FLUid Image analysis and Description (FLUID)
Project No. FP6-513663
FET - Open Domain, Priority IST

Deliverable 4.1
Demonstrator on the motion layers estimation method

(1)L. Alvarez, C.A. Castaño, M. García, K. Krissian, L. Mazorra, A. Salgado, J. Sánchez

(1)AMI group, Univ. Las Palmas Gran Canaria

Due date of deliverable: 30 June 2007
Actual submission date: June 2007
Start date of project: 1 December 2004
Duration: 3 years

Project cofunded by the European Commission within the Sixth Framework Programme (2002-2006).

Dissemination level: PU (Public)  Revision: 2

1lalvarez@dis.ulpgc.es
Contents

1 Introduction 4

2 Dense 2D cloud motion estimation. 6
   2.1 General overview 6
   2.2 Standard variational formulation 7
   2.3 Multi-channel flow computation 8

3 Altitude estimation of each cloud point. 10

4 Cloud altitude regularization. 14

5 Software for satellite image sequence processing 15
   5.1 Basic correlation flow estimation technique 15
   5.2 Basic PDE flow estimation technique 15
   5.3 Multichannel PDE flow estimation technique 16
   5.4 Function to remove noise and outliers 16

6 Software for 3D visualization of cloud structures 16
   6.1 Installing AMILab 17
      6.1.1 Set your dynamic libraries path 18
      6.1.2 Set your path 18
      6.1.3 Set the AMILab script directory 18
      6.1.4 Automatic configuration 18
      6.1.5 Run AMILab 19
      6.1.6 More documentation 19
   6.2 Reading the images 19
   6.3 Tutorial for FLUID 21
      6.3.1 Install the tutorial files 21
      6.3.2 Set the fluid navigation path 21
      6.3.3 Loading an image 21
      6.3.4 Displaying an image 22
      6.3.5 Displaying a satellite image in 3D 23
      6.3.6 Selecting a region of interest 23
      6.3.7 Resampling the image 24
6.3.8 Saving and Setting 3D views ........................................... 24
6.3.9 Saving an animation .................................................... 25
6.3.10 Coloring the clouds .................................................... 26
6.3.11 Drawing 3D vectors ...................................................... 28
6.4 Example of a video ............................................................ 28
6.5 Image ........................................................................ 33
6.6 ReadRawImages ............................................................... 33
6.7 CreateFlatMesh ................................................................. 34
6.8 ElevateMesh ................................................................. 34
6.9 ComputeAtitudes ............................................................ 35
6.10 Altitude2Position ............................................................ 36
6.11 ReadFlow ................................................................. 36
6.12 CreateVectors ............................................................... 37
6.13 SetTransform ............................................................... 37
6.14 Interpolate ................................................................. 38

7 Acknowledgements ............................................................... 38
1 Introduction

This deliverable is devoted to the 3D motion cloud layers estimation method we have developed in the context of FLUID project.

Figure 1: Images from different channels of the MSG satellite. From left to right, respectively, the VIS 0.8, WV 6.2, WV 7.3 and IR 10.8. Top row: Images from the Vince hurricane. Bottom row: Images from the sequence on June 5th, 2004.

Meteosat Second Generation satellites replaced in 2002 the former Meteosat, providing significantly increased amount of information in comparison with the previous version in order to accomplish the target of continuous observation of the Earth’s full disk. In this sense, MSG generates images every 15 min with a 10-bit quantization, a spatial sampling distance of 3 km at subsatellite point in 11 channels, from the visible to the infrared channel, and 1 km in the high resolution visible channel [30]. All these channels provide information that is used for different applications, summarized in Table 1. Among the most important applications, numerical weather prediction combines the information from different channels, mainly from the VIS 0.8, WV 6.2, WV 7.3 and IR 10.8 channels [30], to compute the displacement of the clouds between two time instants, that constitute the most important source of information for this application.

The VIS 0.8 channel provides information on the visible zone of the spectrum, that allows identification of cloud structures in the atmosphere, cloud tracking and land and vegetation monitoring. Water Vapour channels, WV 6.2 and WV 7.3, allow us to observe water vapour and winds, and also supports height allocation of semitransparent clouds [29]. Finally, IR 10.8 is essential for measuring temperatures at sea and land surface and top of clouds, and detection of cirrus cloud [18]. Another important piece of information provided by EUMETSAT is a cloud layer classification which consists on an estimation of the cloud structure altitude using this multichannel information, yielding to a segmentation of the pixels into different types of clouds [31, 3], as shown in figure 2.

In order to obtain a 3D cloud layers motion field, we need to address the following issues:
### 1 Introduction

<table>
<thead>
<tr>
<th>Name of the Channel</th>
<th>Central Wavelength</th>
<th>Main Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIS 0.6</td>
<td>0.63 µm</td>
<td>Cloud detection and tracking, surface identification</td>
</tr>
<tr>
<td>VIS 0.8</td>
<td>0.81 µm</td>
<td>Cloud detection and tracking, surface identification</td>
</tr>
<tr>
<td>NIR 1.6</td>
<td>1.64 µm</td>
<td>Discrimination snow/ice cloud/water cloud</td>
</tr>
<tr>
<td>IR 3.9</td>
<td>3.92 µm</td>
<td>Detection of low cloud/fog at night</td>
</tr>
<tr>
<td>WV 6.2</td>
<td>6.25 µm</td>
<td>Water vapour structures at medium-high level</td>
</tr>
<tr>
<td>WV 7.3</td>
<td>7.35 µm</td>
<td>Water vapour structures at low-medium level</td>
</tr>
<tr>
<td>IR 8.7</td>
<td>8.70 µm</td>
<td>Ice/water distinction</td>
</tr>
<tr>
<td>IR 9.7</td>
<td>9.66 µm</td>
<td>Ozone detection in lower stratosphere</td>
</tr>
<tr>
<td>IR 10.8</td>
<td>10.80 µm</td>
<td>Estimation of temperature of clouds and surface</td>
</tr>
<tr>
<td>IR 12</td>
<td>12.00 µm</td>
<td>Estimation of temperature of clouds and surface</td>
</tr>
<tr>
<td>IR 13.4</td>
<td>13.40 µm</td>
<td>Cloud height estimation</td>
</tr>
<tr>
<td>HRV</td>
<td>0.75 µm</td>
<td>High spatial resolution</td>
</tr>
</tbody>
</table>

Table 1: Characteristics and main applications of the different MSG channels

![Image](image_url)

Figure 2: In this image, we illustrate, using different colours, the original cloud structure layer classification estimated from the meteorological satellite channels.
• **Dense 2D cloud motion estimation.** We estimate, using the different channels of the satellite image sequences, a dense 2D cloud motion.

• **Altitude estimation of each cloud point.** We estimate, from the temperature channel, the altitude of each cloud pixel in the image.

• **Cloud altitude regularization.** Optionally, we filter (independently in each layer classification domain) the altitude, in order to smooth the altitude values and to remove potential outliers.

• **3D motion layer structure visualization.** We create a specific user friendly visualization tool to visualize the 3D motion layer structure obtained.

This document is organized according to the main objectives described above. At the end, we present an overview of the AMILab Software that we use to process and visualize the cloud layers.

2 Dense 2D cloud motion estimation.

2.1 General overview.

Estimation of the cloud motion from satellite images has several applications in meteorology and climate ([15]). In particular, it is an important source of information for numerical weather prediction (NWP) ([5]). It is also useful in understanding the structure and dynamics of hurricanes and severe thunderstorms.

Different classes of techniques have been used to estimate the cloud motion, among them are techniques using the local cross-correlation ([21, 27, 29]) and cross-correlation combined with relaxation labeling ([34, 13]), motion analysis from stereoscopic images ([35, 20]), neural networks ([11]) block-matching techniques ([9]), local fitting ([36]), scale-space classification by matching contour points of high curvatures ([23]), and variational techniques using Partial Differential Equations (PDE) also referred to as Optical Flow techniques in the field of computer vision([10]). The most widely used techniques are based on the local cross-correlation. They have the advantage of being robust to global intensity changes, but they are also computationally expensive and they do not integrate a global regularization that ensures spatial coherence of the results. For these reasons, the displacement motion is usually calculated on a discrete regular lattice of the image domain. Also, local cross-correlation techniques are well suited for rigid-motion but can fail in case of fast non-rigid displacements. On the opposite, methods based on variational optical flow impose a global smoothing constraint on the estimated displacement and are calculated on the whole image domain, leading to a dense estimation of the flow.

