

# MicMac, Apero, Pastis and Other Beverages in a Nutshell!

January 31, 2017



# Contents

<b>I Generalities</b>	<b>5</b>
0.1 Foreword . . . . .	7
<b>1 Introduction</b>	<b>9</b>
1.1 History, Status and Contributors . . . . .	9
1.2 Prerequisites . . . . .	10
1.3 Installation and Distribution . . . . .	10
1.3.1 Documentation . . . . .	10
1.3.2 Install . . . . .	10
1.4 Libraries, Programs and Dependencies . . . . .	11
1.5 Interface for the Tools . . . . .	11
1.5.1 Kinds of Interfaces . . . . .	11
1.5.2 Simple Tools . . . . .	11
1.5.2.1 GUI for command line tools . . . . .	12
1.5.3 Complex Tools . . . . .	13
1.5.4 Where Calling the Tools From (The Mandatory Working Directory) . . . . .	13
1.6 Data Organization and Communication . . . . .	13
1.7 Existing Tools . . . . .	14
<b>2 Some Realization Examples</b>	<b>15</b>
2.1 3D Objects . . . . .	15
2.1.1 Statues . . . . .	15
2.1.2 Architectural Details . . . . .	16
2.2 Indoors Global Modelization . . . . .	16
2.2.1 Architecture . . . . .	16
2.3 Globally Planar Objects . . . . .	16
2.3.1 Elevation and ortho images . . . . .	16
2.3.2 Painting and Fresco . . . . .	16
2.3.3 Bas-relief . . . . .	16
2.3.4 Macro-photo . . . . .	16
2.4 Aerial Photos . . . . .	16
2.4.1 Urban DEM . . . . .	16
2.4.2 Satellite Images . . . . .	16
2.4.3 UAV Missions . . . . .	16
2.5 Miscellaneous . . . . .	16
2.5.1 Industrial . . . . .	16
2.6 Gallery of images . . . . .	16
<b>3 Simplified Tools</b>	<b>27</b>
3.1 All in one command . . . . .	27
3.2 Modification since Mercurial version . . . . .	27
3.2.1 Installing the tools . . . . .	27
3.2.2 The <i>new universal</i> command <code>mm3d</code> . . . . .	28
3.2.3 Help with <code>mm3d</code> . . . . .	28
3.2.4 Log files . . . . .	29

3.3	Computing Tie Points with Tapioca . . . . .	29
3.3.1	General Structure . . . . .	29
3.3.2	Computing All the Tie Points of a Set of Images . . . . .	30
3.3.3	Optimization for Linear Canvas . . . . .	30
3.3.4	Multi Scale Approach . . . . .	31
3.3.5	Explicit Specification of images pairs . . . . .	31
3.3.6	The tool <b>GrapheHom</b> . . . . .	31
3.4	Simple relative orientation and calibration with Tapas . . . . .	32
3.4.1	Generalities . . . . .	32
3.4.2	The data set "Mur Saint Martin" . . . . .	32
3.4.3	Basic usage . . . . .	33
3.4.3.1	Syntax . . . . .	33
3.4.3.2	Distorsion models . . . . .	34
3.4.3.3	Strategy . . . . .	34
3.4.3.4	Results . . . . .	34
3.4.4	Successive calls to Tapas . . . . .	35
3.5	Multiple lenses with Tapas . . . . .	36
3.5.1	The Saint Martin Street data set . . . . .	36
3.5.2	Exploiting the data with Tapas . . . . .	37
3.6	Camera data base and exif handling . . . . .	37
3.6.1	How Tapas initialize calibration . . . . .	37
3.6.2	Camera data base . . . . .	38
3.6.3	Indicating missing xif info . . . . .	38
3.6.4	Modifying exif . . . . .	39
3.6.5	XML "cache" version of xif information . . . . .	39
3.7	Using Raw images . . . . .	40
3.8	Other options of Tapas . . . . .	40
3.8.1	Saving intermediar results with <b>SauvAutom</b> . . . . .	40
3.8.2	Forcing first image with <b>ImInit</b> . . . . .	40
3.8.3	Freezing poses with <b>FrozenPoses</b> . . . . .	40
3.9	Other tools for orientation . . . . .	41
3.9.1	The tool <b>AperiCloud</b> . . . . .	41
3.9.2	The tool <b>Campari</b> . . . . .	41
3.9.2.1	Estimate lever-arm with the tool <b>Campari</b> . . . . .	42
3.9.2.2	Bundle adjustment with pushbroom sensor . . . . .	43
3.9.3	Convention for Orientation name . . . . .	44
3.10	The Bascule's tools . . . . .	44
3.10.1	Generalities . . . . .	44
3.10.1.1	Scene based orientation with <b>SBGlobBascule</b> . . . . .	44
3.10.1.2	Geo-referencing with <b>GCPBascule</b> . . . . .	46
3.10.1.3	Creating local repair with <b>RepLocBascule</b> . . . . .	46
3.10.1.4	Geo-referencing with <b>CenterBascule</b> . . . . .	48
3.10.1.5	Merging Orientation with <b>Morito</b> . . . . .	48
3.10.1.6	Accuracy Control with <b>GCPCtrl</b> . . . . .	49
3.10.2	<b>MakeGrid</b> . . . . .	49
3.11	Tools for full automatic matching . . . . .	49
3.11.1	The tortue data set . . . . .	50
3.11.2	The tool <b>AperoChImSecMM</b> . . . . .	50
3.11.3	The tool <b>MMInitialModel</b> . . . . .	51
3.11.4	The tool <b>MMTestAllAuto</b> . . . . .	52
3.12	Tools for simplified semi-automatic matching . . . . .	52
3.12.1	Basic rectification with <b>Tarama</b> . . . . .	54
3.12.2	Simplified matching in ground geometry with <b>Malt</b> . . . . .	55
3.12.2.1	General characteristics . . . . .	55
3.12.2.2	Optional parameters . . . . .	56
3.12.3	Image geometry with <b>Malt</b> . . . . .	56

3.13 Ortho photo generation . . . . .	57
<b>4 Use cases with Simplified tools</b>	<b>61</b>
4.1 The Vincennes data set . . . . .	61
4.1.1 Description of the data set . . . . .	61
4.1.2 Computing tie points and orientations . . . . .	61
4.1.2.1 Tie points . . . . .	61
4.1.2.2 Relative orientation . . . . .	61
4.1.2.3 Optional, absolute orientation . . . . .	63
4.1.2.4 Optional, scene-based orientation . . . . .	63
4.1.3 Matching . . . . .	63
4.1.3.1 "Standard" option . . . . .	63
4.1.3.2 "Ortho-cylindrical" option . . . . .	63
4.1.3.3 Concrete use of "Ortho-cylindric" option . . . . .	65
4.2 The Saint-Michel de Cuxa data set . . . . .	66
4.2.1 Description of the data set . . . . .	66
4.2.2 Computing tie points and relative orientations . . . . .	69
4.2.2.1 Tie points . . . . .	69
4.2.2.2 Relative orientation . . . . .	69
4.2.3 GCP transforms . . . . .	70
4.2.3.1 Ground control point coordinates conversion . . . . .	70
4.2.3.2 Ground control point image coordinates input . . . . .	70
4.2.3.3 Bascule . . . . .	70
4.2.3.4 Adding points with predictive interface <i>SaisieAppuisPredic</i> . . . . .	70
4.2.3.5 Bascule . . . . .	70
4.2.4 Bundle adjustment with ground control points . . . . .	70
4.2.5 Post-processing . . . . .	71
4.2.5.1 Coordinate system backward transform . . . . .	71
4.3 The Grand-Leez dataset . . . . .	73
4.3.1 Dataset description . . . . .	73
4.3.2 Computing tie points and absolute orientation . . . . .	75
4.3.2.1 Conversion of GPS data in <i>MicMac</i> format . . . . .	75
4.3.2.2 Tie points . . . . .	76
4.3.2.3 Relative orientation . . . . .	76
4.3.2.4 Georeferencing . . . . .	76
4.3.2.5 Bundle adjustment with embedded GPS data . . . . .	76
4.3.3 Dense matching and orthorectification . . . . .	76
4.4 GoPro Video data-set . . . . .	78
4.4.1 Description of the data set . . . . .	78
4.4.2 The commands . . . . .	79
4.4.3 Some comments . . . . .	81
4.4.3.1 Developing still images with <i>ffmpeg</i> . . . . .	81
4.4.3.2 Adding missing xif with <i>SetExif</i> . . . . .	81
4.4.3.3 Selecting sharpest images with DIV . . . . .	81
4.4.3.4 Standard orientation . . . . .	81
4.4.3.5 Seizing the waves . . . . .	82
4.4.3.6 Filtering homologous points . . . . .	82
4.4.3.7 Final orientation . . . . .	82
4.5 The satellite data set . . . . .	82
4.5.1 Description of the data . . . . .	82
4.5.2 From 2D images to 3D objects – the processing commands . . . . .	84
4.5.3 Understanding the bundle adjustment output ( <i>Campari</i> ) . . . . .	85
4.5.4 Understanding the bundle adjustment validation output ( <i>MMTestOrient</i> ) . . . . .	86
4.6 The Viabon dataset . . . . .	86
4.6.1 Processing GPS data . . . . .	87
4.6.1.1 Compile <i>MicMac</i> with <i>RTKlib</i> . . . . .	87

4.6.1.2	Base station processing . . . . .	87
4.6.1.3	UAV trajectories processing . . . . .	88
4.6.2	Computing tie points . . . . .	88
4.6.2.1	Computing exterior orientation . . . . .	89
4.6.2.2	GPS positions & Camera centers matching . . . . .	90
4.6.2.3	Images georeferencing . . . . .	90
4.6.2.4	Bundle Bloc Adjustment with GPS observations and lever-arm offset . . . . .	91
4.6.2.5	Advanced internal camera model . . . . .	92
4.6.2.6	Integrated Sensor Orientation using embedded GPS and 1 GCP . . . . .	93
4.6.2.7	Classical GCPs indirect georeferencing . . . . .	94
4.6.3	Dense Matching and Orthorectification . . . . .	94
<b>5</b>	<b>A Quick Overview of Matching</b>	<b>97</b>
5.1	Installing the Tools . . . . .	97
5.1.1	svn Extraction - obsolete (see 3.2.1) . . . . .	97
5.1.2	Compilation . . . . .	97
5.1.3	Some Important Directories . . . . .	97
5.1.4	Installing the Examples . . . . .	97
5.1.5	Verification and Global Vision of the Main Tools . . . . .	98
5.2	A MicMac Example Using Simplest Parametrization . . . . .	98
5.2.1	Epipolar Geometry . . . . .	98
5.2.2	Analyzing Matching Parameters . . . . .	98
5.2.3	Running the Program . . . . .	100
5.2.4	Analyzing the Results . . . . .	100
5.3	Examples Using MicMac, Algorithmic Aspect . . . . .	101
5.3.1	Using Multi-resolution . . . . .	101
5.3.2	Using Regularization . . . . .	103
5.4	Examples using MicMac, Geometric Aspect . . . . .	105
5.4.1	Ground Geometry . . . . .	105
5.4.2	An Example in Ground Terrain, Parameter Analysis . . . . .	106
5.4.3	An Example in Ground Terrain, Result Analysis . . . . .	108
5.4.4	An Example in Ground Terrain with CUDA . . . . .	109
5.4.5	An Example in "Ground-image" Geometry . . . . .	109
5.4.6	Batching Several Computations . . . . .	113
5.5	Hidden part and individual ortho images generation . . . . .	114
5.5.1	Other Options . . . . .	116
5.6	2D Matching . . . . .	116
5.6.1	Pure Image 2D Matching . . . . .	116
5.6.2	Ground Image 2D Matching . . . . .	117
<b>6</b>	<b>A Quick Overview of Orientation</b>	<b>119</b>
6.1	General Organization of Apero . . . . .	119
6.1.1	Input and Output . . . . .	119
6.1.2	General Strategy . . . . .	119
6.2	A First Example . . . . .	120
6.2.1	Introduction . . . . .	120
6.2.2	Tie Points . . . . .	121
6.2.3	Declaring the Observations . . . . .	121
6.2.4	Declaring the Unknowns, Camera-calibration . . . . .	122
6.2.5	Declaring the Unknowns, Camera-poses . . . . .	122
6.2.6	Running the Compensation . . . . .	123
6.2.6.1	What Is Done During the Compensation . . . . .	123
6.2.6.2	Structure of <code>SectionCompensation</code> . . . . .	124
6.2.6.3	Handling the Constraints . . . . .	124
6.2.6.4	Using Weighted Observations . . . . .	125
6.2.6.5	Weighting of homogeneous and heterogeneous observations . . . . .	125

6.2.7	Understanding the Message . . . . .	127
6.2.8	Storing the Results . . . . .	128
6.3	More Examples . . . . .	128
6.3.1	Adding More Images . . . . .	128
6.3.2	Computing the Internal Parameters . . . . .	129
6.3.3	Automatic Image Ordering . . . . .	130
6.4	Geo-referencing . . . . .	131
6.4.1	External Initialization . . . . .	131
6.4.2	Scene-based Orientation . . . . .	132
6.4.3	Using Embedded GPS . . . . .	134
6.4.4	Ground Control Points . . . . .	135
6.4.4.1	Ground Points Organization . . . . .	135
6.4.4.2	Using GCP . . . . .	136
<b>7</b>	<b>A Quick Overview of Other Tools</b>	<b>139</b>
7.1	Developing raw and jpeg Images with Devlop . . . . .	139
7.2	Making Ortho Mosaic with Porto . . . . .	139
7.2.1	Introduction . . . . .	139
7.2.2	Input to Porto . . . . .	139
7.2.3	Output to Porto . . . . .	141
7.3	V.O.D.K.A. . . . .	141
7.3.1	Theory . . . . .	141
7.3.2	Name and function . . . . .	143
7.3.3	Input data . . . . .	143
7.3.4	Output data . . . . .	144
7.3.5	Options . . . . .	144
7.3.6	How to use VODKA . . . . .	144
7.4	A.R.S.E.N.I.C . . . . .	144
7.4.1	Name and function . . . . .	144
7.4.2	Input data . . . . .	144
7.4.3	Output data . . . . .	144
7.4.4	Options . . . . .	144
7.4.5	Algorithm . . . . .	145
7.4.5.1	Tie point detection . . . . .	145
7.4.5.2	Equalization . . . . .	145
7.4.6	How to use ARSENIC . . . . .	146
7.5	Miscellaneous tools . . . . .	146
7.5.1	TestLib PackHomolToPly . . . . .	146
7.5.2	MeshProjOnImg . . . . .	147
7.5.3	InitOriLinear . . . . .	148
7.5.4	ReprojImg . . . . .	151
7.5.5	ExtractMesure2D . . . . .	151
7.5.6	BasculeCamsInRepCam . . . . .	152
7.5.7	BasculePtsInRepCam . . . . .	152
7.5.8	CorrLA . . . . .	152
7.5.9	ExportXmlGcp2Txt . . . . .	153
7.5.10	SimplePredict . . . . .	153
7.5.11	Export2Ply . . . . .	154
7.5.12	CmpOri . . . . .	154
7.5.13	CmpCalib . . . . .	154
7.5.14	PseudoIntersect . . . . .	155

<b>8 Interactive tools</b>	<b>157</b>
8.1 Generalities . . . . .	157
8.2 Entering mask with <i>SaisieMasq</i> or <i>SaisieMasqQT</i> . . . . .	157
8.2.1 <i>SaisieMasq</i> . . . . .	157
8.2.2 <i>SaisieMasqQT</i> . . . . .	157
8.3 Entering 3D mask with <i>SaisieMasqQT</i> . . . . .	158
8.4 Entering points . . . . .	159
8.4.1 Generalities . . . . .	159
8.4.1.1 Geometry menu . . . . .	159
8.4.1.2 Info menu . . . . .	160
8.4.1.3 Undo menu . . . . .	160
8.4.1.4 Zoom menu . . . . .	160
8.4.2 For initial GCP with <i>SaisieAppuisInit</i> . . . . .	161
8.4.3 For fast predictive entering GCP with <i>SaisieAppuisPredic</i> . . . . .	162
8.4.4 For bascule with <i>SaisieBasc</i> . . . . .	163
8.5 Visualize Tie-points with SEL . . . . .	163
<b>9 New "generation" of tools</b>	<b>165</b>
9.1 Fully automatic dense matching . . . . .	165
9.1.1 Generalities . . . . .	165
9.1.2 Quickmac option . . . . .	165
9.2 Post-processing tools - mesh generation and texturing . . . . .	167
9.2.1 Mesh generation . . . . .	167
9.2.2 Texturing the mesh . . . . .	168
9.3 Parallelizing Apero . . . . .	170
9.3.1 Parallelizing Apero . . . . .	170
<b>10 XML-Formal Parameter Specification</b>	<b>173</b>
10.1 Introduction . . . . .	173
10.1.1 General Mechanism . . . . .	173
10.1.2 Format Specification . . . . .	173
10.1.3 Command Line . . . . .	174
10.2 Tree Matching . . . . .	174
10.3 Types des nœuds terminaux . . . . .	175
10.3.1 Types généraux . . . . .	175
10.3.2 Types énumérés . . . . .	176
10.4 Définition de types d'arbres . . . . .	176
10.5 Mode "différentiel" . . . . .	177
10.6 Autres caractéristiques . . . . .	178
10.6.1 Modification par ligne de commande . . . . .	178
10.6.2 Valeurs par défaut . . . . .	178
10.6.3 Fichiers de paramètres générés . . . . .	178
<b>11 Names Convention Organization</b>	<b>179</b>
11.1 General Organization . . . . .	179
11.1.1 Requirements . . . . .	179
11.1.2 The struct <i>ChantierDescripteur</i> . . . . .	179
11.1.3 The Files Containing <i>ChantierDescripteur</i> . . . . .	179
11.1.4 Overriding and Priority . . . . .	180
11.2 Regular Expression and Substitution . . . . .	180
11.2.1 Regular Expression . . . . .	180
11.2.2 Substitution . . . . .	180
11.3 Helps tools in names manipulation . . . . .	180
11.3.1 <i>TestKey</i> . . . . .	180
11.3.2 <i>TestMTD</i> . . . . .	180
11.3.3 <i>TestNameCalib</i> . . . . .	181

11.4 Describing String Sets . . . . .	181
11.5 Describing String Mapping . . . . .	181
11.5.1 Advanced association . . . . .	181
11.6 Describing String Relations . . . . .	181
11.7 Filters and In-File Definition . . . . .	181
<b>12 Use cases for 2D Matching</b>	<b>183</b>
12.1 Checking orientation . . . . .	183
12.1.1 For Conik Orientation . . . . .	183
12.1.2 For Push-Broom Orientation . . . . .	185
12.2 The Mars data-set . . . . .	186
12.2.1 Description of the data set . . . . .	186
12.2.2 Comment on the parameters . . . . .	186
12.2.2.1 Geometry . . . . .	186
12.2.2.2 Matching . . . . .	186
12.2.2.3 Results . . . . .	187
12.3 The Gulya Earthquake Data-Set . . . . .	187
12.3.1 Introduction . . . . .	187
12.3.2 Description of the data set . . . . .	187
12.3.3 Simplified interface . . . . .	188
12.3.4 Comment on the parameters . . . . .	188
12.3.4.1 Interpolation . . . . .	188
12.3.4.2 Image term . . . . .	188
12.3.4.3 Non isotropic regularization . . . . .	188
12.4 The Concrete Data-Set and civil engineering . . . . .	189
12.4.1 Introduction . . . . .	189
12.4.2 Parametrizing . . . . .	189
12.5 FDSC - a post-processing tool for 2D correlation results . . . . .	189
12.5.1 Drawing the fault trace . . . . .	190
12.5.2 Stacking profiles . . . . .	190
12.5.3 Drawing the slip-curve . . . . .	191
<b>II Reference documentation</b>	<b>193</b>
<b>13 Data exchange with other tools</b>	<b>195</b>
13.1 Generalities . . . . .	195
13.2 Orientation's convention . . . . .	195
13.2.1 Internal orientation . . . . .	195
13.2.2 External orientation . . . . .	196
13.3 Conversion tools . . . . .	197
13.3.1 Ground Control Point Conversion: <code>GCPConvert</code> . . . . .	197
13.3.1.1 File format conversion with <code>GCPConvert</code> . . . . .	197
13.3.2 Orientations conversion for PMVS: <code>Apero2PMVS</code> . . . . .	198
13.3.2.1 Distortion removing with <code>DRUNK</code> . . . . .	199
13.3.3 <code>Apero2NVM</code> . . . . .	199
13.3.4 Embedded GPS Conversion: <code>OriConvert</code> . . . . .	200
13.3.4.1 File format conversion with <code>OriConvert</code> . . . . .	200
13.3.4.2 Taking into account a GPS delay with <code>OriConvert</code> . . . . .	201
13.3.4.3 Selection of a image sub-block with <code>OriConvert</code> . . . . .	202
13.3.5 Extracting Gps from exif . . . . .	203
13.3.5.1 Extracting Gps from exif with <code>XifGps2Xml</code> . . . . .	203
13.3.5.2 Extracting Gps from exif with <code>XifGps2Txt</code> . . . . .	203
13.3.6 Exporting external orientation to Omega-Phi-Kapa . . . . .	204

<b>14 Geo Localisation formats</b>	<b>205</b>
14.1 Overview of conic orientation specification . . . . .	205
14.1.1 Generalities . . . . .	205
14.1.2 Internal orientation . . . . .	206
14.1.3 Kind of projection . . . . .	206
14.1.4 External orientation . . . . .	206
14.1.5 Intrinsic Calibration . . . . .	207
14.1.6 The Verif Section . . . . .	207
14.2 Distorsion specification . . . . .	208
14.2.1 Generalities . . . . .	208
14.2.1.1 Composition of distortions . . . . .	208
14.2.1.2 Structure of basic distortion . . . . .	208
14.2.2 Radial Model . . . . .	209
14.2.3 Photogrammetric Standard Model . . . . .	209
14.2.4 Grids model . . . . .	210
14.3 Unified distortion models . . . . .	210
14.3.1 Generalities . . . . .	210
14.3.2 Unified Polynomial models . . . . .	210
14.3.3 Brown's and Ebner's model . . . . .	211
14.3.4 Fish eye models . . . . .	211
14.3.5 The tag <RayonUtile> . . . . .	212
14.4 The tool <b>TestCam</b> . . . . .	212
14.5 Coordinate system . . . . .	213
14.5.1 Generalities . . . . .	213
14.5.2 XML codage . . . . .	213
14.5.2.1 Generalities . . . . .	213
14.5.2.2 Geocentric . . . . .	213
14.5.2.3 eTC_WGS84 . . . . .	213
14.5.2.4 Exterior file coordinate system . . . . .	214
14.5.2.5 Locally tangent repair . . . . .	214
14.5.2.6 Polynomial coordinate system . . . . .	214
14.6 Tools for processing trajectory and coordinate systems . . . . .	214
14.6.1 SysCoordPolyn . . . . .	214
14.6.2 The TrAJ2 command . . . . .	215
14.6.3 Trajectory preprocessing . . . . .	215
14.6.3.1 The tool SplitBande . . . . .	215
14.6.3.2 The tool BoreSightInit . . . . .	215
<b>15 Advanced Tie Points</b>	<b>217</b>
15.1 Changing default detector in <b>XML_User/</b> . . . . .	217
15.2 Filtering tie points in <b>HomolFilterMasq</b> . . . . .	217
15.3 Merging Tie point from multiple view with <b>HomolMergePDVUnik</b> . . . . .	218
15.4 Tie points on low contrast images usign SFS in <b>MicMac-LocalChantierDescripteur.xml</b> . . . . .	218
15.4.1 Alternative syntax @SFS . . . . .	222
15.5 Tie point reduction in <b>RedTieP</b> . . . . .	222
15.5.1 Algorithm description . . . . .	222
15.5.2 Parallelization . . . . .	223
15.6 Global and order-agnostic tie point reduction with <b>Schnaps</b> . . . . .	223
15.6.1 Algorithm . . . . .	224
15.6.2 Output . . . . .	224
15.7 Tie point reduction , with <b>OriRedTieP</b> and <b>Ratafia</b> . . . . .	224
15.7.1 Generalities . . . . .	224
15.7.2 Tie point reduction , quasi-vertical case with <b>OriRedTieP</b> . . . . .	225
15.7.2.1 Algorithm . . . . .	225
15.7.2.2 "Von Gruber" point . . . . .	225
15.7.2.3 The command . . . . .	226

15.7.2.4 Memory issue and parallelization . . . . .	227
15.7.3 Tie point reduction , general case with Ratafia . . . . .	227
15.7.3.1 The algorithm . . . . .	227
15.7.3.2 The command line . . . . .	227
<b>16 Advanced orientation</b>	<b>231</b>
16.1 Creating a calibration unknown by image . . . . .	231
16.1.1 When is it necessary? . . . . .	231
16.1.2 Examples . . . . .	231
16.1.3 How to create unknowns . . . . .	231
16.1.4 Saving results with variable calibrations . . . . .	231
16.1.5 Loading initial values with variable calibrations . . . . .	232
16.1.6 Examples with group of poses . . . . .	232
16.1.7 Enforcing a smooth evolution . . . . .	232
16.2 Database of existing calibration . . . . .	233
16.2.1 General points . . . . .	233
16.3 Auxiliary exports . . . . .	233
16.3.1 Generating point clouds with <ExportNuage> . . . . .	233
16.4 Using scanned analog images . . . . .	233
16.4.1 Dealing with internal orientation . . . . .	233
16.4.2 Semi-automatic fiducial mark input with Kugelhupf . . . . .	236
16.4.3 FFT variant with FFTKugelhupf . . . . .	236
16.4.4 Resampling images with ReSampFid . . . . .	237
16.5 Adjustment with lines . . . . .	238
16.5.1 Introduction . . . . .	238
16.5.2 Data set . . . . .	238
16.5.3 Organization of information . . . . .	238
16.5.4 Example Apero-2-DroiteStatique.xml . . . . .	238
16.5.5 Example Apero-3-DroiteEvolv.xml . . . . .	239
16.5.6 Example Apero-4-CompensMixte.xml and Apero-5-CompensAll.xml . . . . .	240
16.6 Recent evolution in Tapas and other orientation tools . . . . .	240
16.6.1 Viscosity & Levenberg Marquardt stuff . . . . .	240
16.6.2 Additional distortion . . . . .	240
16.6.3 Non Linear Bascule (swing) . . . . .	241
16.6.3.1 Motivation . . . . .	241
16.6.3.2 Mathematical model . . . . .	242
16.6.3.3 Using it in MicMac . . . . .	242
16.6.3.4 Example of use, and message interpretation . . . . .	243
16.6.4 A detailed example . . . . .	243
16.6.5 Miscellaneous options to Tapas . . . . .	243
16.6.5.1 FreeCalibInit . . . . .	243
16.6.5.2 FrozenCalibs . . . . .	243
16.6.5.3 SinglePos . . . . .	244
16.7 GCP : accuracy and optimal weighting . . . . .	244
16.8 Initial Orientation with Martini . . . . .	245
16.9 Miscellaneous tools about calibration . . . . .	245
16.9.1 ConvertCalib to Calibration conversion . . . . .	245
16.9.2 Genepi to generate artificial perfect 2D-3D points . . . . .	246
16.9.3 Init11P, space resection for uncalibrated camera . . . . .	246
16.9.4 Aspro, space resection for calibrated camera . . . . .	247
16.10 Rigid Block Compensation . . . . .	247
16.10.1 Introduction . . . . .	247
16.10.1.1 Mathematics . . . . .	247
16.10.1.2 Data set . . . . .	248
16.10.1.3 Preprocessing and camera naming . . . . .	248
16.10.1.4 Standard MicMac Processing . . . . .	249

