OBJECT SEGMENTATION WITH REGION GROWING AND PRINCIPAL COMPONENT ANALYSIS

Marco Roggero

Dept. of Georesource and Territory, Politecnico di Torino, C.so Duca degli Abruzzi, Torino, IT – roggero@atlantic.polito.it

KEY WORDS: Principal Component Analysis, Tensorization, Region Growing, Discrete Geometry, Laser Scanning.

ABSTRACT:

The paper considers the problem of object segmentation and shape recognition in discrete noisy data. Two different algorithms combine region growing techniques with principal component analysis. The proposed algorithms are applied to a data set from airborne laser scanners.

INTRODUCTION

The problem of extracting object description from data is common in many scientific applications and technologies. We have begun this research starting from object segmentation and shape detection in airborne laser scanner data, and working on these data we have implemented my algorithms. However we think that some aspects of this work are of general interest, and may be applicable to other situations.
Let's consider the problem of mapping single points into sets of points of similar properties, defining the regions within a points set. The algorithm proposed tries to solve the problem using region growing and principal component analysis.
Region growing is probably less common than edge detection as a low level process, but it is applicable in multi-dimensional cases and in noisy environments. Principal component analysis is also robust to noise, and unable us to define the intrinsic geometric and physic properties of objects.

HIERARCHIC REGION GROWING

This section describes a linking mechanism which is based on local comparison of point properties, with reference to its neighbours. Points should be merged if they are near, not only in sense of euclidean distance, but also in terms of physical or geometrical properties. These properties, or point descriptors, are defined in the following (see "Geometrical descriptors" at *pag. 3*).

Tree structure

The linking algotithm start the aggregation of objects from a randomly extracted point, a seed, that is for example the first non aggregated point in the database index. This point $\boldsymbol{p_1}$ is placed in the *level 0* of a tree stucture, as shown in *fig. 1*, where each point is marked by its database pointer. Then, it is necessary to search the neighbourhood. Neighbourhood is intended now in an Euclidean sense, and this neighbourhood is the same used to calculate the geometric properties of the object at point $\boldsymbol{p_1}$.

The points in the neighbourhood that match the aggregation criteria, are now aggregated to the object and placed at *level 1* in tree structure, as a new branch of the tree. Then the linking algorithm continues the aggregation, starting from the points in the new level, and so on to the terminal branches.
Terminal branches are determined by two events. The first, is when a point is marked as terminal (grey in the figure) if in its neighbourhood there aren't other non aggregated points. The second event is when the points in the neighbourhood don't match the aggregation criteria.
The algorithm stops to aggregate the object when all the branches reach a terminal point; then it starts to aggregate a new object from a new random seed.

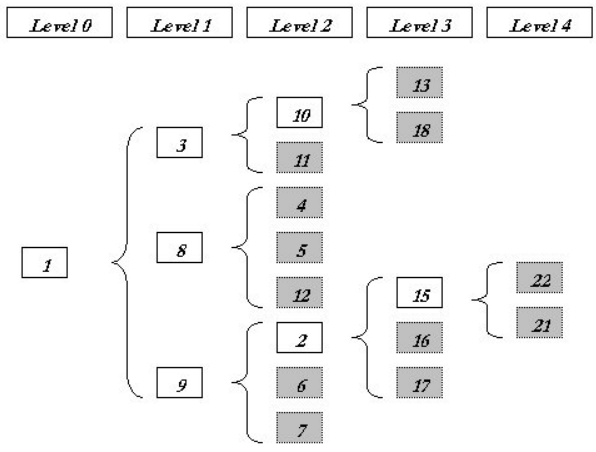

Figure 1 – Tree structure. Points in the structure are marked by a database pointer. Point 1 is the seed; grey points are terminal.
Summarizing:

- An initial set of points are iteratively merged according to aggregation criteria.
- An arbitrary *seed point* is choosen and compared with neighbouring points.
- A region is *grown* from the seed point by adding neighbouring points that match aggregation criteria.