The tracking of cloud motion is usually computed from the infrared (IR: 10.5-12.5 μm) channel ([21, 29]). Low-level cloud motion has also been tracked from the visible channel ([25, 36]). Recently, multispectral motion analysis have been investigated, first by visual superimposition of the cloud motion estimated on each channel individually ([32]) and more recently using a multichannel cross-correlation technique with the visible and IR channels ([13]). Recently, [16] have also proposed a dense multi-layer estimation of the cloud motion.

A large number of methods have been proposed in the computer vision community to address the problem of motion estimation from a set of images. The projection of the 3D object motion
in the scene yields a 2D flow field in the image domain. Most of the methods deal with the problem of estimating the 2D vector field between images based on the image intensities. This problem is generally referred to as “optical flow estimation”. The optical flow is the apparent displacement of pixels in a sequence of images.

During the last two decades a large amount of techniques for computing the optical flow have appeared. These methods can be classified into three different categories: correlation-based, gradient-based and phase-based techniques ([7, 22]). Different works have also evaluated the performance of the most popular algorithms ([6, 19, 14]). The gradient-based techniques are amongst the most accurate and robust strategies to calculate the 2D flow field. They rely on the so-called optical flow constraint which relates the brightness gradient with the vector field, \( h(x) = (u(x), v(x)) \).

The determination of optical flow is a classic ill-posed problem in computer vision and it requires additional regularizing assumptions. The regularization by [17] reflects the assumption that the optical flow field varies smoothly in space. However, since many natural image sequences are better described in terms of piecewise smooth flow fields, much research has been done to modify the Horn and Schunck approach to permit discontinuous flow fields ([24], [28], [4], [8], [12], [33], [26]).

### 2.2 Standard variational formulation

The 2D flow computation is carried out using a PDE based optical flow technique described in [2]. It consists in minimizing an energy defined as a weighted sum of 2 terms: a data term and a regularization term. The data term assumes that the images have similar intensities at the corresponding points and the regularization term assumes smoothness of the fluid flow.

The regularization term uses the approach proposed by [24], with the following improvements: (i) the formulation avoids inconsistencies caused by centering the brightness term and the smoothness term in different images, (ii) it uses a coarse to fine linear scale-space strategy to avoid convergence to physically irrelevant local minima, and (iii) it creates an energy functional that is invariant under linear brightness changes.

The energy to minimize is written as:

\[
E(h) = \int_{\mathbb{R}^2} (I_1(x) - I_2(x + h))^2 \, dx + C \int_{\mathbb{R}^2} \text{tr}(\nabla h^t D \nabla h) \, dx,
\]

where \( x \) is a point in \( \mathbb{R}^2 \), \( h = h(x) = (u(x), v(x))^t \) is the displacement field that we are looking for, \( I_1 \) and \( I_2 \) are the two input images, \( \text{tr}(\cdot) \) is the trace operator, \( C \) is a constant that weights the smoothing term, \( \nabla \) is the gradient operator, and \( D \) is a regularized projection matrix in the direction orthogonal to \( \nabla I_1 \).

The matrix \( D \) is expressed as:

\[
D(\nabla I_1) = \frac{1}{\|\nabla I_1\|^2 + 2\lambda^2} \left( \xi \xi^t + \lambda^2 \text{Id} \right),
\]

where \( \xi = \left( \frac{\partial I_1}{\partial y}, -\frac{\partial I_1}{\partial x} \right)^t \) is a vector orthogonal to \( \nabla I_1 \). The associated Euler-Lagrange equations
are given by the following PDE system:

\[ C \text{div}(D \nabla u) + (I_1(x) - I_2(x + h)) \frac{\partial I_2}{\partial x}(x + h) = 0 \]  
(3)

\[ C \text{div}(D \nabla v) + (I_1(x) - I_2(x + h)) \frac{\partial I_2}{\partial y}(x + h) = 0 \]  
(4)

The system is numerically solved using an iterative Gauss-Seidel algorithm detailed in [2].

### 2.3 Multi-channel flow computation

The variational methods proposed in the literature usually use information from a single channel. Normally, these methods are targeted to solve the optic flow problem using greyscale visual images (one channel). Although methods oriented to colour (multi-channel) images have been investigated, they are not very common.

The satellites have many sensors that capture images in different regions of the wave spectrum. This information is very useful for the estimation of the cloud motion. The clouds produce changes in the water vapour concentration, in the air pressure and in the thermal radiation from the earth. So, using the data given by these channels, we should be able to compute a robust and accurate solution for the cloud motion estimation.

We have extended the optical flow method described above to include information from several channels captured by the satellite sensors. In this section, we explain in details our variational (energy) model and the corresponding numerical scheme.

The energy model proposed for motion estimation using multichannel data is, as in the case of a single channel, based in the addition of two terms: the data term and the smoothness term. Our input data will be pairs of images of different channels. Our energy has to combine the channel information in some way. We have included in the energy a set of weights that specifies the relevance of each channel. The data term takes into account information from all the channels. Thus it consists in a combination of differences between two images weighted by positive real numbers \( \rho_c \), where \( c \in [1,N_c] \) is the channel associated to this number and \( N_c \) is the number of channels. In the single channel method, the Nagel-Enkelmann operator uses the image gradient to decide the direction and the amount of diffusion. In the multichannel method, we want to keep this idea combining the data from different channels.

The energy to minimize is written as:

\[
E(\mathbf{h}) = \int_{\mathbb{R}^2} \sum_{c=1}^{N_c} \rho_c (I^c_1(x) - I^c_2(x + h))^2 dx + C \int_{\mathbb{R}^2} \text{tr}(\nabla h^t D \nabla h) dx,
\]  
(5)

where \( I^c_1 \) and \( I^c_2 \) are the first and second images in the channel \( c \).

In the single channel method the matrix \( D \) is expressed as:

\[
D(\nabla I) = \frac{1}{\|\nabla I\|^2 + 2\lambda^2} (\xi \xi^t + \lambda^2 \mathbf{I}),
\]  
(6)

where \( \xi = (\frac{\partial \nabla I}{\partial y}, -\frac{\partial \nabla I}{\partial x})^t \) is a vector orthogonal to \( \nabla I \). In order to define matrix \( D \) for the multichannel method, we need to define a single vector \( \mathbf{g} \) which plays the role of \( \nabla I \). To define \( \mathbf{g} \) we propose two strategies:
2.3 Multi-channel flow computation

- **Maximum gradient.** At each pixel location, \( \hat{g} \) is computed as the gradient of greatest magnitude among the gradients of all the channels.

\[
\arg\max \{ \|\vec{v}\|, \vec{v} \in \{\nabla I_c, c \in [1, N_c]\}\}
\]

- **Average gradient.** \( \bar{g} \) is computed as a dominant direction in the set of the gradient vectors for all channels. Since the direction of the gradient vector is not relevant, the usual way to estimate the dominant orientation \( \bar{g} \) is using the named structure tensor. This structure tensor is defined as the matrix

\[
\sum_{c=1}^{N_c} \rho_c (\nabla I_c^* \cdot (\nabla I_c^*)^T)
\]

if we note by \( \varepsilon_{\text{max}} \) the normalized eigenvector associated to the maximum eigenvalue \( \lambda_{\text{max}} \) of the above matrix, then we can define \( \bar{g} \) as

\[
\bar{g} = \sqrt{\frac{\lambda_{\text{max}}}{\sum_{c=1}^{N_c} \rho_c}} \varepsilon_{\text{max}}
\]

In fact, we can show that \( \bar{g} \) is the minimum, under the constraint \( \|\bar{g}\| = \sqrt{\lambda_{\text{max}}/\sum_{c=1}^{N_c} \rho_c} \), of the following energy:

\[
E(\bar{g}) = -\sum_{c=1}^{N_c} \rho_c (\bar{g}^T \cdot \nabla I_c^*)^2
\]

The Euler-Lagrange equations associated to the energy (5) are:

\[
Cd_{i,\nabla u} + \sum_{c=1}^{N_c} \rho_c (I_c^1(x) - I_c^2(x + h)) \frac{\partial I_c^2}{\partial x}(x + h) = 0 \quad (7)
\]

\[
Cd_{i,\nabla v} + \sum_{c=1}^{N_c} \rho_c (I_c^1(x) - I_c^2(x + h)) \frac{\partial I_c^2}{\partial y}(x + h) = 0 \quad (8)
\]

To discretize the above system of partial differential equations, we use an implicit finite difference scheme because it is more stable and converges faster than the usual explicit schemes.