16.10.2 Indicating block structure . . . . .	249
16.10.3 Block estimation . . . . .	249
16.10.4 Block compensation . . . . .	251
16.10.4.1 Global with no attachment to known value . . . . .	251
16.10.4.2 Global with attachment to known value . . . . .	252
16.10.4.3 Time relative . . . . .	252
16.10.4.4 Combination . . . . .	252
<b>17 Advanced matching, theoretical aspect</b>	<b>253</b>
17.1 Generalities . . . . .	253
17.1.1 Geometric notations . . . . .	253
17.1.2 Notation for quantification . . . . .	253
17.2 Energetic formulation and regularization . . . . .	254
17.2.1 Generalities . . . . .	254
<b>18 Advanced matching, practical aspect</b>	<b>255</b>
18.1 Cost function . . . . .	255
18.2 Exporting the score . . . . .	255
18.2.1 Default behaviour . . . . .	255
18.2.2 Exporting the correlation cube . . . . .	256
<b>19 Using satellite images</b>	<b>257</b>
19.1 With approximate sensor orientation – RPC bundle adjustment (recommended) . . . . .	257
19.1.1 The RPC conversion to MicMac-format files . . . . .	257
19.2 With approximate or refined sensor orientation	
– the GRID/RTO processing . . . . .	258
19.2.1 Pleiades-Spot or DigitalGlobe very high resolution optical satellite images . . . . .	258
19.2.1.1 Image couple . . . . .	259
19.2.1.2 Set of Images . . . . .	260
19.3 Epipolar geometry of a satellite image pair . . . . .	261
19.4 SAKE - Simplified tool for satellite images correlation . . . . .	261
<b>III Algorithmic Documentation</b>	<b>263</b>
<b>20 Généralités</b>	<b>267</b>
20.1 Notations Géométriques . . . . .	267
20.2 Discréétisation et quantification . . . . .	267
<b>21 Approche multi-résolution</b>	<b>269</b>
21.1 Motivations . . . . .	269
21.2 Modèle prédictif . . . . .	269
21.3 Noyaux utilisés pour la sous-résolution . . . . .	269
<b>22 Mesure de ressemblance et corrélation</b>	<b>271</b>
22.1 Généralité . . . . .	271
22.2 Utilisation pour la ressemblance de deux vignettes . . . . .	272
22.3 Fenêtre de "taille 1" . . . . .	272
22.4 "Fenêtre exponentielle" . . . . .	272
22.4.1 Principe des fenêtres à pondération variable . . . . .	272
22.4.2 Equivalence des tailles de fenêtre . . . . .	273
22.5 Multi-corrélation . . . . .	273
22.6 Interpolation . . . . .	274
<b>23 Algorithmes de filtrages rapides</b>	<b>275</b>

<b>24 Approches énergétiques et régularisation</b>	<b>277</b>
24.1 Généralités . . . . .	277
24.2 Programmation dynamique . . . . .	277
24.3 Programmation dynamique "multi directionnelle" . . . . .	278
24.4 Algorithmes de flots . . . . .	279
24.5 Algorithmes de déquantification . . . . .	279
24.6 Algorithmes variationnels . . . . .	279
<b>25 Algorithm on orientation</b>	<b>281</b>
25.1 Tomasi-Kanabe . . . . .	281
25.2 Triplet selection algorithm . . . . .	283
<b>26 Sensibility Analysis</b>	<b>285</b>
26.1 Theoreticall consideration . . . . .	285
26.1.1 some tricks . . . . .	285
26.1.1.1 tricks . . . . .	285
26.1.1.2 tricks . . . . .	285
26.1.1.3 tricks . . . . .	285
26.1.2 Least square notation . . . . .	285
26.1.3 Variance . . . . .	286
26.1.4 Covariance . . . . .	287
26.1.5 Unknown elimination . . . . .	287
26.1.6 Practicle aspects on unknown elimination in MicMac . . . . .	288
26.1.7 Sensibility . . . . .	288
26.2 Use in MicMac . . . . .	289
<b>IV Documentation utilisateur</b>	<b>291</b>
<b>27 Mécanismes Généraux</b>	<b>293</b>
27.1 Généralités et notations . . . . .	293
27.1.1 Boîtes englobantes . . . . .	293
27.2 Géométries . . . . .	293
27.2.1 Géométries intrinsèque et de restitution . . . . .	293
27.2.2 Géométrie intrinsèque (image) . . . . .	294
27.2.2.1 Description générale . . . . .	294
27.2.2.2 Géométrie <eGeomImageOri> . . . . .	294
27.2.2.3 Géométrie <eGeomImageModule> . . . . .	294
27.2.2.4 Géométries <eGeomImage_Epip>, <eGeomImageDHD_Px> . . . . .	294
27.2.2.5 Lecture des paramètres de géométrie . . . . .	295
27.2.3 Géométries de restitution (terrain) . . . . .	295
27.2.3.1 Description générale . . . . .	295
27.2.3.2 Géométries terrain . . . . .	295
27.2.3.3 Géométries <eGeomMNTFaisceauIm1ZTerrain_Px1D> . . . . .	295
27.2.3.4 Géométries <eGeomMNTFaisceauIm1ZTerrain_Px2D> . . . . .	296
27.2.3.5 Géométries <eGeomMNTFaisceauIm1PrCh_Px?D> . . . . .	296
27.2.4 Caractéristiques liées aux géométries . . . . .	296
27.2.4.1 Combinaisons de géométries, dimension de parallaxe . . . . .	296
27.2.4.2 Unités de parallaxes . . . . .	296
27.3 Patrons de sélection et de transformations de chaînes . . . . .	298
27.3.1 Utilisation avec un nom . . . . .	298
27.3.2 Utilisation avec deux noms . . . . .	298
27.4 Librairies Dynamiques . . . . .	299
27.4.1 Fonctionnement Général . . . . .	299
27.4.2 Utilisation pour la géométrie . . . . .	299
27.4.3 Utilisation pour les pyramides . . . . .	299

27.5 Types réutilisés . . . . .	299
27.5.1 Le type FileOriMnt . . . . .	300
27.5.2 Le type SpecFitrageImage . . . . .	300
27.6 Gestion des erreurs . . . . .	300
27.6.1 Bugs . . . . .	300
27.6.2 Erreurs mal signalées . . . . .	300
27.6.3 Erreurs cataloguées . . . . .	300
<b>28 Sections hors mise en correspondance</b>	<b>301</b>
28.1 Section Terrain . . . . .	301
28.1.1 <IntervParalaxe> et <IntervAltimetrie> . . . . .	301
28.1.1.1 Valeurs moyennes de parallaxe . . . . .	301
28.1.1.2 Incertitude de calcul . . . . .	302
28.1.1.3 Incertitude pour le calcul d'emprise . . . . .	302
28.1.1.4 MNT initial . . . . .	302
28.1.2 Planimétrie . . . . .	302
28.1.2.1 Calcul de l'emprise spécifiée . . . . .	303
28.1.2.2 Calcul de l'emprise par défaut . . . . .	303
28.1.2.3 Résolution terrain . . . . .	303
28.1.2.4 Masque terrain . . . . .	303
28.1.2.5 Recouvrement minimal . . . . .	304
28.1.3 Paramètres liés à la "rugosité" . . . . .	304
28.2 Section Prise de Vue . . . . .	304
28.2.1 Images . . . . .	304
28.2.1.1 Ensemble des images . . . . .	304
28.2.1.2 Masque images . . . . .	305
28.2.1.3 Gestionnaire de pyramide d'image . . . . .	306
28.2.2 Géométrie (intrinsèque) . . . . .	306
28.2.2.1 Type de Géométrie . . . . .	306
28.2.2.2 Association images/géométries, cas standard . . . . .	306
28.2.2.3 Nom calculé sur Im1-Im2 . . . . .	307
28.2.2.4 Points homologues . . . . .	308
28.3 Génération de Résultat . . . . .	308
28.3.1 Image 8 Bits . . . . .	308
28.3.2 Image de corrélation . . . . .	308
28.3.3 Basculement dans une autre géométrie . . . . .	308
28.3.4 Parallaxe relative ?? . . . . .	309
28.4 Modèles analytiques . . . . .	309
28.5 Section Espace de travail . . . . .	309
28.5.1 Directory Image . . . . .	309
28.6 Section dite "Vrac" . . . . .	309
<b>29 Mise en correspondance</b>	<b>311</b>
29.1 Généralité . . . . .	311
29.1.1 Organisation . . . . .	311
29.1.2 Mode différentiel, valeurs par défaut . . . . .	311
29.1.3 Equivalence de noms . . . . .	311
29.2 Paramètres globaux . . . . .	311
29.2.1 Clip de la zone de MEC . . . . .	312
29.2.2 Calcul du masque de MEC . . . . .	312
29.2.2.1 Nombre minimal d'images . . . . .	312
29.2.3 Divers . . . . .	312
29.2.3.1 Valeur par défaut de l'attache aux données . . . . .	312
29.2.3.2 Corrélation dégénérées . . . . .	312
29.3 Géométrie et nappes englobantes . . . . .	313
29.3.1 Gestion des résolutions . . . . .	313

29.3.1.1 Résolution terrain . . . . .	313
29.3.1.2 Résolution image . . . . .	313
29.3.2 Pas de quantification . . . . .	313
29.3.3 Calcul des nappes englobantes . . . . .	314
29.3.4 Redressement des images . . . . .	314
29.3.5 Divers . . . . .	314
29.3.5.1 Différentiabilité de la géométrie . . . . .	314
29.4 Autres paramètres d'entrées . . . . .	314
29.4.1 Sélection des images . . . . .	314
29.4.2 Interpolation . . . . .	315
29.5 Approche énergétique . . . . .	315
29.5.1 Attaché aux données . . . . .	315
29.5.1.1 fenêtres de corrélation . . . . .	315
29.5.1.2 Multi-corrélation . . . . .	316
29.5.1.3 Dynamique de corrélation . . . . .	316
29.5.2 A priori . . . . .	316
29.5.2.1 Régularisation . . . . .	316
29.5.2.2 Post-filtrage . . . . .	317
29.5.3 Minimisation . . . . .	317
29.5.3.1 Choix d'un algorithmes . . . . .	317
29.5.3.2 Paramètre spécifiques à Cox-Roy . . . . .	317
29.5.3.3 Paramètre spécifiques à la programmation dynamique . . . . .	317
29.5.4 Sous résolution des algorithmes . . . . .	318
29.5.5 Option non implantées . . . . .	318
29.6 Gestion mémoire . . . . .	318
<b>30 Cas d'utilisation</b>	<b>319</b>
30.1 MNT Spots . . . . .	319
30.2 MNE Urbains . . . . .	319
30.3 Superposition d'images colorées . . . . .	319
30.4 Points homologues pour l'aéro-triangulation . . . . .	319
<b>31 Programmes Utilitaires</b>	<b>321</b>
31.1 Généralités . . . . .	321
31.1.1 Génération du binaire . . . . .	321
31.1.2 Liste des arguments . . . . .	321
31.1.3 Aide en ligne . . . . .	321
31.2 L'utilitaire de changement d'échelle <code>ScaleIm</code> . . . . .	322
31.2.1 Fonctionnalités . . . . .	322
31.2.2 Paramètres . . . . .	322
31.2.3 Evolutions possibles . . . . .	322
31.3 L'utilitaire d'ombrage <code>GrShade</code> . . . . .	322
31.3.1 Fonctionnalités . . . . .	322
31.3.2 Paramètres . . . . .	322
31.3.3 Evolutions possibles . . . . .	323
31.4 L'utilitaire de déquantification <code>Dequant</code> . . . . .	323
31.5 L'utilitaire d'information sur un fichier tiff <code>tiff_info</code> . . . . .	323
31.6 L'utilitaire de test des expression régulière <code>test_regex</code> . . . . .	323
31.7 SupMntIm to superpose image and DTM . . . . .	323
<b>V Documentation programmeur</b>	<b>325</b>
<b>32 Ateliers</b>	<b>327</b>
32.1 C++ course under MicMac's library : Elise . . . . .	328
32.1.1 Introduction and generalities . . . . .	328

32.1.2 How to create a new .cpp file and compile it using the library Elise? . . . . .	328
32.1.2.1 Hello World ! . . . . .	328
32.1.3 Mandatory or Optional Argument? . . . . .	329
32.1.4 How to load an xml file and read its informations? . . . . .	330
32.1.5 How to get list of files in a folder? . . . . .	331
32.1.6 Epipolar geometry . . . . .	333
32.1.7 Multi Image Correlation . . . . .	335
32.2 C++ course under MicMac's library : Elise . . . . .	345
32.2.1 Introduction and generalities . . . . .	345
32.2.2 Overview of the new classes and interfacing with <i>mm3d</i> . . . . .	345
32.2.3 Preprocessing . . . . .	345
32.2.3.1 Epipolar images . . . . .	345
32.2.3.2 3D mask . . . . .	345
32.2.3.3 Multiscale approach . . . . .	346
32.2.4 The matching . . . . .	346
32.2.4.1 The similarity measure . . . . .	346
32.2.4.2 Matching without regularization . . . . .	347
32.2.4.3 Matching with regularization . . . . .	349
32.3 Visual interfaces "vCommands" . . . . .	352
32.3.1 Introduction . . . . .	352
32.3.2 Compilation and code . . . . .	352
32.3.3 How it works? . . . . .	352
32.3.4 <b>visual_MainWindow</b> class . . . . .	354
32.3.5 Specific functions: vTapioca, vMalt, vC3DC, vSake . . . . .	354
32.3.6 BoxClip and BoxTerrain . . . . .	355
32.4 SaisieQT . . . . .	355
32.4.1 Introduction . . . . .	355
32.4.2 Compilation and code . . . . .	355
32.4.3 How it works? . . . . .	355
32.4.4 SaisieMasqQT . . . . .	356
32.4.4.1 2D mode . . . . .	356
32.4.4.2 3D mode . . . . .	356
32.4.5 SaisieAppuisInitQT and SaisieAppuisPredicQT . . . . .	357
32.4.6 SaisieBascQT . . . . .	357
32.4.7 SaisieCylQT . . . . .	357
32.4.8 SaisieBoxQT . . . . .	357
32.5 Conventions for 3D selection tool . . . . .	358
<b>33 Génération automatique de code</b>	<b>361</b>
<b>VI Annexes</b>	<b>363</b>
<b>A Formats</b>	<b>365</b>
A.1 Calibration formats . . . . .	365
A.2 Grilles de calibration . . . . .	365
A.2.1 Format de codage des déformations du plan . . . . .	365
A.2.1.1 Format . . . . .	365
A.2.1.2 Utilisation . . . . .	365
A.2.2 Application à la calibration interne . . . . .	366
A.2.2.1 Rappels et notation . . . . .	366
A.2.2.2 Codage des distorsion par des grilles . . . . .	367
A.2.2.3 Justification des modèles paramétriques . . . . .	367
A.2.3 Application à une rotation près . . . . .	368
A.2.3.1 Formalisation . . . . .	368
A.2.3.2 Conséquence pour la comparaison de calibration . . . . .	369

A.2.4	Point principal . . . . .	369
A.2.4.1	Le point principal bouge . . . . .	369
A.2.4.2	On a perdu le point principal . . . . .	370
A.2.4.3	On a retrouvé le point principal ? . . . . .	370
A.2.5	Paramètres hors grilles . . . . .	370
A.2.5.1	Paramètres photogramétrique "traditionnel"	370
A.2.5.2	Autres paramètres . . . . .	370
A.3	Points Homologues . . . . .	370
A.4	Fichiers MNT . . . . .	370
<b>B</b>	<b>Vrac</b>	<b>371</b>
B.1	Notation . . . . .	371
B.2	Géométrie épipolaire . . . . .	371
B.3	Matrice essentielle . . . . .	372
B.4	Calcul de l'orientation relative par matrice essentielle . . . . .	372
B.5	Cas planaire . . . . .	372
B.6	Grandes focales, projection axo et points triples . . . . .	372
B.7	Géométrie épipolaire . . . . .	372
B.8	Matrice essentielle . . . . .	372
B.9	Grandes focales, projection axo et points triples . . . . .	372
<b>C</b>	<b>Vrac -Bis</b>	<b>373</b>
C.1	Introduction-Supression des inconnues auxiliaires . . . . .	373
C.2	Formulation algébrique du traitement des inconnues auxiliaires . . . . .	374
C.3	Prcision et corrlation sur les paramres . . . . .	375
<b>D</b>	<b>Vrac -Bis</b>	<b>377</b>
D.1	Utilisation des foncteurs compilés . . . . .	377
D.1.1	Filière non indexee . . . . .	377
D.1.2	Filière indexee . . . . .	378
D.2	Communication avec les systèmes sur-contraints . . . . .	379
D.2.1	Convention d'écritures . . . . .	379
D.2.2	UseEqMatIndexee . . . . .	380
D.3	Mesh computation . . . . .	380
D.3.1	Command Nuage2Ply : . . . . .	380
D.3.2	Command MergePly : . . . . .	380
D.3.3	Example . . . . .	380
<b>E</b>	<b>Various formula</b>	<b>383</b>
E.1	Space resection . . . . .	383
E.2	Stretching (etirement) . . . . .	383
<b>F</b>	<b>Référence bibliographique</b>	<b>387</b>



# **Part I**

# **Generalities**



## 0.1 Foreword

In 2007 I began to write some of MicMac documentation in French. Then, for different reasons (laziness, lack of courage, idleness, ...) I stopped.

In March 2011, as preparing a course on my photogrammetric tools, I decided to start again this documentation. I thought it would be useful to do it in a language (hopefully) close to English. This is the new version you are reading. However, I doubt that it may be complete before a long time and, during this transitional step, I will conserve the existing French chapter at the end of this documentation and there may be some cross references between English and French chapters.



# Chapter 1

## Introduction

### 1.1 History, Status and Contributors

This is the documentation of a set of software photogrammetric tools that, under certain conditions, allow to compute a 3D modelization from a set of images.

MicMac is a tool for image matching. I began to write it in 2005, while working at the French National Geographic Institute (IGN), as a tool integrating several recent results of the scientific community. It is a general purpose tool, probably in many (if not all) specific contexts, one will be able to find a more accurate tool. However, one of its expected advantages is its generality. It has been used in a lot of different contexts, for example:

- digital terrain model in rural context from pairs of satellite images, with exact or approximate orientation;
- digital elevation model in urban context with high resolution multi-stereoscopic images;
- detection of terrain movements;
- 3D modelization of objects (sculptures) or interior and exterior scenes;
- multi-spectral matching of images registration.

Of course this generality comes with a price . . . : it requires a lot of parameterization which sometimes turns to be quite complex. For 3D computation, MicMac works only with oriented images like the ones resulted from classical aero-triangulation process. Early in 2007, there were several opportunities that encouraged me to create a tool that could orientate a set of overlapping images, so that they can be matched in MicMac:

- I bought my first reflex digital camera, and thought it would be fun to be able to make 3D models from my holidays pictures, which turned to be right;
- I discovered the existence of the magical SIFT algorithm from David Lowe, and thought this would make this idea feasible by solving the tie point problem, which turned to be right;
- I had already written several pieces of software, including some calibration tools, which could be reused and made me think it could be done easily and quickly, which turned to be wrong . . .

Since 2008, several tools were added to solve specific requirements: tools for ortho-photo, tools for demosaicing . . . Since 2007, MicMac is an open source software, under the CeCILL-B license (an adaptation to the French law of the L-GPL license); as far as I understand law (not very much) all the other tools described in this document are extensions and evolutions of MicMac and obey to the same license.

Different people have helped me in writing these tools:

- Gregoire Maillet for supporting satellite orientation models (grid of rpc),
- Arnaud Le Bris for adaptation of **Sift++** supporting large images,
- Didier Boldo for the first Windows adaptation,
- Aymeric Godet and Livio de Luca for developing two different user friendly interfaces and also making many tests,
- Christophe Meynard for solving some tricky Linux problems,
- Christian Thom for the first idea of multi-correlation,
- Jean-Michaël Muller for improvements about installation,
- Ana-Maria Rosu for many typo corrections (alas, I can create them faster than she can correct them).

— since september 2012, the *culture 3D* team : Jérémie Belvaux, Gérald Choqueux, Matthieu Deveau.

Of course, there are also many people who helped without knowing by creating free softwares which I integrated:

- AMD (<http://www.cise.ufl.edu/research/sparse/amd/>) for approximate minimum degree ordering;
- SIFT
- DCRAW by Dave Coffin (<http://www.cybercom.net/~dcoffin/dcraw/>) for using raw image;
- image magick for convert, for using jpg images (<http://www.imagemagick.org/>);
- proj4 for handling cartographic projection (<http://trac.osgeo.org/proj/>) ;

## 1.2 Prerequisites

These tools are low-level tools. Although I will try to make this documentation as clear and self-contained as possible, there are some prerequisites:

- the reader must be comfortable with the Linux PC on which the software will be installed; at least, if you are not familiar with installation of software from source code, you should have the support of an administrator;
- some very basic notions of photogrammetry are necessary, not a lot (for example to have a notion of what cross-correlation, epipolar geometry, rotation matrix are).

## 1.3 Installation and Distribution

### 1.3.1 Documentation

This document is rather a "reference" documentation, not specially tuned for an easy begining; for reference to photogrammetry with MicMac see :

- [http://jmfriedt.free.fr/lm\\_sfml\\_eng.pdf](http://jmfriedt.free.fr/lm_sfml_eng.pdf) by JM Friedt;
- <http://forum-micmac.forumprod.com/bibliography-f38.html> a section of the forum dedicated to bibliography;
- several documents in the Documentation folder and sub folder of MicMac distribution (see in particular pdf file and Paper-Algo/ , Paper-MPD/ , Papers-Internship/ , Paper-UseCase/ sub folder)
- micmac-tutoriel-de-photogrammetrie

There is also a forum where you will find answers to the most frequent question you may have : <http://forum-micmac.forumprod.com/>

### 1.3.2 Install

These tools are written in C++ . They are distributed mainly in source code format that you have to compile. As described below, they are relatively low level tools, and the installation, computation and running of these tools require some basic background in practical computer science.

See now the section 3.2 for the new website since 2012, install mercurial and follow the instruction. Several links that may be useful :

- [http://logiciels.ign.fr/?Telechargement\\_20](http://logiciels.ign.fr/?Telechargement_20) for the sources
- <http://forum-micmac.forumprod.com/new-site-of-micmac-since-januray-2013-t340.html>
- <http://forum-micmac.forumprod.com/tuto-install-t518.html>

In this documentation, you will find examples that require some data. The server is now a ftp server :

```
ftp2.ign.fr
user = micmac_user
pwd = scAEf9MR
```

## 1.4 Libraries, Programs and Dependencies

These softwares have few dependencies to other libraries or programs. Basically, if you use tiff or raw files as input, and if you do not use any of the graphical tools provided<sup>1</sup>, there might not be any dependency.

By the way, on Linux, as the graphical interface are by default required, the compiler will require the header file `X.h`, `Xlib.h`, `Xutil.h`, `cursorfont.h`, `keysym.h`. If they are not installed, you can easily get them with something like :

```
sudo apt-get install x11proto-core-dev libx11-dev
```

Most users will want sooner or later to use jpeg files. In this case, it will be necessary to have installed the command `convert`, this command is a part of the excellent `ImageMagick` package.

The `draw` source code I use to handle xif files info is sufficient in most cases. However, when it fails, I try to use the `exiv2` tool. I also recommend that you install this excellent and free package.

I also recommend that you install the excellent package `exiftool`, it is a free open source package and has the ability to read many `xif` information (including GPS tags that will be soon usable in `Apero`).

## 1.5 Interface for the Tools

### 1.5.1 Kinds of Interfaces

There are roughly three kinds of interfaces for softwares:

- user friendly graphical interface, with intuitive menu and window etc. Its advantage is that it may be usable by all final users, the drawback of this solution being the cost for the developer;
- API or application programming interface. Using this level of interface requires you to use one of the programming language the API is functioning with. One of the drawbacks of these API is that they require a lot of documentation;
- a set of programs that you can call on a command line, with parameters being added on a command line or included in a file.

The tools described here use mainly the third kind of interface. This seemed to be the optimal solution as these tools have been primarily developed for my own usage and usage of colleagues from the same building. Since this is not optimal for end users, some user friendly graphical interfaces have been added, to help to set parameters.

### 1.5.2 Simple Tools

The tools described here are all command line tools. Their parameters can be added directly on the command line or, for more complex tools (like `Apero` and `MicMac`) the parameters are provided in an XML file.

Here is an example of calling the command `GrShade` for computing the shading of a depth image:

```
bin/GrShade ../micmac_data/Boudha/F050_IMG_5571_MpDcraw8B_GB.tif Visu=1 FZ=0.1
```

The simple tools described here, that have all their parameters on command lines, include:

- `bin/GrShade` for computing shading;
- `bin/Nuage2Ply` for transforming depth map in cloud point in ply format;
- `bin/ScaleIm` for rescaling an image (with some care on aliasing);
- `bin/ScaleNuage` for scaling a depth map.