- When the growth of one region stops we simply choose another seed point which does not yet belong to any region and start again.
- This whole process is continued until all points belong to some region.

Memory usage and time of work

Programming a tree structure is very simple, if is not necessary to keep all levels in the memory. In fact to aggregate the *level n* we need to know *level n-1* only. These two levels can be placed in a two dimensional array:

$$\begin{bmatrix} Level(n-1) \\ Level(n) \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & \dots & a_p \\ b_1 & b_2 & \dots & b_q \end{bmatrix}.$$

When the aggregation of the level n stops, row 1 of the array is set equal to row 2, and row 2 is set equal to 0. Then, the aggregation continues from the points in row 1.
Note that using this logic the algorithm works using little memory, even if the data set is very large. Moreover, each point is processed once, except the points near discontinuities. So, time of work is $\vartheta(n)$.

Comments

Region growing can have several undesirable effects. Current region dominates the growth process, and ambiguities around edges of adjacent regions may not be resolved correctly. Different choices of seeds may give different segmentation results, and problems can occur if the (arbitrarily chosen) seed point lies on a discontinuity.
To counter the above problems, *simultaneous region growing* techniques have been developed, but they don't exist if the aggregation criteria are invariant in the direction of growing. A simple example of an invariant criterion uses euclidean distance: point $\boldsymbol{p_m}$ is aggregated to point $\boldsymbol{p_n}$ only if the distance $d_{mn} = \|p_m - p_n\|$ is less than a threshold value $\boldsymbol{r_{max}}$. It is clear that $d_{mn} = d_{nm}$, and this criterion can be defined as invariant.
The proposed region growing algorithm performs a region–based segmentation. May it be interesting to use the same aggregation criteria in an edge–based segmentation algorithm and compare the results. The difference between the two ways is that the region–based segmentation directly constructs the regions, and the edge–based segmentation first detects borders between regions. Segmentations resulting from edge-based methods and region growing methods are not usually exactly the same.

PRINCIPAL COMPONENT ANALYSIS

Principal component analysis (PCA) can be used to analyze the structure of a data set. PCA finds basis vectors for a (sub)space which:

- maximise the variance retained in the projected data
- or (equivalently) give uncorrelated projected distributions

This description will not go into too much detail on PCA, as there is a large body of literature on it. More attention will be paid to some aspects involved in region growing logic. Formulas and examples are often referred to the three dimensional case, but it is easy to extend them to n-dimensional cases.

Moments and tensor of inertia

Properties of objects are described by discrete measures in many real cases. Every measure could be seen as a point in a n-dimensional space, and the distribution of these points (measures) is related to the properties of the real object. We can think of the geometrical properties of an object in a 2D or 3D space, but this logic can be extended to other properties (physical for example) in a nD space. A point (measure) in the 3D space can assume one of the three following roles:

- surface patch or volume element
- discontinuity (surface, curve or point junctions)
- outlier

Let's consider the two extreme cases, a point on a smooth surface with a well defined surface (normal) orientation, and a point on a (curve or point) junction whose absolute orientation is uncertain. This whole continuum can be abstracted as a second order symmetric 3D tensor, which can be visualized geometrically as an ellipsoid. Such an ellipsoid can be fully described by the corresponding eigensystem with its three unit eigenvectors, $\hat{V}_{\max}$, $\hat{V}_{mid}$ and $\hat{V}_{\min}$ and the three corresponding eigenvalues $\lambda_{\max} \geq \lambda_{mid} \geq \lambda_{\min}$.
The second order 3D tensor is the symmetric matrix:

$$[I] = \begin{bmatrix} I_{XX} & I_{YX} & I_{ZX} \\ I_{XY} & I_{YY} & I_{ZY} \\ I_{XZ} & I_{YZ} & I_{ZZ} \end{bmatrix}$$

