Object reconstruction and 3D mosaicing

Author

(UNIVR)
U. Castellani, A.Fusiello, V.Murino

(DISI)
L.Papaleo, E.Puppo, M.Pittore

Date: Friday, 02 July 2004

Outline

Introduction: WP5 Objectives

Develop and establish methods for: 3D object reconstruction from sensor data acquired at either a single location, or multiple locations (mosaicing); multi-resolution representation of reconstructed objects.

3D reconstruction occurs through a sequence of operations that process input range maps to build the mosaic mesh. The following software components are present both in the on-line and in the off-line version, possibly with different characteristics and implementation on the two versions:

1. **Data thresholding:** raw data in input are filtered on the basis of sensor intensity values to discard those data that appear to be spurious;
2. **Single frame reconstruction:** a mesh of triangles is built for each frame, which interpolates range data from that frame only;
3. **Mesh filtering:** each single frame mesh is filtered by eliminating small connected components;
4. **Registration:** meshes from different frames are registered in the same coordinate system;
5. **Mosaicing:** registered meshes are used to compose the mosaic mesh.

The 3D object reconstruction software comes in two versions: *on-line* and *off-line.*

The on-line version is intended for use during ROV operation to provide feedback to the pilot on the scene spanned by sensors. In this version, stress is on fast methods that can support real-time operation.

The off-line version is intended for use during surveying sessions on data collected previously. In this case, stress is on accurate methods that can provide high quality models.

The on-line version of reconstruction consists of a strictly sequential execution of the above five components. These components might be pipelined in a multi-processor architecture. However, in the rest of this document, we will refer to a single processor implementation that communicates with the Data Acquisition and Pre-processing module and the POLECAT System through a software interface as specified in [1] . The main difference between the on-line and the off-line version is in the registration task:

- **in the on-line version registration is made only pairwise** (one frame with respect to the previous frame);
- **in the off-line version, registration is made globally** (each frame with respect to all other frames that overlap with it).

In the off-line version, a preliminary pairwise registration and data fusion (which is part of the mosaicing task) is necessary in order to provide a map of overlap among different frames, which is used next for global registration. Therefore, the off-line version consists of a sequential execution of steps: 1, 2, 3, 4a (pairwise), 5a (partial), 4b (global), 5b (global). Mosaicing is indeed obtained by modifying the on-line version and integrating it with the global registration software developed to this specific purpose.

In the following two sections, we describe separately the architectures of the on-line and off-line versions and related components, focusing also data exchange with other software modules, data exchange among different components, as well as parameters that need to be set by the user (through GUI).

Multi-resolution representation is intended for off-line use. Its main purpose is to speed-up the visualisation of large meshes (e.g., sea beds or mosaics build during previous survey sessions) by concentrating the level of detail only where it is necessary, and discarding detail elsewhere. The multi-resolution software consists of: one construction module, which builds the multi-resolution model; and a library to build applications that query the multi-resolution model and obtain meshes at different level of detail for their purposes. One example is a viewer that queries the multi-resolution model at each frame to obtain meshes clipped at the view frustum and with a resolution decreasing with distance from the viewpoint. The multi-resolution software is described in Section 3.

1 Online Reconstruction and Mosaicing

1.1 ARCHITECTURE

The aim of this section is to describe in details the ARROV on-line data processing scheme. The main goal of this data processing is to construct a model of the underwater environment and to track the ROV's motion or pose based on the processing of acoustic images, optical images and the pose information from the motion sensors.

The program for on-line 3D reconstruction receives in input a sequence of frames and produces a corresponding sequence of mesh updates (Figure 1). The program is endowed with a GUI that allows the user to set a number of parameters used during processing. Note that the complete mosaic, built on all meshes received, is maintained as a state of the program, rather than given in output. The GUI allows the user to download the whole mosaic on disk by either interrupting or terminating on-line operations

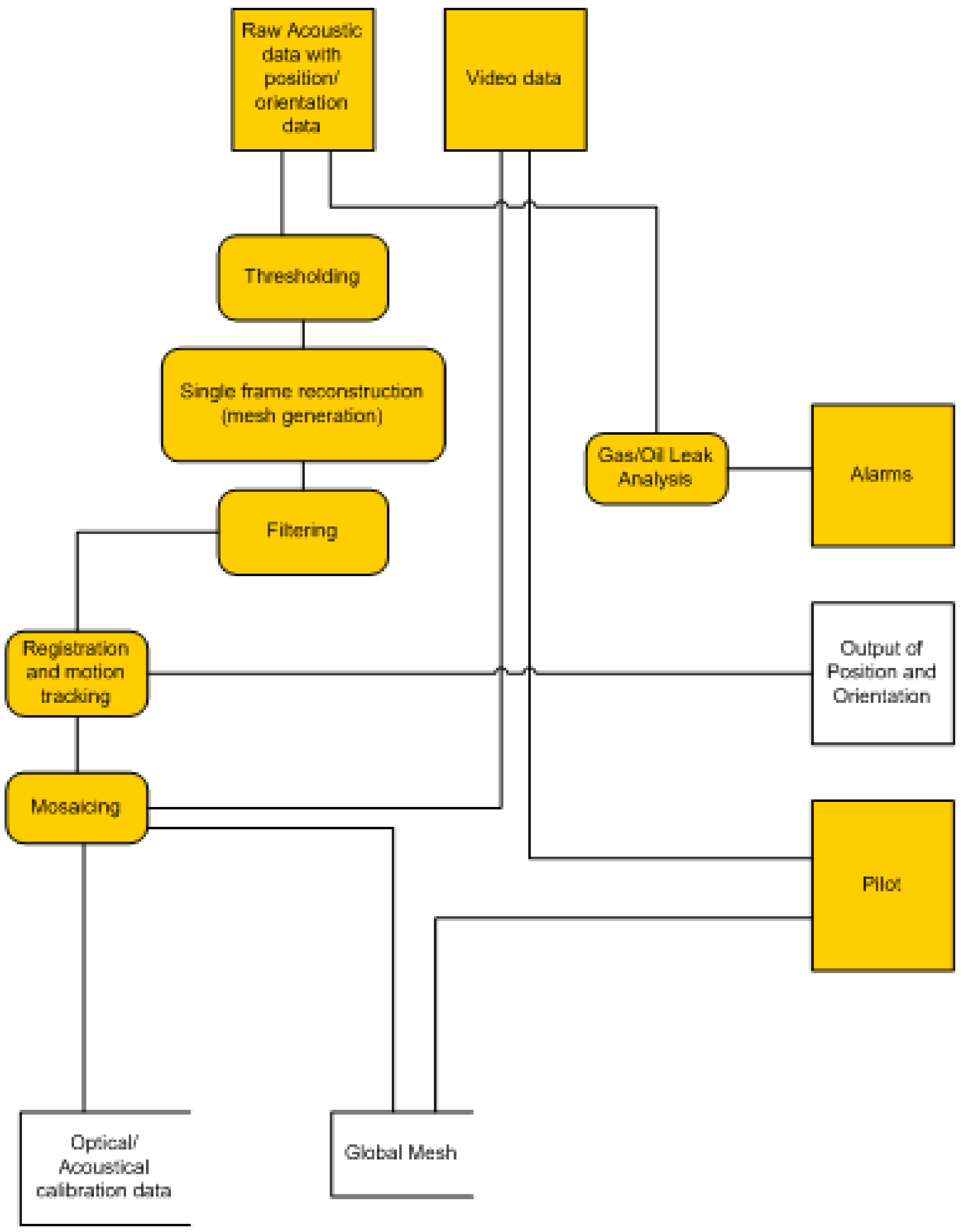

Raw Acoustic data with position/ orientation data
Video data
Thresholding
Single frame reconstruction (mesh generation)
Gas/Oil Leak Analysis
Alarms
Filtering
Registration and motion tracking
Output of Position and Orientation
Mosaicing
Pilot
Optical/ Acoustical calibration data
Global Mesh

Figure 1 - Data Processing Scheme

1.1.1 Input

A *frame* consists of the following data:

- A **time stamp**, in the form of a non-negative integer value
- A **range map**, in the form of two bi-dimensional arrays of size 64×64:
    - Each entry in the first array contains a **3D point** in the form of three Cartesian coordinates *x, y, z* (each being a signed integer on 16 bits) referring to a right-handed coordinate system centered at the sensor, with axes oriented as follows:
        - *x* – pointing from the sensor face and out, perpendicular to the face of the receiver array;
        - *y* – horizontal, pointing to the left when looking in the viewing direction;
        - *z* – vertical, pointing upward.
    - Each entry in the second array contains an **intensity value** in the form of a non-negative integer value (unsigned integer on 8 bits – optionally this may be extended to 16 bits).
- **Pre-registration information**, in the form of a 4×4 matrix of real (double) values. The matrix represents a rototranslation in homogeneous coordinates (actually, only the first three rows of the matrix are necessary, the fourth row being fixed). A point from the range map, when multiplied by this matrix, becomes represented in a *global coordinate system.*

In order to reduce the size of data exchanged between the *Data Acquisition and Pre-processing module* and the other modules, the possibility will be considered to compress the range map prior to transmission, and decompress it prior to local storage, by the following mechanism:

- The array of intensities is run-length encoded. This kind of compression is likely to be effective since this array contains many zero values. Intensity values are transmitted first.
- Only points corresponding to non-zero intensity values are transmitted. Delta encoding of point coordinates may be also considered.

A statistical analysis of time saved during transmission vs additional time spent for compression/decompression is necessary to estimate the advantages of this mechanism.
Compression/decompression will be managed by interface methods provided by the *Data Acquisition and Pre-processing module.*

1.1.2 Output

A *mesh update* consists of the following data:

- A **time stamp**, same as in input
- A **list of meshes**. Each element of the list is a **mesh**, which consists in turn of the following data:
    - A **name**, in the form of a positive integer number. We may initially assume that just one new mesh is generated at each frame: in this case the name of each mesh is the time stamp of the frame that generated it first.
    - The **number of triangles** in the mesh, in the form of a positive integer number.
    - A **list of triangles**, where each triangle is a **triple of points** and a **point** is, in turn a triple of Cartesian coordinates (*x,y,z*) in a *global coordinate system,* each coordinate in the form of a real number (float).

A list of meshes may possibly be empty, which means that no new parts of mosaic have been generated at the current frame.

- A 4×4 **registration matrix** in homogemeous coordinates (it is sufficient to transmit the first three rows of the matrix), each coordinate in the form of a real number (double, because precision is important in this case), which describes current position and orientation of the sensor in a global coordinate system.

Graphical Viewers of the *POLECAT System* will maintain the current mosaic as a **collection of meshes**, each mesh having the same structure described above. Meshes received by a Viewer at a given frame can be of two kinds:

- **New mesh:** a mesh that was generated in the current frame. In this case, this mesh must be added to the collection held by the Viewer.
- **Modified mesh:** a mesh, which was generated in a previous frame and modified in the current frame. In this case, this mesh must *substitute* the mesh having the same name in the collection held by the Viewer. An empty mesh, i.e., a mesh with zero triangles, means that the mesh with the same name must be *deleted* from the collection held by the Viewer.

A modified mesh can be discriminated from a new mesh based on existence of its name in the collection of meshes that form the current mosaic. If the current mosaic is downloaded to disk, it will be encoded as a single mesh, containing all triangles of the mosaic, and having the same structure described above, except for the name. Global download can be slow, depending on the size of the mosaic, so the system/user should assume to either interrupt, or terminate on-line operations during download.

1.1.3 Parameters

The program needs some parameters to be either specified by the user or automatically set at startup. Parameters have influence on trade-off between speed and accuracy of reconstruction. Some parameters can be set only at startup and remain fixed for a whole session, while others can be adjusted through GUI even during on-line operations.

- **Intensity threshold:** a non-negative integer value in the range [0,255] (optionally, [0,65535] if 16 bits are used for intensity values). This is used during data thresholding: only data points having an intensity value equal or greater than the threshold are considered for further processing.

- **Range difference threshold:** a positive short value. This is used during single frame reconstruction. Only points having a difference in their range distance smaller or equal than the threshold may be considered adjacent in the mesh. This threshold can be calculated automatically based on the distribution of the range difference of all points in the frame.

- **Intensity difference threshold:** a positive short value. This is used during single frame reconstruction. Only points having a difference in their intensity smaller or equal than the threshold may be considered adjacent in the mesh. This threshold may be calculated automatically based on the distribution of the intensity difference of all points in the frame.

- **Level of neighbourhood:** a flag. This is used during single frame reconstruction. If flag is set, triangles are built by considering pairs of points that correspond to array entries at a distance of at most two units in the range map (Level 2); otherwise only pairs of points that correspond to array entries at a distance of at most one unit are considered (Level 1).
- **Minimum size of components:** a positive real value. This is used during filtering. Only connected components of the mesh built on single frame, which have a number of triangles greater or equal than the threshold are considered for further processing.
- **Reliability threshold:** a positive integer value. This is used during on-line registration. It is the minimum number of vertices in the current frame to allow a reliable registration. If that number is below the threshold, the results of registration are not used in motion tracking and the current mesh is not inegrated in the mosaic.
- **Subsample threshold:** a positive integer value. This is used during on-line registration.It is the maximum number of points in the current frame used for registration. If the number of vertices in the current frame exceeds the threshold, the frame randomly subsampled.
- **Maximum number of ICP iterations:** a positive integer value. This is used during on-line registration. If the maximum number of iterations is reached, the ICP algorithm will stop.
- **Minimum residual registration error (RRE)**: a positive real value. This is used during on-line registration. If the minimum residual error is measured, the ICP algorithm will stop.
- **Grid resolution:** a positive real value. This is used during mosaicing. It defines the edge length of square cells in which 3D space is subdivided. In practice, it roughly corresponds to the average length of edges of triangles in the resulting triangulation. This value is fixed during setup and cannot be changed during a session. It is important to trade-off between speed and accuracy.
- **Fusion threshold:** a positive real value. This is used during mosaicing. It defines the minimum distance between two vertices generated by different frames. If two such vertices lie along the same grid line and are at a distance smaller than this threshold, they are merged. This value must be smaller than grid resolution. The choice of fusion threshold should also keep into account the RRE. It is also possible to automatically set fusion threshold on the basis of the RRE value.