For the formulation of the iterative procedure we introduce the following notations:

\[
I_{2,i,x}^{c,h} = \frac{\partial I_c^2}{\partial x}(x_i + h^k) \quad \text{and} \quad I_{2,i,y}^{c,h} = \frac{\partial I_c^2}{\partial y}(x_i + h^k).
\]

The matrix \( D \), introduced above, is written in each pixel \( i \) as:

\[
D_i = \begin{pmatrix}
  a_i & b_i \\
  b_i & c_i
\end{pmatrix}
\]

We can discretize the differential operator in each pixel \( i \) and we obtain:

\[
div(D_i \nabla h) = \begin{pmatrix}
  a_i \partial_x h + b_i \partial_y h \\
  b_i \partial_x h + c_i \partial_y h
\end{pmatrix} = \partial_x (a_i \partial_x h) + \partial_x (b_i \partial_y h) + \partial_y (b_i \partial_x h) + \partial_y (c_i \partial_y h)
\]
We define \( N_i^* \) as the set of \( 3 \times 3 \) neighbours around the pixel \( i \) excluding the pixel \( i \) itself. Using standard difference scheme we can write:

\[
\text{div}(D_i \nabla h_i) = \sum_{n \in N_i^*} (d_n h_n) + d_i h_i \tag{9}
\]

for suitable coefficients \( d_n \).

First, the terms of the form \( I(x + h^{k+1}) \) are linearized via Taylor expansion

\[
I_1^c(x) - I_2^c(x + h^k) \approx I_1^c(x) - I_2^c(x + h^k) - \frac{\partial I_2^c}{\partial x}(x + h^k)(u^{k+1} - u^k) - \frac{\partial I_2^c}{\partial y}(x + h^k)(v^{k+1} - v^k)
\]

Finally, the components of the vector displacement \((u_i, v_i)\) are obtained asymptotically by iterations of a Gauss-Seidel type scheme:

\[
u^{k+1} = u_i - \frac{dt}{C} \sum_{n \in N_i^*} (d_n u_n) + \sum_{c=1}^{N_c} \rho_c \left( I_1^c(x_i) - I_2^c(x_i + h^k) + u_i^k I_{2,i,x}^{c,h} - (v_i^{k+1} - v_i^k) I_{2,i,y}^{c,h} \right) \\
1 + dt \cdot \left( C d_i + \sum_{c=1}^{N_c} \rho_c \cdot \left( I_{2,i,x}^{c,h} \right)^2 \right) \tag{10}
\]

\[
v^{k+1} = v_i - \frac{dt}{C} \sum_{n \in N_i^*} (d_n v_n) + \sum_{c=1}^{N_c} \rho_c \left( I_1^c(x_i) - I_2^c(x_i + h^k) + v_i^k I_{2,i,y}^{c,h} - (u_i^{k+1} - u_i^k) I_{2,i,x}^{c,h} \right) \\
1 + dt \cdot \left( C d_i + \sum_{c=1}^{N_c} \rho_c \cdot \left( I_{2,i,y}^{c,h} \right)^2 \right)
\]

To resume, in figure 3, we present a scheme of the multichannel motion estimation algorithm.

In order to increase the convergence rate of the algorithm and to avoid to be trapped in spurious local minima, we use a multiresolution scheme, that is, we solve successively the system at different level of the image resolution, starting from the coarsest grid. In figure 4 we present a scheme of the multiresolution algorithm.

3 Altitude estimation of each cloud point.

The height of the clouds is computed using the technique described in [31], chapter 4: “Height assignment of motion vectors”. An approximation of the height of the clouds is computed from an estimation of the temperature based on the IR channel 10.8.

Let us denote the infrared intensity at the current pixel position as \( C \). The radiance \( R \) \((\text{mW} \cdot \text{m}^{-2} \cdot \text{sr}^{-1} \cdot \text{cm})\) is calculated as \( R = R_0 + \alpha C \), where \( R_0 \) and \( \alpha \) are included in the original MSG files. The brightness temperature \( T_b \) of the observed object can be approximated from the infrared channel, using the formula:

\[
T_b = \frac{1}{A} \left( \frac{(C_2\nu_c)}{\ln(1 + \frac{C_1\nu_c^2}{R})} - B \right), \tag{11}
\]
3 Altitude estimation of each cloud point.

Figure 3: Multichannel motion estimation algorithm.

where \( C_1 = 1.19104 \times 10^{-5} \text{ mW.m}^{-2}.\text{sr}^{-1}.\text{cm}^4 \), \( C_2 = 1.43877 \text{ K.cm} \), and the central wavenumber \( \nu_c \), the parameters \( A \) (dimensionless) and \( B \) (in Kelvin) are constants given by Eumetsat for each satellite and channel. The altitude is then deduced from the temperature as:

\[
a = \frac{T - T_0}{\gamma},
\]

where \( T_0 = 288.15 \text{ K} \) \((15^\circ \text{C})\) is the approximate temperature at sea level, and \( \gamma = -6.510^{-3} \text{ K.m}^{-3} \) is the standard temperature change with respect to the height.

Figure 5 illustrates our 3D visualization. Each pixel of the scene is drawn at its 3D location based on its estimated height and on the parameters of the satellite. Using the cloud classification provided by Eumetsat, low, medium, high and very high clouds are displayed with colours ranging from red to dark blue. The estimated cloud motion is represented by 3D vectors which are displayed on a regular grid within the identified clouds. The 3D vectors are proportional to the estimated displacement and have a vertical component that represents the evolution of the clouds height. This component is based on the height estimation of two successive frames of the sequence, and on the 2D motion field. A video that illustrates our results is also available on internet ([1]) \(^2\).

**Remark: Dense 3D cloud motion estimation.** We observe that by combining the pixel altitude estimation and the dense 2D cloud motion we estimate a 3D cloud motion field.

\(^2\) http://serdis.dis.ulpgc.es/~krissian/HomePage/Demos/Fluid/Video/CVPR_VIDEO_AMI.mpg
Figure 4: Multiresolution Algorithm Scheme for 3 resolution levels.

Notation:
D : DownSampling Operator. The resolution of the images is divided by 2
U() : UpSampling Operator. The estimated motion at one resolution is extended to a higher resolution
ME : Multichannel motion estimation algorithm. (described in figure 3)
Figure 5: 3D view of the hurricane Vince, including colouring the clouds and 3D displacement vectors calculated from our multichannel motion estimation algorithm.

Figure 6: Altitude estimation algorithm
4 Cloud altitude regularization.

Meteorological satellites provide a cloud structure classification based on an estimation of the cloud structure altitude computed using a combination of the multichannel satellite image values (see figure 2 for illustration). In practice, we can assume that the classification areas \( L_i \ (i = 1, ..., N_L) \) are estimated as level set of a classification function \( f_L : \Omega \to R \), that is

\[
L_i = \{ \bar{x} \in \Omega : \lambda_{i-1} < f_L(\bar{x}) < \lambda_i \}
\]

where \( \lambda_0 < \lambda_1 < ... < \lambda_{N_L} \). Each classification area \( L_i \) represents a cloud structure layer. Usually there are 2 main problems concerning the classification areas \( L_i \). The first one is that the multichannel satellite image values are noisy and therefore the classification function \( f_L(.) \) is also noisy and the layers \( L_i \) require some kind of regularization. The second one is that, in order to analyze the cloud structures, we need to assume a model of interaction between the different layers \( L_i \). In this paper, we will assume the simplest case each layer \( L_i \) is at a different altitude and there is no interaction between the different layers. This assumption is usually true, but it obviously fails in the case of complex 3D atmospheric phenomena as, for instance, the hurricanes.