Generally, these tools understand the syntax `bin/Tool -help` that prints the syntactic description of the command. For example `bin/Nuage2Ply -help` will print on the terminal:

```
*****
* Help for Elise Arg main *
*****
Unnamed args :
```

---

1. for handling mask, visualize tie points ...

```
* string
Named args :
* [Name=Sz] Pt2dr
* [Name=P0] Pt2dr
* [Name=Out] string
* [Name=Scale] REAL
* [Name=Attr] string
* [Name=Comments] vector<std::string>
* [Name=Bin] INT
* [Name=Mask] string
* [Name=Dyn] REAL
```

This indicates that `bin/Nuage2Ply` has one mandatory argument, of type `string`; mandatory arguments come first and the order matters. `bin/Nuage2Ply` also admits several optional arguments. For example, there is one optional argument named `Scale`, of type `REAL`. If this argument is to be specified with the value 2.5, the command line will contain `Scale=2.5`. Of course the command `bin/Tool -help` gives information essentially on the syntactic aspect, the semantic has to be found in this documentation (when the chapter exists ...).

### 1.5.2.1 GUI for command line tools

For each command line tool, a graphical interface can be launched to help setting parameters. To run this interface, one should replace command name in command line by `mm3d + "v" + command`. For example, to set parameters for command `GrShade`, one should call:

```
bin/mm3d vGrShade
```

This will raise an interface where parameters can be set, and where all available options are shown:

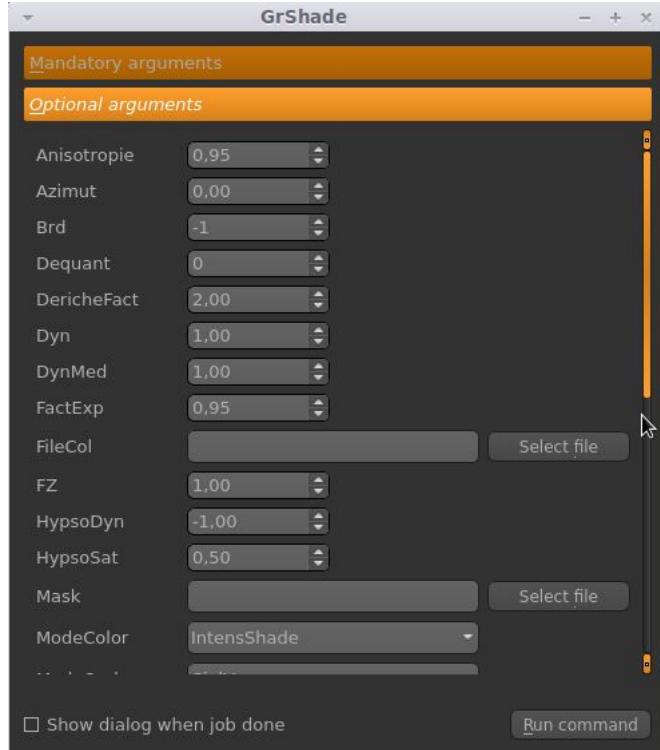


Figure 1.1: Visual interface for command line tools

### 1.5.3 Complex Tools

For a more complex command, that requires arbitrary numbers of arguments, the command line would not be manageable. For this command, it has been decided to use an XML file for specifying the parametrization. XML has the following advantages:

- it is a standard, with current specialized editor;
- the name tagging convention, although heavy for writing, make it easier to read;
- it allows textual description of attributed tree structures, which is exactly what is required for complex parametrization.

Here is an example for calling Apero:

```
bin/Apero ../micmac_data/Ref-Apero/Test-Lion/AperoQuick.xml
```

If you downloaded the data example as described in 3.2.1, you could have a look at the `AperoQuick.xml` file to see what it looks like. For all these complex tools, that exit in an XML file, there is a formal description of the XML file that is correct from the syntactic point of view. These XML formal description files are all located in the `include/XML_GEN/` directory. For example, the file `include/XML_GEN/ParamApero.xml` contains a formal description of the XML files which are syntactically valid XML files for the `Apero` program.

How the formal files are used to specify the valid files is too complex to describe it here. The mechanism is described in chapter 10. Basically, the idea is that the parameter file must be a sub-tree of the specification file satisfying some arity constraints.

Generally, the XML file can be modified using optional command line arguments. For example, you can run one of the example data set with no argument:

```
bin/MICMAC /home/mpd/micmac_data/Jeux1-Spot-Epi/Param-0-Epi.xml
```

But if, for some reason, you want to start the computation directly from the second step you shall add an optional argument and type:

```
bin/MICMAC /home/mpd/micmac_data/Jeux1-Spot-Epi/Param-0-Epi.xml FirstEtapeMEC=2
```

### 1.5.4 Where Calling the Tools From (The Mandatory Working Directory)

At the beginning of `MicMac`, it was mandatory to run the file from the `micmac` directory. This is why in all the examples you will see commands like `bin/MICMAC ....`. As I had a lot of complaints about this not being very convenient, I have corrected this fact for most of the tools. However, I do not guarantee that this has been corrected everywhere. So if you encounter problems, you should try to run the file from the `micmac` directory.

## 1.6 Data Organization and Communication

When you want to use photogrammetric tools for complex tasks, there are a lot of things about data organization that has to be specified to programs. For example:

- at a given step, you want to orientate a certain subset of images of a project; so you need to have the possibility to specify sets and subsets of files;
- sometimes you will want to specify that if an image name is `toto_123.tif` or `toto_0123.tif` then the associated orientation is `123_tata.xml`; so you need to have the possibility to specify the probably complex rules of computation that transform strings to strings;
- sometimes you will want to specify that a matching process (for example tie points computation) must be executed between all pairs of images satisfying certain conditions; so you need to have the possibility to specify relations (in the mathematical way).

All these tasks may be performed by a database management system. Although there are some very efficient systems such as open source systems, this is not what I chose for supporting this functionality (because I wanted my tools to stay relatively autonomous). Maybe it was not a good choice, however it has to be assumed now.

The precise mechanism is quite complex and it is described in the chapter 11. The main ideas are:

- there is a huge use of modern regular expressions to specify string sets and string manipulation. For example, the pattern `Img([0-9]{4}).tif` will describe the name set beginning with `Img`, followed by four digits and ending with `.tif`. If one wants to specify that the file associated to `Img1234.tif` is `Ori/1234-HH.xml`, there will be something like `Ori/$1-HH.xml` associated to `Img([0-9]{4}).tif` (the meaning is that `$1` is to be replaced by the first sub-expression between parenthesis);
- to facilitate the sharing of sets, transformations, relations ... between programs, generally they are not manipulated directly. They are created in a common file and are given a name (or Key). The program refers to these objects by their key which facilitates name convention sharing. For example, if the transformation `Img([0-9]{4}).tif → Ori/$1-HH.xml` is to be used to describe the association between image and orientation, it may be declared in the file `MicMac-LocalChantierDescripteur.xml` under the key `Key-Im2Ori`, this key will then be used in `Apero` for the creation of orientation file and in `MicMac` for using the result of `Apero`;
- a lot of pre-existing conventions are automatically loaded by the tools, and for most of the cases these standard conventions should be sufficient.

For example

## 1.7 Existing Tools

The pipeline for transforming a set of images in a 3D model, and optionally generating ortho-photo, is made essentially of four "complex" tools:

- **Pastis**. In fact, this tool is no more than an interface to the well known `Sift++`, distribution of `Sift`, there is no algorithmic added value. Its advantage is to integrate the tie point generation in a way compatible with the global pipeline;
- **Apero** starts from tie points generated by **Pastis**, and optional complementary measurements, and generates external and internal orientations compatible with these measurements;
- **MicMac** starts from orientation generated by **Apero** and computes image matching;
- **Porto** starts from individual rectified images, that have been optionally generated by **MicMac**, and generates a global ortho-photo; this tool is still in a very early stage.

There are several auxiliary tools that may be helpful for importing or exporting data at different steps of this pipeline:

- **BatchFDC** for batching a set of commands;
- **Casa** for computing analytic surface (cylinder ...), from points cloud, very early stage;
- **ClipIm** for clipping image;
- **ConvertIm** for some image conversion;
- **Dequant** for quantifying an image;
- **GrShade** for compute shading from depth image;
- **MapCmd** transforms a command working on a single file in a command working on a set of files;
- **CpFileVide** to complete
- **MpDcraw**, an interface to the great `dcraw` offering some low-level service useful for image matching;
- **MyRename** for image renaming, using modern regular expression and giving the possibility to integrate xif data in the new name, tricky but necessary in the existing pipeline;
- **Nuage2Ply**, a tool to convert depth map in point cloud;
- **SaisieMasq**, a user friendly (compared to others ...) tool to create mask upon an image;
- **ScaleIm**, tool for scaling image;
- **ScaleNuage**, tool for scaling internal representation of point cloud;
- **tiff.info**, tool for giving information about a tiff file;
- **to8Bits**, tool for converting 16 or 32 bit image in a 8 bit image.
- **SupMntIm**, tool for generating a superposition of image and MNT in hypsometry and level curves.
- **PanelIm**. Gather images in a panel.

## Chapter 2

# Some Realization Examples

This chapter contains some 3D models realized with the tools presented in this documentation. Its purpose is to give an idea of what can be achieved with these tools, it has no didactic purpose. For now it is just a gallery, some comments will be added later.

There are also several interesting sites corresponding to use cases in cultural heritage and environmental application :

- <http://www.tapenade.gamsau.archi.fr>, for architecture and archeology;
- <https://sites.google.com/site/geomicmac/home/documentation>, for geology and surveying;
- micmac-tutoriel-de-photogrammetrie-sous, for architecture;
- <http://c3dc.fr/galerie/>, for cultural heritage

The DocMicMac directory<sup>1</sup> there are also several documents more or less related to these tools :

- the directory Paper-MPD contains some papers describing Apero/MicMac and protocols :
  - MPD-Eurocow-17.docx focus on acquisition protocols for orientation;
  - Collection\_EDYTEM\_12-2011\_Images\_et\_modles\_3D\_en\_milieux\_naturels.pdf focus on application to natural environments;
- the directory Paper-UseCase/ contains papers from colleagues that describe some experiments with Apero/MicMac;
- the directory Papers-Internship/ contains some reports of internships done by students of our school and using Apero/MicMac;
- the directory Paper-Algo// contains papers relative to some algorithmic aspects;

## 2.1 3D Objects

### 2.1.1 Statues

See 2.1

---

1. where this file is originally located

**2.1.2 Architectural Details****2.2 Indoors Global Modelization****2.2.1 Architecture****2.3 Globally Planar Objects****2.3.1 Elevation and ortho images****2.3.2 Painting and Fresco****2.3.3 Bas-relief****2.3.4 Macro-photo****2.4 Aerial Photos****2.4.1 Urban DEM****2.4.2 Satellite Images****2.4.3 UAV Missions****2.5 Miscellaneous****2.5.1 Industrial****2.6 Gallery of images**

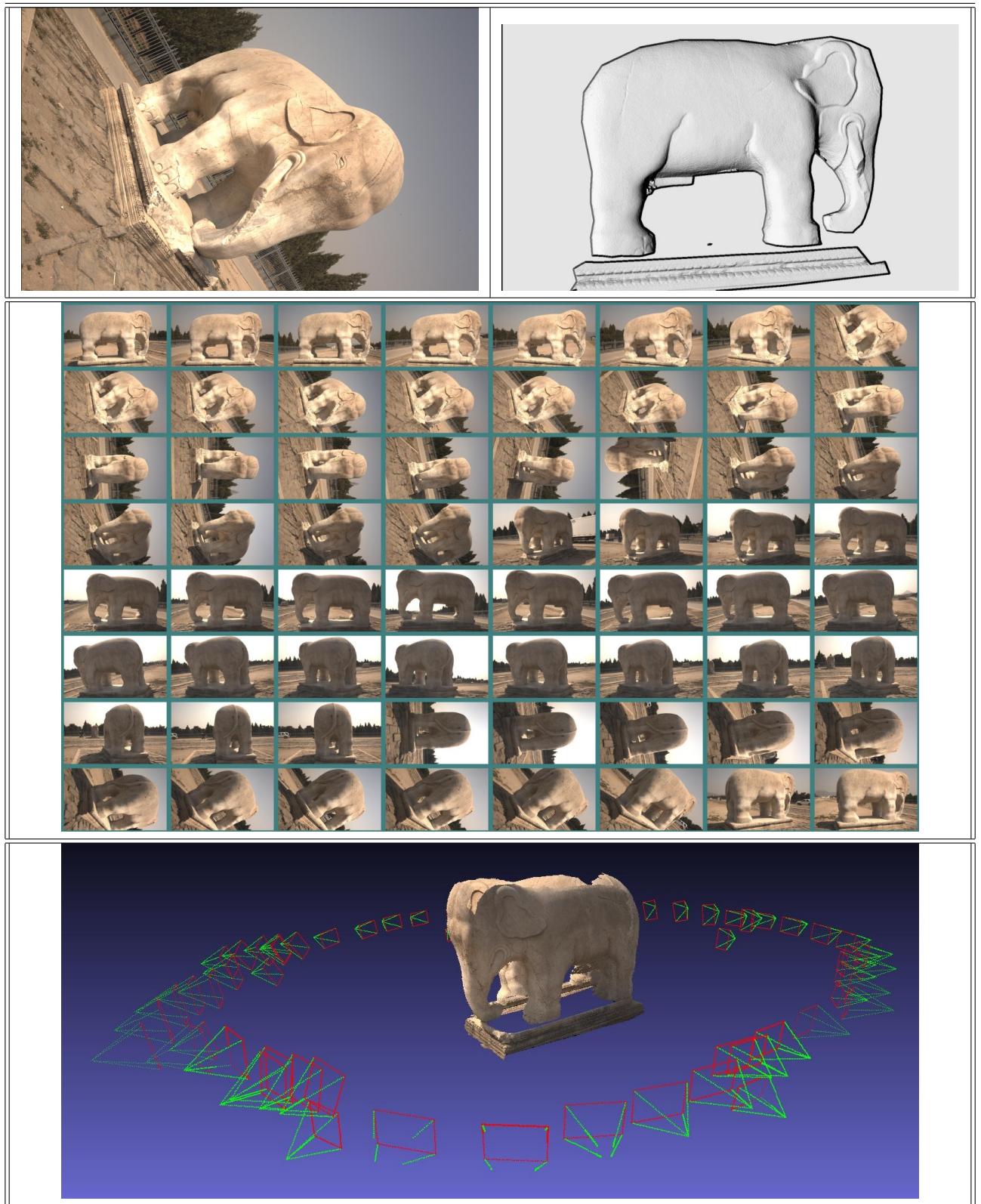


Figure 2.1: **Statues:** elephant in Chian long temple, one of the 60 images, a shaded mode, a global view of the 60 image, the 3D model and the camera position

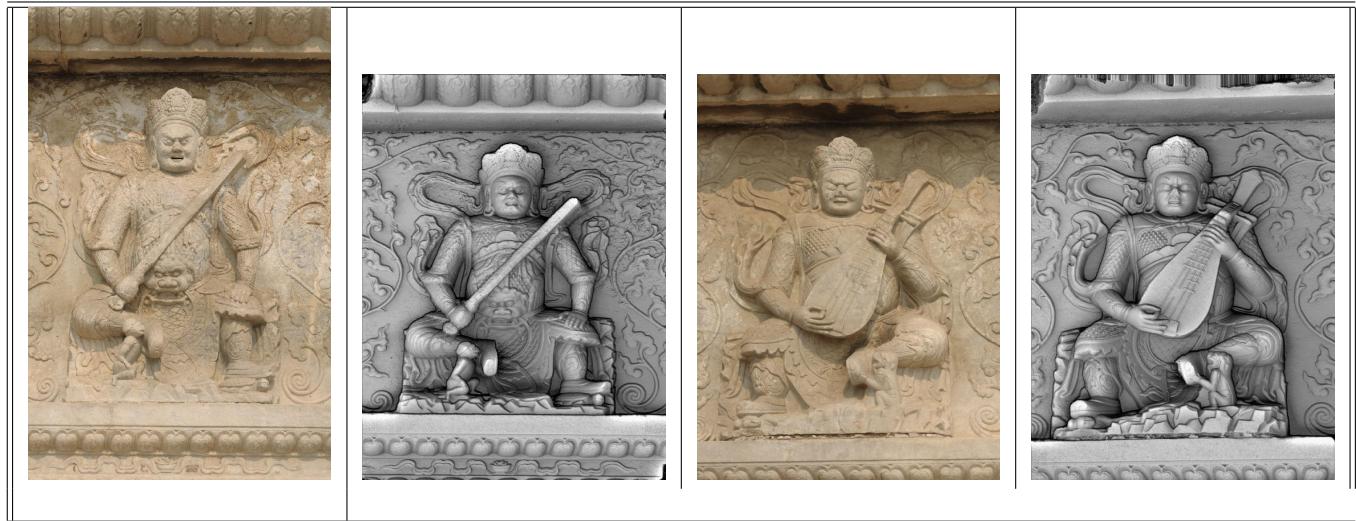


Figure 2.2: **Statues:** Zhenjue temple

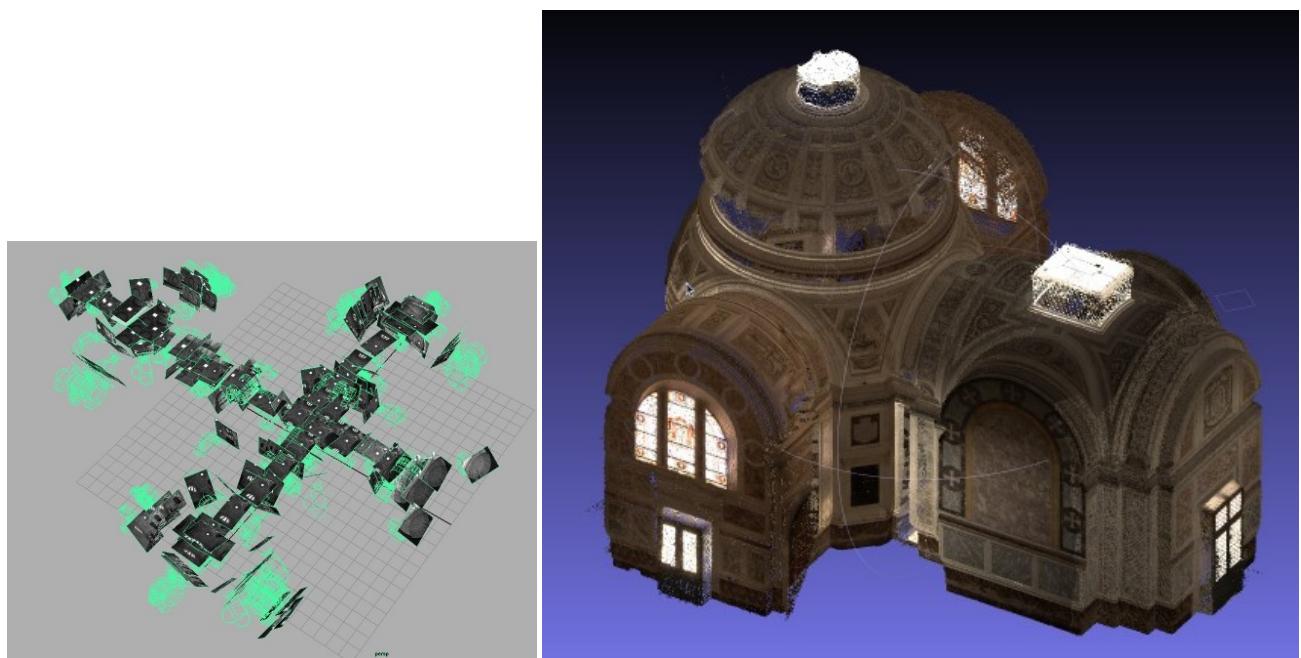


Figure 2.3: **Indoor architecture:** Chapelle imperiale Ajaccio, with 100 fish-eye images; left position of camera, right 3D model



Figure 2.4: The set of images acquired on a wall in Pompeii, a global view of the 3D model of the wall and a global view of orthophoto

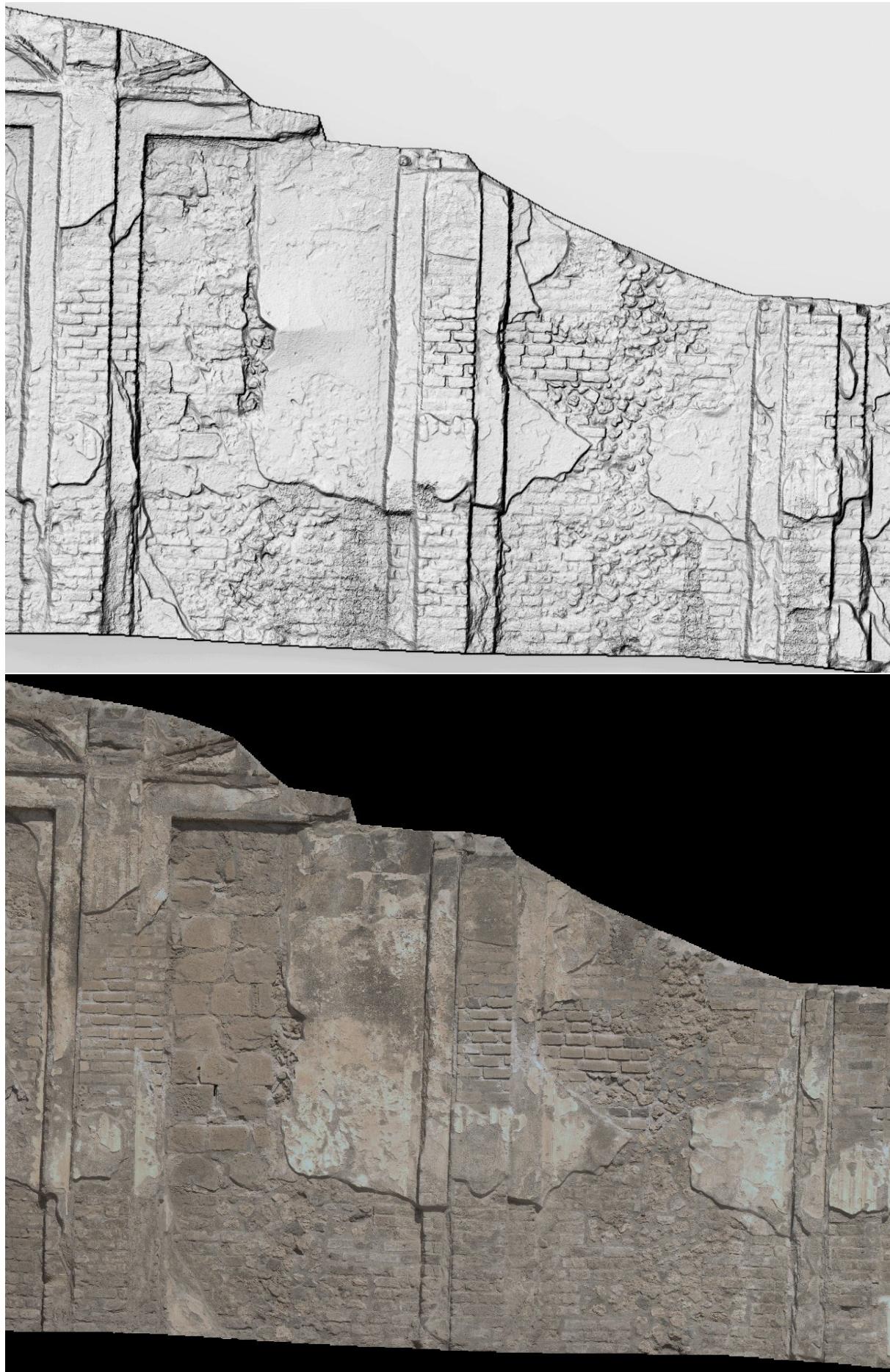


Figure 2.5: Detail on 3D model and ortho photo in Pompei



Figure 2.6: 3D model and ortho photo on "Collégiale Notre Dame de la Garde (Lamballe)"

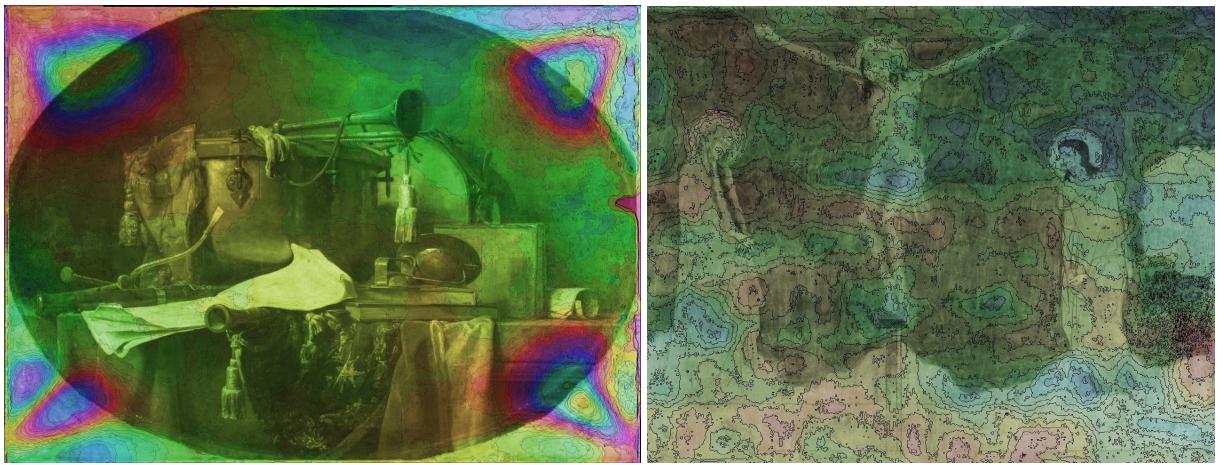


Figure 2.7: **Painting and Fresco:** Fine depth maps computation on painting and fresco, images and in superposition level curves and hypsometry; left image, photo C2RMF/Jean Marsac; right fresco in Villeneuve les Avignon, photo CICRP/Odile Guillon

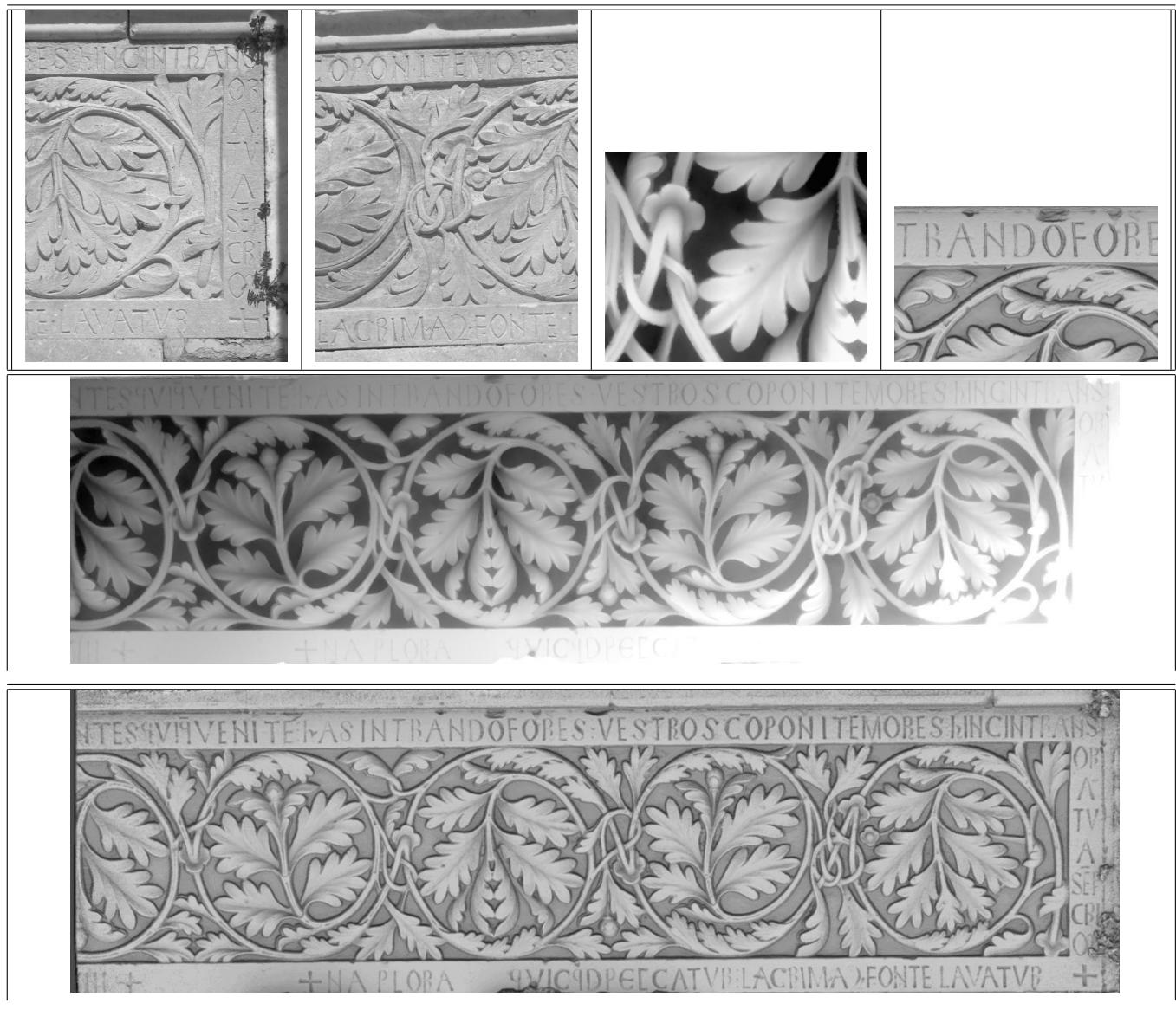


Figure 2.8: **Bas Relief** Frise in Villeneuve-ls-Maguelone ...