1.2 THRESHOLDING

Every acoustical image captured by the acoustical camera Echoscope on board ROV has 64x64 points and each point is described by four numbers: *x*, *y*, *z*, *i*; where *z* represents the range information and *i* is the intensity of the point. A possible way to filter the 3D data is to set up a threshold to the intensity of the points and the points with low intensity are filtered Figure 2. This idea is because the echoes backscattered from objects have stronger response than other cluttering interferences

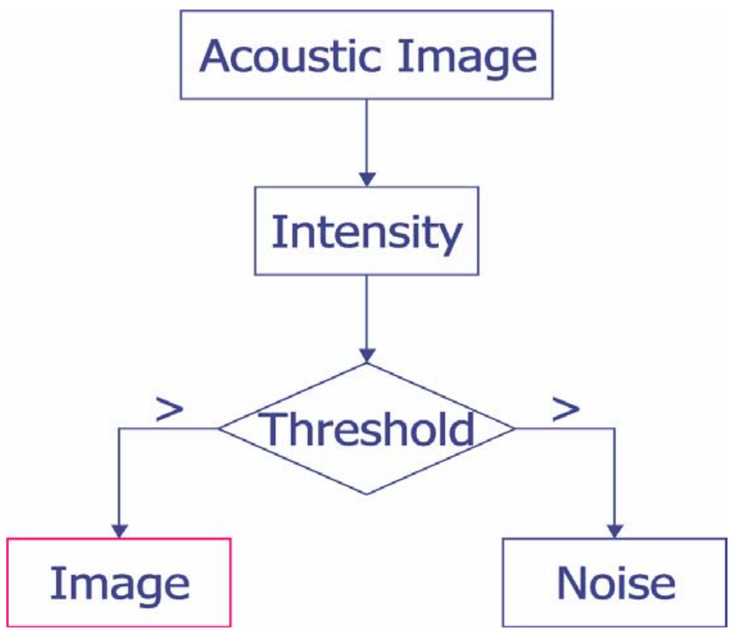


Figure 2 - initial threshold over intensities

Thresholding in ARROV system is performed on the raw 3D point intensities among adjancent points. During the filtering stage, and for each acoustic frame, a threshold is computed, based on a preliminary analysis of the intensity difference distributions among 3D points. Once the distribution (basically a simple histogram) is defined, a threshold is computed that filters out the far queue of the distribution itself.

This threshold is computed as follows:

1. for each point in the input, we compute the intensity difference with respect to its neigbours (*delta matrix construction*) and we create a histogram of the differences;
2. We find the maximum of the histogram through a simple search;
3. We use a user threshold parameter *TH* tuning how far to cut the distribution, e.g. *TH=3%.*
4. Moving from the first bin toward greater values, the integral of the distribution is computed, until it reaches 100-TH percentage of the total integral of the distribution;
5. The bin, at which this condition occurs, is used for computing the maximum difference accepted for the intensities among adjacent points.

Analogously, we compute a parameter threshold on the connectivity distance among 3D points. Note that this procedure relies on the fact the outliers represent noise points and therefore have to be filtered away. Unfortunately, this is true only in some cases, but in some others what seem to be an outlier is instead a *useful point*: for this reason, this filtering procedure is used with care during the whole pipeline.

1.3 SINGLE FRAME RECONSTRUCTION

The next step of the Recostruction Pipeline is the reconstruction of a triangle mesh from a single range image just acquired by the 3D Echoscope. Our input basically consists of (Table 1):

1. a matrix $I[i][j]$ describing a regular lattice (range image), whose entries (i,j) describe either a point in space (x,y,z), or a vacancy,

2. a connectivity threshold (θ), used to evaluate whether or not two nodes in the lattice should be considered adjacent, and
3. the lateral resolution *l* of the sensor.

The output of the method is (Table 1):

1. an adjacency matrix *A*, containing the connection data for each entry in the input frame, and
2. a single frame mesh *M*=(*V*,*F*), where *V* is a set of 3D vertices and *F* is a set of faces. The mesh structure contains also additional information such as *face* and *vertex normals*.

Input		**Output**	
`INPUT MATRIX`	`Vertex I[]`	`ADJACENCY MATRIX`	`Vertex A[]`
`CONNECTIVITY THRESHOLD`	`float t`	`SINGLE FRAME MESH`	`Mesh *M`
`LATERAL RESOLUTION`	`double l`		

Table 1 Input and output of the Single Frame Reconstruction Procedure

The method considers points corresponding to non-vacant entries in the lattice, and it connects them pairwise to form edges: cycles of three edges form triangles of the output mesh. A *potential* edge exists between each pair of points $v \equiv (x_1,y_1,z_1)$ and $w \equiv (x_2,y_2,z_2)$ if their radial distance is below some threshold (θ), depending from the following formula

$$\left| \sqrt{x_1^2 + y_1^2 + z_1^2} - \sqrt{x_2^2 + y_2^2 + z_2^2} \right| < \theta .$$

We implemented an algorithm, which correctly reconstructs a triangle mesh from a range image as input. It can be divided into four main steps:

1. Find connections for non-vacant entries.
2. Recovery vacant entries
3. Improve connections: find feasible semi-diagonals
4. Generate the mesh

In the following, we will go into details of each step, showing results and pseudo-code of each sub-procedure.

1.3.1 Step 1 : Find connections for non-vacant entries

For each *2×2* block $\begin{bmatrix} v_1 & v_2 \\ v_3 & v_4 \end{bmatrix}$ in the lattice where v_1 is a non-vacant entry (Figure 3.left) the procedure tries to find feasible edges. It first tests independently for horizontal and vertical connectivities (basically v_1,v_2 and v_1,v_3), successively it tests for the descending diagonal (v_1,v_4): if it is found the procedure stops otherwise it checks the other diagonal searching for the connection between v_3 and v_2.

Each time an adjacency is found, the procedure updates the adjacency vector.

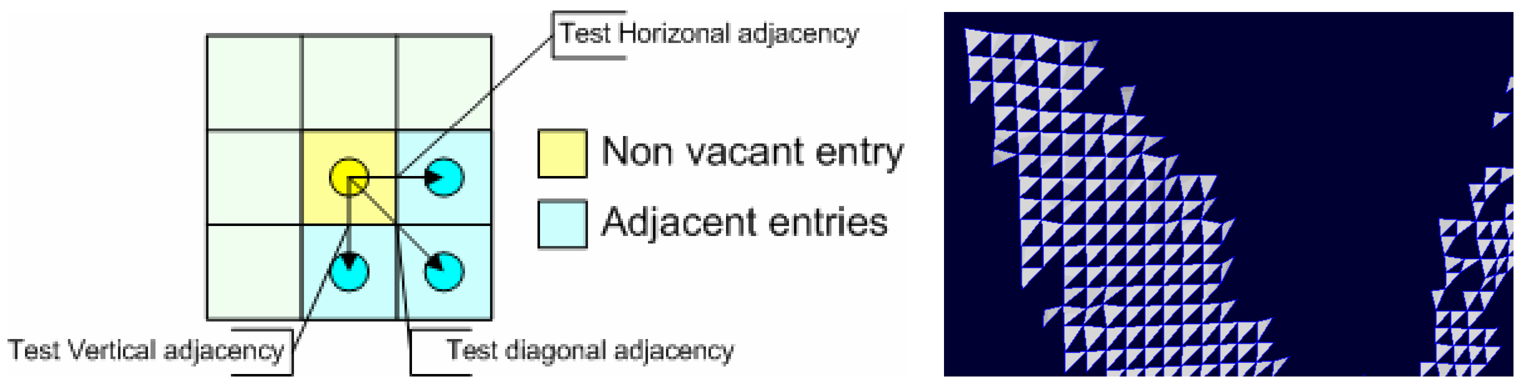


Figure 3 - 2x2 block testing (left) and the result applied on a input range image (right)

```
// add elements to the 2x2 block
for (i=0;i<(m_Height-1)*m_Width-1;i++)
{
 // add a non vacant entry if the intensity of the relative vertex is
 // higher than a given threshold (IntThres) otherwise
 // add a vacant entry (NULL).
 Load(v0,v1,v2,v3,IntThres);
 // Test vertical adjacencies between non-vacant entries
 // if their radial distance is less than the input threshold m_RadThr
 if ((v1!=NULL && v3!=NULL) && (RaDist(v1,v3)< m_RadThr)) {
      // update the Adjacency Matrix for the output
      // update existence and adjacency fields
      UpdateAdjacencyMatrix();
   }
  // Test vertical adjacencies between non-vacant entries
  // if their radial distance is less than the input threshold m_RadThr
  if ((v2!=NULL && v3!=NULL) && (RaDist(v2,v3) < m_RadThr))
      UpdateAdjacencyMatrix();

  // Test Diagonal Adjacencies between non-vacant entries:
  // ascending and descending -> first good is taken
  // if their radial distance is less than the input threshold m_RadThr
  if ((v0!= NULL && v3!=NULL) && (RaDist(v0,v3) < m_RadThr))
      UpdateAdjacencyMatrix();
  else if((v1!=NULL && v2!=NULL)&&(RaDist(v1,v2)<m_RadThr))
      UpdateAdjacencyMatrix();
}
```

Table 2 - pseudo-code of the first step in the Single Frame Reconstruction procedure

1.3.2 Step 2 : Recovery vacant entries

Once all the connections of step 1 from non vacant entries have been discovered, the method considers all the entries in the lattice with no 3D point (*vacant entries*), as in case of a hole in the grid defined by the ARROV sensor.

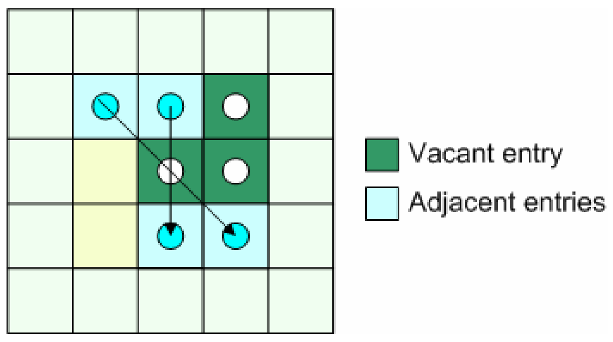
For each vacant entry *v* (Figure 4.left) the procedure considers a *3×3* window W_v surrounding *v* and tests for connectivity between pairs of (non vacant) entries belonging to this W_v:

1. It tests vertical and horizontal adjacencies, checking existence and validity (a connection can exist only if it do not intersect any other already existing connection),
2. It tests the two diagonal adjacencies and validate each potential adjacency only if there are no existing edges crossing it.

```
//  Vacancies Analysis
for (i=2*m_Width+2;i<m_Width*(m_Height-2)-2;i++)
if (adj_mat[i*m_adj_dim+m_adj_dim-1] == 0) {
 BOOL done=FALSE;
 // Load interesting entries with respect to the active 3x3 window
 Load(v1,v6,v3,v4,IntThres);

 double d16,d34;
 if (v1!= NULL && v6 != NULL)   d16=RaDist(v1,v6);
 if (v3!= NULL && v4 != NULL)   d34=RaDist(v3,v4);
 if (d16 < m_RadThr || d34 < m_RadThr)
  if (d16 < d34){
   UpdateAdjacencyMatrix();
   done=TRUE;
  }
  else {
    UpdateAdjacencyMatrix();
    done=TRUE;
  }
   // let's try the crossed ones...  first good is taken
  if (!done) {
   Load(v0,v7,IntThres);
   if ((v0!= NULL && v7!=NULL) && (RaDist(v0,v7)<m_RadThr))
    if ((adj_mat[(i+m_Width-2)*m_adj_dim+14] == 0) &&        // 0 - 16
        (adj_mat[(i-1)*m_adj_dim+14] == 0)){                // 22- 15
            UpdateAdjacencyMatrix();
    } }
   if (!done){
       Load(v2,v5,IntThres);
       if (v2!=NULL && v5!=NULL) && (RaDist(v2,v5)<m_RadThr))
        if ((adj_mat[(i+1)*m_adj_dim+20] == 0) &&             // 10 - 18
             (adj_mat[(i+m_Width+2)*m_adj_dim+20] == 0))      // 16 - 9
             UpdateAdjacencyMatrix();
    }}
```

Table 3 - pseudo-code relative to the computation of step2 in the Single frame Reconstruction procedure



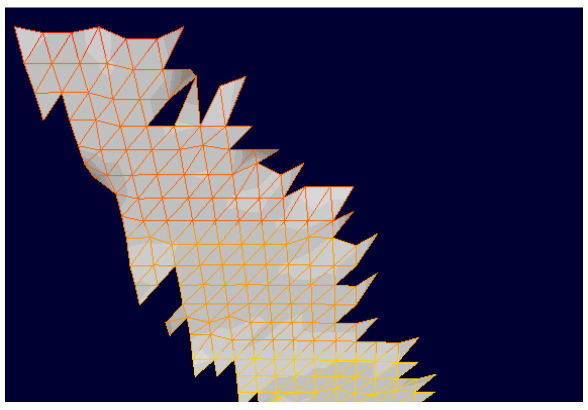


Figure 4 - 3x3 bock testing around a vacant entry (left) and the result of the application of this procedure step applied to an input range image (right)

The results demonstrated that this step is able to fill some holes remained after the application of step_1 and to improve the result of the reconstruction (Figure 4). Unfortunately, too many holes remain in the final mesh and another step is necessary.