In order to regularize the boundary of the cloud layers \( L_i \), we propose to use a median type filter applied to the classification function \( f_L(.) \) that is, for each \( \bar{x} \), we define \( med(f_L)(\bar{x}) \) as:

\[
med(f_L)(\bar{x}) = \left\{ \lambda : \int_{B(\bar{x})} |f_L(\bar{y}) - \lambda| \, d\bar{y} \leq \int_{B(\bar{x})} |f_L(\bar{y}) - \nu| \, d\bar{y} \quad \forall \nu \in R \right\}
\]

where \( B(\bar{x}) \) is a neighbourhood of \( \bar{x} \). So we define the new classification layers \( L'_i \) as

\[
L'_i = \{ \bar{x} \in \Omega : \lambda_{i-1} < med(f_L)(\bar{x}) < \lambda_i \}
\]

this filter regularize the boundary of the layers \( L_i \) and it removes small isolated set in \( L_i \). In figure 7 we illustrate this behaviour.

In order to regularize the multichannel satellite image values \( I^c_j(\cdot) \), we propose to use a variational technique based on the following energy minimization

\[
E(u^c_j) = \int_{L'_i} (u^c_j(\bar{y}) - I^c_j(\bar{y}))^2 \, d\bar{y} + \alpha \int_{L'_i} \|\nabla u^c_j(\bar{y})\|^2 \, d\bar{y}
\]

the parameter \( \alpha \) represents the weight of the regularization process. Since we assume no interaction between the different layers \( L'_i \), we will consider homogeneous Neumann type boundary condition. The Euler-Lagrange equation associated to the above energy is given by:

\[
\begin{cases}
-\alpha \Delta u^c_j(\bar{x}) + u^c_j(\bar{x}) = I^c_j(\bar{x}) & \text{in } L'_i \\
\frac{\partial u^c_j}{\partial n}(\bar{x}) = 0 & \text{in } \partial L'_i
\end{cases}
\]

The solution \( u^c_j(\cdot) \) of the above differential equation represents the smoothed version of the channel \( I^c_j(\bar{x}) \) in the classification area \( L'_i \). We observe that since we regularize \( I^c_j(.) \) in an independent way in each classification area \( L'_i \) the discontinuities across the boundary of the cloud layers \( L'_i \) of the satellite image value \( I^c_j(.) \) are preserved. Figure 7 illustrates this behaviour.
Figure 7: On the left, we present the original cloud structure layer where the altitude is computed from the channel IR 10.8. On the right we present the results obtained with the proposed methods. We observe the regularization of the cloud structure layer boundary as well as the regularization of the height.

5 Software for satellite image sequence processing

In this section, we present an overview of the main C-functions we have implemented to process the satellite image sequence.

5.1 Basic correlation flow estimation technique.

```c
void ami_optic_flow_correlation(
    float **image1, float **image2, /* input images */
    int width, int height, /* dimension of the images */
    float **u, float **v, /* output computed flow */
    int x_init, int y_init, /* initial coordinate point to compute the correlation */
    int x_interval_size, int y_interval_size, /* displacement between points for computing the correlation */
    int size_correlation_window) /* size of correlation window, it has to be a power of 2 */
```

5.2 Basic PDE flow estimation technique.

```c
void ami_optic_flow_pde(
    long height, /* image dimension in x direction */
    long width, /* image dimension in y direction */
    float lix, /* pixel size in x direction */
    float hy, /* pixel size in y direction */
    float *img1, /* first image, unchanged */
    float *img2, /* second image, unchanged */
    float sigma, /* Gaussian scale smoothing image data */
    float T, /* stopping time */
    float alpha, /* smoothness weight */
    long dtype, /* type of smoothness term */
```
5.3 Multichannel PDE flow estimation technique.

```c
void ami_optic_flow_multichannel_pde

(long height, /* image dimension in x direction */
 long width, /* image dimension in y direction */
 float hx, /* pixel size in x direction */
 float hy, /* pixel size in y direction */
 float **fl, /* first channel image sequence, unchanged */
 float **f2, /* second channel image sequence, unchanged */
 int Nc, /* number of image channels */
 float *channel_weight, /* weight for each image channel */
 float sigma, /* Gaussian scale smoothing image data */
 float T, /* stopping time */
 float alpha, /* smoothness weight */
 long dtype, /* type of smoothness term */
 float quantile, /* isotropy fraction */
 float ht, /* time step size */
 float ***u, /* x component of optic flow, changed */
 float ***v, /* y component of optic flow, changed */
 int Niter, /* Number of Iterations for Gauss-Seidel */
 long Nzoom) /* Number of zooms to apply in the pyramidal approach */;
```

5.4 Function to remove noise and outliers

```c
void ami_noise_median_lter_2df_clasication ( 
    float *image, /* Original image (input and output data) */
    unsigned char *cla, /* Classification image */
    int dimX, int dimY, /* Image dimensions */
    int windowRadius, /* Radius of the window we use to compute the median */
    float percentage, /* If relative error (image value point, median) > percentage we apply median filter */
    int nIter, /* Number of the times we apply the procedure */
    int xInit, int yInit, /* Initial point of the lattice */
    int xIntervalSize, int yIntervalSize, /* Point displacement in the lattice */
    unsigned char umbralClassification /* Umbra of clasification image */
);
```

6 Software for 3D visualization of cloud structures

The 3D visualization is based on OpenGL, using our software AMILab available at [http://serdis.dis.ulpgc.es/~krissiam/HomePage/Software/AMILab/](http://serdis.dis.ulpgc.es/~krissiam/HomePage/Software/AMILab/).

Fig. 8 illustrates the different tasks and their inputs.

Fig. 9 shows on the left the 3D layer decomposition obtained using the EUMETSAT original information. On the right, we display the effect our noise reducing filters applied to both the
6.1 Installing AMILab

This section is a tutorial on the use of AMILab for the visualization of cloud structures provided by EUMETSAT. The tutorial explains all the steps needed to generate a 3D animation of the different clouds and of the 3D displacement vectors.

6.1 Installing AMILab

The software is maintained for Linux operating system. You can download the current pre-compiled version at:

http://serdis.dis.ulpgc.es/~krissian/HomePage/Software/AMILab/

The file "AMILab_linux.tgz" contains a directory AMILab_Linux_2.6.6/YYYY_MM_DD/ with the following subdirectories:

- **bin**: contains the binary called 'amilab_Linux_2.6.6'
- **libs**: contains dynamic libraries needed to run amilab

classification image and the temperature channel and the estimated flow for the highest class of clouds. The vertical component of the vectors represents the evolution of the clouds altitude. A video that illustrates our results is available at http://serdis.dis.ulpgc.es/~krissian/HomePage/Demos/Fluid/Video/CVPR_VIDEO_AMI.mpg

Figure 8: Processing and visualization of satellite data

Figure 9: Left: coloring the clouds, right: displacement field as 3D vectors.
• Scripts: current scripts used for different image processing or visualization tasks.

You can uncompress it in a directory of your choice: INSTALLDIR, using:

```bash
> tar zxf AMILab_lin ux.tgz
```

### 6.1.1 Set your dynamic libraries path

In order to run the program, you have to add the path of the dynamic libraries to your variable `LD_LIBRARY_PATH`, if you use tcsh you can do it with:

```bash
> setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:INSTALLDIR/AMILab_Linux_2.6.6_YYYY_MM_DD/libs
```

or

```bash
> export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:INSTALLDIR/AMILab_Linux_2.6.6_YYYY_MM_DD/libs
```

If you don’t have an existing `LD_LIBRARY_PATH` variable in your environment then you can use

```bash
> setenv LD_LIBRARY_PATH INSTALLDIR/AMILab_Linux_2.6.6_YYYY_MM_DD/libs
```

or

```bash
> export LD_LIBRARY_PATH=INSTALLDIR/AMILab_Linux_2.6.6_YYYY_MM_DD/libs
```

### 6.1.2 Set your path

In the same way, you can add the ’bin’ directory to your ’PATH’ environment variable. For example with tcsh:

```bash
> setenv PATH ${PATH}:INSTALLDIR/AMILab_Linux_2.6.6_YYYY_MM_DD/bin
```

### 6.1.3 Set the AMILab script directory

In the same way, you can add the ’bin’ directory to your ’PATH’ environment variable. For example with tcsh:

```bash
> setenv AMI_SCRIPTS INSTALLDIR/AMILab_Linux_2.6.6_YYYY_MM_DD/Scripts
```

The script directory allows to load a script from this directory using the ’func’ command without having to specify the whole path.