This tensor is called inertia tensor if a mass is associated to each point of the data set. The elements of $\boldsymbol{I}$ are the second order moments:

$$I_{xx} = \sum m_i \cdot \left[(y_i - y_G)^2 + (z_i - z_G)^2 \right]$$
$$I_{yy} = \sum m_i \cdot \left[(x_i - x_G)^2 + (z_i - z_G)^2 \right]$$
$$I_{zz} = \sum m_i \cdot \left[(x_i - x_G)^2 + (y_i - y_G)^2 \right]$$
$$I_{xy} = \sum m_i \cdot \left[(x_i - x_G) \cdot (y_i - y_G) \right]$$
$$I_{xz} = \sum m_i \cdot \left[(x_i - x_G) \cdot (z_i - z_G) \right]$$
$$I_{yz} = \sum m_i \cdot \left[(y_i - y_G) \cdot (z_i - z_G) \right]$$

where $\boldsymbol{m_i}$ is the mass associated to point i, $\boldsymbol{x_i}$, $\boldsymbol{y_i}$ and $\boldsymbol{z_i}$ are the coordinates of point i and $\boldsymbol{x_G}$, $\boldsymbol{y_G}$ and $\boldsymbol{z_G}$ are the coordinates of the centre of mass of the points set.
The symmetry of the tensor guarantees that all of $\boldsymbol{I}$'s eigenvalues are real and that there is an orthonormal basis of eigenvectors (principal system), that is:

$$[V] = \begin{bmatrix} v_{1X} & v_{2X} & v_{3X} \\ v_{1Y} & v_{2Y} & v_{3Y} \\ v_{1Z} & v_{2Z} & v_{3Z} \end{bmatrix}$$

where $\bar{v}_1, \bar{v}_2, \bar{v}_3$ are the eigenvectors of $\boldsymbol{I}$. $\boldsymbol{V}$ is also the transition matrix between the canonic and principal system. The coordinates in the principal system are:

$$\begin{bmatrix} \xi \\ \eta \\ \vartheta \end{bmatrix} = V^{-1} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

The principal moments of inertia $\lfloor I_\xi \quad I_\eta \quad I_\vartheta \rfloor$ are the three real eigenvalues of $\boldsymbol{I}$.

Geometrical descriptors

The geometrical properties of an object in a point $\boldsymbol{p_i}$ can be described intrinsically defining an inertial system centered in this point. We mustn't forget that we need to know those properties or descriptors that simplify object segmentation. In order to reach this result, we note that is desirable that these descriptors are singular in discontinuities or, in other words, that they have a peak value. We can't consider sign change of the descriptors, because of the presence of noise in data.
In the following are described three different descriptors, static moment, curvature and junction, but it is possible to define a non-limited number of properties, increasing the dimensions of the space of descriptors. So, the dimension of the problem is not only the dimension of the space of the objects $\boldsymbol{O^m}$, but the dimension of the union of this space with the space of descriptors $\boldsymbol{D^n}$:

$$S^{m+n} \equiv O^m \cup D^n$$

Static moment: the module of the first order moment (or static moment) is maximum at the edge and, in a smaller measure, at a change of curvature. Static moments are defined as:

$$S_x = \sum m_i (y_i + z_i)$$
$$S_y = \sum m_i (x_i + z_i)$$
$$S_z = \sum m_i (x_i + y_i)$$