1.3.3 Step 3: Improve connections, find feasible semi-diagonals

In order to improve the resulting mesh the method tries to find feasible semi-diagonal connections. For each *3*×*3* window in the input lattice we check for the possible *eight* semi-diagonal edges among the entries at the boundary of the window (Figure 5.right). At most two of the feasible semi-diagonal edges can coexist and the method searches for them (Table 4).

```
// Search for  Horizontal Semi-Diagonals in a 3x3 block
for (i=0;i<m_Width*(m_Height-1)-2;i++) {      // first check
 BOOL done = FALSE;
 if (adj_mat[(i+1)*m_adj_dim+4] == 1)  done = TRUE;
 else if ((adj_mat[(i+1)*m_adj_dim+5] == 1) ||
         ((i>=m_Width) && (adj_mat[(i-m_Width+1)*m_adj_dim+5]==1)))
      done = TRUE;
 if (!done) {                                 // descendant semi-diagonal
  Load (v0,v1,IntThres);
  if (v0!=NULL && v1!= NULL)
   if (CheckOnAdjacencyMatrix())   UpdateAdjacencyMatrix(); }
 if (!done) {                                 // ascendant semidiag
   Load (v0,v1,IntThres);
   if(v0!=NULL && v1!=NULL)
    if(CheckOnAdjacencyMatrix()) {
       UpdateAdjacencyMatrix();
       done = TRUE; }   } }
// Search for Vertical Semi-Diagonals
for (i=0;i<m_Width*(m_Height-2)-1;i++) {      // first check
 BOOL done = FALSE;
 if (adj_mat[(i+m_Width)*m_adj_dim+10]==1)  done = TRUE;
 else if ((adj_mat[(i+m_Width+1)*m_adj_dim+23]==1)||
          (adj_mat[(i+m_Width)*m_adj_dim+11]==1))  done = TRUE;
 if (!done) {                                 // descendent semi-diagonal
   Load (v0,v1,IntThres);
```

```
    if (v0 != NULL && v1 != NULL)
     if(CheckOnAdjacencyMatrix()) {
       UpdateAdjacencyMatrix();
        done = TRUE;
     if (!done) {                          // ascendant semi-diagonal
      Load (v0,v1,IntThres);
      if (v0!=NULL && v1!=NULL)
         if CheckOnAdjacencyMatrix()) {
           UpdateAdjacencyMatrix();
           done = TRUE;   } }}}
```

Table 4 – pseudo-code relative to the computation of step3 in the Single frame Reconstruction procedure.

1.3.4 Step 4: Mesh generation

The last step of our procedure generates the final triangular mesh by forming faces as cycles of three edges (Figure 6). For each location *v* in the lattice, the method scans the portion of adjacency list (that has been populated during the three steps above spanned) by the first 12 bits of the bit vector encoding it (Figure 5) (the remaining edges in the list will be considered from other vertices).

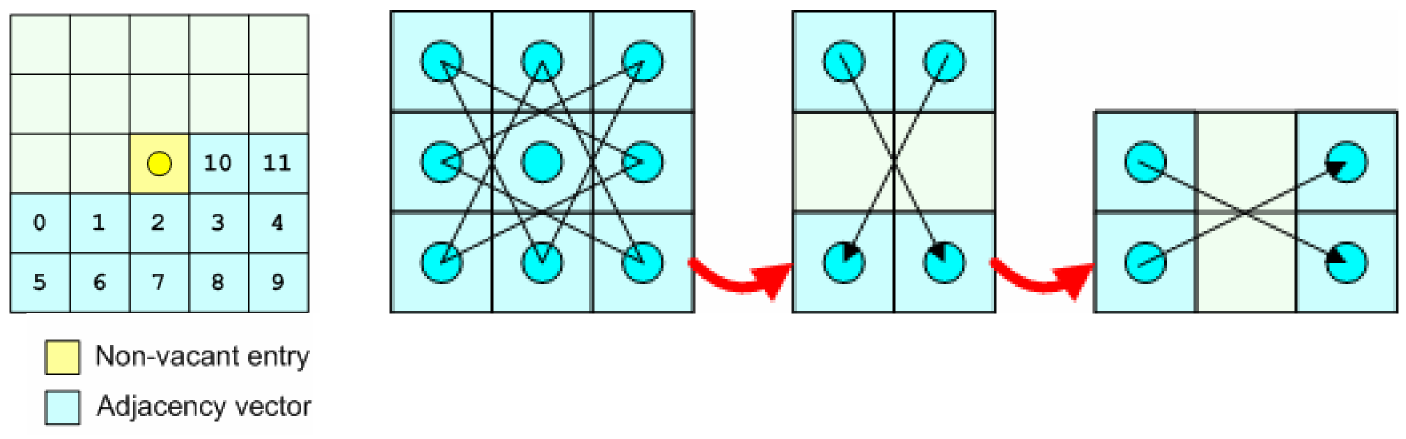


**Figure 5 - the adjacency vector of an entry in the lattice (left)
a 3x3 block for testing for feasible semi-diagonals (right)**

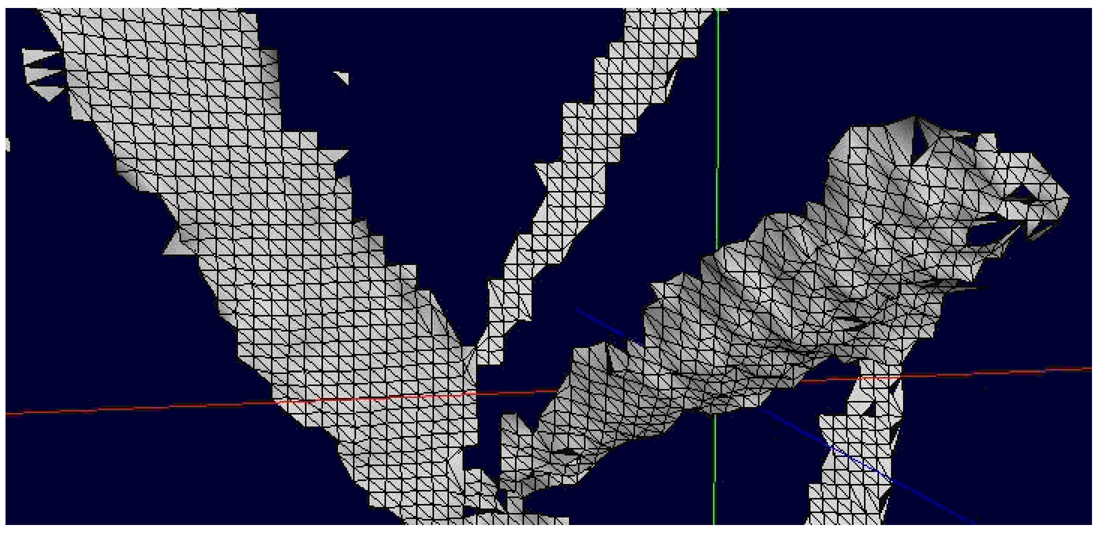


Figure 6 - the result of the application of the SingleFrameReconstruction() procedure.

For each pair of consecutive edges in the list, a triangle of the mesh is generated. A *filter* can be also applied based on the *perimeters* of faces, to obtain a smoother distribution of face dimensions, and to overcome limitations due to the radial nature of the distance (used to test for point connectivity).

```
// Build Mesh Procedure
int p1,p2,p3;  // points
int j,j2, i1,i2;
for (i=0;i<m_Width*(m_Height-2)-2;i++)
 if (adj_mat[i*m_adj_dim+m_adj_dim-1] == 1)
 {
  p1=i; p2=-1; p3=-1; j=0; j2=0;
  while (j<12) {
   if (adj_mat[i*m_adj_dim+j] == 1)
   if (p2 == -1 && p3 == -1) p2=i+alut[j][0]*m_Width+alut[j][1]; j2=j;
   else if (p3 == -1) {
       p3=i+alut[j][0]*m_Width+alut[j][1]; // to check for p2-p3 adj
       i1=alut[j][0]-alut[j2][0]; i2=alut[j][1]-alut[j2][1];
   } if (p2!=-1 && p3!=-1 && adj_mat[p2*m_adj_dim+invalut[i1+2][i2+2]]==1)
   {
       CFace3d *pface = new CFace3d(m_ArrayVertex.GetAt(p1),
                    m_ArrayVertex.GetAt(p2), m_ArrayVertex.GetAt(p3));
       double per = pface->Perimeter(); // filter on face perimeter
       if (per<m_FacePerThres)  pMesh->AddFace(pface);        // add face to mesh
       p2=-1; p3=-1; j=j2; }
   j++; } }
 pMesh->BuildAdjacency(); // first build adjacency list for each vertex
 if (m_bRemoveIsolatedVertices) { // Remove Isolated Vertices
  for (i=0;i<NbVertex();i++) {
    CVertex3d* vert = GetVertex(i);
    int size = (GetArrayFaceNeighbor())->GetSize();
    if (size == 0) DeleteVertex(i); }
 } m_bMeshBuilt = TRUE;
 return 1; }
```

Table 5 - pseudo-code relative to the construction of the Mesh. A Filter to the perimeter of a face is applied in order to improve the goodness of the mesh.

1.4 FILTERING

Once all the faces have been added to the mesh, the isolated vertices are cut off, and the normal for faces and vertices are computed adding them to the mesh structure. The normal computation is split into two phases.

1. Compute normals for faces by taking the external product of two independent unit vectors lying on the face itself.
2. For each vertex, compute the relative normal by taking the vector sum of the faces' normals sharing the active vertex (the *neighborhood faces* of the vertex).

The connectivity information coming from the mesh is used to discard connected components smaller than a given size. This step, called size *filter*, greatly improves the quality of the acoustic images.

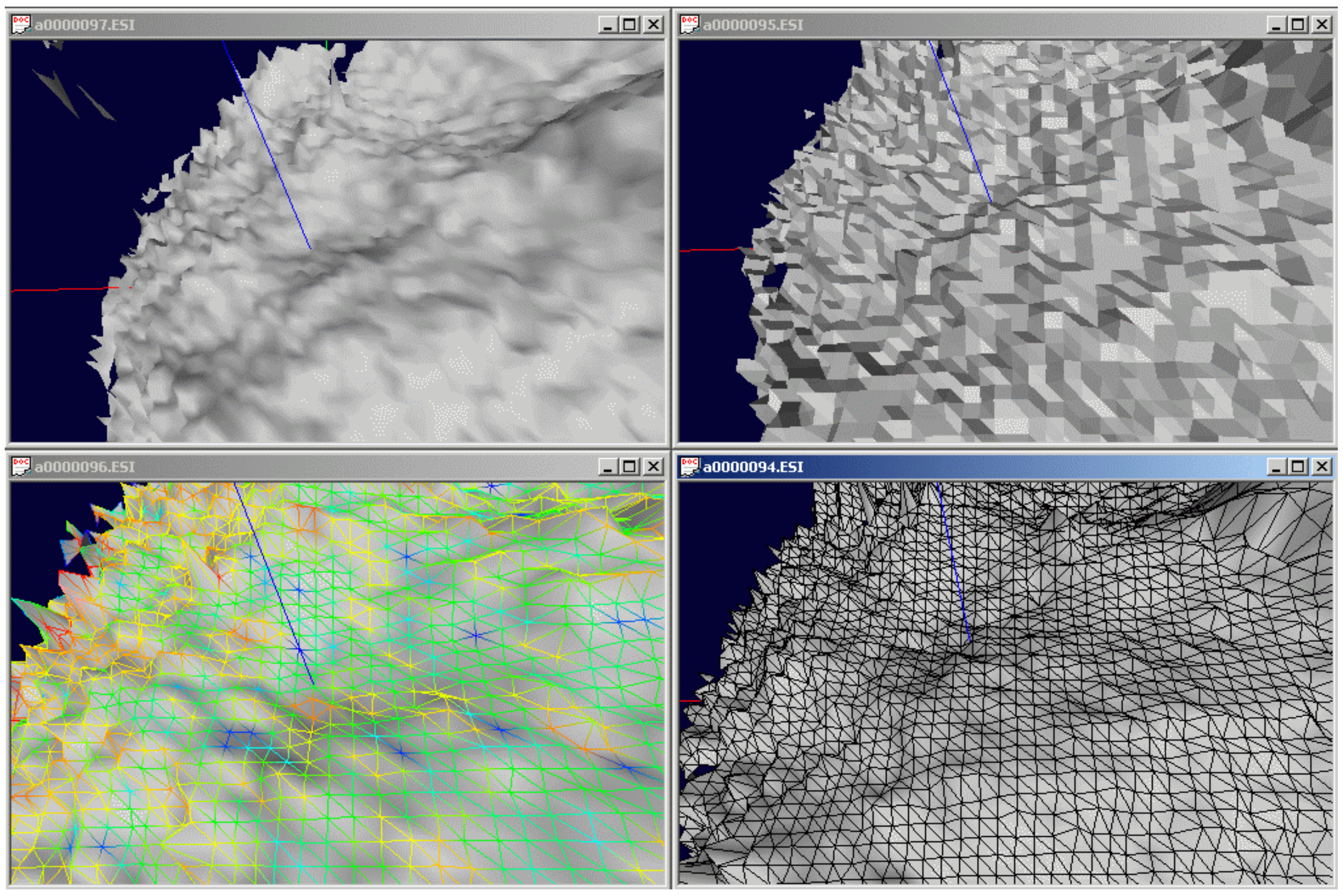
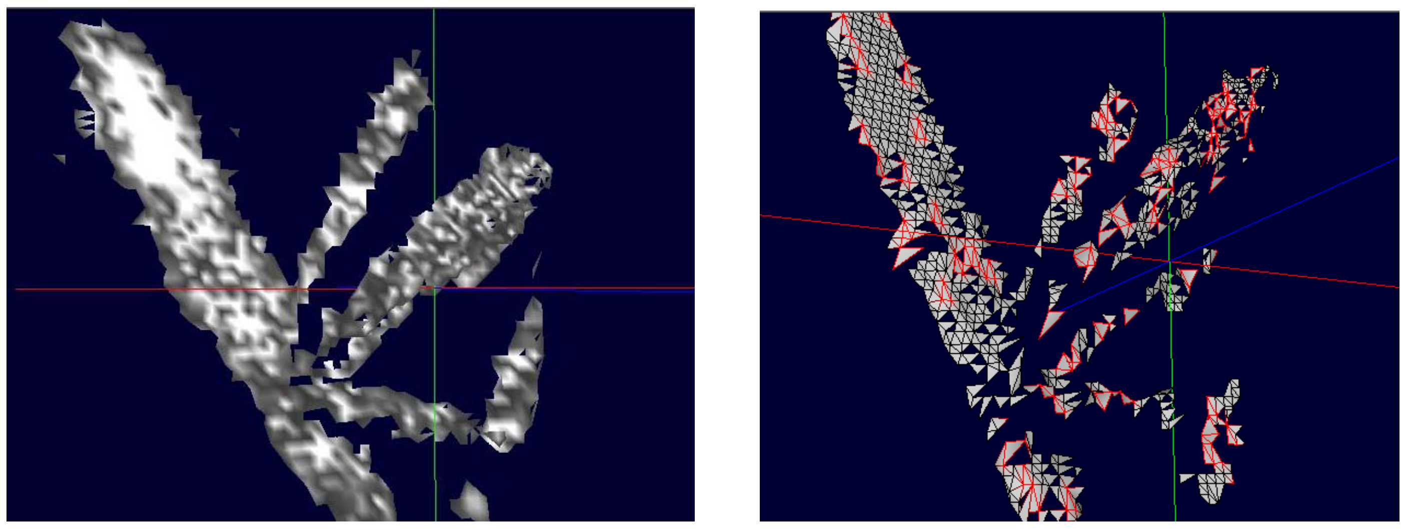


Figure 7 - Tiling of four meshes with different visualization options (from top-left to bottom-right: smooth, simple faces, wireframe with curvature, wireframe and simple faces).