### 6.1.4 Automatic configuration

If you want your environment to be automatically set up for AMILab, you should add all the necessary configuration commands to your file ’.tcshrc’ or ’.bashrc’ in your home directory.
6.1.5 Run AMILab

If you have set up the PATH variable in your environment, you can run the program from any directory by typing:

```
> amilab_Linux_2.6.6
```

If it does not start, you can send an email to 'krissian@dis.ulpgc.es'. Each time you run AMILab from a given directory, the program will create a sub-directory called '.improcess' (if it does not already exists), and a new file called 'improcess worksheet XXX' in this directory, where XXX is an increasing number starting at 000 the first time you run AMILab. This file will automatically save every command you write within AMILab, working as a backup and a history of your commands.

6.1.6 More documentation

AMILab is a scripting language, fully interpreted, which can be compared in some sense to more complex well known languages as MATLAB or Python, although it has much less features. One of the motivations for creating AMILab was to manipulate the images in an intuitive way, combined with good visualization tools for both 2D and 3D images and surfaces.

A first documentation of AMILab can be found downloading docamil.pdf. It is not updated but can be of some help.

AMILab comes with two main visualization functionalities:

- An image viewer: SliceView.
- A surface viewer: GLView.

Both viewers are integrated in AMILab in a natural way and they can interact.

6.2 Reading the images

AMILab can read and write several standard image formats, it uses the Visualization ToolKit (VTK), and ImageMagick to try to read and write unknown formats. It can also read/write the format from INRIA, and it has its own format, which consists of an ASCII header followed by the raw data, or pointing to the raw data in another file (or several files for 3D or 2D+T data sets). The program can read raw data compressed with gzip: if the file is not found, it looks for the same filename with a '.gz' extension. If it can find it, it will open the file using the 'popen' (pipe open) command and the 'gunzip' program.

Most of the data from the Fluid project comes as raw data, for example 'WP01/package_02_LMD/images/AfGG_c0.81.xx'. In order to read this data with AMILab, one option is to create an ASCII header which contains the properties of the images. To read the sequence of images into a single volume data, we used the following header, written into a file called “AfGG_c0.81.ami”:

AMILAB 1.1
The header consists of a list of lines containing information of type 'keyword = value'. The keywords have upper and lower cases, and the lower cases can be omitted. If a keyword is not specified in the header, a default value will be used. Here is a detailed description of this header:

- XDim, YDim, ZDim are the dimensions of the image, in the case of a series of images, ZDim is not relevant and it is calculated automatically based on the number of images.
- VoxSizeX, VoxSizeY, VoxSizeZ are the dimensions of the voxels (or pixels in 2D) in a given unit (for medical images, it is usually in mm).
- TranslX, TranslY, TranslZ define the translation of the first point (0,0,0) of the image in the same unit as the voxel size.
- Repres is the representation type, the possible types are: BIN, UCHAR, SCHAR, USHORT, SSHORT, UINT, SINT, ULONG, SLONG, FLOAT and DOUBLE.
- Type can be SCALAR, 2DVECTOR, 3DVECTOR, RGB, 2DSYMMAT, 3DSYMMAT and COMPLEX. The current version of AMILab only supports SCALAR, 3DVECTOR and RGB.
- Endianness can be BIG or LITTLE, this allows to read images which have been saved from another Operating System with different endianness.
- ScanOrder is not used yet, it will allow to orient the image for the display.
- DF means DataFile and can be INTERN or EXTERN. The default value is INTERN, which means that the header and the data are in the same file, EXTERN means that the raw data is in another file.
- FF mean FileFormat, it tells the program where the raw data is. For a sequence of images, the filenames of each image will be generated from the slice number using the same syntax as 'printf' from the C language to generate the filename from the file format and the integer number (for example "%d" will write the number and "%04d" will write the number using 4 characters and preceded by zeros).
6.3 Tutorial for FLUID

In order to follow this tutorial, you should install some files of Work Package 1 (WP01), from the Fluid website.

6.3.1 Install the tutorial files

Here is how to do it: choose a working directory, let’s call it ‘DIR’ and write:

```
> cd DIR
> mkdir WP01
> mkdir WP01/package_02_LMD
> cd WP01/package_02_LMD

in DIR/ WP01/package_02_LMD download and un tar the file tar_WP1_LMD_images.tar.

> tar xf tar_WP1_LMD_images.tar
```

Then download and un tar the files needed for the tutorial:

```
> cd ../..
> wget http://serdis.dis.ulpgc.es/~krissian/HomePage/Software/AMILab/Tutorial/Fluid/FluidAMILabTutorial.tgz
> tar zxf FluidAMILabTutorial.tgz
```

6.3.2 Set the fluid navigation path

The fluid navigation path is the directory which contains the navigation definition, you can define it using for example (for tcsh):

```
> setenv FLUID_NAV_PATH DIR
```

By default, the program will look for the navigation file in the directory where it was run, and the navigation file is supposed to be ‘nav_MSG_02_05_2003c0_63.dat’. This file is provided with the tutorial files. The command 'SetFluidNavFile(string)' allows to change the navigation file. If the file is not found, then the program will look for it in the directory $FLUID_NAV_PATH.

6.3.3 Loading an image

To load an image, you can use the Image command (subsection 6.5):

```
i47 = Image "AfGG_c0.81.47.ami"
i   = Image "AfGG_c0.81.ami"
```
"i47" contains only one 2D image, whereas 'i' contains all the images from 47 to 55 and is considered as a 3D image.

Another option to read the data directly without a header image is to use the ReadRawImages command (subsection 6.6):

```plaintext
i47 = ReadRawImages(1024,1024,USHORT,0,"WP01/package_02_LMD/images/AfGG_c0.81.47",0,0)
i = ReadRawImages(1024,1024,USHORT,0,"WP01/package_02_LMD/images/AfGG_c0.81.%d",47,55)
```

In the first case, because i47 only reads one 2D image, the first and last slices are not used and we set those parameters to 0. Once an image is read, you can print the information about the image using the 'info' command:

```plaintext
i47.info
i.info
```

it will display the result in the terminal from which AMILab was executed. For example, 'i.info' will display

```plaintext
> format=UNSIGNED SHORT dim=(1024,1024,9) vox=(1.000000,1.000000,1.000000) translation=(0.000000,0.000000,0.000000)
```

### 6.3.4 Displaying an image

The display of the image is obtained using the 'show' command:

```plaintext
show i
```

It will create a window and a variable called 'i_draw' for this window. To see the list of variable in the environment, you can use the command

```plaintext
show vars
```

Because the image 'i' is 3D, it is possible to animate the image as a movie by selecting the option 'Options → Option → Animation'. The animation can be 'forward', 'backward', or 'auto-reverse' and its speed can be controlled from the 'Animation Parameters' menu (see the documentation of the image viewer for more information).

![Figure 10: Display of a satellite image.](image)
6.3.5 Displaying a satellite image in 3D

We suppose that the image i47 is already loaded, here is how to display the data in 3D:

\[ i47 = \text{ReadRawImages}(1024,1024,\text{USHORT},0,"\text{WP01/package_02_LMD/images/AfGG_c10.8.47",0,0}) \]

We read the image of temperatures.

\[ \text{coeff} = \text{Image} "\text{alt\_coeff.ami.gz}" \]

The file ’alt\_coeff.ami.gz’ is an image which contains 5 float values (1D image), those coefficients allow to calculate the altitudes.

\[ \text{alt47} = \text{ComputeAltitudes}(i47,\text{coeff}) \ast 4 \]

We calculate an estimation of the altitudes based on the temperature and we scale them by 4 for the visualization.

\[ \text{earth} = \text{CreateFlatMesh}(i47>0) \]

’CreateFlatMesh’ creates a triangular mesh from a grayscale 2D image. The mesh is flat and lies in a plan.

\[ \text{pos} = \text{Altitude2Position}(\text{alt47},1) \]

’pos’ is an image containing the real positions in 3D.

\[ \text{earth}.\text{SetColors}(i47,\text{min}(i47),\text{max}(i47)) \]

We don’t use texture mapping, but we assign a grayscale color per vertex in the visible image ’i47’ and linearly mapping its intensity from the min and max to black and white.

\[ \text{earth}.\text{ElevateMesh}(\text{pos}) \]

We elevate the mesh, changing the position of each vertex based on the pre-calculated positions in the image ’pos’.

\[ \text{earth}.\text{SetColorMaterial}(1) \]
\[ \text{earth}.\text{Normals} \]
\[ \text{earth}.\text{Recompute} \]
\[ \text{show} \text{earth} \]
\[ \text{earth}_\text{draw}.\text{rotate}(0,90,0) \]
\[ \text{earth}_\text{draw}.\text{rotate}(0,0,90) \]

We allow color material, recompute normals for the new mesh, display it and orient it.