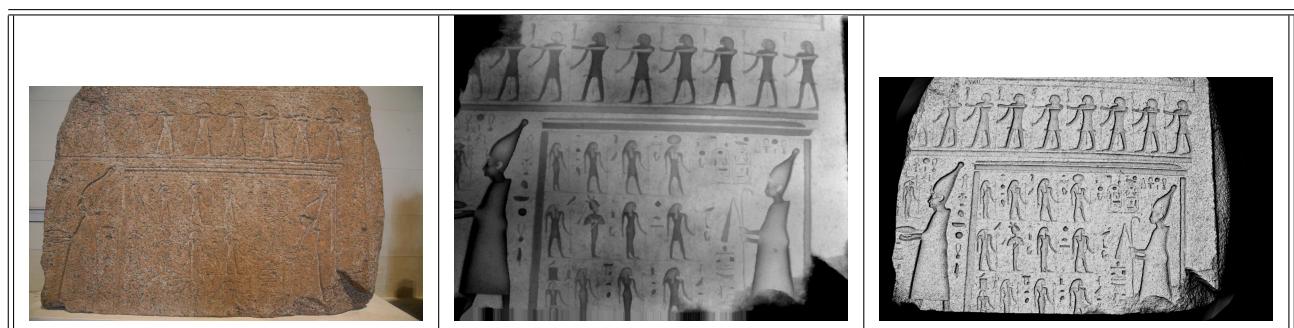


Figure 2.9: **Bas Relief** stone in Louvre, image and 3D model rendered in shading and depth map ...

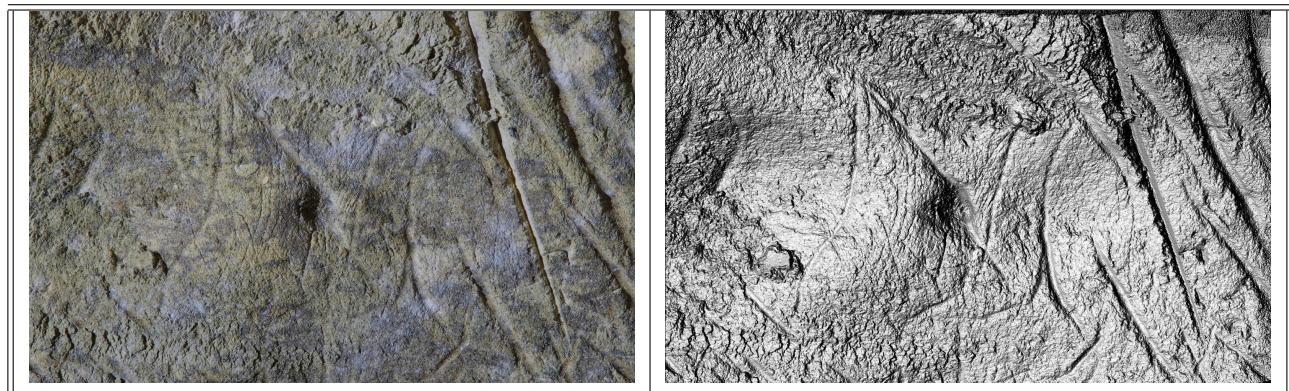


Figure 2.10: Macro photography in Rouffignac cave, at  $\frac{1}{20}$  mm resolution: image and 3D model shading

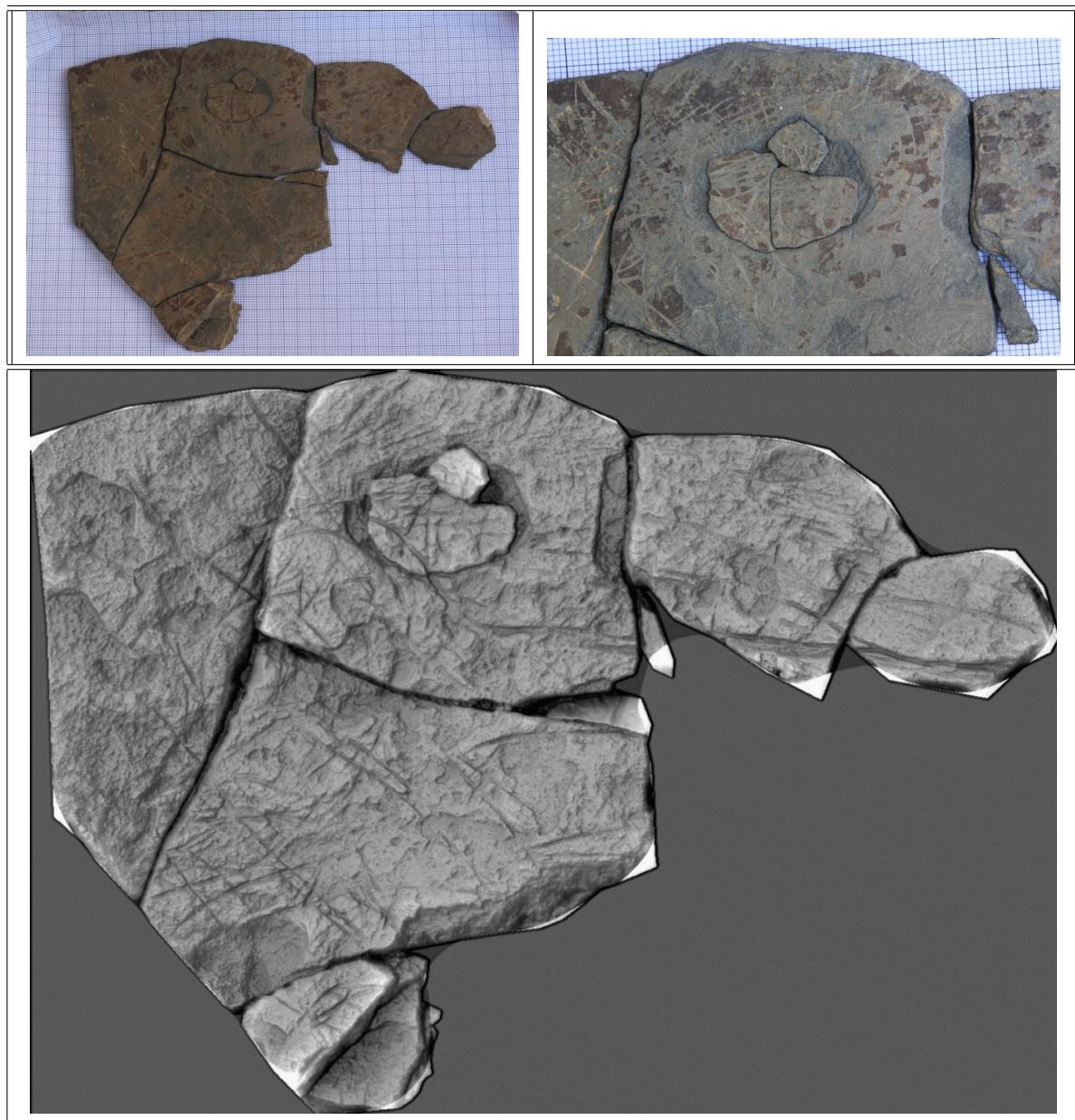


Figure 2.11: Macro photography booklet of Mayenne science cave ...



Figure 2.12: Digital elevation model on semi urban area, 8cm resolution, DGPF data set, for Euro-SDR benchmarking on image matching



Figure 2.13: A more global view of DEM on DGPF data set



Figure 2.14: Forteresse de Salses, photo acquired by drone survey, in collaboration with Map-CNRS; hypsometry and shading, ortho photography, oblique view of 3D model, Euro-SDR benchmarking on image matching

# Chapter 3

## Simplified Tools

This chapter describes simplified tools that allow to make computations without filling XML-files. Of course they cannot deal with all the situation that are handled by complex tools, but I hope that in near future they will be sufficient for 95% of usages.

### 3.1 All in one command

These tools are still in development but, I hope, the first complete version will be available soon. However many tools already exist, what is ready now :

- full automatic tie points computation works (see `Tapioca` tool in 3.3);
- full automatic orientation computation works (see `Tapas` tool in 3.4);
- full automatic matching is not achieved, there exist some piece of code that may be already useful for some users, see 3.11;
- semi-automatic matching works with tool `Malt`, see 3.12;

### 3.2 Modification since Mercurial version

Since end of 2012, several modification on the general organization of the project occurred. This section describes the main modification. Although much care were taken to guarantee a strict compatibility with previous version, it is recommended to use the new mechanisms.

#### 3.2.1 Installing the tools

The main modification on the distribution are :

- the versionning tool is now `mercurial`
- the tools are working on `Linux`, `MacOs` and `Windows`;
- the tools are also distributed on binary version (however, it remains of course an open source project and it is still possible to download the source code).

To get the binary version, go to : <http://logiciels.ign.fr/?Telechargement>, 20

To get the source (you will need to install the mercurial versionning system) type :

```
hg clone https://culture3d:culture3d@geoportail.forge.ign.fr/hg/culture3d
```

To update source code, type:

```
hg pull https://culture3d:culture3d@geoportail.forge.ign.fr/hg/culture3d
```

```
hg up https://culture3d:culture3d@geoportail.forge.ign.fr/hg/culture3d
```

### 3.2.2 The *new universal* command `mm3d`

This section describe a significant modification that occurred since end of 2012. To decrease the size of binary version, and to facilitate and unify the development, the syntax for calling the tools is now based on a unique command `mm3d`. The general syntax is :

```
mm3d Command arg1 arg2 ... argn  NameOpt1=Argot1 ...
```

For example, a possible call to the `Tapas` tool with the new syntax would be :

```
mm3d Tapas RadialStd ".*.PEF" Out=All
```

For backward compatibility (support of existing user's script), the old syntax is still supported for most of the existing tool. For example, it is still valid to write :

```
Tapas RadialStd ".*.PEF" Out=All
```

However, it is recommended for new scripts to be based on the universal command `mm3d`.

### 3.2.3 Help with `mm3d`

When typing only `mm3d`, user can get a list of existing commands :

```
mm3d
mm3d : Allowed commands
AperiCloud Visualization of camera in ply file
Apero Compute external and internal orientations
AperoChImSecMM Select secondary images for MicMac
Bascule Generate orientations coherent with some physical information on the scene
BatchFDC Tool for batching a set of commands
Campari Interface to Apero, for compensation of heterogeneous measures
ChgSysCo Chang coordinate system of orientation
CmpCalib Do some stuff
cod Do some stuff
CreateEpip Tool create epipolar images
Dequant Tool for dequantifying an image
Devlop Do some stuff
ElDcraw Do some stuff
....
```

If the first argument is not an existing command, some indication is given to help to find the "good" name. For example, if one knows that command begin by `ta` :

```
mm3d ta
Suggest by Prefix Match
    Tapas
    Tapioca
    Tarama
    Tawny
```

On the other hand, if you type `mm3d ascul` , as there is no command beginning by `ascul`, one will get all the commands that contain `ascul` :

```
mm3d ascu
Suggest by Subex Match
    Bascule
    GCPBascule
    CenterBascule
    NuageBascule
    RepLocBascule
    SBGlobBascule
```

Finally if arg is nor a prefix nor a subexpression of any command, it will be tested as posix regular expression :

```
mm3d .*C.*asc.*
Suggest by Pattern Match
  GCPBascule
  CenterBascule
  RepLocBascule
```

### 3.2.4 Log files

For the main command, a log file mm3d-LogFile.txt is created, this file stores a global history of all the processing. An example extracted from my dataset :

```
home/marc/MMM/culture3d/bin/mm3d Tapioca MulScale Abbey-IMG_.*.jpg 200 800
  [Beginning at ] Wed Jan  2 22:34:04 2013
  [Ending correctly at] Wed Jan  2 22:35:19 2013
=====
/home/marc/MMM/culture3d/bin/mm3d Tapas RadialBasic Abbey-IMG_02[1-2][0-9].jpg Out=Calib
  [Beginning at ] Wed Jan  2 22:37:11 2013
  [Failing with code 256 at ] Wed Jan  2 22:37:24 2013
=====
/home/marc/MMM/culture3d/bin/mm3d Tapas RadialBasic Abbey-IMG_03.*.jpg Out=Calib
  [Beginning at ] Wed Jan  2 22:39:22 2013
  [Ending correctly at] Wed Jan  2 22:39:24 2013
=====
/home/marc/MMM/culture3d/bin/mm3d Tapas RadialBasic Abbey-.*.jpg InCal=Calib Out=All-Rel
  [Beginning at ] Wed Jan  2 22:39:57 2013
  [Ending correctly at] Wed Jan  2 22:40:21 2013
=====
/home/marc/MMM/culture3d/bin/mm3d Campari Abbey.*.jpg RTL-Bascule RTL-Compense GCP=[AppRTL.xml,0.1]
  [Beginning at ] Mon Jan  7 15:17:22 2013
  [Ending correctly at] Mon Jan  7 15:17:42 2013
....
```

## 3.3 Computing Tie Points with Tapioca

Tapioca is a simple tool interface for computing tie points. I think Tapioca should be sufficient in 95% of cases. If it is not the case, you will have to refer to a more complex and powerful tool named *Pastis* which will be described later. In fact, Tapioca is only an interface to *Pastis*<sup>1</sup>.

### 3.3.1 General Structure

The general syntax of Tapioca is:

```
Tapioca Mode Files Arg1 Arg2 ...Opt1=Val Opt2=Val ...
```

Here are possible use of Tapioca:

```
Tapioca All ".../micmac_data/ExempleDoc/Boudha/IMG_[0-9]{4}.tif" -1 ExpTxt=1
Tapioca Line ".../micmac_data/ExempleDoc/Boudha/IMG_[0-9]{4}.tif" -1 3 ExpTxt=1
Tapioca MulScale ".../micmac_data/ExempleDoc/Boudha/IMG_[0-9]{4}.tif" 300 -1 ExpTxt=1
Tapioca File ".../micmac_data/ExempleDoc/Boudha/MesCouples.xml" -1 ExpTxt=1
```

The meaning of arguments is:

---

1. *Pastis* being itself an interface to *Sift++* ...

- **Mode** is an enumerated value specifying a functioning mode (i.e. a way to compute the pair of images that are to be matched). These values are **All** for all possible pairs, **MulScale** for a multi-scale optimization, **Line** for a selection adapted to linear images acquisition, and **File** for XML file describing the pairs;
- **Files** specifies a set of images to be matched. For all these images, a set of sift descriptor will be computed. However, all the pairs of descriptors sets will not be matched. To optimize the computation, a subset of images pair will be described by the **Mode** parameters. The first part of **Files** is a directory, and the second one is the description of the files to be computed with Tapioca. The results will be written in the subdirectory **Homol** of the specified directory as it was described in 6.2.2;
- **Arg1 Arg2 ...** are mandatory parameters; their number depends upon **Mode**, it always includes a resolution parameter;
- **Opt1=Val Opt2=Val ...** are optional parameters. The possible optional parameters depends upon **Mode**. There is always at least three possible optional parameters:
  - **ExpTxt** indicates if you want an export in text mode; default is 0 (binary mode);
  - **ByP** indicates the number of processors that will be used to parallelize the process; default is the number of processors described in 5.1.2;
  - **Ratio** to choose the ratio between first and second best points in matching. Default is 0.6, lower means that you want less ambiguity (and less points). Only used with ANN match.

If you do not remember the possible mode key words, just type:

```
Tapioca -help
```

The possible values will be printed.

If you do not remember the argument corresponding to a possible mode, just type **Tapioca mode -help**, for example:

```
Tapioca MulScale -help
```

### 3.3.2 Computing All the Tie Points of a Set of Images

The simplest case of use of **Tapioca** is when you only want to compute tie points between all the pairs of a set of images. The syntax is:

```
Tapioca All Files Size ArgOpt=
```

The only optional arguments are those common to all modes (**ExpTxt** and **ByP**).

The parameter **Files** is the concatenation of the directory where the files are located with a regular expression used as a filter on the existing files of the directory.

The parameter **Size** is used to shrink the images. It does not specify a scale but the desired width for shrinking the images. For example, if the initial image has a width of 5000, and the value is 2000, it will specify a scaling of 0.4. If its value is **-1**, it means, conventionally, no shrinking. This is the value chosen in the examples, in order to limit the size of the transmitted data, the images have already been shrunk. With real images, I do not recommend the value **-1** but rather a value corresponding to a scaling between 0.3 and 0.5.

The example:

```
Tapioca All ".../micmac_data/ExempleDoc/Boudha/IMG_[0-9]{4}.tif" -1 ExpTxt=1
```

It generates tie points computation between all the pairs of images of the **Boudha** directory, with names matching **IMG\_[0-9]4.tif**. There is no shrinking, the export is made in text mode.

### 3.3.3 Optimization for Linear Canvas

It often occurs that the photos canvas has a linear structure, for example, when you acquire photos of a facade walking along the street. In this case, you know that  $K^{th}$  can only have tie points with images in the interval  $[K - \delta, K + \delta]$ ; giving this information to Tapioca can save a lot of time. The syntax is:

```
Tapioca Line Files Size delta ArgOpt=
```

**delta** is  $\delta$  and all the other arguments have the same meaning as in the **All** mode.

### 3.3.4 Multi Scale Approach

The mode `MulScale` can save significant computation time on large sets of images. Even if it is not optimal for all canvas, it has the benefit of being general and usable with any data set.

In this mode, a first computation of tie points is made for all the pairs of images at very low resolution (so it is quite fast). Then the computation, at the desired resolution, is done only for the pairs having, at low resolution, a number of tie points exceeding a given threshold.

```
Tapioca MulScale Files  SizeLow Size  NbMinPt=  ArgOpt=
```

`SizeLow` is the size of the images that will be used at low resolution. `Size` is the targeted size. The optional value `NbMinPt` is the threshold on the number of tie points detected at low resolution, its default value is 2. For example:

```
Tapioca MulScale "../micmac_data/ExempleDoc/Boudha/IMG_[0-9]{4}.tif" 300 -1 ExpTxt=1
```

Computes tie points with images of width of 300, and if the pairs have at least 2 tie points, it does the computation at full resolution.

### 3.3.5 Explicit Specification of images pairs

Sometimes, you will have external information (like embedded GPS) that allows you to know which pairs of images are potential candidates for tie points. If you are familiar with computer programming, you will find that the easiest way to communicate your information is to write a file containing the explicit list of pairs of images. It is possible in the `File` mode, the file containing the pairs must have the following structure:

```
<?xml version="1.0" ?>
<SauvegardeNamedRel>
  <Cple>IMG_5564.tif IMG_5565.tif</Cple>
  <Cple>IMG_5574.tif IMG_5575.tif</Cple>
  <Cple>IMG_5580.tif IMG_5579.tif</Cple>
  <Cple>IMG_5581.tif IMG_5582.tif</Cple>
</SauvegardeNamedRel>
```

The syntax is:

```
Tapioca File  NameOfFile  Size  ArgOpt=
```

The pairs contained in the file `NameOfFile` are names relative to the directory indicated by the first part of `NameOfFile`. For example in:

```
Tapioca File  "../micmac_data/ExempleDoc/Boudha/MesCouples.xml" -1 ExpTxt=1
```

In the pair `<Cple>IMG_5564.tif IMG_5565.tif</Cple>`, the first name means `"../micmac_data/ExempleDoc/Boudha/IMG_5564.tif"`.

### 3.3.6 The tool GrapheHom

A tool for generating a file image pairs, as input to `Tapioca File ...` from external Data (GPS or GPS-INS).

```
GrapheHom -help
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string
 * string
 * string
Named args :
 * [Name=TagC] string
 * [Name=TagOri] string
```

```
* [Name=AltiSol] REAL
* [Name=Dist] REAL
* [Name=Rab] REAL
* [Name=Terr] bool
* [Name=Sym] bool
* [Name=Out] string
```

The three first mandatory args :

- directory ;
- a pattern describing the set of images (it can also be a key of set);
- a key association for computing the name of the *a priori* localization file; this file must always contain the image position of type `Pt3dr` (by default in the tag `Centre`); optionally it can contain a tag of type `OrientationConique` defining the approximate orientation (by default this tag is `OrientationConique`);

The optional args :

- `TagC` XML tag for center (default = `Center`);
- `TagOri` XML tag for orientation (default = `OrientationConique`);
- `AltiSol` altitude of ground when it cannot be found in orientation files (default = 0.0);
- `Terr` is a terrestrial or aerial acquisition, (default = `false`, i.e; aerial);
- `Dist` minimal distance between two submit, optional in aerial mission (a default value will be computed) mandatory in terrestrial acquisition;

For example :

```
i
GrapheHom ./    ".*.ARW" NKS-Assoc-Im2Orient@-AO-Navig-UTM
Or
GrapheHom ./    ".*.ARW" -AO-Navig-UTM
```

## 3.4 Simple relative orientation and calibration with Tapas

### 3.4.1 Generalities

The general tool for computing orientation of images is `Apero`, this is a relatively complex tool an overview of which is given in chapter 6. These sections describe a set of basic tools offering a simplified interface to some of the elementary functionalities of `Apero` :

- `Tapas`, in this section, is a tool offering most of the possibilities of `Apero` for computing purely relative orientations;
- `AperiCloud`, in 3.9.1 for generating a visualization of camera position and sparse 3D model;
- `Campari`, in 3.9.2 is a tool for compensation of heterogeneous measures (tie points and ground control points);
- `Bascule`, in 3.10.1, for generating orientations coherent with some physical information on the scene;
- `MakeGrid`, in 3.10.2, for generating orientations in a grid format that is more adapted to some further processing;

Like with many tools, one can type `Tapas -help` to have a brief description of `Tapas`'s arguments.

### 3.4.2 The data set "Mur Saint Martin"

The directory `MurSaintMartin/` in `micmac_data/ExempleDoc/`, contains a first set of data, that will be used for illustrating `Tapas`. This set is made of 23 jpeg images that have been acquired with the same camera and same focal length; it is made of two subset :

- 17 images of a wall : images `IMGP4167.JPG` to `IMGP4183.JPG`, the first 8 images are presented on figure 3.1;
- 6 images of a corner, that can be optionally used for intrinsic calibration : images `IMGP4160.JPG` to `IMGP4165.JPG`, the images are on figure 3.2 ;



Figure 3.1: The Saint-Martin set of images, images of the wall



Figure 3.2: The Saint-Martin set of images, images for intrinsic calibration

To run the example using **Tapas**, tie points will be required, they can be computed by the two commands<sup>2</sup> :

```
Tapioca All "IMGP416[0-5].JPG" 1000
Tapioca Line "IMGP41((6[7-9])|([7-8][0-9])).JPG" 1000 4
```

### 3.4.3 Basic usage

#### 3.4.3.1 Syntax

The basic command to run **Tapas** is :

```
Tapas ModeCalib PatternImage
```

Where :

- **ModeCalib** is an enumerated value specifying a model of calibration;
  - **PatternImage** is a pattern specifying the subset of images to orientate;
- For example the command :

```
Tapas RadialExtended "IMGP41((6[7-9])|([7-8][0-9])).JPG"
```

Means :

- compute the relative orientation of the set of images defined by the regular expression ;
- for the intrinsic calibration use a model **RadialExtended**;
- there is exactly one intrinsic calibration unknown for each focal length, the focal length being extracted from exif metadata; the exif meta-data is used for defining the initial value of each
- use a predefined strategy for computing orientations and intrinsic calibration.