The three static moments can be combined in a comprensive descriptor that is the total static moment:

$$S_t = \left| S_x \cdot S_y \cdot S_z \right|$$

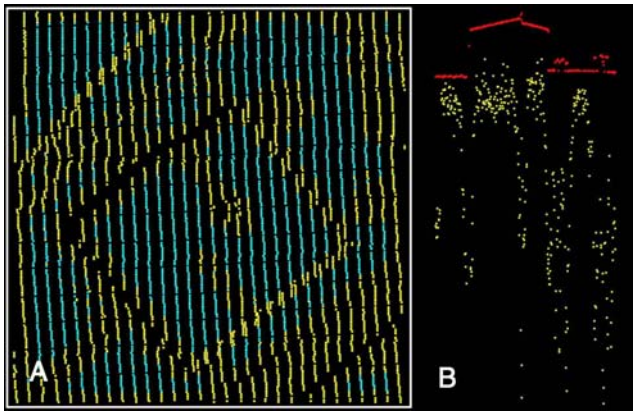

Figure 2 – Total static moments in an airborne laser scanning data set by TopoSys. In (A) the most elevated values of static moments are in yellow. In (B) the red points represent a profile in the data set, the yellow points the static moment (with sign changed). Note the peak value in discontinuities.

Curvature: in discrete geometry it is possible to define curvature in many ways. Fortunately we don't need the exact value of curvature; a functional one is sufficient, beacuse we are looking for the location of peak values, not for their magnitude. In the hypothesis that the point set is a surface, the ratio

$$\tilde{\rho} = \frac{\lambda_{\max}}{\lambda_{\min}}$$

is a functional of the ray of curvature. In fact $\lambda_{\min}$ is null for a planar surface and $\lambda_{\max}$ is not null; that leads to an infinite value for ρ. Increasing curvature, $\lambda_{\min}$ increases and $\lambda_{\max}$ is constant, while $\lambda_{\min}$ is less than $\lambda_{\max}$. When $\lambda_{\min}$ becomes equal to $\lambda_{\max}$, the two eigenvalues swap, $\lambda_{\min}$ becomes constant and $\lambda_{\max}$ increases with curvature. A functional of curvature is the ratio

$$\tilde{\kappa} = \frac{\lambda_{\min}}{\lambda_{\max}}$$

Junction: the eigenvalues encode the magnitudes of orientation (un)certainties, since they indicate the size of the corresponding ellipsoid. At curve or point junctions, where intersecting surface are present, there is no preferred orientation and the eigenvalue $\lambda_{\min}$ has peak values. A junction map that represents the location of the peak value for $\lambda_{\min}$ is very similar to the static moments map of the figure 2.

Data distribution anisotropy

Distribution anisotropy is a problem present in point cloud data, such as range data, and leads to a classification of the scanning shape as a real shape. PCA simplify anisotropic data set processing: principal components are referred to a spheric neighbourhood, but it is possible to refer aggregation criteria to an ellipsoidic neighbourhood, whose semiaxes are defined by RMS coordinates:

$$e_{\max} = r\frac{\sigma_{\max}}{\sigma_{\max}};\quad e_{mid} = r\frac{\sigma_{mid}}{\sigma_{\max}};\quad e_{\min} = r\frac{\sigma_{\min}}{\sigma_{\max}}$$

where $\sigma_{\min}$, σ_{mid} and $\sigma_{\max}$ are the RMS in the directions of $\hat{V}_{\min}$, $\hat{V}_{mid}$ and $\hat{V}_{\max}$, and $\boldsymbol{r}$ is the radius of the spheric neighbourhood.
With reference to figure 3, that represents a laser scanner anisotropic data set, we note that the sampling density in the first principal direction (~7 points/m) is greater than in the second principal direction. (~1 points/m). Using the ellipsoidic neighborhood in aggregation, we can take advantage of the better definition of the measures in the first principal direction.