The ’file Tutorial/tutorial_display3D.amil’ contains the previous commands.

6.3.6 Selecting a region of interest

We can reduce the image to a subregion using the ’[ ]’ operator, this operator can be used in different ways, one way for a 2D image is ’image[xmin:xmax,ymin:ymax]’. For example:

\[ \text{si47} = i47[100:300,100:300] \]
\[ \text{st47} = t47[100:300,100:300] \]
The new image stores the information of its position. Running the command 'si47.info' will show the translation of this image. We then rerun the previous commands on the sub-image:

```plaintext
salt47 = ComputeAltitudes(st47,coeff)*4
searth = CreateFlatMesh(si47>=0)
spos = Altitude2Position(salt47,1)
searth.SetColors(si47,min(si47),max(si47))
searth.ElevateMesh(pos)
searth.SetColorMaterial(1)
searth.Normals
searth.Recompute
show searth
```

and we can compare the two supercies with the 'compare' command:

```plaintext
earth_draw.compare(searth_draw)
```

Changes of window size position, zoom, orientation will be automatically applied to the compared window ('searth_draw' in this case).

### 6.3.7 Resampling the image

The image can be resampled using the 'Resize' command. 'Resize' will update the voxel (or pixel) size of the new image.

```plaintext
i47_2 = Resize(i47,512,512,1,0)
```

The last parameter select the kind of interpolation used (0 for closest point, 1 for linear).

### 6.3.8 Saving and Setting 3D views

You can save, load and set 3D views using Transform, GetTransform and SetTransform commands. To generate continuous transitions between 2 views, we also create a command that interpolates views between 2 selected views called Interpolate. Here is an example:

```plaintext
show earth
t1 = earth_drow.GetTransform
```

You can move the model using the mouse, save a second view, and come back to the first view:
t2 = earth_draw.GetTransform 
earth_draw.SetTransform(t1)

You can save both views in files, load them and compare them using the ‘print’ command. 
The ‘print’ command displays the information of the transform as a combination of Rotation, 
Translation and Scale. It prints the information on the standard output (the terminal from which 
AMILab was run).

t1.save "t1.mat"
t2.save "t2.mat"
t1_2 = Transform("t1.mat")
t1.print
t1_2.print

Here is how to generate a continuous transition between the first and the second views:

```plaintext
for n=1 to 20 {
    t3 = Interpolate(t1,t2,n/20)
    earth_draw.SetTransform(t3)
    del t3
}
```

We need to delete the variable ‘t3’ because the language does not accept to overwrite an existing 
transform (or view).

### 6.3.9 Saving an animation

To save an animation in a standard format (like MPEG), we first save a set of snapshots from 
the drawing window into a 3D image in RGB format. We can then run a script that converts 
the images into a series of 2D images in a standard format (like jpeg), and another script that 
saves the series of images into an animation. The following lines initialize the 3D image that 
will contain the animation and saves the first image. It uses ‘getimage’ which takes a snapshot 
of the 3D display using OpenGL. Another equivalent command is ‘GetImageFromX’, which uses 
X11 to get the snapshot. ‘putimage’ pastes an image into another image at a given location or 
based on the image translation information if no position is given.

```plaintext
nb_images=20
earth_draw.SetTransform(t1)
im1 = earth_draw.getimage
anim = Image(RGB,im1.tx,im1.ty,nb_images)
anim_current = 0
anim.putimage(im1,0,0,anim_current)
```

Once the animation initialized, we save all the remaining 20 images. The variable ‘anim_current’ 
is the index of the current image to be saved.

```plaintext
for n=1 to nb_images−1 {
    t3 = Interpolate(t1,t2,n/(nb_images−1))
    earth_draw.SetTransform(t3)
    del t3
    im1 = earth_draw.getimage
    anim.putimage(im1,0,0,anim_current)
    anim_current = anim_current + 1
}
```
The 3D image can be seen using the command 'anim.show', it can also be saved as a series of 2D images using for example:

```plaintext
for n=0 to n_b_images − 1 {
    im1 = anim[:,:, n : n + 1] // extract one image from the sequence
    im1.save sprint("tutorial_anim.%03.0f.jpg",n)
}
```

The command 'sprint' is similar to 'sprintf' in C. Because numbers return float values by default, the syntax "%03.0f" will return an integer of 3 characters including leading zeros. This loop will save the files: tutorial_anim.000.jpg, ..., tutorial_anim.020.jpg.

Next we can use the script `mkmpg4` written in Python to generate the video using for example 8 frames per seconds:

```
> mkmpg4 −f 8 −o tutorial_anim.mpg tutorial_anim∗.jpg
```

6.3.10 Coloring the clouds

**Using one mesh per category of clouds** One way to color the clouds is to generate one mesh per category of cloud and then to set a different ambient color for each mesh, disabling the colormaterial option. This is achieved by the following script 'Tutorial/tutorial_coloring.ami'.

First, we read all the files that we need: the visual image, the classification image, the temperature image and the coefficients:

```plaintext
i48 = ReadRawImages(1024,1024,USHORT,0,"WP01/package_02_LMD/images/AfGG_c0.81.48",0,0)
c48 = ReadRawImages(1024,1024, UCHAR,0,"WP01/package_02_LMD/misc/AfGG_CLA_scene_analysis.48",0,0)
t48 = ReadRawImages(1024,1024,USHORT,0,"WP01/package_02_LMD/images/AfGG_c10.8.48",0,0)
coeff = Image "alt_coeff.ami.gz"
```

Second, we create flat meshes for each class, using `CreateFlatMesh` and based on the intensity of the classification image. The mesh created using bilinear interpolation on the image intensity. We introduce an epsilon to separate the different meshes with a small space (epsilon=0.3). The variable 'clouds' is declared as an array of 5 supercies, for the 5 different categories of clouds.

```plaintext
epsilon = 0.5 − 0.2
earth = CreateFlatMesh(c48,0,100−epsilon)
clouds = Surface[5]
clouds[0] = CreateFlatMesh(c48,100−epsilon,100+epsilon)
clouds[1] = CreateFlatMesh(c48,101−epsilon,103+epsilon)
clouds[2] = CreateFlatMesh(c48,104−epsilon,106+epsilon)
clouds[3] = CreateFlatMesh(c48,107−epsilon,109+epsilon)
clouds[4] = CreateFlatMesh(c48,110−epsilon,200)
```

Then, we elevate the meshes based on the 3D position of the pixels, scaling the estimated altitudes by a factor 4. We color the 'earth' using the visual image information and we enable colormaterial for it.

```plaintext
alt_factor = 4
alt48 = ComputeAltitudes(t48,coeff)
alt48 = alt48−alt_factor
```
6.3 Tutorial for FLUID

```plaintext
pos = Altitude2Position(alt48, 1)
earth.SetColors(i48, min(i48), max(i48))
earth.ElevateMesh(pos)
earth.Normals
earth.SetColorMaterial(1)
show earth
for n = 0 to 4 {
    clouds[n].ElevateMesh(pos)
    clouds[n].Normals
    earth_draw += clouds[n]
}

Finally, we set an grey light to avoid affecting the colors, we set a standard color for each class of clouds, and we re-orient the scene.

earth_draw.SetLightDiffuse(0, 180, 180, 180)

# Set Colors of the different clouds
c0 = SetAmbient(195, 195, 195)
c1 = SetAmbient(224, 0, 0)
c2 = SetAmbient(222, 222, 0)
c3 = SetAmbient(0, 254, 252)
c4 = SetAmbient(195, 195, 195)
earth_draw.rotate(0, -90, 0)

Figure 12: Coloring the clouds, two different views.

Using only one mesh Another way is to generate a RGB image containing all the colors and to set these colors to only one supercy. This is achieved by the script 'Tutorial/tutorial_coloring_onemesh.ami':