The single frame reconstruction algorithm has been successfully implemented and tested over echoscope 3D data. The results show that a good mesh reconstruction is obtained via a local analysis of the spatial properties of the 3D data, by exploiting the structure of the echoscope sensor. The reconstruction algorithm depends upon two thresholds, but not critically. The first threshold is over the radial distance between 3D points in the lattice, and is used to evaluate the adjacency relationships among the lattice entries. The second threshold is over the intensity of the 3D points. As we can remark looking at the next figure, the intensity values are unevenly distributed over the 3D points, and represent a measure of the reliability of the sensor data itself.



If we decide not to rely on the data in the reconstruction we can rise up the intensity threshold, therefore excluding points out of the mesh. Particularly in this case, as showed in the figure below, the resulting mesh will be much more noisy but our approach will show an optimal behavior. (in red in the figure are sketched the faces obtained from step two and three). Therefore it is possible to obtain both more reliable and more dense meshes.

1.5 FAST REGISTRATION

In order to be able to work on-line, ICP needs to be modified. In general, the speed enhancement of ICP algorithm can be achieved by:

i) reducing the number of iterations necessary to converge and

ii) reducing the time spent in each iteration (i.e., time spent for the calculation of the correspondences)[26] .

Basically, the first and the simplest step is to restrict the number of points to be registered, which is the trade-off between the registration speed and the measurement precision. However, this technique cannot be applied to the registration of the underwater images extremely since the already poor quality of the images.

A number of classical approaches, such as k-D tree [27] , [28] , exist for speed up the calculation of the correspondences, which has been reputed to account for the bulk of the computational complexity of ICP algorithm. Another approach is based on the substitution of the point-to-point distance metric with the point-to-surface distance, which [26] report to yield a faster convergence. The acceleration of the extraction of corresponding points is still open to be addressed and different innovative techniques [29] [30] [31] have been proposed recently.

1.5.1 Reverse calibration approach

In this report we propose an acceleration method based on the so-called reverse calibration technique [32] . The method is based on the fact that from the sensor we obtain data stored in both the structures of unorganized cloud of point $x_i = (x; y; z)$ and range image $r(i; j)$ (i.e., a 2.5 dimensional image) [33] . Given a 3D point $y_i \in Y$ of data set and given the camera parameters it is possible to project y_i onto the range image of the model set $r_m(i\ j)$. The 3D point $x_i \in X$ associated to $r_m(i; j)$ will be the hypothetic corresponding point of y_i.

In order to improve the accuracy of finding correspondences it is possible to use the information of the connectivity given by the range image.

We define the neighborhood N^x_i of x_i as:

$$N_i^x = \{x_{k,h} \in X \quad s.t. \quad x_{k,h} = RP(r_m(i+k, j+h)) \qquad k,h = -w, \ldots +w\} \quad (1.5.1)$$

where w is the dimension of a window centered on $r(i; j)$ and $RP(r(i; j))$ is the operator that re-projects the range point $r(i; j)$ onto the Euclidean 3D space. It is worth noting that the range image is not dense since after the filtering a lot of points are discarded. For each range point $r(i;j)$ there is a flag that indicates if the associated point is survived or not. More precisely, the operator $RP(r(i\ j))$ gives 0 if the corresponding 3D point was discarded. Finally, the definitive corresponding point of y_i will be the closest x'_i of the points belonging to the neighborhood N^x_i of x_i. If the projection of the point y_i falls onto an empty area, this point remains without correspondence.

As we pointed out before, the main important step is the projection of the 3D point onto the range image. This process can be carried out by the following equation:

$$i = \frac{\alpha - I_{OFF}}{s_\alpha}; \qquad j = \frac{\beta - J_{OFF}}{s_\beta} \tag{1.5.2}$$

where s_α and s_β are respectively elevation and azimuth increments (according to the spherical scanning principle), I_{OFF} and J_{OFF} are offsets and finally α,β are given by:

$$\alpha = arctg\frac{y}{z}; \qquad \beta = arctg\frac{x}{z} \tag{1.5.3}$$

The parameters s_α, s_β, I_{OFF} and J_{OFF} are fixed by the acquisition sensor and they determine the aperture of the acquisition (i.e., field of view and resolution). By considering the high computational cost of the *arctg* operator a variation of *i* and *j* indices extraction is introduced.

From equation (1.5.3) we easily find that:

$$tg\alpha = \frac{y}{z}; \qquad tg\beta = \frac{x}{z} \tag{1.5.4}$$

Because the possible angles α_i and β_j on the model image are known a priori (i.e., according to the spherical scanning principle) the values $tg\alpha_i$ and $tg\beta_j$ are stored into a table.

When a 3D point $(x; y; z)$ is coming from a dataset, the values x/z and y/z are computed and they are successively compared with the table values. In this way, by avoiding the *arc-tangent* computation, a dramatic improvement of the speed is obtained.

Another improvement of the speed is gained by changing the computation of the distance in the corresponding points search phase. In particular, as customary in literature, we compare the square of the distance by avoiding the computation of the root as following:

$$d^2 = (x_m - x_d)^2 + (y_m - y_d)^2 + (z_m - z_d)^2 \tag{1.5.5}$$

Summary of the algorithm In summary, the algorithm for speed up the finding of corresponding points is given by the following steps:

For each 3D data-point $y_i \in Y$

- find *i* and *j* by using equations 1.5.4 and 1.5.3
- project y_i on to the model range image $r_m(i; j)$

- find the hypothetic corresponding point x_i
- find the neighborhood N^x_i of x_i
- find the definitive corresponding point x'_i (if it exists)

1.5.2 Subsampling

The points of the data-set are sub-sampled in order to reduce the number of point used for registration. We test two different subsampling methods: *random* and *uniform*.

In both of them, the size of the sub-set is fixed a priori. The random method computes the subset by generating different random number (and consequently by calling the "*random function*" several times). The uniform method simply calculates the subsampling rate S_n and select one point every S_n, according to the acquiring order.

Although the random method is more accurate, the uniform method is quite faster. For this reason, we decide to implement the uniform method.

1.5.3 Pre-alignment

We verified that, the alignment based on the reverse projection, could fail when the two views are not enough close. In order to increase the robustness of the registration, we applied, for some iteration, the classical ICP method without reverse projection. This action permits to obtain a good pre-alignment from which the reverse-calibration alignment converges fast and correctly to the optimal solution.

1.5.4 Speed vs. accuracy

As we stressed before, this work aim at finding the best trade-off between speed and accuracy. In this section we summarize the parameters and the features of the algorithm that are crucial for the performance of the algorithm:

- **Vanish Threshold**: this parameter defines the stop criteria of the ICP iterations. If the residual is less than this threshold, the alignment is stopped. By reducing the vanish, the accuracy is improved but more iteration must be carried out and, consequently, more time is spent. Although this parameter could be tuned by a user, we have found experimentally its best estimation.
- **Automatic outliers rejection:** This procedure makes the algorithm robust to the outliers that rise when the two views are poorly overlapped. Because this computation is based on statistic a lot of time is spent for it. Especially when the motion of the ROV is very slow, it could be possible to disable this feature. Even if there is a flag in the algorithm for its disable, for the most of the cases this procedure is recommended.
- **Subsampling level :** As we mentioned before, the number of points of the subsampled image influences the performance of the alignment. This parameter is tunable by the user that can adapt the subsampling to the environment conditions. Also for this parameter we have find a reasonable estimation that can be used as a default value.

- **Number of pre-aligning iterations:** Some iterations of the classic method for computing the closest points (without re-projection) are needed when the two images are not close enough even if this phase is very computational expensive. By testing different sequences of image pairs we verified that two iterations are sufficient for a good pre-alignment.

1.6 MOSAICING

In general, the meshes built on the single frames, when represented in the same coordinate system, after registration, will freely overlap and intersect. Jumps, cracks and redundant surfaces will occur at overlaps (Figure 8). Thus, the simple collection of such meshes is not sufficient to give a final mosaic of the sensed objects. Meshes must undergo a process of *geometric fusion*, which produces a single mesh starting at the various registered meshes. If the fusion is to be done in off-line framework, algorithms proposed in [24] , [25] serve well the purpose.

In the case of the *on-line version*, at a given instant of time, a mesh *A* is given, which represents the mosaic obtained from all previous frames, and a new mesh *B* comes, which is built from the current frame as explained in the Single Frame Reconstruction Step.

The aim is to merge such meshes in a consistent way, in order to obtain a new mesh *R*, which represents the scene spanned by both A and *B*. This must be a *fast* process, since the system must support real time visualization of the mosaic at each frame. In particular, the output of the algorithm must be used to update the graphical system with the new primitives to be rendered. Unfortunately, this is not simply an incremental process in which new graphical primitives are added to the existing ones at each new frame. Indeed, the *fusion* operation may change the portion of *A* that overlaps with *B* (and it must overlap, otherwise registration would fail).

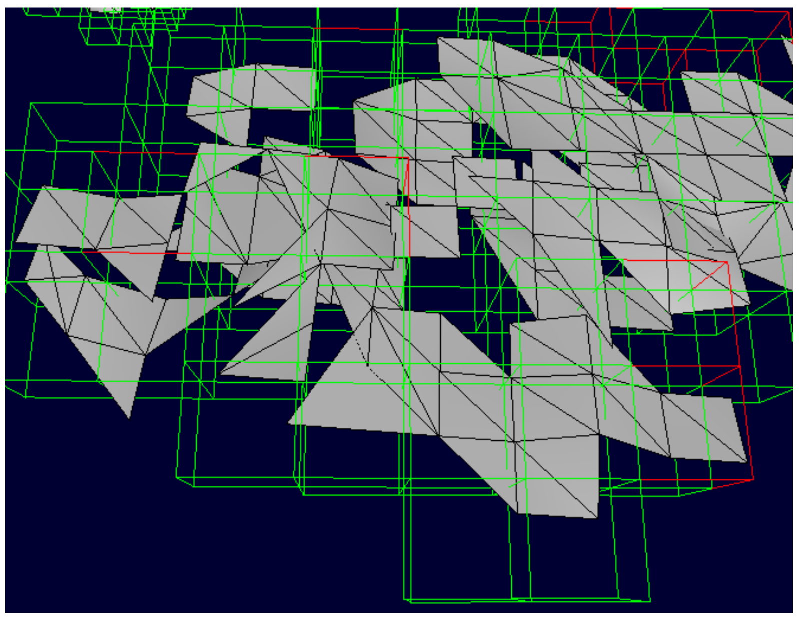


Figure 8 - some faces may overlap after multiple single frame meshes have been added at the model.

This means that some primitives rendered in previous frames will have to be substituted with new ones. Eventually, this means that display lists used by the graphics system will have to be regenerated. This operation can be expensive if applied to the entire mesh, and it could slow down the system.

Because of observations above, we decided to modify the method of the *Marching Intersection Algorithm* (MIA) [24] , which exhibits an interesting approach, since it allows a valid trade-off between speed and accuracy and, by the use of quality parameters, it takes into account the reliability of input data. The original MIA is based on a volumetric approach that locates the geometric intersections between the meshes built on single frames and a virtual 3D reference grid. Intersections are first merged and the output mesh is found by joining intersection points that belong to edges of the same cell, through a scheme defined in a lookup table.

In order to make it applicable to an *on-line setting*, we have modified the algorithm to deal with a pre-computed mesh *A* (as explained before) which *fits* the reference grid (i.e., it has its vertices on grid edges, and each face is completely contained in a grid cell), and a new mesh *B* which intersects the grid properly. Furthermore, in order to support *real time rendering*, we have developed a **lazy update strategy** for display lists, which reduces the load per frame on the graphics system.

So the mosaicing algorithm can be divided into three main phases:

1. Rasterization and Intersection management
2. Fusion, with cleaning and removal operations
3. Mesh generation and Lazy Update

It takes as input:

- a single frame mesh *SFM* with time stamp *t*
- a registration matrix *RM* in homogeneous coordinates, which describes current position and orientation of the sensor in a global coordinate system
- a grid resolution value *GR* (positive real) representing the edge length of square cells in which 3D space is subdivided
- a fusion threshold value *FT* (positive real) representing the minimum distance between two vertices generated by different frames (The Fusion threshold is usually kept lower than grid resolution)
- an updating threshold value *UT* (positive real) representing the maximum number of changes in a mesh before a new list must be generated for visualization

The Fusion Procedure produces in output:

- a time stamp *t* (the same of the input)
- a list of meshes, in which each mesh represents a new portion of mosaic or a modified one, and has a name and the information on the number of triangles in that mesh
- a list of triangles where each triangle is a triple of points (each point represented in a global coordinate system with Cartesian coordinates (x, y,z)).

1.6.1 Phase 1: Rasterization and Intersection Management

This part of the algorithm analyzes every face f_q of one of the registered meshes, searching for intersections with the virtual underlying grid G (the other is the actual mosaic and therefore is already aligned at the grid).

For every face f_q, the minimum enclosing box ($EBox_{fq}$) is computed and the f_q-projection on each plane (called rasterization planes) in the x,y and z directions (XY , XZ, $Y Z$) is analyzed (Figure 9).

