodate further discrete computations.

V. DISCUSSION

We further address the approximation error as well as the data reduction rate that are introduced by spline modeling. The measurements are computed on a set of 4,800 shape samples which represent actual gesture trajectories as they were acquired from 10 subjects [8]. Sample shapes include: triangle, rectangle, caret, check, question mark, curly braces, etc. The entire set of gestures as well as details on the acquisition process and the database are given in [8].

Approximation error may be handled using the Hausdorff distance [7]. By definition, two sets of points are within Hausdorff distance *r* from each other iff any point of one set is within distance *r* from some point of the other set. Otherwise put, the Hausdorff distance between two curves is the greatest lower bound of the distance between the two curves. Given two curves *p* and *q* sampled into *|p|* and *|q|* points, the Hausdorff distance between them *h(p, q)* is computed as being the maximum distance between every point from the 1st curve *p* to its closest point on the 2nd curve

q:

$$h(p,q) = \max_{i=1,|p|}\left\{\min_{j=1,|q|}\left\{d\left(p_i,q_j\right)\right\}\right\} \tag{8}$$

We define the approximation error (AE) as percentage by dividing the Hausdorff distance to the length of the bounding square of our shapes in the $[-1,1]^2$ space:

$$AE(p,q) = \frac{h(p,q)}{2} \cdot 100 \tag{9}$$

We may also compute the average Hausdorff distance as:

$$\bar{h}(p,q) = \frac{1}{|p|}\sum_{i=1}^{|p|}\min_{j=1,|q|}\left\{d\left(p_i,q_j\right)\right\} \tag{10}$$

which gives the corresponding average approximation error:

$$\overline{AE}(p,q) = \frac{\bar{h}(p,q)}{2} \cdot 100 \tag{11}$$

We also compute the data reduction rate (RR) as the percentage decrease between the number of control points needed to store the spline (N_{Spline}) and the original number of points ($N_{Original}$):

$$RR = \frac{N_{Original} - N_{Spline}}{N_{Original}} \cdot 100 \tag{12}$$

Figure 7 presents the variation of the approximation error (computed as Hausdorff distance and average Hausdorff distance) with the variation of *eps* (threshold value used for filtering data). Figures 8 and 9 present the data reduction rate as absolute values as well as reduction percentage. It may be clearly seen that the greater the threshold value *eps*, the greater is the approximation error and the smaller the number of points needed to store the spline. In the average case, choosing *eps* of 2.5% of the length of the shape bounding rectangle ($2.5 \cdot 2 / 100 = 0.05$), we obtain a spline representation which approximates the original data within a maximum Hausdorff distance of 5% (= 0.1) and average of 1.5% (= 0.03) and we get a reduction rate of 77.77% (reduction from 70 to 14 points or 5:1).
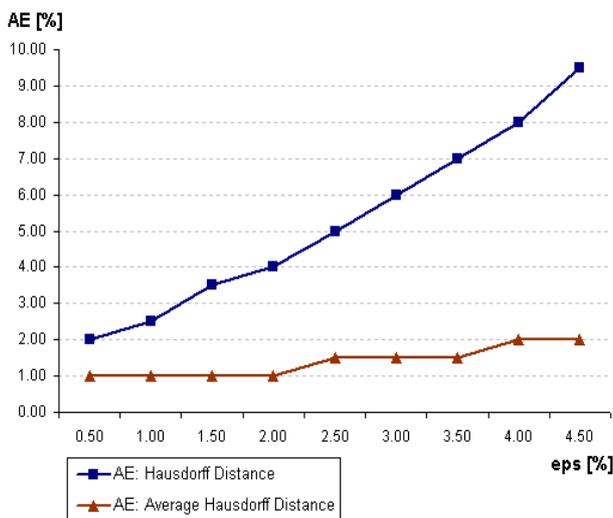

Figure 7. Approximation errors (AE) as Hausdorff and average Hausdorff distance versus the filtering threshold value *eps*.
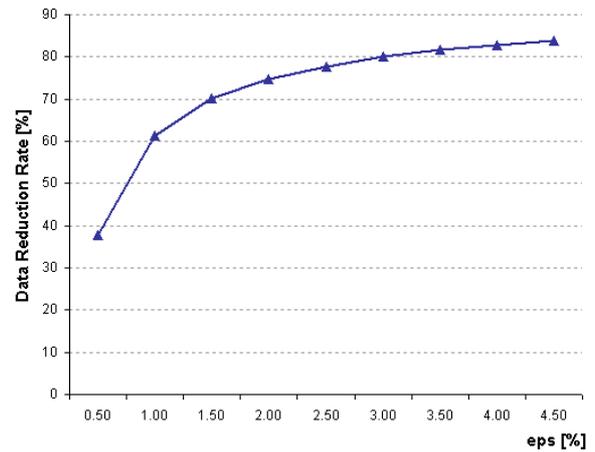