```
$$i48_{\min} = \text{min}(i48)$$
$$i48_{\max} = \text{max}(i48)$$
$$i48_{\text{uchar}} = \frac{(\text{FLOAT}\cdot i48 - i48_{\min})}{(i48_{\max} - i48_{\min})} \cdot 255$$

$$c1[0] = (c48 < 101) \cdot i48_{\text{uchar}} + (c48 \geq 101) \cdot (c48 \leq 103) \cdot 224 + (c48 \geq 104) \cdot (c48 \leq 106) \cdot 222$$
$$c1[1] = (c48 < 101) \cdot i48_{\text{uchar}} + (c48 \geq 104) \cdot (c48 \leq 106) \cdot 222 + (c48 \geq 107) \cdot (c48 \leq 109) \cdot 254$$
$$c1[2] = (c48 < 101) \cdot i48_{\text{uchar}} + (c48 \geq 107) \cdot (c48 \leq 109) \cdot 252$$

# set the colors to the vertices of the mesh
earth.SetColors(c1)

# Elevate the mesh
alt_factor = 4
alt48 = ComputeAltitudes(t48, coeff)
alt48 = alt48 \cdot alt_factor
pos = Altitude2Position(alt48, 1)
earth.ElevateMesh(pos)
earth.Normals
earth.SetColorMaterial(1)

show earth

6.3.11 Drawing 3D vectors

Vectors showing the evolution of the clouds can be displayed in 3D. For this purpose, we created the function 'CreateVectors' which will display the vectors in 3D from the estimated flow. To avoid loading too many show the graphic card, each vector is generated by only 2 tetrahedra (6 triangles), one for the head and one for the tail. The vectors showing the displacement between to images need the estimated altitudes from both images. The distance between successive vectors in term of pixels can be selected for both X and Y directions. The user can also choose a scaling factor for the vectors, which are drawn proportional to their magnitude.

In addition to the previous script:

$$t49 = \text{ReadRawImages}(1024, 1024, \text{USHORT}, 0, "WP01/package_02_LMD/images/AfGG_c10.8.49", 0)$$
coef = Image "alt_coef.ami.gz"
alt49 = ComputeAltitudes(t49, coeff) \cdot alt_factor

# Read the flow
displ = ReadFlow("test_arrows/01test_AfGG_SF_4_0.3_0.0_0.0_0.3_.uv_v2.txt")

# Create the vectors
mask = (c48 \geq 107) \cdot (c48 \leq 109)
vector = CreateVectors(alt48 \cdot mask, displ \cdot mask, alt49 \cdot mask, 6, 6, 3, 1)
vector.SetAmbient(10, 10, 200)

earth_draw += vector

The image 'mask' allows to select only vectors belonging to the higher clouds.

6.4 Example of a video

In this section, we detail the script which allows to create a video showing the data from the satellite image in 3D, from different points of view and with the displacement vectors in 3D. In order to generate this video, several commands have been added to AMILab.

These commands are: CreateFlatMesh, ElevateMesh, ComputeAltitudes, Altitude2Position, ReadFlow, CreateVectors. In order to manipulate 3D views we also created the following
Figure 13: Selected views displaying the displacement field as 3D vectors. Left, with colormap and disabled transparency. Right, with colormap.

commands: Transform, SetTransform, GetTransform, Interpolate (documentation is available in appendix).

The following script, which is divided in several listings, is available as 'Tutorial/tutorial_animation amatil'.

Listing 1: Initialization

```plaintext
filtering = 0
type_string="AfGG"
type_image=1
# Adds an 2D image to the animation
# anim and anim_current are global variables
# if the animation 'anim' is full, add 20 more images
proc AnimAddImage( IMAGE newim) {
  #
  if (anim_current<anim.tz) {
    anim.putimage(newim,0,0,anim_current)
    anim_current=anim_current+1;
  }
  else {
    global
    _tmpim = Image(RGB,anim.tx,anim.ty,anim.tz+100);
    print sprint("new animation size: %03.0f
",_tmpim.tz)
    _tmpim.putimage(anim,0,0,0)
    del anim
    _tmpim.putimage(newim,0,0,anim_current)
    anim_current=anim_current+1;
  }
}
# rescaling of the altitudes
alt_factor = 4
```

Listing 2: Loading the images

```plaintext
imdir ="WP01/package_02_LMD/images/"
clouddir="WP01/package_02_LMD/misc/"
if = RealRawImages(1024,1024,USHORT,0,imdir+"AfGG_c0.81.48",0,0)
# Without filtering
```
c48 = ReadRawImages(1024,1024, UCHAR, 0, clouddir + "AfGG_CLA_scene_analysis.48", 0, 0)
t48 = ReadRawImages(1024,1024, USHORT, 0, imdir + "AfGG_c10.8.49", 0, 0)
i48min = min(i48)
i48max = max(i48)

Listing 3: Creating the flat meshes

```plaintext
# Creates the mesh for the earth
earth = CreateFlatMesh(c48 >= 0)
# Creates the meshes for the different types of clouds
# epsilon allows letting some space between the different
# categories of clouds
epsilon = 0.5 - 0.2
clouds = Surface[5]
clouds[0] = CreateFlatMesh(c48, 100 - epsilon, 100 + epsilon)
clouds[1] = CreateFlatMesh(c48, 101 - epsilon, 103 + epsilon)
clouds[2] = CreateFlatMesh(c48, 104 - epsilon, 106 + epsilon)
clouds[3] = CreateFlatMesh(c48, 107 - epsilon, 109 + epsilon)
clouds[4] = CreateFlatMesh(c48, 110 - epsilon, 200)
```

Listing 4: Elevating the meshes

```plaintext
# proc ProcessMesh(SURFACE _s, IMAGE _j1, NUM _j1min, NUM _j1max, IMAGE _pos) {
    _s.SetColors(_j1, _j1min, _j1max)
    _s.ElevateMesh(_pos)
    _s.Normals
    _s.Recompute
}
proc ProcessMesh(SURFACE _s, IMAGE _j1, NUM _j1min, NUM _j1max, IMAGE _pos) {
    _s.SetColors(_j1, _j1min, _j1max)
    _s.ElevateMesh(_pos)
    _s.Normals
    _s.Recompute
}
```

Listing 5: Visualizing the meshes

```plaintext
# Add all the meshes into the same visualization window
show earth
for k=0 to 4 { earth_draw += clouds[k]; }
earth_draw.SetLightDiffuse(0,180,180,180)
# Set color material ON
earth.SetColorMaterial(1)
for k=0 to 4 { clouds[k].SetColorMaterial(1); }
earth_draw.rotate(0,-90,0)
earth_draw.rotate(0,0,-90)
# Set the view
```
```plaintext
view1 = Transform("view1.txt")
earth_draw.Transform(view1)
earth_draw.Paint
earth_draw.SetBackgroundColor(5,5,40)
earth_draw.SetWindowSize(600,670)

Listing 6: Setting the colors of the clouds

Listing 7: Moving the point of view
```

# Saving a screen shot of the image with initial colors
im1 = earth_draw.GetImageFromX

# Creating the image containing the whole animation
anim = Image(RGB,im1.tx,im1.ty,300)
anim_current=0

# 1st part: from 2D image to 2D image with colors
# Set Colors of the different clouds
clouds [0]. SetAmbient( 195, 195, 195 )
clouds [1]. SetAmbient( 224, 0, 0 )
clouds [2]. SetAmbient( 222, 222, 0 )
clouds [3]. SetAmbient( 0, 254, 252 )
clouds [4]. SetAmbient( 195, 195, 195 )
for k=0 to 4 {clouds[k].SetColorMaterial(0);} 
earth_draw.Paint

# Saving a screen shot of the image with colors corresponding
# to each category of clouds
im2 = earth_draw.GetImageFromX

