Milestone 4.1

Method for motion layers estimation without physical coupling between layers

FLUID project (Contract n. FP6-513663)

AMI Research Group
University of Las Palmas of Gran Canaria
Las Palmas, SPAIN
## Contents

1 Introduction 3

2 Satellite image analysis using PDE techniques 3
   2.1 2D flow computation ........................................ 3
   2.1.1 Variational formulation ................................... 3
   2.1.2 Multiscale strategy ...................................... 4
   2.1.3 Invariance under linear greyvalue transformation ......... 4
   2.2 Altitude estimation ......................................... 4
   2.3 Noise reduction ............................................. 5
   2.4 Visualization ............................................ 5

3 Software for 3D visualization of cloud structures 6
   3.1 Installing AMILab ........................................ 7
       3.1.1 Set your dynamic libraries path ....................... 7
       3.1.2 Set your path .......................................... 7
       3.1.3 Set the AMILab script directory ..................... 8
       3.1.4 Automatic configuration .............................. 8
       3.1.5 Run AMILab ........................................... 8
       3.1.6 More documentation .................................. 8
   3.2 Reading the images ....................................... 9
   3.3 Tutorial for FLUID ...................................... 10
       3.3.1 Install the tutorial files ............................ 10
       3.3.2 Set the fluid navigation path ....................... 11
       3.3.3 Loading an image .................................... 11
       3.3.4 Displaying an image .................................. 11
       3.3.5 Displaying a satellite image in 3D .................. 12
       3.3.6 Selecting a region of interest ....................... 13
       3.3.7 Resampling the image ................................ 14
       3.3.8 Saving and Setting 3D views ......................... 14
       3.3.9 Saving an animation .................................. 15
       3.3.10 Coloring the clouds .................................. 16
       3.3.11 Drawing 3D vectors .................................. 18
   3.4 Example of a video ..................................... 18
4 Acknowledgements

5 APPENDIX
5.1 Image ................................................. 24
5.2 ReadRawImages ................................. 24
5.3 CreateFlatMesh ................................. 25
5.4 ElevateMesh ................................. 25
5.5 ComputeAttitudes ................................. 26
5.6 Altitude2Position ................................. 27
5.7 ReadFlow ................................. 27
5.8 CreateVectors ................................. 28
5.9 SetTransform ................................. 28
5.10 Interpolate ................................. 29
1 Introduction

One important activity in WP 4, 3D flow estimation, of FLUID European research project (Contract n. FP6-513663) is the estimation of the motion layers without physical coupling between the layers and Milestone 4.1 have been devoted to this topic. In this document we present an overview of our activities concerning Milestone 4.1.

In section 2, we describe the different techniques used to process the satellite image sequences and to estimate the cloud motion. It includes the 2D flow motion estimation, the computation of the height of the clouds, the noise reduction and the 3D visualization of the clouds. In section 3, we present a tutorial of our software in the context of satellite images, allowing to create 3D animations of the clouds and of the motion vector field.

2 Satellite image analysis using PDE techniques

The European organization EUMETSAT provides multichannel satellite image sequences, including visible channels, temperature channels and water vapor channels. We use a combination of these channels to perform the following tasks: 2D flow computation, altitude estimation and noise reduction, 3D position computation and 3D visualization of the clouds and of the vector flow.

2.1 2D flow computation

The 2D flow computation is carried out using a PDE based optical flow technique described in [1]. 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 are similar at the corresponding points and the regularization term assumes a smoothness of the fluid flow. The regularization term uses the approach proposed by Nagel and Enkelmann [3], 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 (ii) it creates an energy functional that is invariant under linear brightness changes. Other approaches introduce a second order div-curl regularization to better preserve the vorticity and divergence structures [2], or use block-based matching methods [4].

2.1.1 Variational formulation

The energy to minimize is written as:

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

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, $\text{tr}$ is the trace operator,
2.2 Altitude estimation

$C$ 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:

$$D(\nabla I_1) = \frac{1}{\|\nabla I_1\|^2 + 2\lambda^2} (\xi \xi^t + \lambda^2 I),$$

(2)

where $\xi = (\frac{\partial I_1}{\partial y}, -\frac{\partial I_1}{\partial x})^t$ 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 [1].

2.1.2 Multiscale strategy

The method is embedded into a linear scale-space framework to improve its robustness to local minima of the energy and to allow the recovery of large displacements. To this end, the two images $I_1$ and $I_2$ are convolved with a Gaussian kernel of standard deviation $\sigma$, where $\sigma$ stands for the current scale. The spatial derivatives of the images are obtained by convolution with the corresponding spatial derivatives of the Gaussian kernel. A coarse to fine scale strategy is used, where the solution at a given scale is used as an initialization for the next finer scale, and the scales are sampled logarithmically.