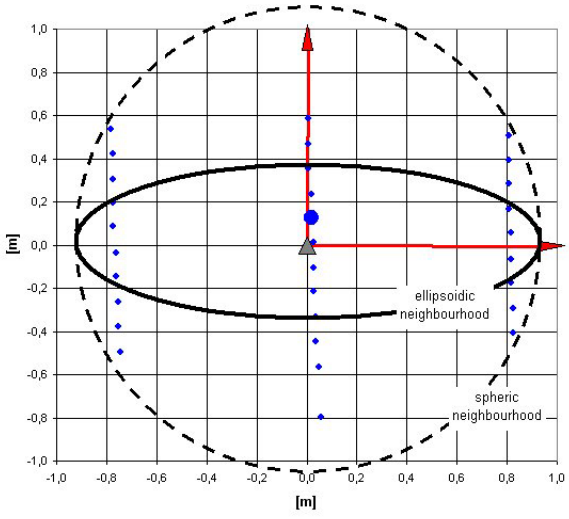

Figure 3 – Normalized ellipsoidic neighborood in anisotropic case; the arrows rapresent the direction of the principal axis (● data points, ▲ centre of mass). The point set is from a TopoSys I airborne laser scanner.

An n-dimensional case

In laser scanning applications, we find a simple n-dimensional example. Some laser scanners measure not only the range, but even radiance. This data can be considered in PCA as a 4-th dimension, and used by aggregation criteria exactly as the first three dimensions. One can find a lot of other examples in image processing.

ALGORITHMS

In this section, two different algorithms that combine region growing techniques and principal component analysis are presented. PCA is used to define the aggregation criteria and to describe the geometrical properties of the objects.
The strategy implemented for object detection and segmentation is based on range data only, and was tested on an airborne laser scanner data set. The result was obtained by working on raw data, so we were able to take advantage of the full resolution potential of laser scanning.
The two algorithms differ in PCA and in aggregation criteria, but they use the same hierarchic region growing technique.

Algorithm 1

This algorithm is based on descriptor mapping; one or more properties are calculated and mapped for every point in the data set. Then the algorithm perform the region growing with reference to the property map, and mark as terminal point those points in which is located a peak value. Is also possible to map different properties and compare the results. The algorithm performs the following steps:

A. Calculates the descriptor for every point $\boldsymbol{p_i}$ in the data set (descriptor mapping):
   1. at the point $\boldsymbol{p_i}$ searches a neighbourhood for other points;
   2. calculates the second order moments using the neighbouring points;
   3. writes the second order symmetric tensor at the point $\boldsymbol{p_i}$;
   4. calculates the three real eigenvalues and the related eigenvectors;
   5. calculates the descriptor value (for example total static moment) at the point $\boldsymbol{p_i}$ in the reference system defined by the three eigenvectors;
   6. writes the descriptor on file.

B. Starts region growing:
   1. extracts a seed and places it in level 1 of the tree structure;
   2. searches a spherical neighbourhood for other points;
   3. sorts neighbouring points in function of their Euclidean distance from the seed;
   4. searches the distance from seed of the descriptor peak value, and sets $\boldsymbol{d_{max}}$ equal to this distance;
   5. marks peak value as terminal point of the tree structure;
   6. aggregates the points with distance from the seed less than $\boldsymbol{d_{max}}$ and places them in level 2 of the tree;
   7. grows the region by adding neighbouring points that match aggregation criteria until all the branches reach a terminal point;
   8. when the growth of the first region stops chooses another seed point which does not yet belong to any region and starts again.

Let's consider the phase A, in wich the algorithm performs the descriptor mapping. Note that the tensor is centrated in the point $\boldsymbol{p_i}$, not in the centre of mass! This is beacause we need to calculate the value of descriptor at the point $\boldsymbol{p_i}$.

The phases B.3-6 are typical of this algorithm and solve in a simple way, using the non-parametric threshold value $\boldsymbol{d_{max}}$, the problem of stop growing at discontinuities.
Time of work is $\vartheta(2n)$, that is $\vartheta_A(n)$ for descriptor mapping plus $\vartheta_B(n)$ for region growing.