# Recording intermediate images between im1 and im2
# so that the colors will appear progressively

total_class = 40
for k=0 to total_class-1 {
    tmp = k/(total_class-1)
    im = im1*(1-tmp) + im2*tmp
    AnimAddImage(im)
}
del tmp

proc GenerateViews(STRING _st1, STRING _st2, NUM _total) {
    # Load the 2 views from files _st1 and _st2,
    # and creates a continuous transformation between them
    # using _total images.
    # All the images are added to the global animation 'anim'
    # and the global variable anim_current is updated.
    _viewinit = Transform(_st1)
    _viewend = Transform(_st2)
    for _n=0 to _total-1 {
        _vtmp = Interpolate(_viewinit,_viewend,_n/(_total-1))
        earth_draw.SetTransform(_vtmp)
        del _vtmp
        _im1 = earth_draw.GetImageFromX
        AnimAddImage(_im1)
    }
del _viewinit
```
Next we will present an overview of the primitives used by AMILab Software.
6.5 Image

***** Image *****

Tokens:
OBJ_IMAGE Image

Rules:
1. image -> OBJ_IMAGE ASTRING
2. image -> OBJ_IMAGE ( expr_string )
3. image -> OBJ_IMAGE
4. image -> OBJ_IMAGE ( basic_type, expr, expr, expr )

Description:
These rules allow to create a new variable. The first rule reads an image from the disk, where the image name is given by a string. The program can read several image formats. It can read all image formats accepted by the library ImageMagick, our own AMImage format, and the format of INRIA. The second rule, with parenthesis, allows to read an image from any string expression. Thus, it accepts string operations like ‘+’ and ‘−’ or commands that generate strings like ‘sprint’. The third rule opens a graphic filename browser to choose the image name from the disk. The fourth rule allows creating a new image in memory, by specifying the image type, and the three dimensions.
The possible image types are: CHAR UCHAR SHORT USHORT FLOAT DOUBLE RGB FLOAT_VECTOR

Examples:
i = Image "test.ami.gz"
// use the file browser
i = Image
i = Image(UCHAR,100,100,100)

6.6 ReadRawImages

***** ReadRawImages *****

Tokens:
T_ReadRawImages ReadRawImages

Rules:
1. image -> T_ReadRawImages ( expr , expr , basic_type , expr ,
expr_string , expr , expr )

Description:
ReadRawImages permits to read 2D raw data (or a sequence of 2D raw data) as an image. The first 2 parameters are the dimensions of the image in X and Y. The third parameter is the type of the pixel information, one of: CHAR, UCHAR, SHORT, USHORT, FLOAT, DOUBLE, RGB, FLOAT_VECTOR. The fourth parameter is the endianness, 0 for little, 1 for big. The fifth parameter is the filename or
file format. The sixth and seventh parameters are the first and last
slices. The program can read raw data compressed with gzip: if the file
is not found, it looks for the same filename with a `.gz` extension. If it
can find it, it will open the file using the `popen` (pipe open) command
and the `gunzip` program.

**Examples:**

```python
i = ReadRawImages(1024,1024,USHORT,0,"AfGG.%d",48,48)
```

6.7 **CreateFlatMesh**

***** Flat mesh from 2D image *****

**Tokens:**

```
T_CreateFlatMesh CreateFlatMesh
```

**Rules:**

1. `surface` ➔ `T_CreateFlatMesh(expr_image)`
2. `surface` ➔ `T_CreateFlatMesh(expr_image, expr, expr)`

**Parameters:**

- `#1 expr_image: input_image`
- `#2 expr: min`
- `#2 expr: max`

**Description:**

Creates a triangulated flat mesh from a 2D image; if 2 values min and max
are specified, only the mesh of the intensity region between min and
max will be created, using linear interpolation.

**Examples:**

```python
s = CreateFlatMesh(i)
s1 = CreateFlatMesh(i,10,100)
```

**SeeAlso:**

- `vtkCreateFlatMesh`, `ElevateMesh`

6.8 **ElevateMesh**

***** Elevate mesh using image information *****

**Tokens:**

```
T_ElevateMesh ElevateMesh
```

**Rules:**

```python
```
6.9 Compute Altitudes

***** Calculate altitudes based on the temperature *****

**Tokens:**
T_ComputeAltitudes ComputeAltitudes

**Rules:**
1. image -> T_ComputeAltitudes '(' expr_image ',' expr_image ')' 

**Parameters:**
#1 expr_image: input temperatures
#2 expr_image: coeff

**Description:**
Compute the altitudes based on the temperature image and an image of coefficients. Coefficients is an image of 5 float values of size 5x1x1 (1D), created from the satellite information.

**Examples:**
alt1 = ComputeAltitudes(i,coeff)

**SeeAlso:**
CreateFlatMesh, ElevateMesh, CreateVectors, Altitudes2Position

******************************************************************************
6.10 Altitude2Position

***** From 2D image of altitudes to image of positions in 3D *****

**Tokens:**
T_Altitude2Position  Altitude2Position

**Rules:**
1. image -> T_Altitude2Position ( expr_image , expr )

**Parameters:**
#1 expr_image: input_image  
#2 expr : type

**Description:**
Compute the position in space of the points based on their position in the image and on their altitude.
The input is the image of altitudes and the type of the image: 0 for Atlantic and 1 for Africa.
The result is a image of 3D vectors in float, where each vector is the position in 3D of the corresponding pixel.

**Examples:**

```
ipos = Altitude2Position(i,1)
```

**SeeAlso:**
CreateFlatMesh, ElevateMesh

6.11 ReadFlow

***** Reads the flow as a 2D vector field *****

**Tokens:**
T_ReadFlow  ReadFlow

**Rules:**
1. image -> T_ReadFlow ( expr_string )

**Parameters:**
#1 expr_string: name of the ASCII file containing the flow information

**Description:**
Reads the flow information in ASCII format and returns a vectorial image of the flow.

**Examples:**

```
disp = ReadFlow("test_arrows/01test_NAtl_SF_3_0.3_0.0_0.0_0.3_.uv_v2.txt")
```

**SeeAlso:**
6.12 CreateVectors

***** Create a 3D vector field to display movement of clouds *****

Tokens:
T_CreateVectors CreateVectors

Rules:
1. surface -> T_CreateVectors (expr_image, expr_image, expr_image, expr, expr, expr, expr)

Parameters:
#1 expr_image: altitudes1
#2 expr_image: displacement
#3 expr_image: altitudes2
#4 expr: step x
#5 expr: step y
#6 expr: scale
#7 expr: type

Description:
Creates a 3D vector field for visualization of the displacement of clouds between times t1 and t2. The input parameters are:
- altitudes1: scalar image, altitudes of the clouds at time t1
- displacement: vectorial image, displacement as a 2D vector field
- altitudes2: scalar image, altitudes of the clouds at time t2
- stepx: spacing in pixels between successive vectors in X direction
- stepy: spacing in pixels between successive vectors in Y direction
- scale: scaling of the vectors
- type: 0 for Atlantic, 1 for Africa.

Examples:
vects = CreateVectors(alt1, displ, alt2, 8, 8, 3, 1)

SeeAlso:
CreateFlatMesh, ElevateMesh, ComputeAltitudes, Altitudes2Position

6.13 SetTransform

***** Set the 3D View for surface visualization *****

Tokens:
T_SetTransform SetTransform

Rules:
38 CONTENTS

commande <-- VAR_SURFDRAW '.' T_SetTransform ('gltransform')

**Parameters:**
1. VAR_SURFDRAW
2. gltransform

**Description:**
Sets a given transform (defining a point of view) to the surface drawing window.

**SeeAlso:**
GetTransform, print, save, Interpolate, TRANSFORM

-------------------------------

6.14 Interpolate

***** Interpolation between 3D transforms *****

**Tokens:**
T_Interpolate Interpolate

**Rules:**
gltransform <-- T_Interpolate (VAR_GLTRANSFORM, VAR_GLTRANSFORM, expr)

**Parameters:**
# VAR_TRANSFORM: transf1
# VAR_TRANSFORM: transf2
# expr: interpolation coefficient (0–1)

**Description:**
Interpolates the transformation between transf1 and transf2 using the interpolation coefficient: 0 gives transf1 and 1 gives transf2. The interpolation separates translation, rotation and scaling. The rotation is decomposed into 3 rotation around the main axes, and the angles are linearly interpolated.

**SeeAlso:**
TRANSFORM, GetTransform, save

-------------------------------

7 Acknowledgements

We would like to acknowledge the Laboratoire de Météorologie Dynamique (CNRS, UMR 8539, France) for providing the experimental data and especially Pierre Lopes and André Szantai who helped and participated in improving this document.
References