2.1.3 Invariance under linear greyvalue transformation

The invariance under linear greyvalue transformation of the form $(I_1, I_2) \rightarrow (kI_1, kI_2)$ is achieved by setting two parameters $\alpha$ and $s$ in $[0, 1]$ as:

$$C = \frac{\alpha}{\max_x(||(\nabla_{\sigma}I_1(x))||^2)}$$

(5)

$$s = \int_0^\lambda \mathcal{H}(z)dz,$$

(6)

where $\nabla_{\sigma}I_1 = \nabla G_{\sigma} * I_1$ is the gradient of $I_1$ at the current scale $\sigma$ and $\mathcal{H}$ represents the normalized histogram of $|\nabla_{\sigma}I_1|$. Using the new parameters $\alpha$ and $s$ instead of $C$ and $\lambda$ ensures the invariance of the solution under linear greyvalue transformation.

2.2 Altitude estimation

More details can be found in Report 1: Basic information on MSG images, 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 infrared channel. Let us denote the infrared intensity at the current pixel position as $C$.

The radiance $R$ (mW.m$^{-2}$.sr$^{-1}.cm$) is calculated as:

$$R = R_0 + \alpha C,$$

where $r_0$ and $\alpha$ are included in the original MSG files by EUMETSAT.

The brightness temperature $T_b$ of the observed object can be approximated from the infrared channel, using the formula:

$$T_b = \frac{1}{A} \left( \left( \frac{C_2 \nu_c}{\ln(1 + \frac{C_1}{\nu_c R})} - B \right) \right),$$

where

- $C_1 = 1.19104.10^{-5}$ mW.m$^{-2}$.sr$^{-1}.cm^4$,
- $C_2 = 1.43877$ K.cm,
- the central wavenumber $\nu_c$, the parameters $A$ (dimensionless) and $B$ (in Kelvin) are given constants for a given by EUMETSAT for each satellite and channel.

The altitude is then deduced from the temperature as:

$$a = \frac{T - T_0}{\gamma},$$

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

### 2.3 Noise reduction

A median filter is applied to the classification image. For the temperature image, each classification area is treated independently, using two types of filters: a validation data filter and an energy based smoothing filter which yields to a linear PDE (heat equation). The validation data filter is based on the median filter within the classification area of the current pixel. The energy-based smoothing filter consists in minimizing

$$E(T) = \int_{x \in C} (T(x) - T_0(x))^2 dx + \alpha \int_{x \in C} \| \nabla T(x) \|^2 dx.$$  

### 2.4 Visualization

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

Figure 1: Processing and visualization of satellite data

Fig. 1 illustrates the different tasks and their inputs.

Fig. 2 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 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 2: Left: coloring the clouds, right: displacement field as 3D vectors.

3 Software for 3D visualization of cloud structures

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
3.1 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
- **Scripts**: current scripts used for different image processing or visualization tasks.

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

```
> tar zxf AMILab_linux.tgz
```

### 3.1.1 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:

```
> setenv LD_LIBRARY_PATH $({LD_LIBRARY_PATH}:INSTALLDIR/AMILab_Linux_2.6.6/YYYY-MM-DD/libs
```

Or

```
> 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

```
> setenv LD_LIBRARY_PATH INSTALLDIR/AMILab_Linux_2.6.6/YYYY-MM-DD/libs
```

Or

```
> export LD_LIBRARY_PATH=INSTALLDIR/AMILab_Linux_2.6.6/YYYY-MM-DD/libs
```

### 3.1.2 Set your path

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

```
> setenv PATH $({PATH}:INSTALLDIR/AMILab_Linux_2.6.6/YYYY-MM-DD/bin
```

Or

```
> export PATH=${PATH}:INSTALLDIR/AMILab_Linux_2.6.6/YYYY-MM-DD/bin
```
3.1 Installing AMILab

3.1.3 Set the AMILab script directory

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

```
> 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.

3.1.4 Automatic configuration

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

3.1.5 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.worksheetXXX’ 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.

3.1.6 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.
3.2 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”:

```
AMIMAGE 1.1
(  
XDim = 1024
YDim = 1024
ZDim = 1
VoxelX = 1.000000
VoxelY = 1.000000
VoxelZ = 1.000000
TransX = 0.000000
TransY = 0.000000
TransZ = 0.000000
Repres = USHORT
Type = SCALAR
Endianness = LITTLE
ScanOrder = LR
DF = EXTERN
FF = /home/karl/projects/Fluid/WP01/package_02_LMD/images/
    AfGG_c0.81.%d
FS = 47
LS = 55
)
```

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.
- **VoxelX, VoxelY, VoxelZ** are the dimensions of the voxels (or pixels in 2D) in a given unit (for medical images, it is usually in mm).
3.3 Tutorial for FLUID

- TransIX, TransIY, TransIZ 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).