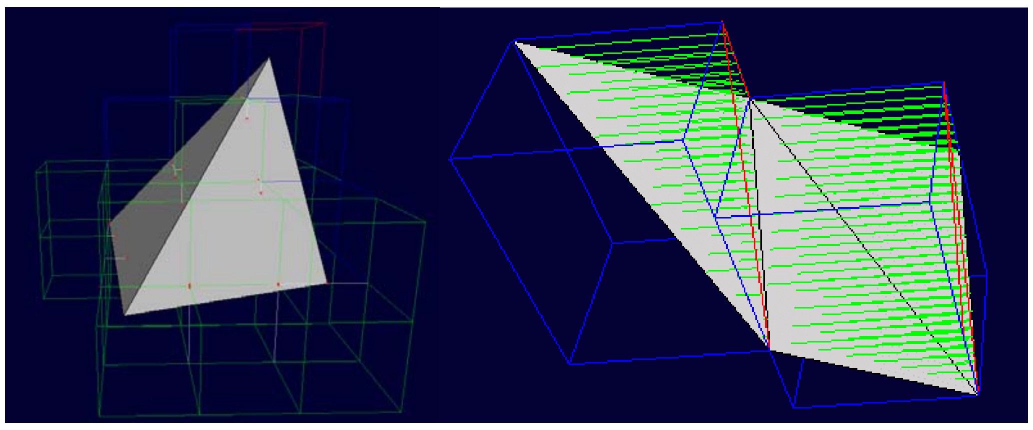


Figure 9 – Rasterization procedure: intersections are marked in red (left) the rasterization lines (X-axis) in green (right)

For doing this, every plane is subdivided in rasterization lines ($l_1,l_2,...,l_m$), each line being orthogonal to the plane and passing through a node of the grid. These lines may intersect a face at some points. Then, for each line l_h, all the intersections ($i_1,i_2,...,i_n$) are computed and the collection of all lines ($l_1,l_2,...,l_m$) gives the set of intersections related to a given rasterization plane p_s (Figure 9, Figure 10). Each intersection i_k is described through a class with the following instance members:

- *ic* as intersection value,
- *name* for the name of the original mesh (or timestamp),
- *sgn* an integer representing the sign of the face normal projection on raster line
- *dir* a number representing the value of the face normal projection on raster line
- *w* a number representing the *weight* of the intersection (computed as a bi-linear interpolation of the weight of surrounding vertices)

Intersections generated by each face f_q are ordered with respect to their value. In the Fusion phase of the algorithm, if two intersections i_1, i_2 are closer than the fusion threshold FT, a merging procedure is invoked. When an intersection i_k is computed, it is inserted into a bi-linked list of the corresponding rasterization line l_h: as the number of rasterization lines is very big, we chose to represent a rasterization plane as an associative collection (hash table) of lists, keeping therefore only those structures containing real intersections.

Every time we instantiate a new list, it is also added to a structure that contains all the updated or generated list of the current frame: this structure is a queue (FIFO type) that is scanned during the fusion phase.

Note that, we assume that underlying grid to have no bounds, that is, to be virtually infinite. Therefore we have to generate for each pair of coordinates a unique key as a function $k=f(x,y)$ that describes the rasterization line: unfortunately it is quite difficult to find such a function. For sake of simplicity we chose $k(x,y) = A\cdot x+y$, where A is a constant (usually big). Of course, it is possible for collisions to occur, but for a medium sized mosaic this is quite unlikely.

Actually the hash function is defined in three dimensions as $k(x,y,z)=A\cdot x+B\cdot y+C\cdot z$ and it is used both on the rasterization planes (setting z to zero) and on the whole 3-D mosaic for cell's indexing purposes

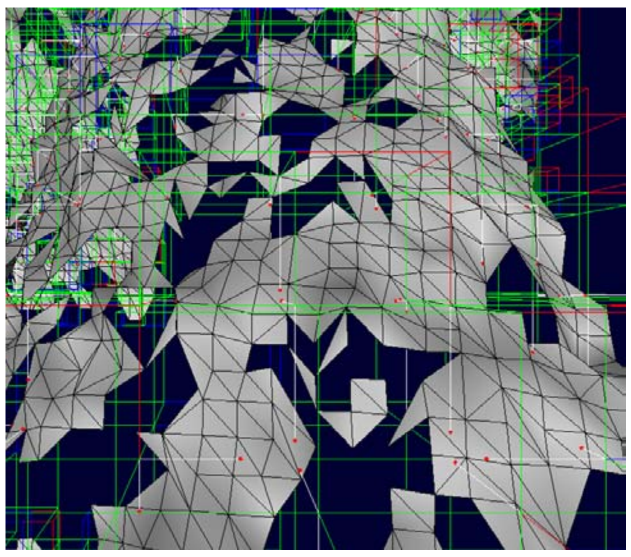


Figure 10 - Cells, edges and intersection (in red)

As we said before, each intersection (object) has a weight w, which measures the *reliability* of the data and it is used in the fusion phase. The weight is computed by linearly interpolating the intensity values of the vertices that describe the current face. If d_0, d_1, d_2 are the distances of the intersection point from the vertices, and I_0, I_1, I_2 are their intensities, then the weight w is defined as follow:

$$w = \frac{\left(I_0 \cdot d_1 \cdot d_2 + I_1 \cdot d_2 \cdot d_0 + I_2 \cdot d_1 \cdot d_0\right)}{d_1 \cdot d_2 + d_2 \cdot d_0 + d_1 \cdot d_0}$$

In Figure 11 the result of the rasterization step is presented in the left, while on the right the original mesh of a single frame is depicted.

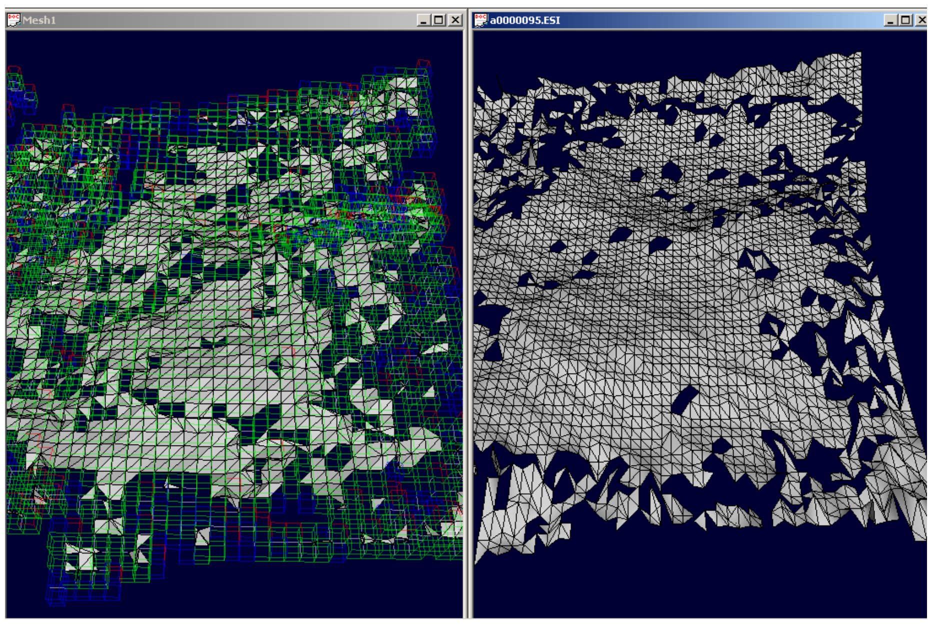


Figure 11 –Surface after rasterization (left) and before (right)

1.6.2 Phase 2: Fusion

After the rasterization phase has been completed on every rasterization plane p_s, the updated intersection lists are scanned and fusion is performed. As we said before, we follow the same approach as in the original MIA to merge intersections generated by A and by B, which are close along an edge of the grid.

Note that: (i) while in the original MIA many possible intersections could need to be merged together (because many different frames overlap on the same region), in this case only pairs of intersections will need merge. This suggests how the *on-line approach* is able to distribute the workload through time, by performing only a relatively small number of operations as a new frame comes; (ii) our fusion process uses only those cells intersecting B independently from the dimension of A.

Merge can be viewed as a warping process that acts on meshes A and B by moving their vertices along edges of the reference grid in order to align them in regions of overlap.

For every intersection lists (associated with a rasterization line l_h), each couple of consecutive intersections i_1 and i_2 with different name (or timestamp - $t_1 \neq t_2$) and concordant signs are merged together if their distance is shorter than the fusion threshold FT taking into account the reliability of the measurement:

```
$IC \quad i_1, i_2, i_3;$
$if((t_{i1} \neq t_{i2}) \wedge (\mathrm{sgn}_{i1} = \mathrm{sgn}_{i2}))$ {
    $d = \frac{w_{i1} \cdot dir_{i1} + w_{i2} \cdot dir_{i2}}{w_{i1} + w_{i2}};$
    $D = \frac{|ic_{i1} - ic_{i2}|}{d};$
    $if(D < FT)$
        $i_3 = Fusion(i_1, i_2);$
}
```

The term d in the previous expression takes into account the fact that fusion is performed along the grid lines rather than on minimum distance direction. This operation corresponds to merge two concordant surfaces, which cross the same virtual cell edge.

If the two intersections i_1, i_2 lie on different virtual cells, then the fusion operator performs a shift of the intersections toward the new average location.

During this operation, whenever we move from a virtual cell to another, a couple of artificial intersections are created on all the perpendicular virtual edges the intersection touches. When a new intersection is generated, the algorithm analyzes the other intersections on the same virtual edge in order to verify if a removal operation can be applied.

The removal operation is performed on the currently updated raster lines $(l_1,\ldots,l_q)$, that is where a new intersection has been created. A removal action (which removes two intersections from the current raster line) is performed on each pair of intersections (i_1,i_2) which:

- lie on the same edge cell e_c,
- have the same name (or timestamp $t_{i1}=t_{i2}$), and
- have *different* signs $\mathrm{sgn}_{i1} \neq \mathrm{sgn}_{i2}$.

1.6.3 Phase 3: Mesh generation

The mesh generation phase uses a modified versions of the popular marching cubes algorithm for polygonising a scalar field. The basic idea is that the reconstruction of a 3D surface is completely defined if all the signed intersections of the surface with the lines of a regular grid are known. The mesh construction can be seen as a sequence of single cell meshing.

Given the virtual 3D-grid G, each cell c can be indexed by its edges $(e_{0c},\ldots,e_{11c})$ or its vertices $(v_{0c},\ldots,v_{7c})$. By the use of a proper container structure, such as a hash table or a map, we keep in memory all the necessary structure without allocating too much memory or degrading the performances of the overall system. For cell's indexing we use the same hash function defined above, using the coordinates of the origin vertex, that is the vertex 0 , for generating the key $k(x,y,z)=A\cdot x+B\cdot y+C\cdot z$.

For each virtual cell c, we analyze the intersections along its edges in order to construct (similarly to how we would find out an isosurface) the vertices (actually, the intersections themselves) and the faces of the relative portion of mesh.

If an edge e_{ic} (i=$0,\ldots,11$) contains an intersection *int* than the classification of its vertices depends on the orientation of that intersection, i.e. sgn_{int}. Successively, faces are generated from the c-edges intersections configuration through a lookup table implementing all the possible intersections configurations on edges.

1.7 LAZY UPDATE

Every time a mesh m_c is created, it is stored as a display list that refers directly to a list of virtual cells and the relative vertices/faces. The current mosaic can be seen as a collection of meshes, each mesh having the same structure described above.

In order to support the graphics system, the algorithm keeps **a geometric structure** consisting of lists of *active cells*, and a corresponding **graphic structure** consisting of different display lists: each list of cells in the geometric structure has a corresponding display list in the graphic structure. Each new frame generates one or more new lists (both geometric and graphic). Such lists contain the new active cells and their corresponding graphical primitives, respectively.

Cells containing portions of surface originated from data with different reliability are distributed among different display lists. This is done because highly reliable data are likely to *survive* longer without modifications.

Old active cells that were modified by processing the new frame *B* are updated in the geometric structure, and each update increases a request value for the relative display list.

In order to speed-up the process of the on-line mosaicing, a *lazy update approach* of the display lists has been implemented, giving out to the Viewer only the newly generated display lists and those display lists for which the amount of changes exceeds the update threshold *UT*.

So, meshes received by a Viewer at a given frame can be of two kinds:

- **New mesh**: a mesh that was generated in the current frame. In this case, this mesh must be added to the collection held by the Viewer.
- **Modified mesh**: a mesh that was generated in a previous frame and modified in the current frame. In this case, this mesh must substitute the mesh having the same name in the collection held by the Viewer. An empty mesh, i.e., a mesh with zero triangles, means that the mesh with the same name must be deleted from the collection held by the Viewer.

A modified mesh can be discriminated from a new mesh on the basis of existence of his name in the collection of meshes that form the current mosaic.

1.8 USER MANUAL

A **Data Proc Module** has been developed for real time processing of data from Echoscope sensor and SIT camera, thereby producing mesh data for graphics visualization. The **Data Proc Module** receives Acoustical and Optical data from the POLECAT Data transfer Interface, process it and sends out meshes to the POLECAT Pilot (or Surveyor) Module via the Mesh Transfer Interface.

The Elaboration Settings are managed by a Settings Dialog, which stores the settings into the host machine registry. Each time the Data Proc Module is started, it reads the settings from the registry and applies them (See section Settings in the User Manual Document – Appendix1).

For more information on the see the Appendix 1 of this document that reviews the main features of the **Data Proc Module** and its usage hints.

2 Off-line data processing

Off-line data processing is obtained from the on-line data processing simply by replacing the fast registration sub-module with another module for accurate registration. Another difference is that the mosaic is not delivered in input frame by frame as it is updated, but all together at the end of processing, in the form of a whole mesh.

In the following, we will describe just the new sub-modules that are inserted to perform accurate registration.

2.1 *ACCURATE PAIRWISE REGISTRATION*

Pairwise registration was addressed using the classical Iterative Closest Point (ICP) algorithm [1] to which we added an outlier rejection rule (called X84) in order to cater for non-overlapping areas between views.

2.1.1.1 Two view point set registration

Let us suppose that we have two sets of 3-D points which correspond to a single shape but are expressed in different reference frames. We call the first of these the model set X, while the other will be the dataset Y.