---

2. the command used in this example can be found in the file **ExCmd.txt**

### 3.4.3.2 Distortion models

The possible value of `ModeCalib` are :

- **RadialExtended** a model with radial distortion (as specified in 14.2.2); in this model there are 10 degrees of freedom: 1 for focal length , 2 for principal point, 2 for distortion center , 5 for coefficients of radial distortion ( $r^3, r^5 \dots r^{11}$ );
- **RadialBasic** a "subset" of previous model: radial distortion with limited degrees of freedom ; adapted when there is a risk of divergence of **RadialExtended**; in this model there are 5 degrees of freedom : 1 for focal length , 2 for principal point and distortion center<sup>3</sup> , 2 for coefficients of radial distortion ( $r^3$  and  $r^5$ );
- **Fraser** a radial model, with decentric and affine parameters (as specified in 14.2.3); there are 12 degrees of freedom: 1 for focal length , 2 for principal point, 2 for distortion center , 3 for coefficients of radial distorsion ( $r^3, r^5 r^7$ ), 2 for decentric parameters, 2 for affine parameters; the optional parameters `LibAff` and `LibDec` (def value true) can be set to false if decentric or affine parameters must stay frozen;
- **FraserBasic** same as previous with for principal point and distortion center constrained to have the same value (so 10 degree of freedom);
- **FishEyeEqui** a model adapted for diagonal fisheyes equilinear ( with *atan* physical model completed with polynomial parameters, as specified in 14.3.4); there are 14 degrees of freedom: 1 for focal length , 2 for principal point, 2 for distortion center , 5 for coefficients of radial distortion ( $r^3, r^5 r^7$ ), 2 for decentric parameters, 2 for affine parameters; by default the ray defining the useful mask, see 14.3.5 is 95% of the diagonal;
- **HemiEqui** same model as previous, but by default the ray defining the useful mask, see 14.3.5 is 52% of the diagonal; adapted to hemispheric equilinear fisheye;
- **AutoCal** and **Figeé** , with this tag no model is defined, all the calibration must have a value (via `InCal` or `InOri` options); with **AutoCal** the calibration are re-evaluated while with **Figeé** it stay frozen.

For all the mode, except of course **AutoCal** and **Figeé**, the initial value of intrinsic calibration is computed this way :

- focal length is computed from exif data using the rules described in 3.6.2;
- principal point and, when apply, distortion center are at the middle of image (except when using the `Decentre` option);
- initial distortion is equal to the physical ideal model : null for the standard lenses and equal to *atant* for fisheye;

### 3.4.3.3 Strategy

With `Tapas`, the user has very few control on the strategy used to compute orientation. The predefined strategy used by `Tapas` is :

- initialize all the intrinsic calibration using exif data (or already computed calibration provided by existing files), then freeze all these unknown;
- choose a central images (the image that has the maximum of tie points);
- compute the orientation of images with the "standard" strategy described in 6.3.3;
- once all the image are ordered, free in a predefined order all the intrinsic parameters;

### 3.4.3.4 Results

The result of `Tapas` are stored in a subdirectory `Ori-OUTDIR` , where `OUTDIR` is specified by the optional `out` argument of `Tapas`, when `out` is not specified the value of `ModeCalib` is used. With this basic command, the result are stored in the directory `Ori-RadialExtended/` :

- the file `AutoCal280.xml` contains the intrinsic calibration; the name has been automatically computed from the focal length got in exif file (here 28mm); there is only one file because there was only one focal length;
- the files `Orientation-IMGPXXXX.JPG.xml` contain the external orientations;

---

3. they are constrained to have the same value

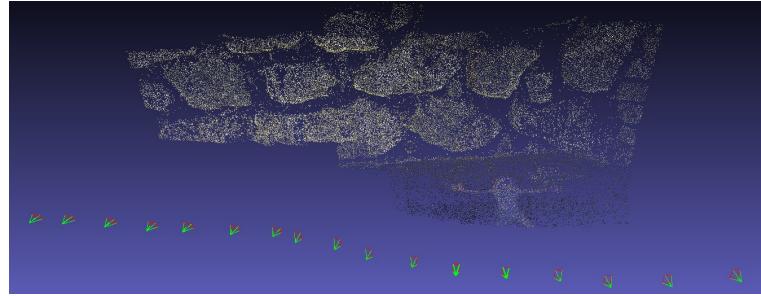


Figure 3.3: Visualization of the orientation obtained

- the detailed specification of intrinsic calibration and external orientation can be found in 14, by the way it's not necessary to have a full understanding of this format for using it in **MicMac** and other tools .

#### 3.4.4 Successive calls to Tapas

Even with simple acquisition, where all the images have been acquired with the same lenses, the usage of **Tapas** presented in 3.4.3 may be too basic. The risk is that, starting from a very rough estimation from the intrinsic calibration, the computation of orientation do not converge to a good solution. With large data set, it is often preferable to proceed in two step :

- compute on a small set of image a value of intrinsic calibration, this set of image should be favorable to calibration; ideally, it should fulfill the following requirements :
- all image converging to same part of the scene,to facilitate the computation of external orientation
- a scene with sufficient depth variation ,to have accurate focal length estimation;
- a image acquisition where there position of the same ground points are located at very different position in the different images where they are seen, this is to have accurate estimation of distortion; this can be obtained by rotating the camera like acquisition of figure 3.2;
- use the calibration obtained on the small set as an initial value for the global orientation;

The set for calibration can be a subset of the images used for the scene reconstruction ; often having a separate acquisition is preferable to ensure that it fulfill all the requirements. In the "Mur Saint Martin" example, it is a separate example; the call to **Tapas** can then be :

```
Tapas RadialExtended "IMGP416[0-5].JPG" Out=Calib
Tapas AutoCal "IMGP41((6[7-9])|([7-8][0-9])).JPG" InCal=Calib Out=Mur
```

Some comments :

- the first line is equivalent to 3.4.3, the only difference is that the out directory is specified; here, the results are then written in **Ori-Calib**;
- in the second line, the argument **InCal=Calib** specifies that for each unknown calibration of focal  $F$  , if there exist a file **Ori-Calib/AutoCal(F\*10).xml**, this file must be used as an initial value ; here, with a 28mm focal, the file **Ori-Calib/AutoCal280.xml** has been created by previous line and is used;
- here, with **ModeCalib=AutoCal**<sup>4</sup>, when the file **Ori-Calib/AutoCal(F\*10).xml** do not exist an error occurs;
- with other mode of **ModeCalib**<sup>5</sup> when the file do not exist, a default initial value is created using the **ModeCalib** as described in 3.4.3.2;

Figure 3.3 show a visualization of the orientation obtained, using the program **AperiCloud** described in 3.9.1.

4. also with **ModeCalib=Figeo**

5. RadialBasic, RadialExtended, Fraser, FishEyeEqui, HemiEqui



Figure 3.4: Some of 10mm-Saint martin's street photos



Figure 3.5: One of the 17mm-Saint martin's street subset

## 3.5 Multiple lenses with Tapas

### 3.5.1 The Saint Martin Street data set

The data set used in this section is still quite basic and a direct orientation of all the images should work; however it illustrates a general strategy, proceeding in a kind of multi-scale approach, that can be adapted for complex architectural modelizations; in a two step version, for the acquisition phase, this approach can for example be:

- acquire with a short focal lenses, a set of images with a wide overlapping that will form a highly connected set;
- acquire with a longer focal length convergent sets of images on areas (possibly covering all the scene) interesting for 3d modelization;
- when acquiring the second set (longer focal) do not worry about connectivity, as it will be the "job" of the first data set.

The data set is on directory `StreetSaintMartin`<sup>6</sup>. It has been made using a "zoom fisheye" at two different focals 10mm and 17mm. There is three subset of images :

- `IMGP4118.JPG` to `IMGP4122.JPG` , 5 images for the calibration of the 10mm;
- `IMGP4123.JPG` to `IMGP4151.JPG` , 29 images of a narrow street, mixing 10mm and 17mm focal; although the images have been acquired for the purpose of the documentation<sup>7</sup> , one could imagine the 10mm are used for the global context and the 17mm is used for modelization of some details (the 17mm are made of two convergent subset : 44 to 47 and 48 to 51); figure 3.4 shows some of the 10mm images and figure 3.5 shows some of the 17mm images;
- `IMGP4152.JPG` to `IMGP4158.JPG` , 7 images for the calibration of the 17mm.

To obtain the necessary tie points, one can type (you can find it inside the file `ExCmd.txt`) :

```
Tapioca All "IMGP41((1[8-9])|(2[0-2])).JPG" 1000
Tapioca All "IMGP41((5[2-8])).JPG" 1000
Tapioca All "IMGP41((2[3-9])|[3-4][0-9]|(5[0-1])).JPG" 1000
```

6. under `micmac.data/ExempleDoc/`

7. and so the dataset is a bit artificial

### 3.5.2 Exploiting the data with Tapas

To exploit the acquisition strategy described above, a pertinent processing strategy will be also separated in several steps; for example:

- orientate at first the short focal lenses, that will constitute the global canvas
  - compute the orientations of other images based on the canvas of the already oriented images;
- Using this general strategy , with the Saint Martin Street dataset, we would like to proceed this way:
- compute the calibrations of the camera;
  - compute the orientations of the  $10mm$  images only;
  - compute the orientations of the  $17mm$  images, in the same coordinate system that the already oriented  $10mm$  images;

The following commands could realize this program :

```
Tapas FishEyeEqui    "IMGP41((1[8-9])|(2[0-2])).JPG"  Out=Calib10
Tapas FishEyeEqui    "IMGP41((5[2-8])).JPG"    Out=Calib17

Tapas AutoCal    "IMGP41((2[3-9])|[3-4][0-9]|(5[0-1])).JPG"  InCal=Calib10 Focs=[9,11] Out=Tmp1
cp Ori-Calib17/AutoCal170.xml Ori-Tmp1/
Tapas FishEyeEqui    "IMGP41((2[3-9])|[3-4][0-9]|(5[0-1])).JPG"  InOri=Tmp1  Out=all

AperiCloud "IMGP41((2[3-9])|[3-4][0-9]|(5[0-1])).JPG" all
```

Let's comment :

- the first two lines generates an initial calibration with the calibration subsets; it is quite similar to 3.4.4, the difference being that the `FishEyeEqui` specifies that we have a fisheye;
- the third line uses the `InCal` options, already seen; it also uses the option `Focs=[9,11]`, the effect is that only the images with a focal lens between  $9mm$  and  $11mm$  will be loaded; so here we orientate the  $10mm$  subset;
- the line `cp Ori...`, is necessary because we will need, on next call to Tapas, to have all our required input on the same directory `Ori-Tmp1`
- in the next call to `Tapas`, we specify `InOri=Tmp1`, this has two effects :
  - as before when files `Ori-Tmp1/AutoCalXXX.xml` (`XXX` being the required focal) exist they are used to initialize intrinsic calibrations;
  - when files `Ori-Tmp1/OrientationXXX.xml` (`XXX` being the images name) exist they are used to initialize the external orientations;
- so here, we will start directly from "good" initial value for both intrinsic calibration and external orientation of  $10mm$  images.

## 3.6 Camera data base and exif handling

### 3.6.1 How Tapas initialize calibration

For each camera it has to handle, when the user do not provide a calibration file, Tapas has to build an initial value. For all parameters, but the focal it's relatively easy :

- for the distortion, the initial value is null<sup>8</sup>;
- for the principal point the initial value is at center of image<sup>9</sup>.

For the focal lens Tapas must compute an initial value in pixel, this information is computed with the help of xif data. To see some examples of xif data, go to MurSaint martin and try `ElDcraw -i -v` or `exiv2` or `exiftool` :

```
ElDcraw -i -v IMGP4182.JPG
...
Camera: PENTAX K-5
....
```

---

8. more precisely, it is the initial physical model, for example with fish-eye the initial value is a  $\tan^{-1}$  function, see 14.3.4  
 9. of course, this wouldn't be suitable with shift lenses

Focal length: 28.0 mm

Focal Equi35: 42.0 mm

....

As the xif meta data never contains directly the focal in pixel, several cases can occur:

- if the xif meta data contain the value  $F_{35}$  of the focal in equivalent 35mm<sup>10</sup>, then the focal is estimated by  $\frac{F_{35} * W_{Pix}}{35.0}$ , where  $W_{Pix}$  is the width (= number) of image in pixel;
- if the xif meta data contain the value  $F_{mm}$  of the focal in millimeter, and the width  $W_{mm}$  of the sensor is known in millimeter, the focal is estimated by  $\frac{F_{mm} * W_{Pix}}{W_{mm}}$ .

The size of the sensor is not a xif tag, so the information has to come from somewhere else; this is the role of the camera data base.

### 3.6.2 Camera data base

With all camera sold for people, xif meta data contain a tag indicating the name of the camera. For example you can see that the `MurSaintMartin` has been acquired with a PENTAX K-5 camera. This camera name is used by the different tools as an entry in data bases containing information missing from xif files.

These data base can be located in three different files :

- `include/XML_MicMac/DicoCamera.xml` , this global file always exist as it is part of the `MicMac` distribution, I put here the camera necessary for the examples; I also update it when I meet a new camera so that users can take benefit of it; NEVER modify this file to add your own camera, as your may loose all your modification at next update;
- `include/XML_User/DicoCamera.xml` , this file does not exist when you make the first installation of `MicMac`; so you have to create it, and put inside the description of all the camera that you will use currently; as this file is not handled by subversion, there is no risk of over writing it when you update;
- `MicMac-LocalChantierDescripteur.xml`, in your working directory when this file exists; the ENAC example contains an example of such usage;

Naturally, if the same camera is described in several files, the more local file has the priority<sup>11</sup> . Take a look at `include/XML_MicMac/DicoCamera.xml`, the structure is quite simple :

- a `MMCameraDataBase` contains a `CameraEntry` for each camera to describe;
- a `CameraEntry` contains :
  - a `Name` that is used to make the link with information in xif file;
  - a `SzCaptMm` that contains the size of sensor in millimeter;
  - a `ShortName` which usage will be explained later in 16.2; (just give the value `XXXX` until here)

### 3.6.3 Indicating missing xif info

Sometimes, the xif file does not contain the expected information. This can be the case for example when images where acquired by industrial camera, or when the images result from conversion by various software. In this case, the information can be indicated "dynamically" by creating specific key in the `MicMac-LocalChantierDescripteur.xml`.

The ENAC dataset illustrates this usage :

- to indicate camera names, the user must define a rule (see 11.5) of key `NKS-Assoc-STD-CAM`, this rule must transform the name of the file into the name of the camera; in ENAC there is only one camera, the rule specify then that, whatever be the image name, the camera name will be `TheGOPRO`;
- to indicate focal length, the user must define a rule of key `NKS-Assoc-STD-FOC` that associate to each image name its focal; here we have only one focal 3.8, the rule is simple;
- the association mechanism described in 11.5 may seem over complicated, however it has to be very flexible to handle case where there are several cameras or focal lengths not present in xif files;

Here is a copy of the ENAC's dataset `MicMac-LocalChantierDescripteur.xml` :

---

10. try `ElDcraw FileName` to see

11. e.g. `MicMac-LocalChantierDescripteur.xml` highest priority `XML_MicMac/DicoCamera.xml` lowest

```

<Global>

    <ChantierDescripteur >

        <LocCamDataBase>

            <CameraEntry>
                <Name> TheGOPRO  </Name>
                <SzCaptMm> 4.9 8.7  </SzCaptMm>
                <ShortName> TheGOPRO </ShortName>
            </CameraEntry>
        </LocCamDataBase>

        <KeyedNamesAssociations>
            <Calcs>
                <Arrite> 1 1 </Arrite>
                <Direct>
                    <PatternTransform> .*test[0-9]{4}.jpg  </PatternTransform>
                    <CalcName> TheGOPRO </CalcName>
                </Direct>
            </Calcs>
            <Key> NKS-Assoc-STD-CAM </Key>
        </KeyedNamesAssociations>
        <KeyedNamesAssociations>
            <Calcs>
                <Arrite> 1 1 </Arrite>
                <Direct>
                    <PatternTransform> .*test[0-9]{4}.jpg  </PatternTransform>
                    <CalcName> 3.8 </CalcName>
                </Direct>
            </Calcs>
            <Key> NKS-Assoc-STD-FOC  </Key>
        </KeyedNamesAssociations>

    </ChantierDescripteur>
</Global>

```

### 3.6.4 Modifying exif

Another solution to deal with missing xif info is to use the modifying facilities offered by the exiv2 command . I recommend that you read carefully the exiv2 documentation before using it. Here is a short example, without any guarantee.

First create a command file, let name it `Cmd.txt` with the appropriate syntax :

```
set Exif.Photo.FocalLength 120/1
set Exif.Photo.FocalLengthIn35mmFilm 180
```

Then execute this command file on the desired images by something like :

```
exiv2 -m"Cmd.txt" *.PEF
```

### 3.6.5 XML "cache" version of xif information

As exif extraction can be relatively slow, since mercurial revision 3293, there exist a "cache" mechanism to save this xif information in `xml` file and reload it more quickly. The tool `MMXmlXif` can be used for explicitly creating this files :

- call `mm3d MMXmlXif Pattern`
- for each file `aFile` in `Pattern`, a xml version of the xif information is created in `Tmp-MM-Dir/aFile-MDT-Num.xml` where `Num` is some versioning number;
- when the exif information will be required, if the xml file exist, it will be used to load it;

If for example with ENAC data set you run `mm3d MMXmlXif test00.*jpg` you will have a file `Tmp-MM-Dir/test0050.jpg-MD` to contain :

```
<XmlXifInfo>
  <HGRev>3293</HGRev>
  <FocMM>3.7999999999999982</FocMM>
  <Foc35>16.4660314153628171</Foc35>
  <Sz>1280 720</Sz>
  <Cam>TheGOPRO</Cam>
  <BayPat>RVWB</BayPat>
</XmlXifInfo>
```

In fact you probably do not need to explicitly call `MMXmlXif` as it called by `Tapioca` and `Tapas` "just in case". For now, there is no way to avoid this mechanism<sup>12</sup>, but if it happens to have unwanted side effect, some option will be added to suppress it when necessary.

## 3.7 Using Raw images

Sometime the images are in a pure Raw format, with no header describing the physical representation of the image on hard disk. It is possible to use these images in `Apero/MicMac` but user has to explain the missing information. This is done this way :

- for each kind of format create file containing a structure `SpecifFormatRaw`;
- in `MicMac-LocalChantierDescripteur.xml` change the value of the key `NKS-Assoc-SpecifRaw` to associate to each Raw images its file containing the `SpecifFormatRaw`;

An example is given in folder `Documentation/NEW-DATA/RAW-IMAGES`. This case corresponds to :

- a set of images with name `"*.dlr"`;
- each image has the same size `2448 * 2048` and are 8 bits unsigned integer images;
- in this case the optional `<BayPat>` tag indicate that the image are bayer images;
- the `<Focalmm>`, `<FocalEqui35>`, `<Camera>` tags are used to fit the minimal xif information for using at the end in `Apero/Tapas`.

If the tag `<BayPat>` is present the image is interpreted as a color image acquired by bayer matrix specified the string. If it is not present, images are considered as gray scale images. This is illustrated by figure 3.6. This figure presents crops of results of the command `ConvertIm` with different value of `<BayPat>` tag in the file `SpecRaw.xml` :

- left : with the correct value for these images (RGGB here), a standard RGB value is obtained;
- middle : the optional `<BayPat>` tag is absent, the image is interpreted as a gray value image (which generate the checkboard effect);
- right : a false value (here GBRG), the color are swapped.

## 3.8 Other options of Tapas

### 3.8.1 Saving intermediair results with SauvAutom

### 3.8.2 Forcing first image with ImInit

### 3.8.3 Freezing poses with FrozenPoses

The optional arg `FrozenPoses` of `Tapas`, can be used to indicate a subset of images for which orientation will be frozen during all the compensation; `FrozenPoses` is a generalized regular expression (pattern or key) describing this subset.

---

12. As there is no identified drawback

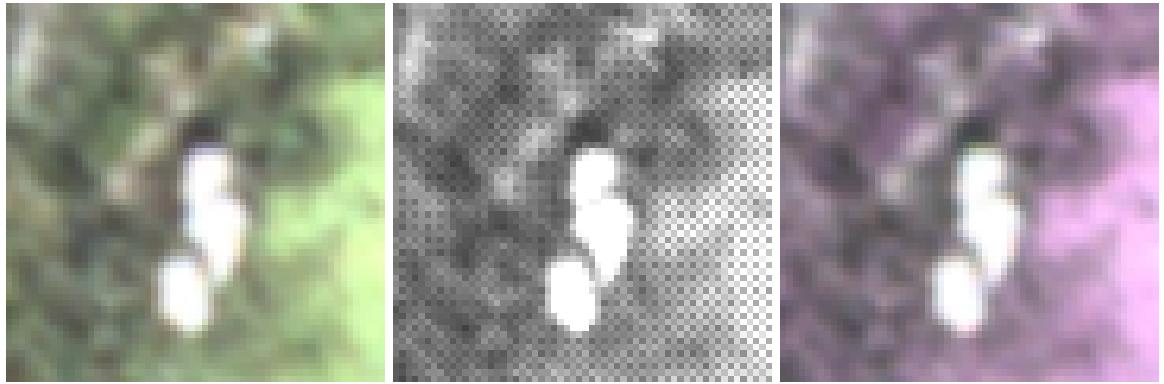


Figure 3.6: Results (crop) of command `ConvertIm` with different value `<BayPat>` : RGGB, absent, GBRG

## 3.9 Other tools for orientation

### 3.9.1 The tool AperiCloud

This section describes `AperiCloud` a simplified version of the `<ExportNuage>` section of `Apero` described in 16.3.1.

For example with Mur Saint Martin:

```
AperiCloud "./IMGP41((6[7-9])|([7-8][0-9])).JPG" Mur
```

Typing `AperiCloud -help`, one gets:

```
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string
 * string
Named args :
 * [Name=ExpTxt] INT
 * [Name=Out] string
 * [Name=Bin] INT
 * [Name=RGB] INT
```

Meaning of args is:

- First arg: pattern specifying the set of images ;
- Second arg: pattern specifying the directory where orientations are located;
- optional `ExpTxt`, def = 0, set to 1 if tie points are to be red in text format;
- optional `Out`, def = `AperiCloud.ply` , specify the name of the generated ply file;
- optional `Bin`, def = 1 , set to 0 if ply file are to be generated in text format;
- optional `RGB`, def = 1 , set to 0 if the point are to be coloured with black and white images (usefull to save time);

### 3.9.2 The tool Campari

This section describes `Campari` an interface to `Apero`, for compensation of heterogeneous measures, that is: tie points and ground control points.

For example:

```
Campari "MyDir\IMG_.*.jpg" OriIn OriOut GCP=[GroundMeasures.xml,0.1,ImgMeasures.xml,0.5]
```

Typing `Campari -help`, one gets:

```
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {Full Directory (Dir+Pattern)}
 * string :: {Input Orientation}
 * string :: {Output Orientation}
Named args :
 * [Name=GCP] vector<std::string> :: {[GrMes.xml,GrUncertainty,ImMes.xml,ImUnc]}
```

Meaning of args is:

- First arg: pattern specifying the set of images ;
- Second arg: pattern specifying the directory where orientations are located;
- Third arg: pattern specifying the directory where to write output orientations;
- optional GCP, specifying the ground and image measures files, with their respective uncertainties;

Mandatory part of GCP:

- xml file with 3D coordinates for GCP
- GCP ground uncertainty
- xml file with 2D coordinates for GCP
- GCP image uncertainty

The xml file containing 3D coordinates has to verify a specific format. A tool to convert existing coordinates listing files into this format is proposed with `GCPConvert` and described in 13.3.1.

The xml file containing 2D coordinates can be generated using interfaces `SaisieAppuisInit` and `SaisieAppuisPredic` in Linux, described in 8.4.2.

For a detailed example on how to use `Campari` in a typical aerial surveying workflow, see 4.2.1.

### 3.9.2.1 Estimate lever-arm with the tool Campari

As seen, the tool `Campari` deals with compensation of heterogeneous measures. Not only tie points and ground control points but also GPS data. In direct georeferencing case for example, specially for UAV acquisitions, one gets a GPS antenna on the top of the UAV and camera embedded on back side. The vector which separates the phase center of the GPS antenna and the optical center of the camera is called : lever-arm vector. The value of this vector expressed in the camera frame must be constant (in practice, very slight variations). To include GPS data in the compensation and estimate lever-arm, for example :

```
mm3d Campari "MyDir\IMG_.*.jpg" OriIn OriOut GCP=[GroundMeasures.xml,0.1,ImgMeasures.xml,0.5]
EmGPS=[Ori-Nav-Brut/,0.02,0.05] GpsLa=[0,0,0]
```

The directory `OriIn` must contain orientations in the same frame as the directory `Ori-Nav-Brut/`. This can be done with a relative orientation and ground control points using the tool `GCPBascule` described in 3.10.1.2 or even with the tool `CenterBascule` described in 3.10.1.4.

Typing `Campari -help`, one gets:

```
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {Full Directory (Dir+Pattern)}
 * string :: {Input Orientation}
```

```

* string :: {Output Orientation}
Named args :
* [Name=EmGPS] vector<std::string> :: {Embedded GPS [Gps-Dir,GpsUnc,?GpsAlti?], GpsAlti if != P}
* [Name=GpsLa] Pt3dr :: {Gps Lever Arm, in combinaision with EmGPS}

```

Meaning of optional args is:

- **EmGPS**, specifying the directory where orientations generated from GPS data are located; this directory can be generated from a file using the tool **OriConvert** described in 13.3.4. The uncertainty for the height component, for GPS coordinates, must be different from the uncertainty of the horizontal components. These values depend on if GPS coordinates are derived from a processing based on pseudo-range (2-5 m) or carrier-phase measurements (1-5 cm). Also, these values are the same for all GPS coordinates
- **GpsLa**, the initial value of the lever arm vector expressed in the camera frame

What is displayed while **Campari** running :

```

Lever Arm, Cam: DSC02925.JPG Residual [-0.0011,0.0071,-0.0131] LA: [-0.1532,-0.0230,-0.0417]
Lever Arm, Cam: DSC02926.JPG Residual [-0.0261,0.0280,0.02031] LA: [-0.1532,-0.0230,-0.0417]
Lever Arm, Cam: DSC02927.JPG Residual [0.01024,-0.006,0.00408] LA: [-0.1532,-0.0230,-0.0417]
RES:[DSC02925.JPG] ER2 0.339613 Nn 99.6381 Of 48080 Mul 22837 Mul-NN 22781 Time 0.842611
RES:[DSC02926.JPG] ER2 0.332633 Nn 99.5899 Of 38769 Mul 16613 Mul-NN 16553 Time 0.668483
RES:[DSC02927.JPG] ER2 0.339088 Nn 99.6616 Of 49646 Mul 23107 Mul-NN 23056 Time 0.883703
...

```

For each image, value of residual is given with the estimated value of lever-arm in camera frame. The LA value is the same for all images.

To estimate the delay between the GPS data recording and the triggering camera, please refere to the tool **OriConvert** described in 13.3.4.