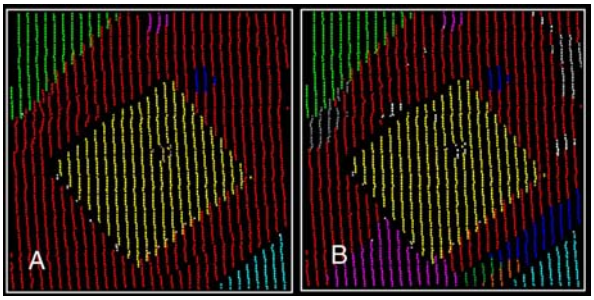

Figure 4 – Segmentation results on a test area of 25·25 m. In (A) region growing was performed with a distance criterion only. In (B) was used the algorithm 1 and the total static moment as descriptor. TopoSys data set.

Algorithm 2

This algorithm uses PCA only in region growing aggregation criteria and don't performs descriptor mapping, so it is faster than algorithm 1. The tensor is centred in centre of mass and in this reference system a nucleus is defined. The aggregation criteria are based on this nucleus and different definitions of the nucleus are possible. The algorithm performs the following steps:

A. Starts region growing:
   1. extracts a seed $\boldsymbol{p_i}$ and places it in level 1 of the tree structure;
   2. searches a spherical neighbourhood for other points;
   3. calculates the centre of mass using $\boldsymbol{p_i}$ and the neighbouring points;
   4. calculates the second order moments with reference to the centre of mass, using $\boldsymbol{p_i}$ and the neighbouring points;
   5. writes the second order symmetric tensor centred in the centre of mass;
   6. calculates the three real eigenvalues and the related eigenvectors;
   7. transforms the coordinates of $\boldsymbol{p_i}$ and of the neighbouring points in the principal reference system;
   8. calculates the RMS coordinates in the principal reference system;
   9. defines a normalized elliptical nucleus (see at "Data distribution anisotropy");
   10. if $\boldsymbol{p_i}$ is internal at nucleus, aggregates $\boldsymbol{p_i}$ and the neighbouring points in the nucleus; else marks $\boldsymbol{p_i}$ as a terminal point of the tree structure.

In the phase A.9 different definitions of the nucleus are possible. Time of work of this algorithm is $\vartheta(n)$.

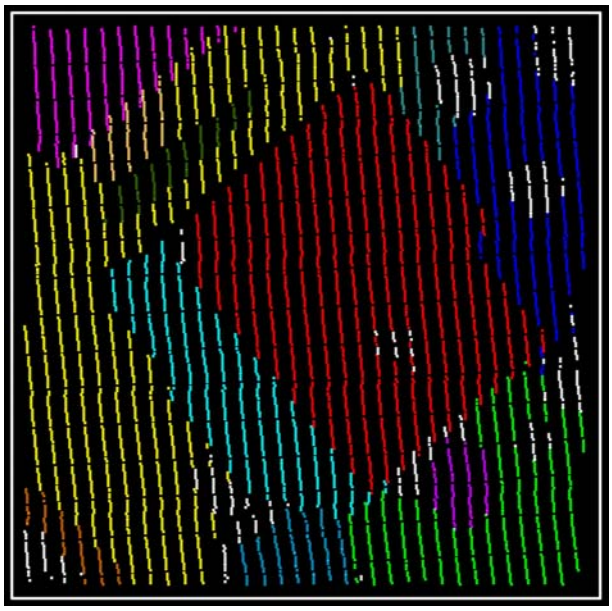

Figure 5 – Segmentation results obtained by algorithm 2 on a test area of 27·27 m. TopoSys data set.

TESTS

A 5-dimensional case

A series of tests in the n-dimesional case was performed using a laser data set provided by FOTONOR AS for the ISPRS test on estracting DEMs. The test area was scanned with an Optech laser scanner, and both first and last pulse data were recorded. The sampling density is of 1.8 points per square meter. The scanner recorded the received pulse intensity also, and we have used this data with descriptor mapping to improve the segmentation results. In this test we have mapped the total static moment only. The total dimension of the problem is:

$$S^5 \equiv O^4 \cup D^1$$

and the best robustness reached using five dimensions, supplies the low sampling density.