Assuming that for each point in the data set the corresponding point in the model set is known, *the point set registration problem* consist in finding a 3D transformation which, when applied to the data set Y, minimizes the distance between the two point sets. The goal of this problem can be stated more formally as follows:

$$\min_{\mathbf{R},\mathbf{t}} \sum_{i=1}^{N} \|\mathbf{x}_i - (\mathbf{R}\mathbf{y}_i + \mathbf{t})\|^2, \tag{2.1.1}$$

where R is a 3x3 rotation matrix, t is a 3x1 translation vector, and the subscript i refers to corresponding elements of the sets X and Y. Efficient, non-iterative solutions to this problem were compared in [7] , and the one based on Singular Value Decomposition (SVD) was found to be the best, in terms of accuracy and stability.

2.1.1.2 Iterated Closest Point

In general, when point correspondences are unknown, the Iterated Closest Point (ICP) algorithm may be used. For each point y_i from the set Y, there exists at least one point on the surface of X which is closer to y_i than all other points in X. This is the closest point, x_i. The basic idea behind the ICP algorithm is that, under certain conditions, the point correspondence provided by sets of closest points is a reasonable approximation to the true point correspondence. The ICP algorithm can be summarized:

1.For each point in Y, compute the closest point in X;

2.With the correspondence from step 1, compute the incremental transformation ($R; t$);

3.Apply the incremental transformation from step 2 to the data Y;

4.If the change in total mean square error is less than a threshold, terminate. Else goto step 1.

Besl and McKay [1] proved that this algorithm is guaranteed to converge monotonically to a local minimum of the Mean Square Error (MSE). ICP can give very accurate results when a set is a subset of the other, but results deteriorate with outliers, created by non-overlapping areas between views. In this case, the overlapping surface portions must start very close to each other to ensure convergence, making the initial position a critical parameter.

Modifications to the original ICP have been proposed to achieve accurate registration of partially overlapping point sets [8] [9] [10] . We implemented a variation similar to the one proposed by Zhang [10] , using robust statistics to limit the maximum allowable distance between closest points.

2.1.1.3 Robust outlier rejection

As pointed out by Zhang, the distribution of the residuals for two fully overlapping sets approximates a Gaussian, when the registration is good. The non-overlapped points skew this distribution: they are *outliers*. Therefore, good correspondences can be discriminated by using an outlier rejection rule on the distribution of closest point distances. To do this, we employ a simple but effective rejection rule, X84 [6] , which use robust estimates for location and scale of a corrupted Gaussian distribution to set a rejection threshold. The median is a robust location estimator, and the Median Absolute Deviation (MAD), defined as

$$\mathrm{MAD} = \operatorname*{med}_i\{|\epsilon_i - \operatorname*{med}_j \epsilon_j|\}. \tag{2.1.2}$$

is a robust estimator of the scale (i.e., the spread of the distribution). It can be seen that, for symmetric (and moderately skewed) distributions, the MAD coincides with the *interquartile range*:

$$\mathrm{MAD} = \frac{\xi_{3/4} - \xi_{1/4}}{2}, \tag{2.1.3}$$

Where $\xi_q$is the q-th quartile of the distribution (for example, the median is $\xi_{½}$). For normal distributions, we infer the standard deviation from

$$\mathrm{MAD} = \Phi^{-1}(3/4)\sigma \approx 0.6745\sigma. \tag{2.1.4}$$

The X84 rule prescribes to reject values that are more than *k* Median Absolute Deviations away from the median. A value of *k*=5:2, under the hypothesis of Gaussian distribution, is adequate in practice (as reported in [6]), since it corresponds to about 3.5 standard deviations, and the range [μ-3:5σ; μ+3:5σ] contains more than the 99.9% of a Gaussian distribution. The rejection rule X84 has a breakdown point of 50%: any majority of the data can overrule any minority.

2.2 *PRE-MOSAICING*

Pre-mosaicing consists essentially of the first two phases of the mosaicing sub-module, as described in the previous section. Such two phases are performed on the sequence of input frames initially registered with the accurate pairwise registration method described above.

Once vertices undergo the fusion phase, instead of generating the mesh, an incidence matrix is updated. The incidence matrix is a square matrix with as many rows and columns as the number of frames (only the triangular part of the matrix below the diagonal is used, though). For each entry (i,j) of the matrix a score is recorded, which counts the number of points in the mosaic that are obtained by fusing one point from frame I with another from frame j. The higher the score, the more overlap between two different frames.

All entries of the incidence matrix are initially set to zero. In order to update the matrix, a list of indices is attached to each vertex of the pre-mosaic, which contains the indices of all frames that contributed to generate that vertex through successive steps of fusion. At each new frame j, fusion occurs between one existing vertex of the mosaic (which may have a list of one or more indices attached to) and one vertex from frame j. So the list attached to vertex resulting from fusion is obtained by appending index j to the existing list. Then the incidence matrix is updated by incrementing all entries of the type (j,i) for each I that were present in the list of the vertex that participated in fusion.

At the end of this processing, all data from pre-mosaicing are discarded, while the incidence matrix is passed to the next phase.

2.3 *GLOBAL REGISTRATION*

Global registration aims at representing all the registered frames with respect to the same reference system. A widely used approach (such as the method for the on-line phase) to the registration of many views is to sequentially apply pairwise registration until all the views are combined. These scheme do not use all the available information, and do not compute the optimal solution, because of the accumulation of registration errors.

Global registration could exploit information present in the unused overlapping view pairs, distributing the registration error evenly between every pairwise registration.

The proposed method for the off-line phase consists of locally registering all spatially overlapping image pairs, in addition to those that are adjacent in the frame sequence. The main idea is based on the introduction of the constraints arising from the pairwise registration directly on the transformation matrices, without the need to go over data points again, after the initial pairwise registration between all the overlapping views.

Global registration needs information on the overlapping between the image pairs, in order to select correctly those pairs that can be registered (i.e., the images with a sufficient overlapping). This information is given by the incidence matrix computed in the pre-mosaicing phase. We now turn our attention to the simultaneous registration of several point sets.

2.3.1.1 Chaining pairwise transformations

Assume that there are M overlapping point sets (or views) $V^1,\dots,V^M$, each taken from a different viewpoint. The objective is to find the best rigid transformations $G^1,\dots,G^M$ to apply to each set, bringing them a common reference frame where they are seamless aligned.

Let G^{ij} be the rigid transformation matrix (in homogeneous coordinates) that registers view j onto view i, i.e.,

$$V^i = \mathbf{G}^{ij} V^j \qquad (2.3.1)$$

where the equality holds only for the overlapping portions of the two points sets V^i and $G^{ij}V^j$.

If we choose (arbitrarily) view k as the reference one, then the unknown rigid transformation $G^1,\dots,G^M$ are respectively $G^{k;1},\dots,G^{k;M}$. As customary, we will take k=1. These rigid transformations are not independent each other, being linked by the composition relationship:

$$\mathbf{G}^{k,j} = \mathbf{G}^{k,i}\mathbf{G}^{i,j} \qquad (2.3.2)$$

We can therefore estimate the alignment G^i of image V^i on the reference view (defined by the image V^1), by first registering V^i onto any view V^j and then using G^j to map the result into the space of V^1.

$$\mathbf{G}^j = \mathbf{G}^i\mathbf{G}^{i,j} \qquad (2.3.3)$$

This relationship, can be used to compute G^i when all the matrices $G^{i;i-1},\dots,G^{2;1}$ are known, by simply chaining them

$$\mathbf{G}^i = \prod_{j=2}^{i} \mathbf{G}^{j,j-1} \qquad (2.3.4)$$

The global registration matrix G^i will map V^i into the space of V^1 (the reference view). As it is well known, the combination of pairwise registration does not yield the optimal result. For example, if $G^{k;i}$ and $G^{i;j}$ are optimal on the sense that they minimize the mean square error distance between the respective sets, then $G^{k;j}$ computed with equation (2.3.2) does not necessarily minimizes the mean square error between views V^j and V^k.

Small registration errors accumulate so that images near the end of a sequence have a large cumulative error.

2.3.1.2 Global transformations adjustment

In order to improve the quality of global registration, let us suppose we have locally registered all spatially overlapping image pairs, in addition to those that are adjacent in the image sequence. Especially for underwater images, in which the ROV moves back and forth we can find significant overlapping also between distant views in the temporal sequence.

The aim of our method is to optimize the information coming from every pairwise registrations, obtained by the alignment of all overlapped range images. The original contribution consists in obtaining a global registration by introducing algebraic constraints on the transformations, instead of data points. We first perform pairwise registration between every view and each of its overlapping views, thereby computing the Gij whenever it is possible. By considering many equations as (2.3.3), we can build a system of equations in which the Gi;j are known quantities obtained by pairwise image registration, and the matrices Gi1 = Gi (1 · i · N) are unknowns to be found. By decomposing the homogeneous transformation matrices G into a rotation and translation, equation (2.3.3) splits in two:

$$\begin{cases} \mathbf{R}^j = \mathbf{R}^i \mathbf{R}^{i,j} \\ \mathbf{t}^j = \mathbf{R}^i \mathbf{t}^{i,j} + \mathbf{t}^i \end{cases} \tag{2.3.5}$$

where *R* is a rotation matrix and *t* is a translation vector. Although this system of equations is essentially linear, a number of problems arise when formulating solutions that account for the non-linear constraints on the components of *R*. In order to respect these constraints, the rotation matrices must be suitably parameterized, ending up with a system of non-linear equations.

The problem can be cast as a minimization of an objective function:

$$\min \sum_{i,j} \left(\mathrm{angle}(\mathbf{R}^i \mathbf{R}^{i,j} (\mathbf{R}^j)^\top) + ||\mathbf{R}^i \mathbf{t}^{i,j} + \mathbf{t}^i - \mathbf{t}^j|| \right) \tag{2.3.6}$$

where *angle(...)* takes a rotation matrix and returns the angle of rotation around a suitable axis. Starting from the global registration obtained by chaining pairwise transformation [equation (2.3.4)] a solution is found using a Quasi-Newton method.

The estimated transformation $G^1,\ldots,G^M$ are influenced by all the pairwise observed transformations, and the registration error is distributed over all the estimated transformations. In this sense, the final registration is very close to a well balance graph as defined in [1] . Moreover, the complexity of the proposed algorithm is independent from the number of points involved. Because the objective function includes only the matrix components, the complexity depends only on the number of (overlapping) views.

2.3.1.3 Dealing with rotations

A number of techniques have been developed to represent rotations. One of the most convenient is the quaternions representation. They have a number of mathematical properties that make them particularly well suited to requirements of iterative gradient-based search for rotation and translation [4] .

Rotations are represented by unit quaternions. Instead of requiring the quaternion $q=[u,v,w,s]$ to be a unit vector, we can enforce the constraint that the rotation matrix is orthogonal by dividing the matrix by the squared length of the quaternion [4]

$$\mathbf{R}(\mathbf{q}) = \frac{1}{\mathbf{q} \cdot \mathbf{q}} \mathbf{R}_u(\mathbf{q}) \tag{2.3.7}$$

where $R_u(q)$ is the rotation matrix given by

$$\mathbf{R}_u(\mathbf{q}) = \begin{pmatrix} s^2 + u^2 - v^2 - w^2 & 2(uv - sw) & 2(uw + sv) \\ 2(uv + sw) & s^2 - u^2 + v^2 - w^2 & 2(vw - su) \\ 2(uw - sv) & 2(vw + su) & s^2 - u^2 - v^2 + w^2 \end{pmatrix} \tag{2.3.8}$$

This constraint is necessary in general to ensure the gradient accurately reflect the differential properties of a change in the quaternion parameters.

3 Multi-resolution representation

The purpose of multi-resolution modeling is to speed-up the exploration of large pre-computed geometric models, such as seabeds and rig structures, as well as mosaics of generic structures recorded from the on-line data processing system during previous explorations.

The multi-resolution modeling engine consists of:

- an off-line phase that takes in input a large model in the form of a mesh, and builds a multi-resolution version of it;
- an on-line that takes in input a multi-resolution model and, depending on parameters specified by the application using it, returns a mesh providing a simplified version of the object to be represented.

The on-line version works in real time and can provide a different mesh from frame to frame. The resolution of the output mesh can be decided based on a size budget and/or of accuracy requirements, and can be variable in space, so as to support view-dependent rendering.

Because the multi-resolution model is built off-line, through a time-intensive procedure, multi-resolution cannot be used on-line to render the scene that is being constructed through mosaic. Therefore, possible applications of multi-resolution modelling are:

- During exploration of a (partially) known environment - being the position of the ROV with respect to such environment available – in order to provide virtual geometries to the augmented reality module of pilot interface;
- During off-line processing of data, to ease exploration of large models.

The multi-resolution modelling engine has been developed on the basis of the Multi-Triangulation (MT), a general and powerful multi-resolution model that is described in detail in the Appendix D5.1.1 and in references therein.

A software package is made available at URL http://disi.unige.it/person/MagilloP/MT/ , which includes an object-oriented C++ library and related software tools. The library supports the design of applications that make use of the MT such as:

- MT constructors based on mesh simplification algorithms. In particular, a constructor is provided that can take in input either a triangle mesh at high resolution, or simply a cloud of points (terrain data only).
- MT clients that query an MT on the basis of resolution parameters such as:
    - size of the extracted mesh;
    - focus on an area of interest (e.g., view frustum);

- space-dependent accuracy (e.g., accuracy decreasing with distance from viewer).

The source codes of some demo clients are provided to ease the design of applications. For further details, see **Appendix D5.1.1** and URL http://disi.unige.it/person/MagilloP/MT/ .

4 References

[1] M. Hall, ARROV Project Data Transfer Specification, ARROV Project Report GRL/MSH/127, 1/15/2002 (Mid-term Report Annex VII).

[2] R. Bergevin, M. Soucy, H. Gagnon, and D. Laurendeau. Towards a general multiview registration technique. IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(5):540–547, May 1996.

[3] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, andWerner Stuetzle. Surface reconstruction from unorganized points. Computer Graphics, 26(2):71–78, July 1992.

[4] K.Ikeuchi M.D.Wheeler. Iterative estimation of rotation and tranlslation using quaternion. Technical Report CMU-SC-95-215, Carnegie Mellon University, 1995.

[5] P. Besl and N. McKay. A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(2):239–256, February 1992.