- FS and LS are the first and last slices in the image sequence.

Remark: Create the ASCII header from Windows can create problems since Microsoft Windows characters are not always compatible with Linux characters.

3.3 Tutorial for FLUID

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

3.3.1 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
download and untar the file tar_WP1_-LMD_images.tar.
> tar xf tar_WP1_LMD_images.tar
```

Then download and untar the files needed for the tutorial:
3.3 Tutorial for FLUID CONTENTS

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

3.3.2 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 les. 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.

3.3.3 Loading an image

To load an image, you can use the Image command (subsection 5.1):

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 5.2):

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:

i47.info
i.info

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

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

3.3.4 Displaying an image

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

```
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).

![Image](image.png)

**Figure 3**: Display of a satellite image.

### 3.3.5 Displaying a satellite image in 3D

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

```plaintext
i47 = ReadRawImages(1024,1024,USHORT,0,"WP01/package_02_LMD/images/AfGG_c10.8.47",0,0)
```

We read the image of temperatures.

```plaintext
coeff = Image "alt_coeffami.gz"
```

The file 'alt_coeffami.gz' is an image which contains 5 float values (1D image), those coefficients allow to calculate the altitudes.

```plaintext
alt47 = ComputeAltitudes(i47,coeff)*4
```

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

```plaintext
earth = CreateFlatMesh(i47>=0)
```

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

```plaintext
pos = Altitude2Position(alt47,1)
```

'pos' is an image containing the real positions in 3D.
3.3 Tutorial for FLUID CONTENTS

```plaintext
earth. SetColors(i47, min(i47), max(i47))
```

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

```plaintext
earth. ElevateMesh(pos)
```

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

```plaintext
earth. SetColorMaterial(1)
earth. Normals
earth. Recompute
show earth
earth_dra\_draw.rotate(0,-90,0)
earth_dra\_draw.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.

![Figure 4: Display of a satellite image in 3D.](image)

3.3.6 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:

```plaintext
si47 = i47[100:300, 100:300]
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(salt47>=0)
spos = Altitude2Position(salt47,1)
searth. SetColors(si47, min(si47), max(si47))
```
3.3 Tutorial for FLUID

searth.ElevateMesh(pos)
searth.SetColorMaterial(1)
searth.Normals
searth.Recompute
show searth

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

searth_draw.compare(searth_draw)

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

3.3.7 Resampling the image

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

\[ i47_2 = \text{Resize}(i47, 512, 512, 1, 0) \]

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

3.3.8 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:

\[ \text{show earth} \]
\[ t1 = \text{earth_draw.GetTransform} \]

You can move the model using the mouse, save a second view, and come back to the first view:

\[ t2 = \text{earth_draw.GetTransform} \]
\[ \text{earth_draw.SetTransform}(t1) \]

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).

\[ \text{t1.save "t1.mat"} \]
\[ \text{t2.save "t2.mat"} \]
\[ \text{t1_2 = Transform("t1.mat")} \]
\[ \text{t1.print} \]
\[ \text{t1_2.print} \]

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

### 3.3.9 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.

```
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.

```
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:

```
for n=0 to nb_images−1 {
    im1 = anim[:,:,n:n] // 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 `mkmp4eg` written in Python to generate the video using for example 8 frames per seconds:

```bash
> mkmp4eg -f 8 -o tutorial_anim.mpg tutorial_anim*.jpg
```

### 3.3.10 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.amil'.

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

```python
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_coef.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.

```python
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.

```python
alt_factor = 4
alt48 = ComputeAltitudes(t48,coeff)
alt48 = alt48*alt_factor
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
}
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.

```plaintext
earth_draw.SetLightDiffuse(0,180,180,180)
# 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 )
earth_draw.rotate(0,-90,0)
earth_draw.rotate(0,0,-90)
```

Figure 5: 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 those colors to only one supercy. This is achieved by the script `tutorial/tutorial_coloring_onemesh.amil`:

```plaintext
# Read the input images
i48 = ReadRawImages(1024,1024,USHORT,0,"WP01/package_02_LMD/images/AFGG_c0.81.48",0,0)
c48 = ReadRawImages(1024,1024,CHAR,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"
# Create the mesh
earth = CreateFlatMesh(c48>=0)
# Create one RGB image
c1 = (RGB) (c48)
i48min=min(i48)
i48max=max(i48)
i48 uchar = ((FLOAT)i48-i48min)/(i48max-i48min)*255
c1[0] = (c48<101)*i48 uchar+(c48>=101)*(c48<=103)*224+(c48>=104)*(c48<=106)*222
   <=100)*222
   <=100)*254
```
3.4 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 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.anim'.

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 printf("new_animation_size=%03.0f\n",_tmpim.tz)
    _tmpim.putimage(anim,0,0,0)
    del anim
    anim = _tmpim
    del _tmpim
    local
    anim.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/"
```
3.4 Example of a video