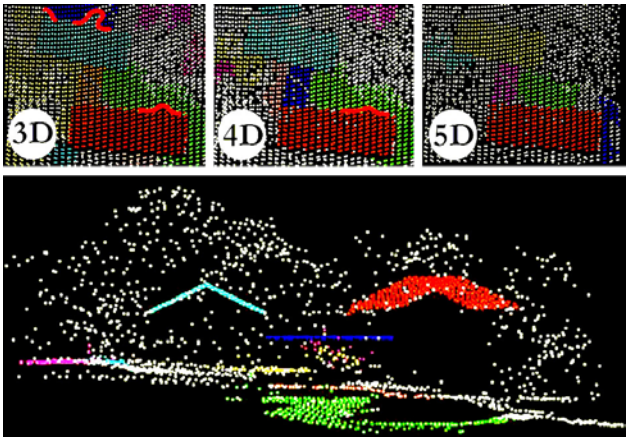

Figure 6 – Segmentation results obtained on a test area of 64·52 m working in three, four and five dimensions. The red line marks the junctions and the edges in which the algorithm fails, because a tree touch the roof. The entities with less than 50 points are in white.

Let's consider three cases.
Case 1: we have segmented a test area of 64·52 m, working at first in the three dimensions X, Y and Z only and using the algorithm 2. The algorithm fails in the junctions between different entities.
Case 2: working in the four dimensions of the space of the objects (X, Y, Z and radiometry) the algorithm 2 don't fails in the junctions.
Case 3: we have used the algorithm 1 and the total static moment as descriptor, working in five dimensions. The algorithm don't fails in the junctions and the result is more precise on the edges.
The results are in figure 6.

Test on a large data set.

We have performed a test on a large data set (about 500000 points) of laser measuraments, scanned on urban area. We have used algorithm 1 working in five dimensions, as in the case 3 of the previous test, and we want to look at the time of work of the algorithm. The processor used is an AMD Athlon with CPU clock of 900 MHz. The result of the test is satisfying, for the segmentation output (figure 7) and for the short time of work (only about ten minutes!).

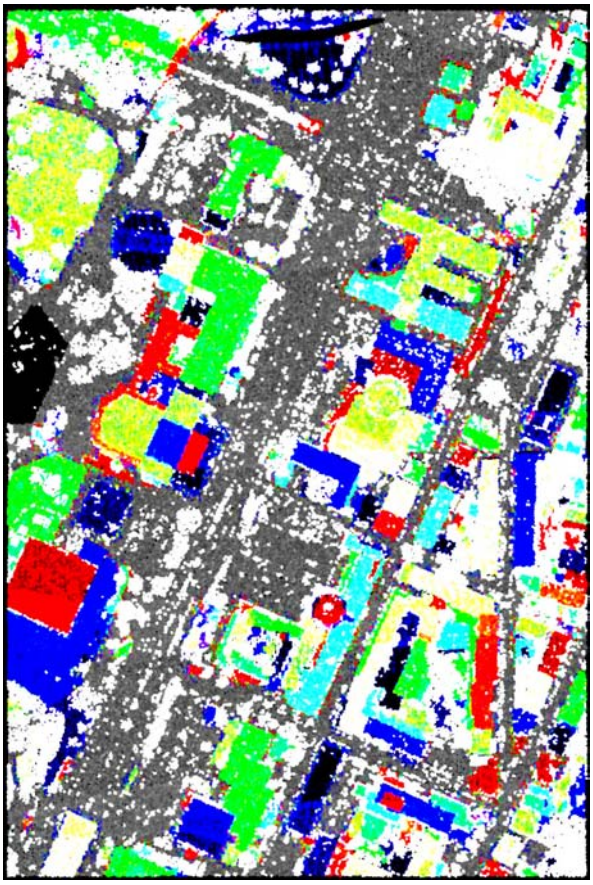

Figure 7 – Segmentation results obtained on a large data set, covering an urban area of 422·634 m. Note ground, gruped in a unique entity, drawed in gray. The entities with less than 50 points are in white.