[6] F.R. Hampel, P.J. Rousseeuw, E.M. Ronchetti, and W.A. Stahel. Robust Statistics: the Approach Based on Influence Functions. Wiley Series in probability and mathematical statistics. John Wiley & Sons, 1986.

[7] Lorusso, D. W. Eggert, and R. B. Fisher. A comparison of four algorithms for estimating 3-D rigid transformations. Machine Vision and Applications, 9:272–290, 1997.

[8] E. Trucco, A. Fusiello, and V. Roberto. Robust motion and correspondence of noisy 3-D point sets with missing data. Pattern Recognition Letters, 20(9):889–898, September 1999.

[9] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In Andrew Glassner, editor, Proceedings of SIGGRAPH ’94 (Orlando, Florida, July 24–29, 1994), Computer Graphics Proceedings, Annual Conference Series, pages 311–318. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[10] Z. Zhang. Iterative point matching of free-form curves and surfaces. International Journal of Computer Vision, 13(2):119–152, 1994.

[11] R. K. Hansen and P. A. Andersen, A 3-D underwater acoustic camera – properties and applications, In P. Tortoli and L. Masotti (eds.), *Acoustical Imaging*, Plenum Press, London, 607 – 611, 1996.

[12] V. Murino, A. Trucco, Three-dimensional image generation and processing in underwater acoustic vision, *Proceedings of the IEEE*, 88(12): 1903-1946, 2000.

[13] M. Okino, Y. Higashi, Measurement of seabed topography by multibeam sonar using CFFT, *IEEE J. Oceanic Engineering*, 11(4): 474 – 479, 1986.

[14] L. Henriksen, Real-time underwater object detection based on an electrically scanned high-resolution sonar, *Proc. IEEE Symposium on Autonomous Underwater Vehicle Technology*, Cambridge, 1994.

[15] D. Sauter, L. Parson, Spatial filtering for speckle reduction, contrast enhancement, and texture analysis of GLORIA images, *IEEE J. Oceanic Engineering*, 19(4): 563 – 576, 1994.

[16] S. Z. Li, *Markov Random Field Modeling in Computer Vision*, Springer-Verlag, Tokyo, 1995.

[17] V. Murino, A. Trucco, Markov-based methodology for the restoration of underwater acoustic images, *International J. of Imaging Systems and Technology*, 8(4): 386 – 395, 1997.

[18] Mignotte, C. Collet, P. Pérez, P. Bouthemy, Markov random field model and fuzzy formalism-based data modeling for the sea-bed classification in sonar imagery, *SPIE Conference on Mathematical Modeling, Bayesian Estimation and Inverse Problems,* Colorado, 3816(29): 229-240, 1999.

[19] S. Dugelay, C. Graffigne, J. M. Augustin, Segmentation of multibeam acoustic imagery in the exploration of the deep sea bottom, *Proceedings of 13th International Conference on Pattern Recognition*, Vienna, 437 – 445, 1996.

[20] V. Murino, A. Trucco, Confidence-based approach to enhancing underwater acoustic image formation, *IEEE Transactions on Image Processing*, 8(2): 270 – 285, 1999.

[21] V. Murino, A. Trucco, C.S. Regazzoni, A probabilistic approach to the coupled reconstruction and restoration of underwater acoustic images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1): 9 – 22, 1998.

[22] D. A. Danielson, *Vectors and Tensors in Engineering and Physics*, Addison-Wesley Publishing Company, London, 1996.

[23] L.V. Subramaniam, R. Bahl, Segmentation and Surface Fitting of Sonar Images for 3D Visualization, *Proc. 8th Int. Symp. on Unmanned Untethered Submersible Technology*, Durham (NH, USA), pp. 290-298, September 1995

[24] Cignoni P., Montani C., Scopigno R and Rocchini C, The Marching Intersections algorithm for merging range images. Technical Report B4-61-00, I.E.I. - C.N.R., Pisa, Italy, June 2000.

[25] Curless B., Levoy M., *A volumetric Method for building complex models from Range Images*. Computer Graphics: Siggraph '96 Proceedings, pp.221-227, 1996

[26] M. Levoy S. Rusinkiewicz. E±cient variants of the icp algorithm. In IEEE Int.Conf. on 3-D Imaging and Modeling, 3DIM '01, Quebec City (Canada), 2001.

[27] J. L. Bentley. Multidimensional binary search trees used for associative searching. CACM, 19, September 1975.

[28] R. Rivest. the optimality of elias's algorithm for performing best-match searches,1974.

[29] I. Pitas C. A. Kapoutsis, C.P. Vavoulidis. Morphological iterative closest point algorithm. IEEE Transaction on Image Processing, 8(11), 1999.

[30] C. Dorai, J. Weng, and A. Jain. Optimal registration of object views using range data. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(10):1131 – 1138, 1997.

[31] G. Godin M. Greenspan. A nearest neighbor method for e±cient icp. In IEEE Int. Conf. on 3-D Imaging and Modeling, 3DIM '01, Quebec City (Canada), 2001.

[32] G. Blais and M. D. Levine. Registering multiview range data to create 3-D computer objects. IEEE Transactions on Pattern Analysis and Machine Intelligence, 17(8):820–824, 1995.

[33] Paul J. Besl. Active, optical imaging sensors. Machine Vision and Applications, pages 127–152, 1988.

Appendix 1 – Data_Proc User Manual

4.1 PURPOSE

The Data Proc Module has been developed for real time processing of data from Echoscope sensor and SIT camera, thereby producing mesh data for graphics visualization. The Data Proc Module receives Acoustical and Optical data from the GRL Data transfer Interface, process it and sends out meshes to the GRL Pilot (or Surveyor) Module via the Mesh Transfer Interface, provided by GRL.

The Elaboration Settings are managed by a Settings Dialog, which stores the settings into the host machine registry. Each time the Data Proc Module is started, it reads the settings from the registry and applies them (See section Settings below).

In the following we shall review the main features of the Data Proc Module and its usage hints.

4.2 MAIN DIALOG

The main dialog gathers all the controls and feedback windows necessary for controlling the elaboration of the incoming data.

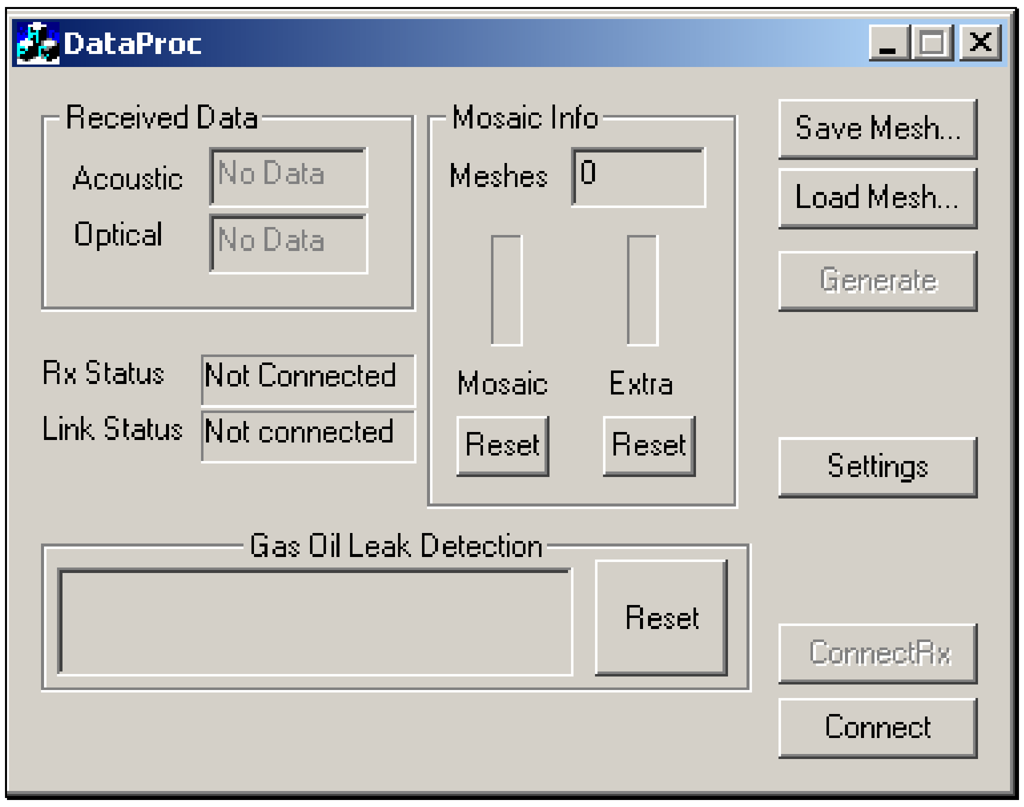


Figure 12 - data_proc main Dialog Box

Mosaic Info:
this section contains information about the generated mesh. The left progress bar indicates the percentage of meshes in the Mosaic buffer. The reset button deletes all the meshes from the mosaic and from the Pilot/Surveyor visualization module. The right progress bar shows the percentage of meshes contained in the Extra-Mosaic buffer.

The Extra-Mosaic meshes are those meshes that are not generated by the mosaic module, but are single frame reconstructed and (possibly) registered. The reset button deletes all the meshes from the Extra-Mosaic mesh buffer and from the Pilot/Surveyor visualization module.

Gas Oil Leak Detection:
This window shows every message coming from the Gas-Oil Leak detector, if activated in the pipeline. The showed message can be one of the following:

- "DETECTOR NOT INITIALIZED”: Leak Detection is active but has not been properly initialised;
- "WAITING FRAMES ... ": Leak Detection is active but waiting for frames;
- "LEAK DETECTOR IS WORKING": Leak Detection is active and computing;
- "NOTHING DETECTED ": No motion has been detected;
- " DETECTED A BUBBLE COLUMN": Leak Detection detected a Bubble Column;
- " DETECTED A LEAK": Leak Detection detected a Leak;

The reset button re-initialises and restarts the Leak Detection module.

4.3 SETTINGS

The settings dialog contains a roll-up control that allows for changing the elaboration pipeline and the settings for every processing module. The Settings Dialog is modeless, so it can be shown through the usual operations.

Note:
in order to make effective the changes, user has to push on the “Apply” button

When the Data Proc Module is active, it reads all the settings from the registry key “Software\\ARROV_DATAPROC”. The settings are written back in the registry when the application is closed.

Note:
in case of abnormal termination or application crash, the settings can be partially or completely lost. In this case user has to check at next execution if the settings are correctly set.

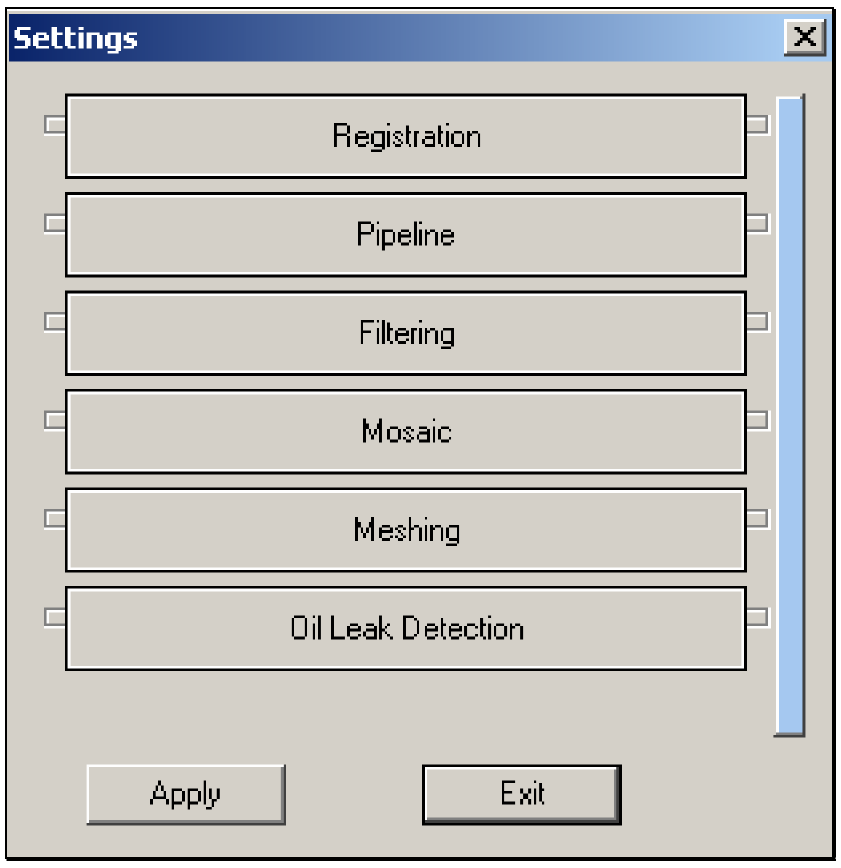


Figure 13 - for choosing the settins

4.4 ELABORATION MODULES

4.4.1 Pipeline

The Pipeline Settings manages the overall elaboration module. Each check box can activate or exclude a processing module from the main pipeline. Each elaboration module has a dialog for its internal settings

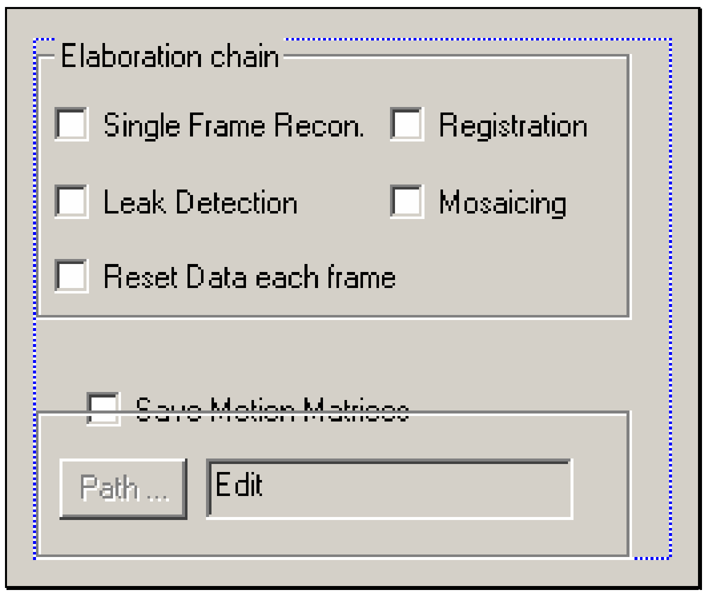


Figure 14 - Dialog Box for internal settings

1. **Registration:** The incoming Acoustic Frame is registered with the previous one, and roto-translated according with the transformation computed by the registration module

2. **Single Frame Reconstruction:** The incoming Acoustic frame is meshed using the Single Frame Algorithm
3. **Mosaicing:** the mesh produced by the Single Frame Reconstruction is added to the current Mosaic Mesh, one or more meshes are sent out to the Visualization module.
4. **Leak Detection:** the incoming Acoustic Frame is sent to the Leak Detection Module, that process it and sends out a result string as described above.
5. **Reset Data each Frame:** each time a frame arrives, the old one is deleted.