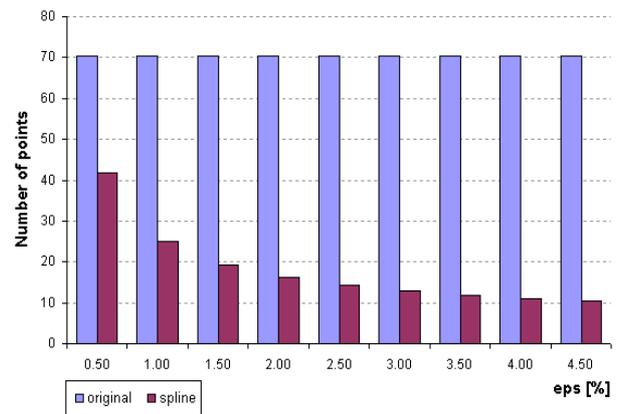**Figure 8.** Data reduction as reduction rate (%) versus the filtering threshold value *eps*.

Figure 9. Data reduction as absolute number of points (original and spline) versus the filtering threshold value *eps*.

VI. CONCLUSIONS

We presented a simple procedure for modeling planar shapes with cubic polynomial splines. The procedure is easy to implement and provides good results in practice as shown by computing the approximation error and data reduction rate on a large set of samples. We only discussed the 2D case but, apart from the examples illustrated, there is no actual limitation of the equations or algorithms to the two-dimensional plane. The same ideas may be directly applied to 3D modeling.

We hope that (novice) researchers working in pattern recognition dealing with shape recognition and analysis will find the representation procedure useful due to the many advantages it brings (probably the most important one for practitioners is the affine invariance of the representation) yet its simplicity in implementation.

APPENDIX A – PSEUDOCODE

We list the pseudo code procedures for the various algorithms described in this paper. We do this in order to demonstrate the simplicity of the technique. In the following procedures, *P* is an array of points indexed from *0* to *length[P]*, *eps* is a real-valued constant and *r* and *t* are arrays of real numbers.

```
DATA-REDUCTION-1(P, eps)
1    for i ß 1, length[P]
2        ▸ Remove point P[i] if it is
3        ▸ too close to its previous neighbor
4        if EUCLID-DIST(P[i],P[i-1]) ≤ eps then
5            P ß P – {i}
```

```
DATA-REDUCTION-2(P, eps)
1   st ß 0
2   end ß 2
3   while end ≤ length[P]
4       totalDist ß 0
5       allPointsUnderDist ß true
6       for i ß st+1, end
7           ▶ Update the total distance
8         dist ß EUCLID-DIST(P[i],P[st]P[end])
9         totalDist ß totalDist + distance
10
11          ▶ Check average distance condition
12          if totalDist / (i- st) > eps then
13              allPointsUnderDist ß false
14              ▶ Exit for
15              i ß end
16
17      if allPointsUnderDist == true then
18          end ß end + 1
19      else
20          ▶ Remove points
21          P ß P - {st+1, st+2, .. end-2}
22          ▶ Increment start and end pointers
23          st ß st + 1
24          end ß start + 2
25
26  ▶ Remove points at the end
27  P ß P - {st+1, st+2, .. end-2}
```

```
DATA-ENHANCEMENT-1(P, N)
1   P' ß nil
2   for i ß 0, length[P] - 1
3       for j ß 1, N
4           t ß (j - 1) / N
5           p' ß (1-t)·P[i] + t·P[i+1]
6           ▶ Add interpolated point to the P'
7           P' ß P' + {p'}
8
9   ▶ Add last point
10  P' ß P' + {P[length[P]}
11  return P'
```

```
COMPUTE-TENSIONS(P)
1   ▶ 1st tension needs to be approximated
2   ▶ because the previous point is n/a
3   ▶ use default value
4   r[0] ß 0.5
5
6   for i ß 0, length[P] - 1
7       distPrev ß EUCLID-DIST(P[i], P[i-1])
8       distNext ß EUCLID-DIST(P[i], P[i+1])
9       r[i] ß distPrev/(distPrev + distNext)
10
11  ▶ last tension needs to be approximated
12  ▶ because the next point is n/a
13  ▶ use default value
14  r[length[P]] ß 0.5
15  return r
```

```
COMPUTE-TANGENTS(P, r)
1   ▶ 1st tangent needs to be approximated
2   ▶ because the previous point is n/a
3   t[0] ß r[0]·(P[1] - P[0])
4
5   for i ß 1, length[P] - 1
6       t[i] ß r[i]·(P[i]-P[i-1]) +
7           r[i+1]·(P[i+1]-P[i])
8
9   ▶ last tangent needs to be approximated
10  ▶ because the previous point is n/a
11  n ß length[P]
12  t[n] ß r[n]·(P[n] - P[n-1])
13  return t
```

```
GENERATE-CATMULL-ROM-SPLINE(P)
1   r ß COMPUTE-TENSIONS(P)
2   t ß COMPUTE-TANGENTS(P, r)
3   Time ß 0
4   for i ß 0, length[P] - 1
5       ▶ Solve the 4-variable system
6       ▶ which has a unique solution
7         c[i][0..3]ß
8   SOLVE(P[i],t[i],P[i+1],t[i+1])
9       ▶ Each segment is defined over [0..1]
10      Time ß Time + 1
    return c, Time
```

```
EVALUATE-SPLINE(splineCoeff, Time, t)
1   if t < 0 or t > Time then
2       return nil
3
4   ▶ Compute the segment index j
5   j ß FLOOR(t)
6   ▶ and the local parameter u
7   u ß t - FLOOR(t)
8   return splineCoeff[j][0] +
9       splineCoeff[j][1]·u +
10      splineCoeff[j][2]·u² +
11      splineCoeff[j][3]·u³
```