### 3.9.2.2 Bundle adjustment with pushbroom sensor

This section describes how to utilize the **Campari** simplified tool to refine the orientation parameters provided in form of the rational polynomial coefficients (RPCs).

Typing **Campari -help**, one gets:

```

*****
* Help for Elise Arg main *
*****
Unnamed args :
* string :: {Full Directory (Dir+Pattern)}
* string :: {Input Orientation}
* string :: {Output Orientation}
Named args :
* [Name=FactElimTieP] REAL :: {Fact elimination of tie point (prop to SigmaTieP, Def=5)}
* [Name=AcceptGB] bool :: {Accepte new Generik Bundle image,
                           Def=true, set false for perfect backward compatibility}
* [Name=PdsGBRot] REAL :: {Weighting of the global rotation constraint
                           (Generic bundle Def=0.002)}
* [Name=PdsGBId] REAL :: {Weighting of the global deformation constraint
                           (Generic bundle Def=0.0)}
* [Name=PdsGBIter] REAL :: {Weighting of the change of the global rotation constraint
                           between iterations (Generic bundle Def=1e-6)}

```

The optional arg **AcceptGB** indicates a generic input camera geometry (central perspective, RPCs, GRIDs). If cameras defined by RPCs are handled (which **MicMac** will automatically find out given the convention of the input orientation files), the trajectory of the satellite platform is fixed within the

adjustment, and only small camera rotations are estimated. The rotations are forced to cause pixel displacements in the sensor's plane being equal to a 2D polynomial function. The coefficients of the functions likewise act as observed unknowns in the adjustment (see E. Rupnik, M. Pierrot-Deseilligny, A. Delorme, and Klinger Y. *Refined satellite image orientation in the free open-source photogrammetric tools Apero/Micmac*. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 2016.).

The parameters of interest that steer the algorithm are : `NbLiais`, `PdsRot`, `PdsGBId`, `PdsGBIter`. The first parameter gives more force to image observations within the adjustment (as opposed to the constraints or ground control points); the `PdsRot` parameter is the weight for the rotation constraint (the lower the value, the *softer* the constraint), the `PdsGBId` imposes that the deformation field is small, and `PdsGBIter` is the weight that control the evolution of the deformation field from iteration to iteration. For datasets with very poorly estimated RPCs (e.g. Cartosat), it is suggested to change the `FactElimTieP` parameter to a higher value.

### 3.9.3 Convention for Orientation name

In "old" version of Apero/MicMac :

## 3.10 The Bascule's tools

### 3.10.1 Generalities

This section, describe the simplified version of the `<BasculeOrientation>` mechanisms described in 6.4.2. This mechanism is used when global transformation of the orientation is required. The tools for bascule are :

- `SBGlobBascule` is a tool for "scene based global" bascule, it is used when no absolute information is available but the user still wishes to give some physical meaning to the orientation;
- `GCPBascule` for using ground control point (GCP) to make a global transformation from a generally purely relative orientation to an orientation in the system of the GCP;
- `RepLocBascule` a tool useful to define a local repair without changing the orientation;
- `CenterBascule` for using embedded GPS on submit to make a global transformation from a generally purely relative orientation to an absolute orientation;
- the `Bascule` used to do, more or less, all of the previous functionality; it is obsolete, still maintained for compatibility, but no longer documented;

#### 3.10.1.1 Scene based orientation with `SBGlobBascule`

A current case in architectural modeling, is when a part of the scene is *globally* plane and we want to do computation in coordinate system where this plane is the horizontal plane. This can be done with the tool `SBGlobBascule`:

- `SBGlobBascule` uses a selected number of images, on which the user has created mask, these mask must define part of the image belonging to the plane (see figure 3.7 as an example);
- `SBGlobBascule` select the tie points belonging to the mask, and compute by least square fitting an estimation of this plane;
- finally `SBGlobBascule` compute the rotation that transform current coordinates in a new system where the fitted plane correspond to the plane  $Z = 0$ ;
- `SBGlobBascule` fix also the orientation inside the plane;
- optionally `SBGlobBascule` can fix the the global scale;

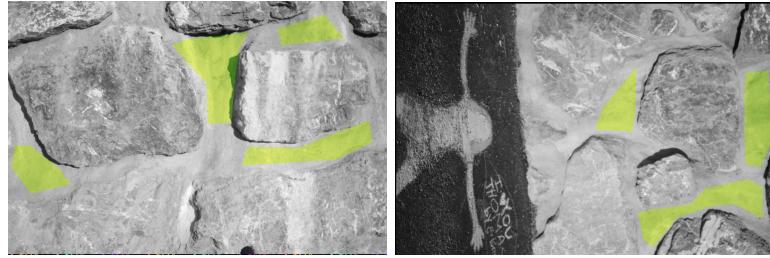


Figure 3.7: Example of masks defining a plane

With the dataset of street Saint Martin, an example of use is :

```
SBGlobBascule "IMGP41((6[7-9])|([7-8][0-9])).JPG" Mur MesureBasc.xml LocBasc PostPlan=_MasqPlan DistFS=0.6
```

The meaning of the arguments are:

- first arg, is the pattern defining the image we want to use;
- second arg **Mur** defines the input orientation;
- third arg **MesureBasc.xml** is a file that contains image measurement for defining orientation;
- fourth arg **Basc** defines the output orientation;
- optional args **PostPlan=\_MasqPlan** means that if image is **IMGP4171.JPG** (or **IMGP4171.CR2** or ...), then the associated mask **IMGP4171\_MasqPlan.tif**. Else, if **PostPlan=NONE** means that no mask is available and we do not want to change the input orientation and may be only want to fix its scale;
- if there are several masks it will use all them for fitting the plane (which can be useful with wide dataset when high accuracy is required);
- optional args **DistFS=0.6** is used to fix the scale;

Open the file **MesureBasc.xml**, you will see that it contains measurement of points in image. Although the syntax should be quite obvious, it is described in section 6.4.4.1. To create a file like **MesureBasc.xml** user can of course do it with a text editor, alternatively he can, on Linux, use the interactive tool **SaisieBasc** described in 8.4.4. Once created, the following information will be looked for by **SBGlobBascule** in this file:

- measurement of points named **Line1** and **Line2**; they will fix orientation in the plane by imposing that line **Line1-Line2** is parallel to **Ox**;
- these points need only to be measured in one image, as they are assumed to be in the plane computed on the mask; if they have been measured several times, a warning will occur;
- optionally a point **Origine** to fix the origin of the repair;
- optionally two points **Ech1** and **Ech2** to fix the scale, each point must be measured in two images, so that a *3d* position can be computed; when **DistFS** is entered, new coordinate system is computed with the constraint that the distance between the *3d* positions of **Ech1** and **Ech2** is equal to **DistFS** ; if **DistFS** is entered and **Ech1** and **Ech2** do not exist in at least two images, an error occurs;

To have the full syntax, as usual:

```
mm3d SBGlobBascule -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Full name (Dir+Pat)}
 * string :: {Orientation in}
 * string :: {Images measures xml file}
 * string :: {Out : orientation }

Named args :
 * [Name=ExpTxt] bool
 * [Name=PostPlan] string :: {Set NONE if no plane}
 * [Name=DistFS] REAL :: {Distance between Ech1 and Ech2 to fix scale (if not given no scaling)}
 * [Name=Rep] string :: {Target coordinate system (Def = ki, ie normal is vertical)}
 * [Name=CPI] bool :: {Calibration Per Image (Def=false)}
```

The **Rep** is described in section 4.1.2.4.

### 3.10.1.2 Geo-referencing with GCPBascule

In the Mur Saint Martin data set, you can find two files:

- `Ground-Pts3D.xml` contains the definition of 3D points, using the syntax detailed in 6.4.4.1;
- `GroundMeasure.xml` contains the 2D measurement of these points in images, using the syntax detailed in 6.4.4.1.

The `GCPBascule` command, allows to transform a purely relative orientation, as computed with `Tapas`, in an absolute one, as soon as there is at least 3 GCP whose projection are known in at least 2 images. Here for example:

```
GCPBascule "IMGP41((6[7-9])|([7-8][0-9])).JPG" Mur Ground Ground-Pts3D.xml GroundMeasure.xml
```

To know the syntax of `GCPBascule`:

```
GCPBascule -help
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {Full name (Dir+Pat)}
 * string :: {Orientation in}
 * string :: {Orientation out}
 * string :: {File for Ground Control Points}
 * string :: {File for Image Measurements}
Named args :
 * [Name=L1] bool :: {L1 minimization vs L2; (Def=false)}
```

The meaning should be quite obvious:

- first arg defines the set of images, which you want to change orientation;
- second arg defines the location of input orientations (generally it will be purely relative orientation generated using `Tapas` as described above);
- third arg defines the location of output orientation that will be generated by `GCPBascule`
- fourth arg defines the file containing the GCP and their 3d measures ;
- fifth arg defines the file containing the image measurement of GCP;
- optional arg `L1` indicates if the transformation from relative to absolute must be done using  $L_1$  or  $L_2$  minimization (as currently the measurements will have some redundancy).

Although it is generally difficult to analyze in detail the results of orientation by inspecting the file, you can check quickly that there is some coherence in the result. For example if you type `grep Centre Ori-Ground/*`, you get:

```
Ori-Ground/Orientation-IMGP4167.JPG.xml: <Centre>5.2020... 1.4890... 2.1157...</Centre>
Ori-Ground/Orientation-IMGP4168.JPG.xml: <Centre>5.2002... 0.7974... 2.1018...</Centre>
...
Ori-Ground/Orientation-IMGP4177.JPG.xml: <Centre>4.1747... -3.4504... 2.1690...</Centre>
...
Ori-Ground/Orientation-IMGP4182.JPG.xml: <Centre>3.7778... -6.1801... 2.2321...</Centre>
Ori-Ground/Orientation-IMGP4183.JPG.xml: <Centre>3.6802... -6.6812... 2.2037...</Centre>
```

So you can verify that in the ground repair, where  $Z$  axis coincides with the vertical, all the image center are approximately at the same height, which is quite coherent with the acquisition I did.

See also 16.6.3.1 for non linear transformation using more GCP.

### 3.10.1.3 Creating local repair with RepLocBascule

When `MicMac` and derived tools are used in ground geometry, the default convention is:

- for rectification, the images are generated with  $Z = Cste$  ;
- for matching, the generated grid represents  $Z = f(x, y)$  ;

These conventions are perfectly ok in aerial photogrammetry, the context in which `MicMac` was originally developed. However, it won't work in architectural context, to make the orthophoto of a vertical wall in some ground coordinate system where  $Z$  axis is vertical. For example, suppose we want to use the rectification tool `Tarama` (see 3.12.1) with the orientation `Ground` defined, in 3.10.1.2. If we type :

```
Tarama "IMGP41((6[7-9])|([7-8][0-9])).JPG" Ground Zoom=32
```

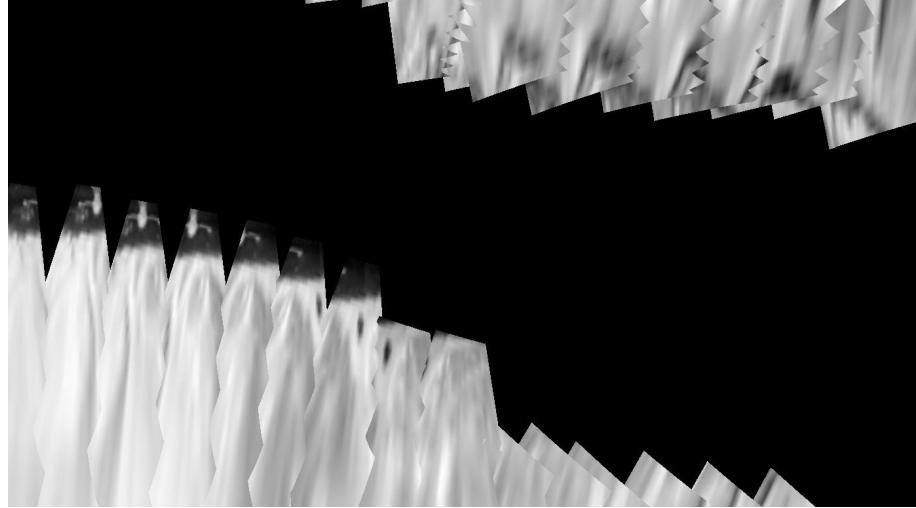


Figure 3.8: Problem with Tarama applied directly in ground geometry

Then we get the "rectified" image of figure 3.8. This is probably not what we wanted.

In such case, what we want to do is to define, for the matching, rectification ... purpose a *local* repair with  $Z$  axis orthogonal to the wall. These local repairs will not be used to change the orientation (we want to maintain all the data in the same ground coordinate system), but they will be used to specify, during matching and rectification, a geometry adapted to the scene. The command to define such repair is `RepLocBascule`, the syntax is:

```
RepLocBascule -help
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {Full name (Dir+Pat)}
 * string :: {Input Orientation}
 * string :: {XML File of Images Measures}
 * string :: {Out Xml File to store the results"}
Named args :
 * [Name=ExpTxt] bool :: {Are tie point in ascii mode ? (Def=false)}
 * [Name=PostPlan] string :: {Pots fix for plane name, (Def=_Masq)}
 * [Name=OrthoCyl] bool :: {Is the repere in ortho-cylindric mode ?}
```

The meaning of the three first args is similar to `SBGlobBascule`. The fourth arg specifies the output xml file. In the data set of Mur Saint Martin, one could type:

```
RepLocBascule "IMGP41((6[7-9])|([7-8][0-9])).JPG" Ground MesureBasc.xml RepCorr.xml PostPlan=_MasqPlan
```

The generated XML file `RepCorr.xml` contains the necessary information to describe a local repair: an origin and 3 axis. Here:

```
<RepereLoc>
    <RepereCartesien>
        <Ori>-0.0303368933943062302 0.0014864175131534263 -0.0145221323781670186</Ori>
        <Ox>-0.00848394971784020499 0.99996077502067815 -0.00254381941768354975</Ox>
        <Oy>-0.00208480031723078411 0.00252621752215371033 0.99999463590194726</Oy>
        <Oz>0.999961837374218176 0.00848920756463100723 0.00206328623854717171</Oz>
    </RepereCartesien>
</RepereLoc>
```

Next sections will describe how they can be used. The general principle is simply to give these repair as optional argument to the program. For example:

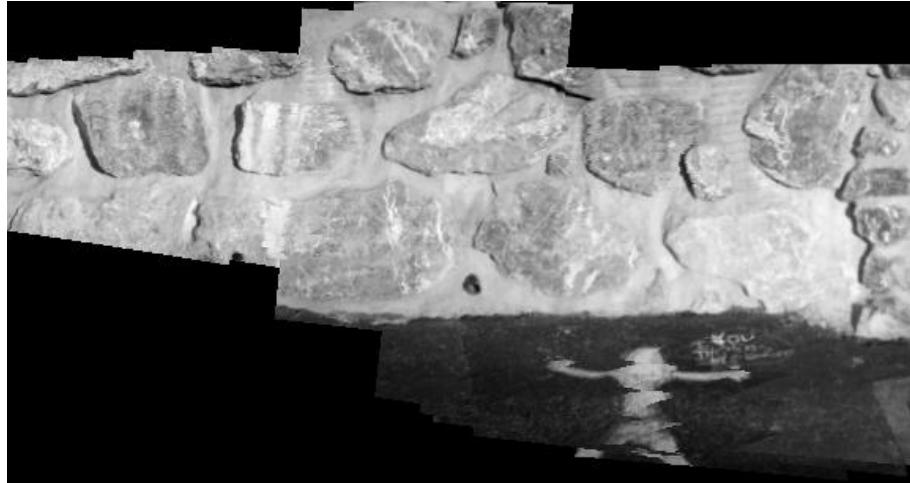


Figure 3.9: Rectified image using the local repair

```
Tarama "IMGP41((6[7-9])|([7-8][0-9])).JPG" Ground Repere=RepCorr.xml
```

This will generate the rectified image of figure 3.9.

#### 3.10.1.4 Geo-referencing with CenterBascule

In the UAS Grand-Leez data set, you can find the file `GPS_WPK_Grand-Leez.csv` which contains embedded GPS and aircraft attitude data. These data are converted with `OriConvert` (section 13.3.4) in a data base of center, which have a similar structure than an orientation database.

The `CenterBascule` tool allows to transform a purely relative orientation, as computed with `Tapas`, in an absolute one. Here for example:

```
CenterBascule "R.*.JPG" All-Rel Nav-adjusted-RTL All-RTL
```

To know the syntax of `CenterBascule` :

```
CenterBascule -help
Unnamed args :
 * string :: {Full name (Dir+Pat)}
 * string :: {Orientation in}
 * string :: {Localization of Information on Centers}
 * string :: {Orientation out}
Named args :
 * [Name=L1] bool :: {L1 minimization vs L2; (Def=false)}
 * [Name=CalcV] bool :: {Use speed to estimate time delay (Def=false)}
```

The meaning of the arguments is:

- first arg defines the set of images, which you want to change orientation;
- second arg defines the location of input orientations (generally it will be purely relative orientation generated using `Tapas` as described above);
- third arg defines the location of the data base of center;
- fourth arg defines the location of output orientation that will be generated by `CenterBascule`
- optional arg `L1` indicates if the transformation from relative to absolute must be done using  $L_1$  or  $L_2$  minimization (as currently the measures will have some redundancy).
- optional arg `CalcV` indicates if GPS delay has to be computed from camera relative speed (see 13.3.4).

The command line `grep Centre All-RTL/*` enable you to quickly check the result.

#### 3.10.1.5 Merging Orientation with Morito

When one has 2 sets of orientation, with at least 2 images common to the two sets, the `Morito` command can be used to merge the two orientations.

An example with the Cuxha Data set, first we create (a bit artificially here) two set of images:

```
Tapas RadialBasic "Abbey-IMG_02[4-8].*.jpg" Out=48
Tapas Figuee "Abbey-IMG_02[0-4].*.jpg" Out=04 InCal=Ori-48/
```

Note that for obvious coherence reason, we have force the two set of images to have the same internal orientation. Now we want to merge this two orientations, taking benefit that images IMG\_0240.jpg ...IMG\_0249.jpg are common to the two subsets. We use:

```
mm3d Morito Ori-48/Orientation-Abbey-IMG_02.*xml Ori-04/Orientation-Abbey-IMG_02.*xml 08
```

The merged orientation are in **Ori-48/**, the program has estimated a *3d* similitude (7 parameter) to do the merging. The merging is done in the system of the first set of images. As the system is redundant, the residual can be used as estimation of the accuracy. They are printed by **Morito**, the **DMatr** represent the residual in rotation and **DPt** the residual in center :

```
...
Orientation-Abbey-IMG_0240.jpg.xml DMatr 0.000457438 DPt 0.000962311
Orientation-Abbey-IMG_0247.jpg.xml DMatr 0.000374341 DPt 0.000914102
Orientation-Abbey-IMG_0248.jpg.xml DMatr 0.00018792 DPt 0.000749945
Orientation-Abbey-IMG_0249.jpg.xml DMatr 0.000361987 DPt 0.000892905
```

### 3.10.1.6 Accuracy Control with GCPCtrl

In photogrammetry, a standard method to control the accuracy of georeferencing result is to use couple of points called Check Points (CP). Check points, unlike Ground Control Points (GCPs), are used either to estimate the *3d* similarity (**GCPBascule**) or for the compensation (**Campari**) on ground measurements. Residuals on check points allow to qualify the accuracy of the georeferencing result.

If for a dataset, several ground measurements are available, a part may serve to perform georeferencing step (**GCPBascule**) while the rest of ground measurements can be used to qualify the accuracy and used as check points. An example of the use of **GCPCtrl**:

```
mm3d GCPCtrl ".*JPG" Ori-RTL-Bascule CP.xml MesuresFinales-S2D.xml
```

Where:

- Ori-RTL-Bascule : are sets of orientations computed using Ground Control Points with (**GCPBascule**)
- CP.xml : are Check Points coordinates (not used to compute orientations of Ori-RTL-Bascule)
- MesuresFinales-S2D.xml : 2d Check Points coordinates

Residuals are used to estimate the accuracy. It is printed by **GCPCtrl** where **D** is the 3d residual and **P** is vector of deviations in axial components:

```
...
Ctrl 1 GCP-Bundle, D=0.00711013 P=[0.00157584,0.00215343,0.0065904]
Ctrl 2 GCP-Bundle, D=0.00213116 P=[-0.000812961,-7.10012e-05,0.00196873]
Ctrl 3 GCP-Bundle, D=0.00503373 P=[-0.00225834,0.000504016,0.00447037]
Ctrl 4 GCP-Bundle, D=0.013416 P=[-0.00209372,0.00171374,0.0131404]
Ctrl 5 GCP-Bundle, D=0.00778608 P=[-0.00129931,-0.000609828,-0.00765265]
Ctrl 6 GCP-Bundle, D=0.00432863 P=[-0.00166909,-0.00255773,-0.00306743]
Ctrl 7 GCP-Bundle, D=0.00536168 P=[0.00157972,-0.0050588,0.000812824]
Ctrl 8 GCP-Bundle, D=0.00493167 P=[-0.000829516,0.00146171,-0.00463645]
Ctrl 9 GCP-Bundle, D=0.00217372 P=[-0.000790291,-0.00131989,0.0015357]

===== ERROR MAX PTS FL =====
|| Value=0.80781 for Cam=DSCO8643.JPG and Pt=7 ; MoyErr=0.698737
=====
```

### 3.10.2 MakeGrid

## 3.11 Tools for full automatic matching

This section is incomplete.



Figure 3.10: Set of tortoise images and visualization of orientation with AperiCloud

### 3.11.1 The tortue data set

This data set is made of 53 photo acquired to model a statue of a tortoise in the pagode of Hue (Vietnam), see figure 3.10 . Although I take some precaution when taking the images, with several point of view having "optimal" angles for stereoscopy, I did not take any note; in these situation, it is hard and boring for a human to recover *a posteriori* the organization of the acquisition. This is a typical situation where to use these tools.

The processing begins classically with tie points and orientation :

```
Tapioca MulScale ".*JPG" 400 1500
Tapas RadialBasic "IMGP68(5[0-9]|6[0-5]).*JPG" Out=Calib
Tapas RadialStd ".*JPG" InOri=Calib Out=All
```

The result of orientation can be seen in figure 3.10.

### 3.11.2 The tool AperoChImSecMM

To compute *a posteriori* the structure of an acquisition, the first thing is to know, for each potential master image, what would be the best set of secondary images. This is what does the tool **AperoChImSecMM**. The meaning of its argument should be quite obvious from inline help :

```
$ mm3d AperoChImSecMM
*****
* Help for Elise Arg main *
*****
Unnamed args :
* string :: {Dir + Pattern}
* string :: {Orientation}
Named args :
* [Name=ExpTxt] bool :: {Have tie point been exported in text format (def = false)}
* [Name=Out] string :: {Out Put destination (Def= same as Orientation-parameter)}
```

Using it with the tortoise data-set we enter :

```
mm3d AperoChImSecMM ".*JPG" All
```

For further use, it is not strictly necessary to understand what is done by **AperoChImSecMM**; however, it can never be bad to understand the tool you use<sup>13</sup> ... When it is finished, we can take a look at **Ori-All** directory, it contains 53 files **ImSec-XXXX.xml**. Each file contains possible sets of secondary images. For example, if we open **ImSec-IMGP6857.JPG.xml** :

```
<ImSecOfMaster>
  <Master>IMGP6857.JPG</Master>
  <Sols>
    <Images>IMGP6860.JPG</Images>
    <Images>IMGP6858.JPG</Images>
```

---

13. at least, this is the "philosophy" with free open source products

```

<Coverage>0.55945759911913906</Coverage>
<Score>0.487036128053463691</Score>
  </Sols>
  <Sols>
    <Images>IMGP6860.JPG</Images>
    <Images>IMGP6858.JPG</Images>
    <Images>IMGP6861.JPG</Images>
    <Coverage>0.816071725376866897</Coverage>
    <Score>0.655094691337392177</Score>
      </Sols>
      <Sols>
        <Images>IMGP6860.JPG</Images>
        <Images>IMGP6859.JPG</Images>
        <Images>IMGP6858.JPG</Images>
        <Images>IMGP6863.JPG</Images>
        <Coverage>0.874862347681766073</Coverage>
        <Score>0.663021676898716383</Score>
          </Sols>
          <Sols>
            <Images>IMGP6860.JPG</Images>
            <Images>IMGP6859.JPG</Images>
            <Images>IMGP6858.JPG</Images>
            <Images>IMGP6861.JPG</Images>
            <Images>IMGP6863.JPG</Images>
            <Coverage>0.874862347681766073</Coverage>
            <Score>0.634082438117069547</Score>
              </Sols>
              <Sols>
                <Images>IMGP6860.JPG</Images>
                <Images>IMGP6859.JPG</Images>
                <Images>IMGP6858.JPG</Images>
                <Images>IMGP6861.JPG</Images>
                <Images>IMGP6863.JPG</Images>
                <Images>IMGP6862.JPG</Images>
                <Coverage>0.874862347681766073</Coverage>
                <Score>0.611377533752188174</Score>
              </Sols>

```

Each **Sol** contains a possible set of secondary images. There are several sets because there is no universal criteria to define the best set : it must covers all the direction around the master image (to avoid hidden part), it must contains images with the "good" angles for stereoscopy, it must avoid redundancy and have a minimal number of images (for efficiency). A these criteria are generally contradictory, **AperoChImSecMM** propose an optimal set for each cardinal (between 2 and 6). Although to facilitate an automatic exploitation, each set has an associated score. Here the "best" set according to **AperoChImSecMM** has 4 image with a score of 0.663.....

These result will be used implicitly in the following automatic tools. It is also possible to use them explicitly "by hand" in creating your own **xml** file for **MicMac**<sup>14</sup>. See for example of how to use this inside **MICMAC** the section **ImSecCalcApero** of file **include/XML\_MicMac/MM-TieP.xml**.

### 3.11.3 The tool MMInitialModel

The tool **MMInitialModel** is used to create fully automatically a regularly dense but coarse **3D** model out of a set of images. Although its main purpose is to create a model that will be used to drive future tools, it can already be used now for those who only need a regular coarse **3D** model. This tool requires that the **AperoChImSecMM** command has been executed before. The syntax is:

```

$ mm3d MMInitialModel
*****
* Help for Elise Arg main *
*****
Unnamed args :

```

---

14. Use in **Malt** will come soon



Figure 3.11: Some view point of the coarse model create with MMInitialModel on tortoise data-set

```
* string :: {Dir + Pattern}
* string :: {Orientation}

Named args :
* [Name=Visu] bool :: {Interactif Visualization (tuning purpose, program will stop at breakpoint)}
* [Name=DoPly] bool :: {Generate ply ,for tuning purpose, (Def=false)}
* [Name=Zoom] INT :: {Zoom of computed models, (def=8)}
* [Name=ReduceExp] REAL :: {Down scaling of cloud , XML and ply, (def = 3)}
```

When using it now, do not forget the arg **DoPly=1**, else you won't get any ply file! An example of use :

```
mm3d MMInitialModel ".*JPG" All DoPly=1
```

Figure 3.11 present the result of this command.