Note:
this mode is useful for system monitoring purposes, where user don't want to keep frames into the main memory.

6. **Save Motion Matrices**: if this checkbox is selected, after every registration the resulting motion matrix is saved on disk.

4.4.2 Filtering

Filtering applies partly before the single frame reconstruction, on raw acoustic data, and partly after, by removing small connected components in the resulting single-reconstructed mesh.

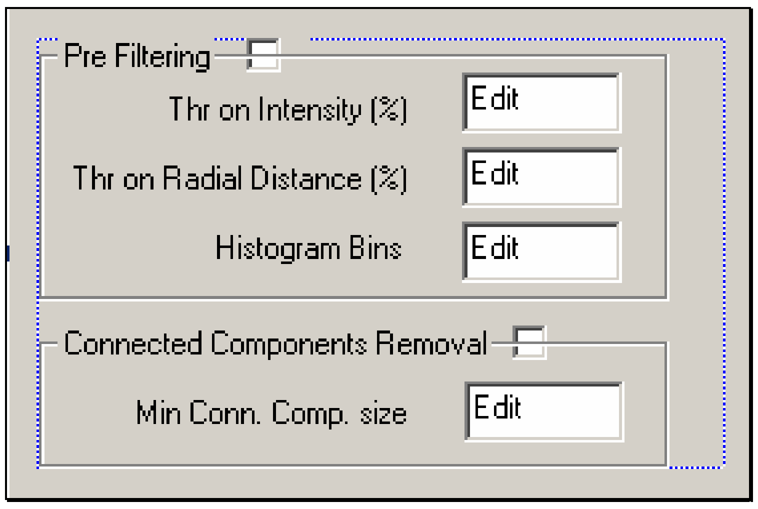

Pre Filtering
Thr on Intensity (%) Edit
Thr on Radial Distance (%) Edit
Histogram Bins Edit
Connected Components Removal
Min Conn. Comp. size Edit

Figure 15 - Pre-filtering Settings

- **Pre-filtering**: if active filters out points in the acoustic frame, on the base of their intensity or their radial distance. (DEFAULT: TRUE)
- **Threshold on Intensity**: cut-off threshold (float, DEFAULT: 1. RANGE: 0-100).
- **Threshold on Radial Distance**: cut-off threshold (float, DEFAULT: 1. RANGE: 0-100)
- **Histogram Bins**: Number of Histogram Bins used to estimate the values distributions (DEFAULT: 100).
- **Connected Components Removal**: if active removes small connected components in the single frame reconstruction (DEFAULT: FALSE).
- **Minimal Connected Components size**: minimum number of vertices of connected components in the mesh (int, DEFAULT: 6)

4.4.3 Registration

Registration computes the geometric transformation that registers the current frame with the previous one. The resulting transformation is a 4x4 matrix describing rotation and translation in the world coordinates.

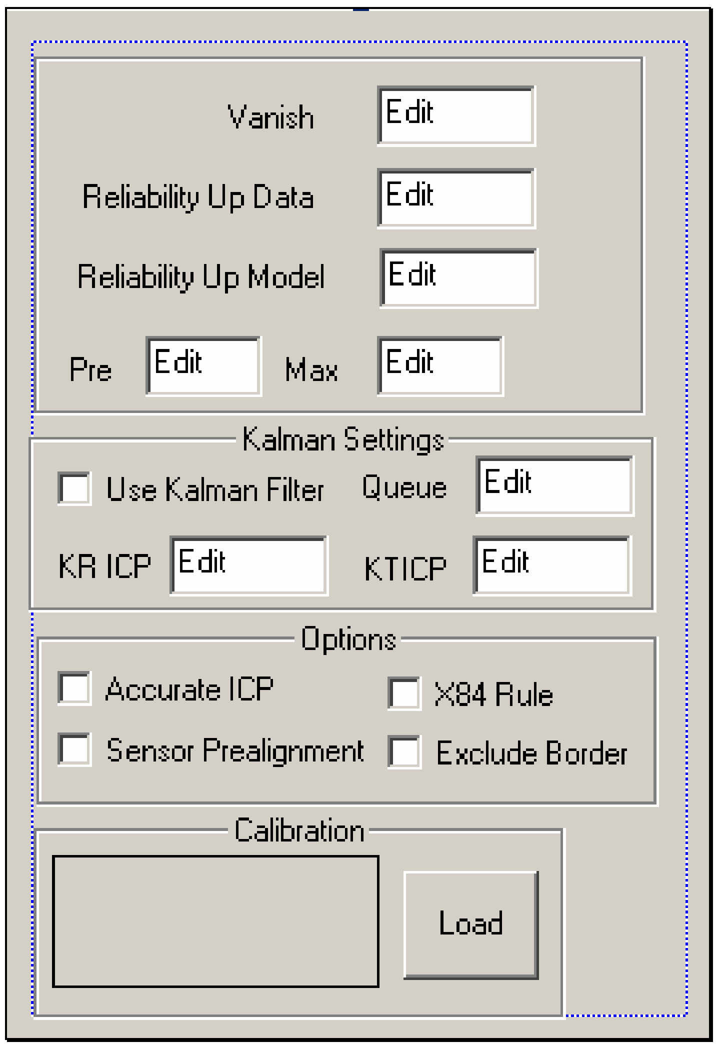


Figure 16 - Registration Settings

- **Vanish**: cut-off threshold for stopping iterations (double, DEFAULT: 0.001)
- **Reliability Up Data Threshold**: used to decimate points from the data (int, DEFAULT: 400)
- **Reliability Up Model Threshold**: used to decimate points from the model (int, DEFAULT: 1000)
- **Pre Alignment iterations**: number of slow ICP iterations used to pre-align data (int, DEFAULT: 0).
- **Max iterations**: max number of iterations (int, DEFAULT: 100)
- **Use Kalman Filter**: if selected Kalman Filter is activated.
- **Queue**: Kalman buffer queue size (int, DEFAULT: 5)
- **KR ICP parameter**: used to set the confidence on Rotation measure from the echoscope sensor (double, DEFAULT: 0. RANGE: 0-100);

- **KT ICP parameter**: used to set the confidence on Translation measure from the echoscope sensor (double, DEFAULT: 0. RANGE: 0-100);
- **Accurate ICP**: uses slow ICP
- **Sensor Pre-Alignment**: uses alignment data coming from echoscope sensor
- **X84 Rule**: if selected uses the X84 rule in the ICP computation
- **Exclude Border**: if selected excludes the border points from the computation
- **Calibration**: Opto-Acoustic Calibration section.
- **Calibration Matrix**: is the 4x3 matrix that binds 3D data from acoustic sensor with 2D data from SIT camera. This matrix is used to colorize or texturise the final meshes. The button **Load** browses for the calibration matrix file.

4.4.4 Mosaic

The Mosaic gathers together the meshes after filtering and single frame reconstruction, applies the transformation computed by the registration module and produces a output mosaic mesh.

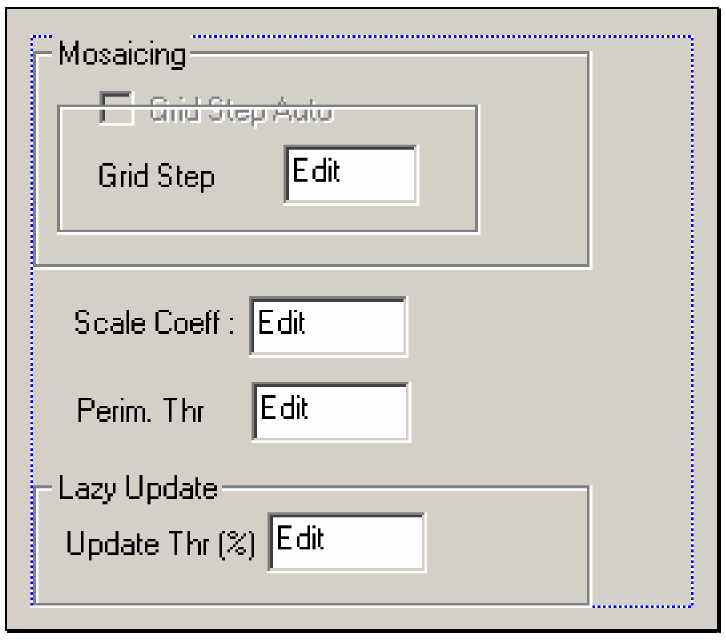


Figure 17 - Mosaicing internal settings

- **Grid Step**: size of the main grid in the mosaic. This parameter tunes the overall resolution of the mosaic.

Note: as the grid step increases, it deeply affects the computation time and therefore the on-line performance of the system (double, DEFAULT: 30);

- **Scale Coefficient**: used to multiply the incoming 3D points. Used for compatibility issues with data formats in the registration module (double, DEFULT:100).
- **Mesh Face Perimeter Threshold**: this threshold cut off meshes with high perimeter. Used to produce better looking meshes. (int, DEFAULT: 300)

- **Lazy Update Threshold**: this threshold tunes the refresh rate of the mosaic. (double, DEFAULT: 0.6, RANGE: 0 – 2).

4.4.5 Meshing

The meshing section controls the setting of the output mesh and the pseudo-texturing of the optical images.

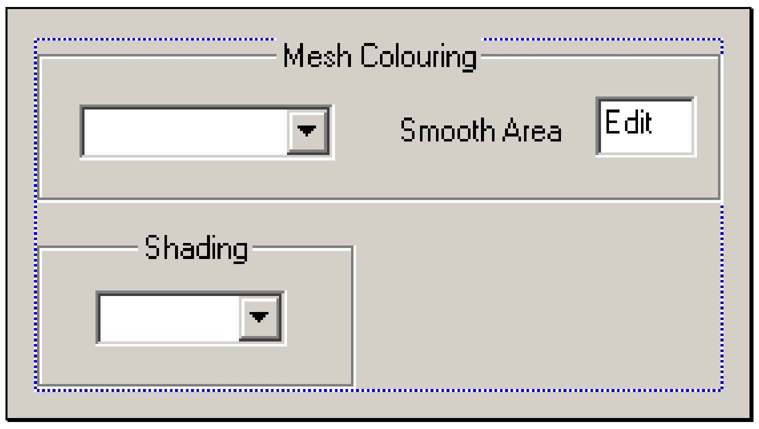


Figure 18 - Meshing Settings

- **Mesh Colouring**: sets the mesh colouring mode, the possible values are:
  - **fixed**: each mesh has a single (default white) colour, for improved visualisation.
  - **Multi**: each mesh has a different colour depending on the current frame number
  - **Colour from Images**: Each vertex is coloured using the information from the Optical image. The colour of each face of the mesh is computed by bi-linear interpolation of colours at face's vertices.
- **Smooth Area:** in the case of mesh pseudo-texturing the colour of each vertex is obtained by computing the mean value in the area around the target point in the image. This area is a square whose side is 2*(Smooth Area)+1. (int, DEFAULT:1. if 0 the target point colour is used).
- **Shading Mode:** this parameter controls whether the resulting mesh is shaded using the normals to the faces (set to "Flat") or the normals to the vertices (set to "Smooth") (DEFAULT: Smooth).

Note:
in case of mosaic mesh generation the shading mode is set to "Flat" by default.

4.4.6 Leak Gas Oil Detection

Those settings refer to the Oil Leak Detection Module, whose purpose is the on-line detection of Oil and Gas Leaks.

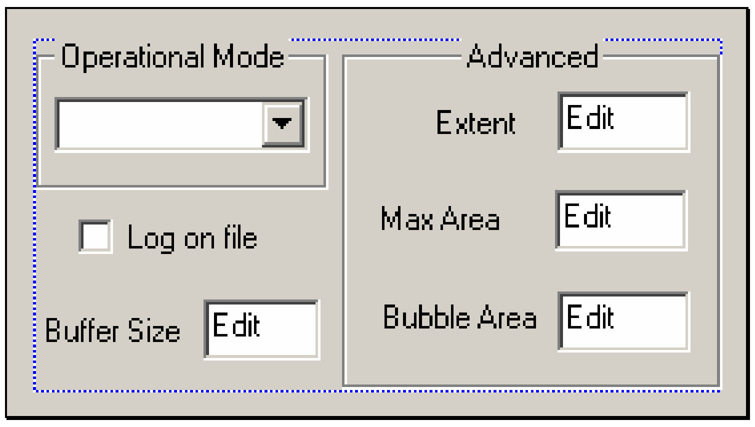


Figure 19 - Leak/Gas Detection Settings

- **Operational Mode**: chooses the operational mode of the Leak Detector
- **Log on File**: is selected saves to a file the verbose log from the Detector
- **Buffer Size**: size of frame buffer used for detecting leaks
- **Extent**: Advanced parameter. Please refer to Oil Leak Detector Manual
- **Max Area**: Advanced parameter. Please refer to Oil Leak Detector Manual
- **Bubble Area**: Advanced parameter. Please refer to Oil Leak Detector Manual

4.5 Example

This a usage example of the Data Proc module during usual operations.

1. If necessary change the elaboration settings activating the Settings dialog by clicking on the **Settings button**.
2. First click on the **Connect button** in main dialog interface.
3. If a proper data source (example echoscope module) is active and connected, click on the **RX button** and wait until a valid connection is set up by the data transfer mechanism.
4. Click on **generate button** to activate the timer that polls on the data transfer and, if any, elaborates the incoming data.
5. In case push on the **reset buttons** to reset graphics visualization.