REFERENCES

[1] J. Hershberger, J. Snoeyink, "Speeding Up the Douglas-Peucker Line-Simplification Algorithm", Proceedings of the 5th Symposium on Data Handling, pp. 134-143 , 1992.

[2] Carl De Boor, "A Practical Guide to Splines", Springer, 2001.

[3] Phillip J. Barry and Ronald N. Goldman, "A recursive evaluation algorithm for a class of Catmull-Rom splines", SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques, pp. 199-204, ACM Press, New York, 1988.

[4] E. Catmull, R. Rom, D. Knuth, "A class of local interpolating splines", Computer Aided Geometric Design (Eds: R. E. Barnhill and R. F. Reisenfeld), pp. 317-326, Academic Press, New York, 1974.

[5] Michael Unser, "Splines and Wavelets: New Perspectives for Pattern Recognition", Pattern Recognition, pp. 244-248, LNCS 2781/2003.

[6] Sambhunath Biswas, Brian C. Lovell, "Bézier and Splines in Image Processing and Machine Vision", Springer, 2008.

[7] Daniel P. Huttenlocher, Klara Kedem, "Computing the minimum Hausdorff distance for point sets under translation", Proceedings of the sixth annual symposium on Computational geometry, pp. 340-349, ACM Press, 1990.

[8] J.O. Wobbrock, A.D. Wilson, Y. Li, "Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes", Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '07). Newport, Rhode Island (October 7-10, 2007). New York: ACM Press, pp. 159-168.

[9] Radu-Daniel Vatavu, Ştefan-Gheorghe Pentiuc, Christophe Chaillou, Laurent Grisoni, Samuel Degrande, Visual Recognition of Hand Postures for Interacting with Virtual Environments, Advances in Electrical and Computer Engineering, Volume 6 (13), Number 2(26), 2006, University „Stefan cel Mare" of Suceava, ISSN 1582-7445,pp. 55-58.

[10] Radu-Daniel Vatavu, Ştefan Gheorghe Pentiuc, Applied Hands-On Interfaces: Enhancing Object Manipulation inside Virtual Environments, European Conference on the Use of Modern Information and Communication Technologies, ECUMICT 2008, Gent, Belgium, March 13-14, 2008.

[11] George MAHALU, Radu PENTIUC, Acquisition and Processing System for the Photometry Parameters of The Bright Objects, Advances in Electrical and Computer Engineering, Suceava, Romania, ISSN 1582-7445, No 1/2001, volume 1 (8), pp. 26-31