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
}
```

```plaintext
# Add all the meshes into the same visualization window
for k=0 to 4 { earth_draw += clouds[k]; }
earth_draw.SetLightDiffuse(0, 180, 180, 180)
# Set color material ON
```

Listing 5: Visualizing the meshes
3.4 Example of a video

earth.SetColorMaterial(1)
for k=0 to 4 {clouds[k].SetColorMaterial(1);}
earth_draw.rotate(0,−90,0)
earth_draw.rotate(0,0,−90)

Listing 6: Setting the colors of the clouds

# 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

Listing 7: Moving the point of view

Listing 8: Generating views
3.4 Example of a video

```csharp
3.4 Example of a video CONTENTS

```del _vtmp
_im1 = earth_draw.GetImageFromX
AnimAddImage(_im1)
}
del _viewinit
del _viewend

```

```csharp
# total_view = 40
# Rotate
GenerateViews("view1.txt", "view4.txt", total_view)
# Get Closer
GenerateViews("view4.txt", "view5.txt", total_view)
GenerateViews("view5.txt", "view5_1.txt", total_view)
GenerateViews("view5_1.txt", "view5_1_1.txt", total_view)
# Rotate 360 degrees

total_rotate = 100
for n=1 to total_rotate {
    earth_draw.rotate(0,360/total_rotate,0)
im1 = earth_draw.GetImageFromX
AnimAddImage(im1)
}
GenerateViews("view5_1_1.txt", "view12.txt", total_view)

Listing 8: Adding 3D vectors

```csharp
# without filtering
	t49 = ReadRawImages(1024,1024,USHORT,0,"WP01/package_02_LMD/images/AFGG_c1.0.8.499,0,0")
alt49 = ComputeAltitudes(t49,coeff)∗alt_factor
# reading the vector fields
displ = ReadFlow("test_arrows/01test_AFGG_SF_4.0.3_0.0_0.0_0.3_.uv_v2.txt")
displ1 = displ*(c48>107−epsilon)∗(c48<109+epsilon)
alt48_2 = alt48+(c48>107−epsilon)∗(c48<109+epsilon)
alt49_2 = alt49+(c48>107−epsilon)∗(c48<109+epsilon)
vect1 = CreateVectors(alt48_2,displ1,alt49_2,6,6,3,type_image)
vect1.SetColor(0,0,255)
vect1.SetColorMaterial(0)
vect1.SetAmbient(10,10,200)
vect1.SetOpacity(0)
earth_draw += vect1
# Progressive display of the vectors

total_vectors = 40
for n=1 to total_vectors {
    vect1.SetOpacity(n/total_vectors)
    earth_draw.Paint
    im1 = earth_draw.GetImageFromX
    AnimAddImage(im1)
}
# Copy the last image a few times

total_final = 20
for n=1 to total_final {
    AnimAddImage(im1);
}
# Only keep the real number of images that have been used
ssanim = anim[;,,0:anim_current−1]
```
4 Acknowledgements

We would also like to acknowledge the Laboratoire de Météorologie Dynamique (CNRS, UMR 8539, France) for providing the experimental data and especially Pierre Lopes and André Szántai who helped and participated in improving this document. This work has been funded by the European research project FLUID, number FP6-513663.
5 APPENDIX

5.1 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)


5.2 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.
5.3 CreateFlatMesh

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)
```

5.3 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`

5.4 ElevateMesh

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

Tokens:

```
T_ElevateMesh ElevateMesh
```

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

Tokens:

```
T_ElevateMesh ElevateMesh
```

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

Tokens:

```
T_ElevateMesh ElevateMesh
```
5.5 **ComputeAltitudes**

***** 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
5.6 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

5.7 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:
5.8 CreateVectors

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

**Tokens:**
T_CreateVectors CreateVectors

**Rules:**
1. surface \( \rightarrow \) 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: stepx
- \#5 expr: stepy
- \#6 expr: scale
- \#7 expr: type

**Description:**
Creates a 3D vector field for visualization of the displacement of clouds between times \( t_1 \) and \( t_2 \). The input parameters are:
- altitudes1: scalar image, altitudes of the clouds at time \( t_1 \)
- displacement: vectorial image, displacement as a 2D vector field
- altitudes2: scalar image, altitudes of the clouds at time \( t_2 \)
- 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

5.9 SetTransform

***** Set the 3D View for surface visualization *****
5.10 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

References

REFERENCES