For curious users who wants to see the real time progression, the **Visu=1** option can be used on system where **X11** is installed. Of course, it's then better to use it with only one image. For example try :

```
mm3d MMInitialModel IMGP6857.JPG All Visu=1
```

Figure 3.12 presents some snapshot of the windows obtained when using this option.

### 3.11.4 The tool MMTTestAllAuto

The tool **MMTTestAllAuto** is a *precursor* of the fully automatic tool that will be executed, driven by the coarse model of **MMInitialModel**. It assumes that **AperoChImSecMM** has been executed, and compute a "fine" 3D model for a given master image using a predefined parametrization of **MICMAC**.

The tool works generally well when the master image has "good" secondary image. Conversely, the result can be almost empty in the other cases. Figure 3.13 illustrates the results.

## 3.12 Tools for simplified semi-automatic matching

Even when the full automatic matching will be available, many users will need to have in some circumstances a finer control of the process. This is the aim of these tools.

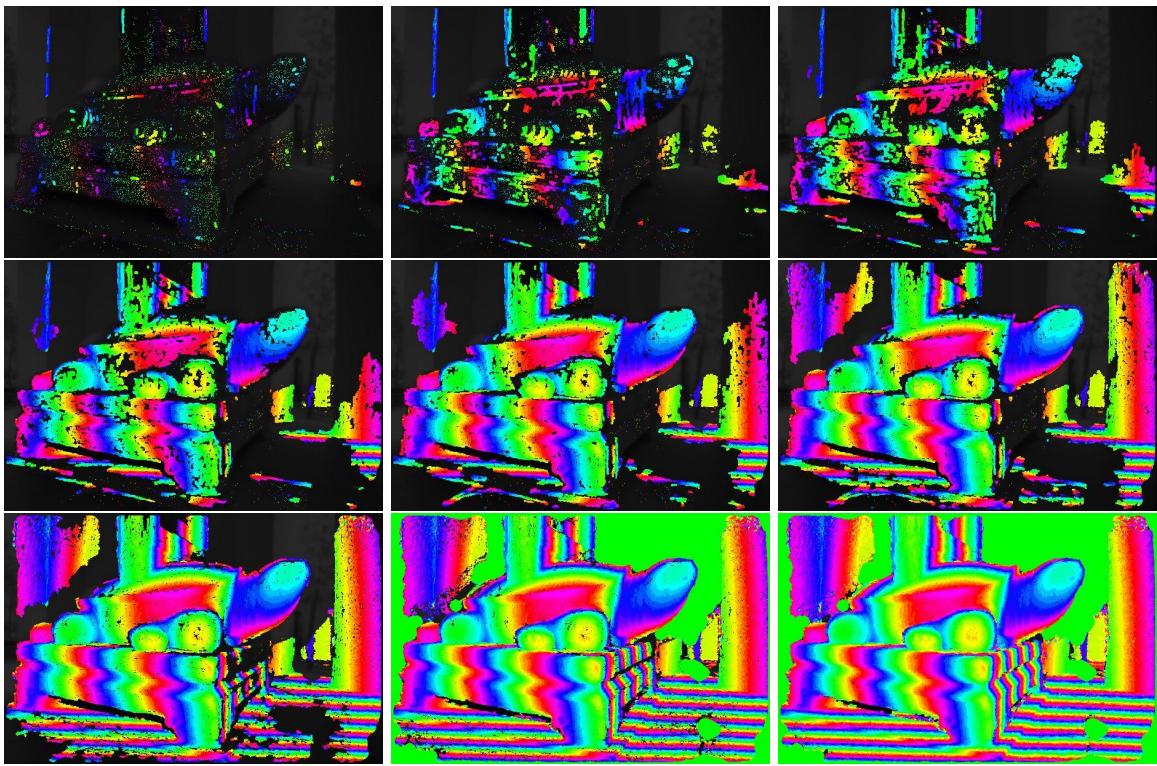


Figure 3.12: Evolution of 3D model, as can be seen with `Visu=1` option of `MMInitialModel`

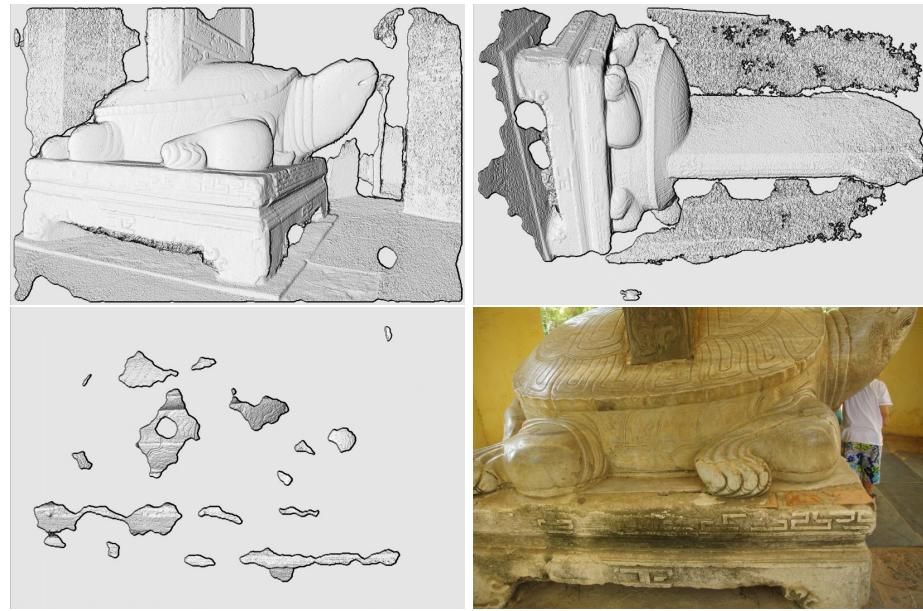


Figure 3.13: Illustration of the `MMTestAllAuto` tool; up two views where it works correctly; down one view (and its corresponding original image) where, due to lack of good secondary image, the result is almost empty



Figure 3.14: Result of Tarama of Mur Saint Martin data set

### 3.12.1 Basic rectification with Tarama

To run the matching process (with **MicMac** or **Malt**) the program must decide which space has to be explored. If no information is given, the program will adopt a default strategy: it will select points of the scene that are visible on at least  $N$  images<sup>15</sup>, making the assumption that the scene is globally flat. Although this strategy may be perfectly ok on aerial acquisition, in the general case it may create a uselessly too large area. In general case, the area selection requires some semantic interpretation of the scene and can only be made by, you, the user who only knows what he wants. For example, in the Saint Martin data set, an automatic program has no information to determine that you want to do the matching on the wall and not on the street.

An easy way to indicate which part of the scene you want to use, is to create a mask on a rectified mosaic. The tool Tarama allows to create such mosaic. At this step the relief is unknown and the rectification is made with the assumption that  $Z = Z_{Moy}$ , so of course these mosaic are not very accurate.

The syntax is:

```
Tarama -help
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {Full Image (Dir+Pat)}
 * string :: {Orientation}
Named args :
 * [Name=Zoom] INT :: {Resolution, (Def=8, must be pow of 2)}
 * [Name=Repere] string :: {local repair as created with RepLocBascule}
 * [Name=Out] string :: {directory for output (Def=TA)}
```

The meaning of the two first mandatory arguments should be quite obvious now. For the optional arguments:

- **Zoom** indicates the resolution of the mosaic, it must be a power of 2 as it is made using the image pyramid of **MicMac**;
- **Repere** indicates an optional repair created by **RepLocBascule** as described in 3.10.1.3
- **Out** describes the directory where the mosaic will be created.

With the Mur Saint Martin data set, using the orientation created with **SBGlobBascule** (3.10.1.1), one can type:

```
Tarama "IMGP41((6[7-9])|([7-8][0-9])).JPG" LocBasc
```

The mosaic image will be created on **TA/TA.LeChantier.tif**. Figure 3.14 shows the result.  
Example using local repair, have been done in 3.10.1.3.

---

<sup>15</sup>. generally  $N = 2$  or  $N = 3$

### 3.12.2 Simplified matching in ground geometry with Malt

#### 3.12.2.1 General characteristics

Malt is a simplified interface to MicMac. Currently it can handle matching in ground geometry (see 5.4.1) and ground-image geometry (see 5.4.5).

Ground geometry is adapted when the scene can be described by a single function  $Z = f(X, Y)$  (with  $X, Y, Z$  being euclidean coordinates); this case occurs quite often when the scene is relatively flat and the acquisition is made by photo acquired orthogonaly to the main plane. The main use cases are:

- modelization of facades to generate ortho photo in architecture;
- modelization of earth surface from aerial acquisition;

Ground image geometry is very general and flexible and can be used in almost all acquisition. Its main drawbacks is that it requires<sup>16</sup> some interaction to select the master images, the mask of these images and the associated secondary images.

The basic syntax requires 3 args:

```
Malt Type Pattern Orient
```

The meaning being:

- first args is a enumerated value, specifying the kind of matching required; the two possible values are:
  - Ortho , for a matching adapted to ortho photo generation;
  - UrbanMNE , for a matching adapted to urban digital elevation model;
  - GeomImage , for a matching in ground image geometry;
- second arg specifies the subset of images;
- third arg specifies the orientation;

An example with data set of Mur Saint Martin:

```
Malt Ortho "./IMGP41((6[7-9])|([7-8][0-9])).JPG" Basc
```

A basic help can be asked with Malt -help:

```
Malt -help
Valid Types for enum value:
  Ortho
  UrbanMNE
  GeomImage
*****
* Help for Elise Arg main *
*****
Unnamed args :
  * string :: {Mode of correlation (must be in allowed enumerated values)}
  * string :: {Full Name (Dir+Pattern)}
  * string :: {Orientation}
Named args :
  * [Name=Master] string :: { Master image must exist iff Mode=GeomImage}
  * [Name=SzW] INT :: {Correlation Window Size (1 means 3x3)}
  * [Name=UseGpu] bool :: {Use Cuda acceleration, def=false}
  * [Name=Regul] REAL :: {Regularization factor}
  * [Name=DirMEC] string :: {Subdirectory where the results will be stored}
  * [Name=DirOF] string
  * [Name=UseTA] INT :: {Use TA as Masq when it exists (Def is true)}
  * [Name=ZoomF] INT :: {Final zoom, (Def 2 in ortho,1 in MNE)}
  * [Name=ZoomI] INT :: {Initial Zoom, (Def depends on number of images)}
  * [Name=ZPas] REAL :: {Quantification step in equivalent pixel (def is 0.4)}
  * [Name=Exe] INT :: {Execute command (Def is true !!)}
  * [Name=Repere] string :: {Local system of coordinat}
  * [Name=NbVI] INT :: {Number of Visible Image required (Def = 3)}
  * [Name=HrOr] bool :: {Compute High Resolution Ortho}
  * [Name=LrOr] bool :: {Compute Low Resolution Ortho}
  * [Name=DirTA] string :: {Directory of TA (for mask)}
  * [Name=Purge] bool :: {Purge the directory of Results before compute}
  * [Name=DoMEC] bool :: {Do the Matching}
  * [Name=UnAnam] bool :: {Compute the un-anamorphosed DTM and ortho (Def context dependant)}
  * [Name=2Ortho] bool :: {Do both anamorphosed ans un-anamorphosed ortho (when applicable) }
  * [Name=ZInc] REAL :: {Incertitude on Z (in proportion of average depth, def=0.3) }
  * [Name=DefCor] REAL :: {Default Correlation in un correlated pixels (Def = 0.2) }
  * [Name=CostTrans] REAL :: {Cost to change from correlation to uncorrelation (Def = 2.0) }
  * [Name=Etape0] INT :: {First Step (Def=1) }
  * [Name=AffineLast] bool :: {Affine Last Etape with Step Z/2 (Def=true) }
  * [Name=ResolOrtho] REAL :: {Resolution of ortho, relatively to images (Def=1.0; 0.5 mean smaller images) }
  * [Name=ImMNT] string :: {Filter to select images used for matching (Def All, usable with ortho) }
  * [Name=ImOrtho] string :: {Filter to select images used for ortho (Def All) }
  * [Name=ZMoy] REAL :: {Average value of Z}
  * [Name=Spherik] bool :: {If true the surface for redressing are spheres}
```

16. for now, I hope it will change in 2013

Name	Meaning	Ortho	UrbanMNE	GeomImage
<b>SzW</b>	Sz correlation window	2	1	1
<b>Regul</b>	Regularization factor	0.05	0.02	0.02
<b>UseGpu</b>	Use Cuda	false	false	false
<b>UseTA</b>	Masq with TA	true	true	true
<b>ZoomF</b>	Final resolution	2	1	1
<b>ZoomI</b>	Initial resolution	XXX	XXX	XXX
<b>ZPas</b>	Z quantification in pixel	0.4	0.4	0.4
<b>Exe</b>	Execute the command	true	true	true
<b>Repere</b>	Name of a local repere for matching	None	None	??
<b>NbVI</b>	Minimal number of image visible in each ground point	3	3	3
<b>HrOr</b>	Compute High Resolution individual ortho photo	true	false	??
<b>LrOr</b>	Compute Low Resolution individual ortho photo	true	false	??
<b>DirTA</b>	Directory where the mask is to search	TA/	TA/	???

Figure 3.15: Default values of malt parameters according to main options

For optional parameters, the default value generally depends on the first parameter. For example the parameter **SzW**, defines the correlation windows size, its default value is:

- 1 (ie window 3x3) for DEM urban generation and ground image geometry because we want to preserve discontinuities;
- 2 (ie window 5x5) for ortho generation, because priority here is robustness;

As usual these default values can be changed explicitly, for example:

```
Malt Ortho "./IMGP41((6[7-9])|([7-8][0-9])).JPG" Basc SzW=5
```

### 3.12.2.2 Optional parameters

Table 3.15 presents a summary of meaning and default value of **Malt** parameters;

Some comments:

- the **ZPas** does not specify directly a value in ground geometry; for a given value of a the **ZPas** parameters, MICMAC will compute the step, in ground geometry, such that, two consecutive projected point in images are on average separated of **ZPas**; in the simple case where there would be two images, with a constant base-to-height ratio  $R = \frac{B}{H}$ , the step in ground geometry would be  $\frac{ZPas}{R}$ ,
- the **NbVI**
- Warning : If you use **UseGpu** and **GeomImage**, it's not a matching in ground image geometry, but in geometry terrain.

### 3.12.3 Image geometry with Malt

An example of using **Malt** in mode **GeomImage** with the **Ramses** data set:

```
Malt GeomImage .*CR2 All Master=IMG_0355.CR2
```

There are some specificities when using **Malt** in the mode **GeomImage**:

- **Master** parameter must have a value, as it is the only way to distinguish the master image from the global set of image given by the pattern;
- masq is not search in **TA/TA.LeChantier.tif**; if, for example, the master image is **IMG\_0355.CR2** then masq is **IMG\_0355\_Masq.tif** for example
- directory storing results depends on the master image, with **IMG\_0355.CR2** it will be **MM-Malt-Img-IMG\_0355**;

See 16.4.1 for an example of using the **Spherik** option.



Figure 3.16: Individual ortho image, and mask image for image IMGP4182.jpg

### 3.13 Ortho photo generation

The simplified tool for generating ortho mosaic is **Tawny**, it is an interface to the **Porto** tool described in 7.2. Using **Tawny** is quite simple because it assumes that data have been correctly prepared and organized during matching process. Practically this is done when matching has been made using **Malt** and I recommend to only use **Tawny** in conjunction with **Malt**. In Ortho Mode, **Malt** has created a set of individual ortho images, associated mask, incidence image, ... in a directory **Ortho-MEC-Malt/**; see for example 3.16. The job of **Tawny** is essentially to merge these data and to optionally do some radiometric equalization.

For the radiometric equalization, **Tawny** will compute for each individual ortho image  $O_i$ , a polynom  $P_i$  such that,  $\forall i, j, x, y$  where ortho image  $O_i$  and  $O_j$  are both defined in  $x, y$  we have relation:

$$O_i(x, y)P_i(x, y) = O_j(x, y)P_j(x, y) \quad (3.1)$$

The problem with such formula is that it can lead to important drift in radiometry. So there is also a global polynom  $R$  that is computed, this polynom is such that:

$$O_i(x, y)P_i(x, y)R(x, y) = O_i(x, y) \quad (3.2)$$

The radiometry of each image used for ortho photo will finally be  $O_i(x, y)P_i(x, y)R(x, y)$ . Of course for equation 3.1 and 3.2, there is much more observations than unknowns and they are solved using least mean square. User can control the radiometric equalization by specifying the degree of the polynom. The syntax is:

```
Tawny -help
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {Directory where are the datas}
Named args :
 * [Name=DEq] INT :: {Degree of equalization (Def=1)}
 * [Name=DEq] INT :: {Degree of equalization (Def=1)}
 * [Name=DEqXY] Pt2di :: {Degree of equalization, if diff in X and Y}
 * [Name=AddCste] bool :: {Add unknown constant for equalization (Def=false)}
 * [Name=DegRap] INT :: {Degree of rappel to initial values, Def = 0}
 * [Name=DegRapXY] Pt2di :: {Degree of rappel to initial values, Def = 0}
 * [Name=RGP] bool :: {Rappel glob on physically equalized, Def = true}
 * [Name=DynG] REAL :: {Global Dynamic (to correct saturation problems)}
 * [Name=ImPrio] string :: {Pattern of image with high prio, def=.*}
 * [Name=SzV] INT :: {Sz of Window for equalization (Def=1, means 3x3)}
 * [Name=CorThr] REAL :: {Threshold of correlation to validate homologous}
 * [Name=NbPerIm] REAL :: {Average number of point per image}
```

The only mandatory argument is the directory where the elementary ortho images have been created by **Malt**. Parameters **DEq**, **DEqXY**, **AddCste**, **DegRap**, **DegRapXY** are relative to the correction function used in radiometric equalization process:

- **DEq** specifies the degree of polynomials  $O_i$ , the default value is 1 , which means that for each ortho image,  $A_i, B_i, C_i$  are computed to satisfy according to least mean square the equation 3.3;

- **DEqXY** specifies the case where the degree of  $O_i$  are different in  $x$  and  $y$ , if **DEqXY=[DX, DY]**, the unknown monoms will be  $x^n, y^m$  such that  $n \leq D_X, m \leq D_Y, n + m \leq \text{Max}(D_X, D_Y)$ ;
- **AddCste** in this case an unknown constant  $K_i$  is add to each ortho  $O_i$  and the equation 3.1 is replaced by 3.4; in almost every case, it is preferable to let the default value **AddCste=false**;
- **DegRap** fix the degree of global polynom  $R$ ;
- **DegRapXY** fix the degree of global polynom  $R$  when different in  $x$  and  $y$ ;

$$O_i(x, y)(A_i + B_i x + C_i y) = O_j(x, y)(A_j + B_j x + C_j y); \quad (3.3)$$

$$O_i(x, y)P_i(x, y) + K_i = O_j(x, y)P_j(x, y) + K_j \quad (3.4)$$

Table 3.17 illustrates the influence of this parameter:

- first line, with the minimum degree parameter, **DEq=0 DegRap=0** some frontier are visible;
- second line, with the default parameter, **DEq=1 DegRap=0**, the frontier are almost not visible but there is clearly a drift in radiometry;
- third line line, with degree 1 polynom per image and a degree 2 global attachment, the frontier are almost not visible and the drift has decreased;
- fourth line line, with degree 1 polynom per image and a degree 4, the has disappeared except at points close to the border;

Note however that this data-set is surprisingly difficult to equalize for such a small set. With many data sets, the default parameters already give an acceptable result.

The parameters **SzV**, **CorThr**, **NbPerIm** are relative to the choice of the point used for the radiometric equalization:

- **SzV** is the size of the patch used for each sample of the radiometric equalization, this patches will be used for computing an average value and for correlation ...
- ... on each patch, correlation coefficient  $C_{i,j}$  are computed between pair of images, they are used only if  $C_{i,j} > \text{CorThr}$  ;
- **NbPerIm** indicates the number of sample that will be approximately used on each image;

On many data sets default values should be OK. However it happened with difficult data sets that all the measures were refused for some images pair, which obviously led to an error. Here is an example of the command I used on a data set with such difficulties:

```
Tawny Ortho-MEC-Malt/ DEq=1 DegRap=1 ImPrio=Ort_IMG_.* SzV=3 CorThr=0.6 NbPerIm=5e4
```

Images	Args opt
	DEq=0
	
	DEq=1 De-gRapXY=[2,0]
	DEq=1 De-gRapXY=[4,1]

Figure 3.17: Exemple of influence of polynomial degré parameter on ortho equalization



# Chapter 4

## Use cases with Simplified tools

### 4.1 The Vincennes data set

#### 4.1.1 Description of the data set

On `micmac_data/ExempleDoc/` the directory `Vincennes` contains 106 images of the Vincennes' castle<sup>1</sup>. This data set illustrate how the tools described here can be used to achieve a typical architectural task: compute for each of the main facade an ortho photo, these ortho photo must be referenced in the same coordinate system. Although the ortho-cylindrical option for geometry described to process this acquisition seems a very specific and narrow technical case, practically it corresponds to very current case for facade processing.

The 106 images of Vincennes' data set are organized in 4 subset :

- images `Face1.*` correspond to the first facade;
- images `Face2.*` correspond to the second facade;
- images `Lnk12.*` images acquired to make the link between the two facades;
- images `Calib.*` acquired to have easily a first calibration.

Note that, *before any processing*, the images have been renamed taking into account the acquisition structure. It is highly recommended to do the same thing before processing data sets having some complexity. It avoids the creation of tricky regular expression. These images are jpeg low resolution images to limit the bandwidth when one upload the data, but of course in real case the full resolution raw image will be preferred.

The file `ExeCmd.txt` contains all the commands that we will need to process these images.

#### 4.1.2 Computing tie points and orientations

##### 4.1.2.1 Tie points

The computation of tie points and relative orientation is quite classic now.

For tie points we want to compute:

- points between all pairs of calibration data set;
- points of `Face1` and `Face2` using the linear structure of the acquisition;
- points between `Lnk12` and connected subset of `Face1` and `Face2`;

This is done by :

```
Tapioca All "Calib-IMGP[0-9]{4}.JPG" 1000
Tapioca Line "Face1-IMGP[0-9]{4}.JPG" 1000 5
Tapioca Line "Face2-IMGP[0-9]{4}.JPG" 1000 5
Tapioca All "((Lnk12-IMGP[0-9]{4})|(Face1-IMGP529[0-9])|(Face2-IMGP531[0-9])).JPG" 1000
```

##### 4.1.2.2 Relative orientation

Then we want to make a first calibration with the calibration data set, and use this calibration as an initial value to the global orientation of `Face1`, `Face2` and `Lnk12`. This is done by :

```
Tapas RadialStd "Calib-IMGP[0-9]{4}.JPG" Out=Calib
Tapas RadialStd "(Face1|Face2|Lnk12)-IMGP[0-9]{4}.JPG" Out=All InCal=Calib
```

---

1. they are low resolution images to limit the downloading time

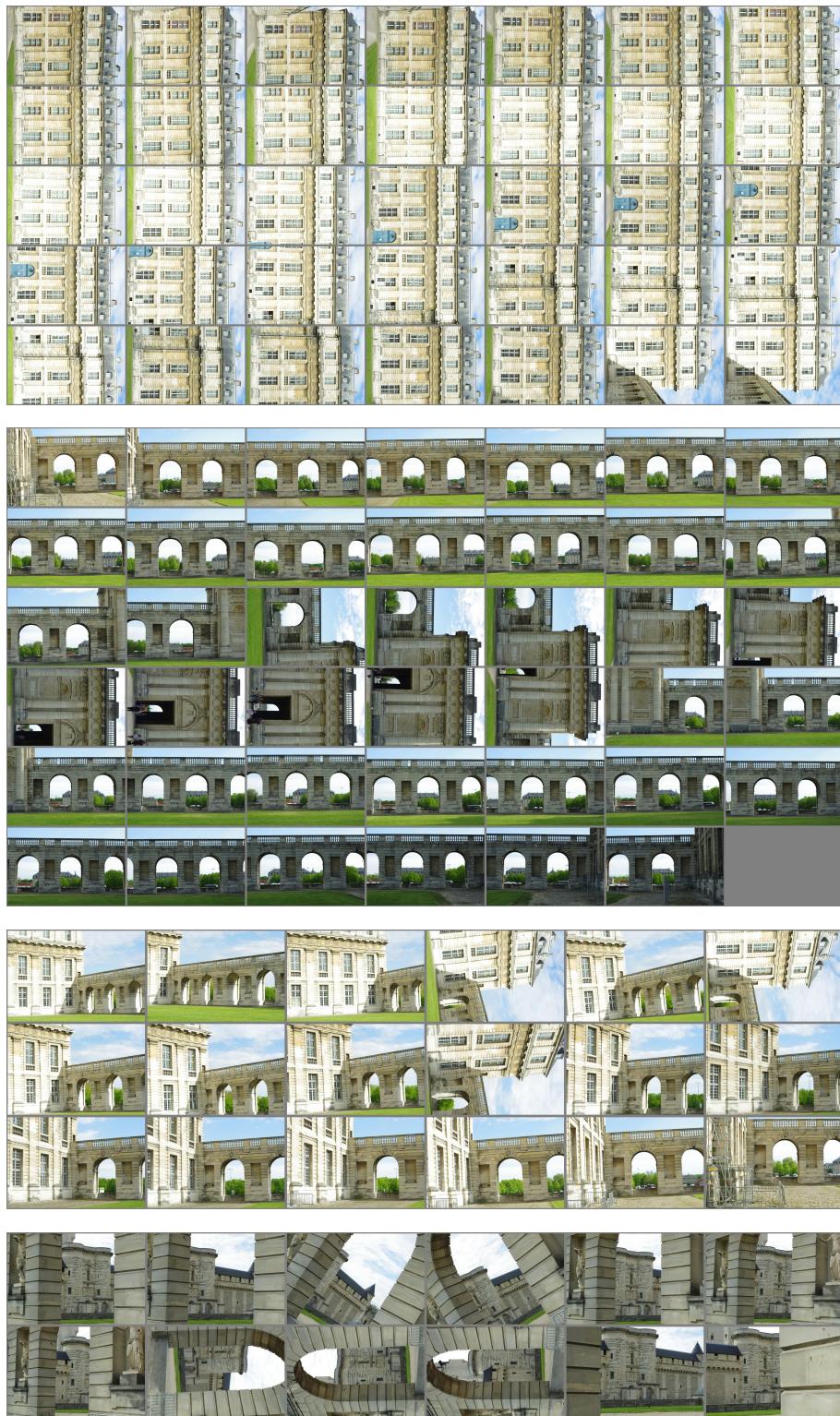


Figure 4.1: Image of Vincenne's data set : Face1, Face2, Lnk12 and Calib

#### 4.1.2.3 Optional, absolute orientation

Finally, we want to transform the orientation from an arbitrary relative orientation to some physically based orientation. If we have some ground control points, this can be done using the **GCPBascule** command (see 3.10.1.2). To generate orientation in **Ori-Ground** :

```
GCPBascule "(Face1|Face2|Lnk12)-IMGP[0-9]{4}.JPG" All Ground Mesure-TestApInit-3D.xml\
Mesure-TestApInit.xml
```

#### 4.1.2.4 Optional, scene-based orientation

Alternatively, if we do not have any GCP and want to put the data in an orientation having some physical meaning, we can use the **SBGlobBascule** command (see 3.10.1.1) :

```
SBGlobBascule "(Face1|Face2|Lnk12)-IMGP[0-9]{4}.JPG" All MesureBascFace1.xml Glob \
PostPlan=_MasqPlanFace1 DistFS=2.0 Rep=ij
```

There is a new option **Rep=ij**, the meaning of this option is :

- it is a string that describe a repair;
- it must contain 2 symbols, each symbols can be in  $\{i,-i,j,-j,k,-k\}$  and describe a vector;
- the global orientation will be such that in the final orientation the line defined by **Line1-Line2** is aligned on first vector, and the normal to the plane is aligned on second vector;
- here in final orientation  $i$  will be the horizontal of the wall and  $j$  will be the normal to the wall, consequently  $k = i \wedge j$  will be the vertical;

### 4.1.3 Matching

#### 4.1.3.1 "Standard" option

The "standard pipeline" for generating an ortho photo of facade, as seen in 3, is for each facade :

- compute a local repair to define the facade with **RepLocBascule**;
- compute a rectified image with **Tarama**;
- make the matching with **Malt**;
- generate the ortho image with **Tawny**;

This can be done by the succession of commands:

```
RepLocBascule "(Face1)-IMGP[0-9]{4}.JPG" Ground MesureBascFace1.xml Repere-F1.xml\
PostPlan=_MasqPlanFace1
Tarama "(Face1)-IMGP[0-9]{4}.JPG" Ground Repere=Repere-F1.xml Out=TA-F1 Zoom=4
Malt Ortho "(Face1)-IMGP[0-9]{4}.JPG" Ground Repere=Repere-F1.xml \
SzW=1 ZoomF=1 DirMEC=Malt-F1 DirTA=TA-F1
Tawny Ortho-Malt-F1/
```

The results are quite deceiving !!! Figure 4.2 illustrate the encountered problem :

- on first line, the ortho photo; it suffer several problem; the main problem are located on the roof (due to bad incidence angles) and on horizontal lines;
- on second line, a snapshot from Meshlab, showing the camera position; it illustrates the fact that in this acquisition all the camera centers are located on the same line;
- the third line, focus on the matching problem that occurs on linear detail that are parallel to the line of acquisition;

#### 4.1.3.2 "Ortho-cylindrical" option

Intuitively it is obvious that when the camera center are all aligned on the same line, the matching problem is ambiguous for line parallel to the acquisition, consequently the quality of result is poor. Obviously, the default would decrease (in fact disappear) if the camera were not aligned, using an UAV or a scaffolding , we could have an optimal geometry similar to aerial acquisition. But it is not always possible to have such material and, for economical reason, it would be interesting to be able to obtain a relatively good quality ortho photo even when all the camera are aligned.