CONCLUSIONS

The proposed algorithms combine efficiently the region growing techniques with principal component analysis. They are fast and also robust in noisy environments, and their robustness increase with increasing the number of dimensions. Many aspects are not yet explored, such as the applications in image processing. The definitions of geometrical properties of objects need to be studied better in discrete and noisy case, and applied in aggregation criteria.
The algorithms have given good and useful results on laser scanning data, preparing the way to vectorialization, aspect that is very important for cartographic production.

APPENDIX

NOTES ON ALGORITHM IMPLEMENTATION

Eigensystem analysis

Eigensystem analysis is performed using EVCRG and GVLSP routines of IMSL Math Library.

- Routine EVCRG computes the eigenvalues and eigenvectors of a real matrix. The matrix is first balanced. Orthogonal similarity transformations are used to reduce the balanced matrix to a real upper Hessenberg matrix. The implicit double-shifted QR algorithm is used to compute the eigenvalues and eigenvectors of this Hessenberg matrix. The eigenvectors are normalized such that each has Euclidean length of value one. The largest component is real and positive. The routines used in EVCRG are based on the EISPACK library.
- Routine GVLSP computes the eigenvalues and eigenvectors of Az = lBz, with A symmetric and B symmetric positive definite. The Cholesky factorization B = RTR, with R a triangular matrix, is used to transform the equation Az = lBz, to

  (R-T AR-1)(Rz) = l (Rz)

  The eigenvalues and eigenvectors of C = R-T AR-1 are then computed. The generalized eigenvectors of A are given by z = R-1 x, where x is an eigenvector of C.

Sorting

Sorting is performed using SVRGP and SVRGN routines of IMSL Math Library.

- Routine SVRGP sorts the elements of an array, A, into ascending order by algebraic value, keeping a record in P of the permutations to the array A. The routine SVRGP uses the algorithm discussed in SVRGN.
- Routine SVRGN sorts the elements of an array, A, into ascending order by algebraic value. The array A is divided into two parts by picking a central element T of the array. The first and last elements of A are compared with T and exchanged until the three values appear in the array in ascending order. The elements of the array are rearranged until all elements greater than or equal to the central element appear in the second part of the array and all those less than or equal to the central element appear in the first part. The upper and lower subscripts of one of the segments are saved, and the process continues iteratively on the other segment. When one segment is finally sorted, the process begins again by retrieving the subscripts of another unsorted portion of the array.

REFERENCES

Curvature estimation

S. Pulla, A. Razdan and G. Farin (2001), Improved curvature estimation for watershed segmentation of 3-dimensional meshes.

P. Kresek, G. Lukács and R. R. Martin, Algorithms for computing curvatures from range data.

C. K. Tang and G. Medioni, Robust estimation of curvature information from noisy 3D data for shape description.

Eigensystem analysis

G. H. Golub and C. F. Van Loan (1996), Matrix computations, The John Hopkins University Press, Baltimore and London.

Hanson, Richard J., R. Lehoucq, J. Stolle, and A. Belmonte (1990), Improved performance of certain matrix eigenvalue computations for the IMSL/MATH Library, IMSL Technical Report 9007, IMSL, Houston.

Martin, R.S., and J.W. Wilkinson (1968), Reduction of the symmetric eigenproblem Ax = lBx and related problems to standard form, Numerische Mathematik, 11, 99-119.

Smith, B.T., J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema, and C.B. Moler (1976), Matrix Eigensystem Routines - EISPACK Guide, Springer-Verlag, New York.

Sorting

Griffin, R., and K.A. Redish (1970), Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, Communications of the ACM, 13, 54.

Petro, R. (1970), Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, Communications of the ACM, 13, 624.

Singleton, R.C. (1969), Algorithm 347: An efficient algorithm for sorting with minimal storage, Communications of the ACM, 12, 185-187.