In fact for theoretical reasons described in [Penard L. 2006 ], this problem are much more important in ground geometry than in image geometry. With the option we have seen until now, we have basically this alternative:

- use the ground geometry with a simple process but obtain bad quality results such those of figure 4.2;

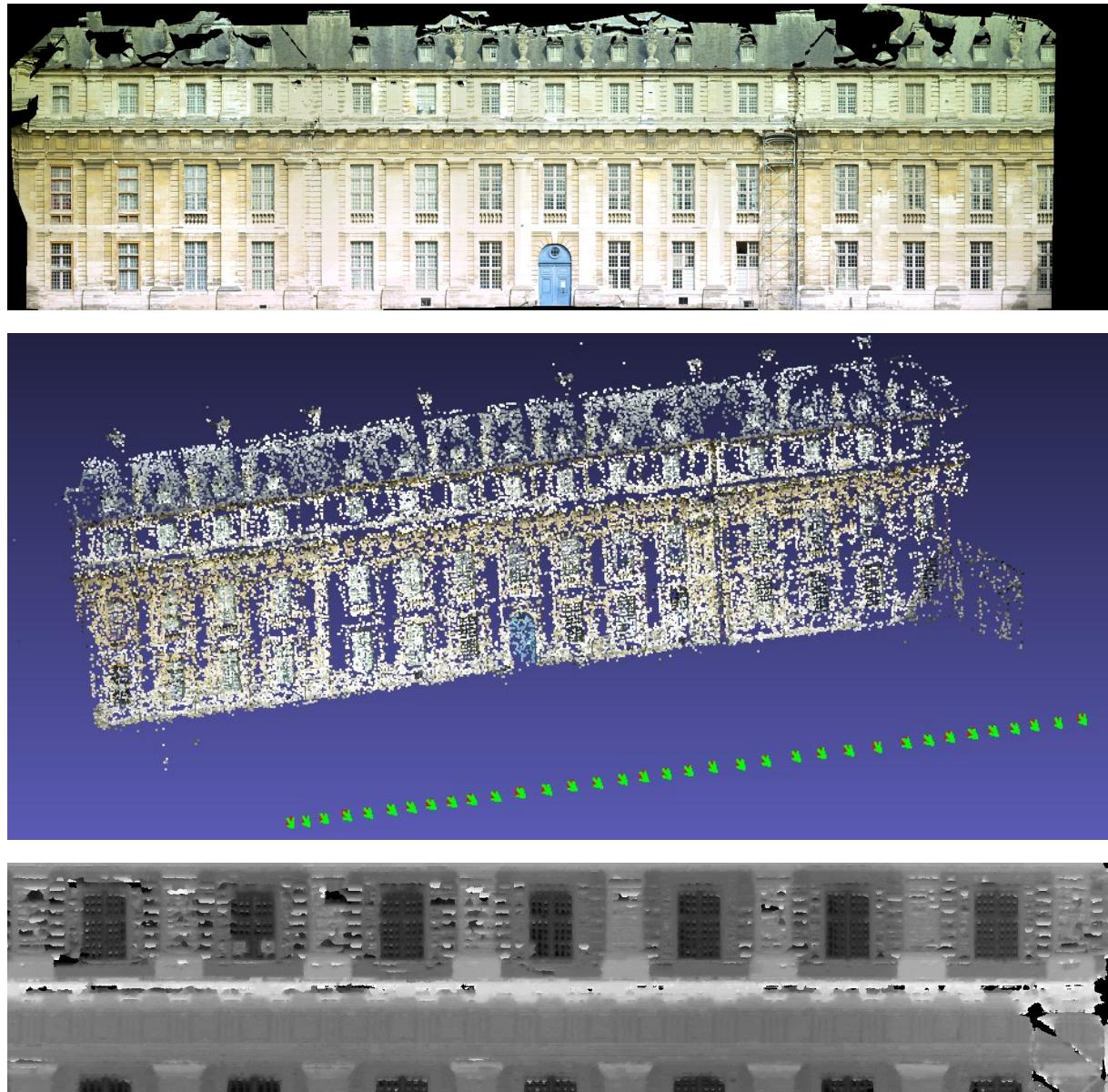


Figure 4.2: Problem with standard processing on Vincennes Facade : low quality ortho photo, alignment of cameras, poor dept map especially for linear structure parallel to camera alignment

- use the image geometry with good results but have a complicated workflow with many depth map that must be merged.

With such acquisition, the ortho-cylindrical geometry combine the benefit of these two geometries. Intuitively this geometry is equivalent to the geometry of a virtual push-broom camera, the line of this virtual push-broom being the line on which are located the camera center. More formally :

- let  $X, Y, Z$  be a coordinate system such that  $Y = 0$  be approximately the line on which the camera are located, and  $Z = D$  be approximately the plane of the wall;
- let  $U, V, L$  be the coordinate system defined by
  - $U = D \tan^{-1}(\frac{X}{Z})$
  - $V = Y$  and  $L = Z$ ;
- we will then compute the DSM as a function  $L = F(U, V)$ .

To use this geometry, we just need to set `OrthoCyl=true` in the command `RepLocBascule` :

```
RepLocBascule "(Face1)-IMGP[0-9]{4}.JPG" Ground MesureBascFace1.xml Ortho-Cyl1.xml \
PostPlan=_MasqPlanFace1 OrthoCyl=true
```

With this option, `RepLocBascule` will also compute, using least mean square, the line that fit the best the alignment of camera perspective centers. If we take a look at file `Ortho-Cyl1.xml` we can see this line coded by `<P0>` and `<P1>` (plus the previous local repair `<Repere>`) :

```
<XmlModeleSurfaceComplexe>
  <XmlOneSurfaceAnalytique>
    <XmlDescriptionAnalytique>
      <OrthoCyl>
        <Repere>
          <Ori>-0.00573 -2.7113574 -0.4521156 </Ori>
          <Ox> 0.00029 0.9999998 -0.0003715 </Ox>
          <Oy> -0.00043 0.0003716 0.9999998 </Oy>
          <Oz> 0.99999 -0.0002960 0.0004372 </Oz>
        </Repere>
        <P0>30.392821 -2.720358 -0.438823</P0>
        <P1>30.391561 -1.720359 -0.43974</P1>
        <AngulCorr>true</AngulCorr>
      </OrthoCyl>
    </XmlDescriptionAnalytique>
    <Id>TheSurf</Id>
    <VueDeLExterieur>true</VueDeLExterieur>
  </XmlOneSurfaceAnalytique>
</XmlModeleSurfaceComplexe>
```

#### 4.1.3.3 Concrete use of "Ortho-cylindric" option

It is then sufficient to give the file created by `RepLocBascule` as an optional parameter to `Tarama` and `Malt` to compute in the adequate geometry; for facade one, we can enter:

```
RepLocBascule "(Face1)-IMGP[0-9]{4}.JPG" Ground MesureBascFace1.xml Ortho-Cyl1.xml \
PostPlan=_MasqPlanFace1 OrthoCyl=true
Tarama "(Face1)-IMGP[0-9]{4}.JPG" Ground Repere=Ortho-Cyl1.xml Out=TA-OC-F1 Zoom=4
Malt Ortho "(Face1)-IMGP[0-9]{4}.JPG" Ground Repere=Ortho-Cyl1.xml \
SzW=1 ZoomF=1 DirMEC=Malt-OC-F1 DirTA=TA-OC-F1
Tawny Ortho-UnAnam-Malt-OC-F1/
```

And for facade 2 :

```
RepLocBascule "(Face2)-IMGP[0-9]{4}.JPG" Ground MesureBascFace2.xml Ortho-Cyl2.xml \
PostPlan=_MasqPlanFace2 OrthoCyl=true
Tarama "(Face2)-IMGP[0-9]{4}.JPG" Ground Repere=Ortho-Cyl2.xml Out=TA-OC-F2 Zoom=4
Malt Ortho "(Face2)-IMGP[0-9]{4}.JPG" Ground Repere=Ortho-Cyl2.xml SzW=1 ZoomF=1 \
DirMEC=Malt-OC-F2 DirTA=TA-OC-F2 NbVI=2
Tawny Ortho-UnAnam-Malt-OC-F2/
```

Note some options of these commands:

- in `RepLocBascule`, the `OrthoCyl=true` as described above;
- in `Tarama`, the `Out=TA-OC-F1` (and `Out=TA-OC-F2`) to specify the directory of output; this is naturally to avoid that each call to `Tarama` overwrite the result of previous calls;
- in `Malt`, the `DirTA=TA-OC-F1` to get the adequate entry from `Tarama` and `Out=DirMEC=Malt-OC-F1` to specify the results; this change the place are written the results of matching, and also the result of individual ortho photo (here it will be `Ortho-UnAnam-Malt-OC-F1/`);

If the ortho-cylindrical geometry is "optimal" for computation, this is generally not a proper geometry for the final user , so at the end of the process, `MicMac` generate an "un-anamorphosed" version of this depth map in euclidean geometry. For example on directory `Malt-OC-F1/`, there exists 9 files `Z_NumX_DeZoomY_STD-MALT.tif` corresponding to the different level of matching in ortho-cylindrical geometry, and a single file `Z_Num1_DeZoom1_Malt-Ortho-UnAnam.tif` corresponding to the euclidean version of the last file ( this is the version presented on second line of figure 4.3). Note that in general there will be very few hidden part on ortho-cylindrical depth map; conversely, they are quite current on euclidean version, but it is intrinsic to what we want to restitute with such acquisition. By default, the ortho photo are also generated in euclidean geometry using the unanamorphosed depth-map. Here for example, they are generated under `Ortho-UnAnam-Malt-OC-F1/` and `Ortho-UnAnam-Malt-OC-F2/`.

Figure 4.3 present some results obtained after this process:

- first line present the depth-map computed in ortho-cylindric geometry using color code;
- second line, euclidean version of the depth map, remark the hidden part;
- third line, ortho photo of facade.

Although all the tool described in this section are rather optimized for ortho-photo generation, it is still possible to generate 3D cloud points. As usual in ground geometry, we use the result of matching for the 3D and the ortho-photo for textures. For example:

```
Nuage2Ply Malt-OC-F1/NuageImProf_Malt-Ortho-UnAnam_Etape_1.xml \
Attr=Ortho-UnAnam-Malt-OC-F1/Ortho-Eg-Test-Redr.tif Scale=3
```

```
Nuage2Ply Malt-OC-F2/NuageImProf_Malt-Ortho-UnAnam_Etape_1.xml \
Attr=Ortho-UnAnam-Malt-OC-F2/Ortho-Eg-Test-Redr.tif Scale=3
```

The meta data file `NuageImProf_Malt-Ortho-UnAnam_Etape_1.xml` contains all the information relative to the local repair use for computation (inside the `<RepereGlob>` balise):

```
<?xml version="1.0" ?>
<XML_ParamNuage3DMaille>
  <NbPixel>5972 1834</NbPixel>
  <PN3M_Nuage>
  ..
  </PN3M_Nuage>
  <RepereGlob>
    <Ori>-0.00573682224569793675 -2.71135741550217935 -0.452115668474152133</Ori>
    <Ox>0.000296255688442622397 0.999999887087029138 -0.000371562236912849938</Ox>
    <Oy>-0.000437158066873386052 0.000371691728275074906 0.999999835369028367</Oy>
    <Oz>0.999999860562685972 -0.000296093208240548543 0.000437268133301418503</Oz>
  </RepereGlob>
  ...
  <Orientation>
  ....
  </Orientation>
  ...
</XML_ParamNuage3DMaille>
```

The point cloud are then generated in the same global repair and are naturally mergeable as can be seen on figure 4.4.

## 4.2 The Saint-Michel de Cuxa data set

### 4.2.1 Description of the data set

On `micmac_data/ExempleDoc/` the directory `MiniCuxha` contains 48 images of the St-Michel de Cuxa's abbey<sup>2</sup>. This data set illustrates how to do a bundle adjustment with ground control points.

---

2. they are low resolution images to limit the downloading time

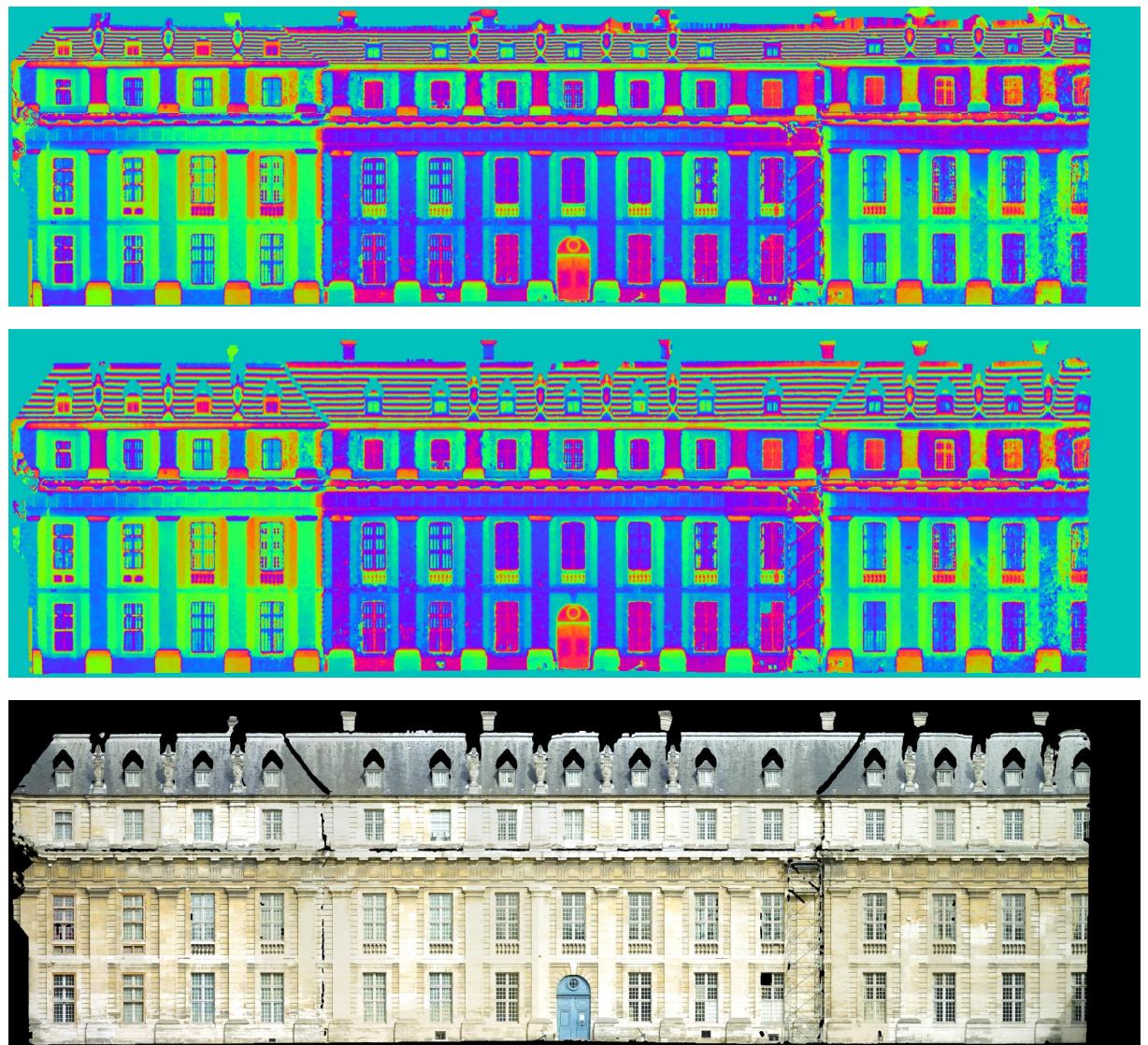


Figure 4.3: 1-Depth map in ortho cylindric geometry, 2-The same, anamorphosed in euclidean geometry, 3-Ortho photo in euclidean geometry

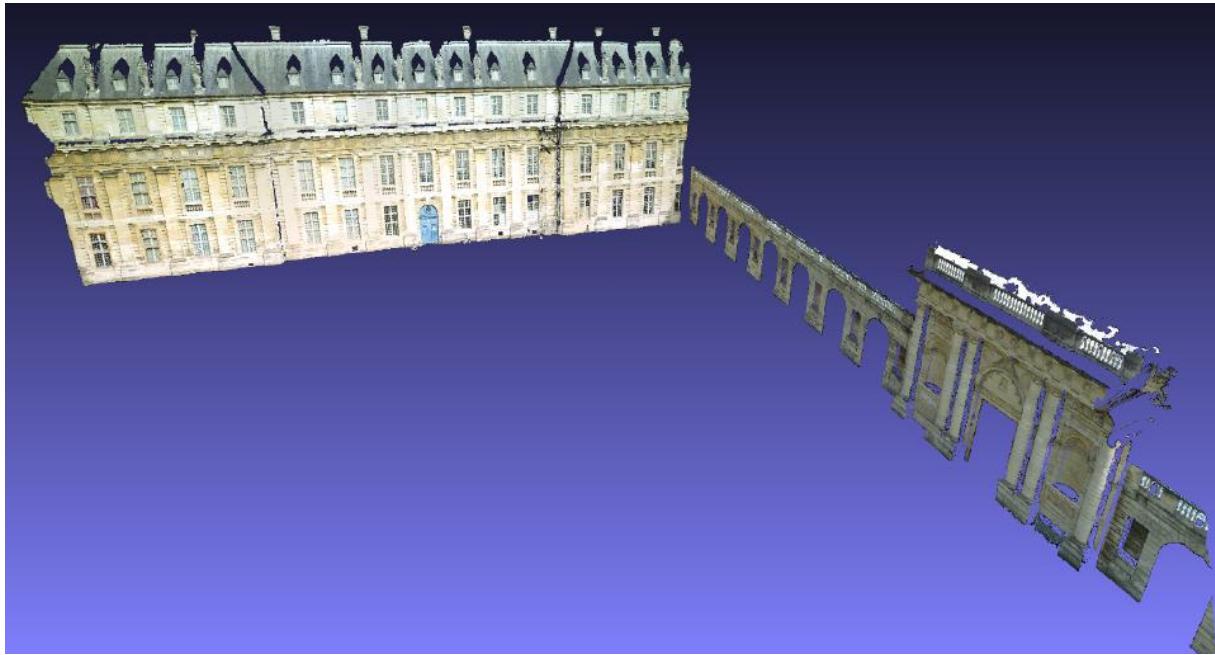


Figure 4.4: Snapshot of two point clouds of the facade

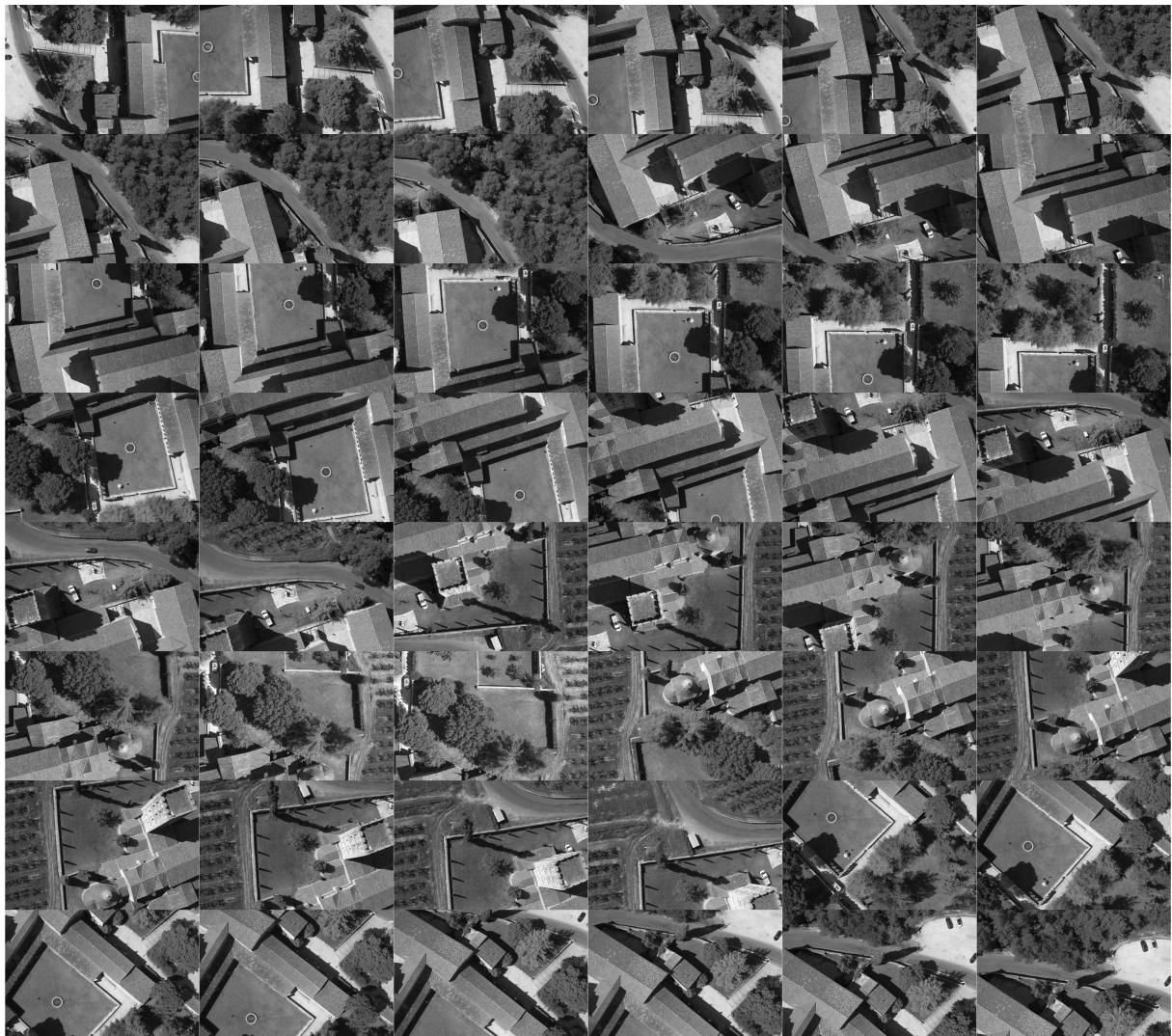


Figure 4.5: Image of Saint-Michel de Cuxa's data set

These images have been taken with an helicopter drone at an approximate height of 100 meters, in a typical aerial photogrammetric setup.

The "standard pipeline" to do a bundle adjustment with ground control points with **MicMac** is:

- compute images relative orientations, with **Tapioca** and **Tapas**;
- transform GCP coordinates into a local euclidean coordinate system, with **GCPConvert**;
- measure image coordinates for a small set of GCP, with **SaisieAppuisInit**;
- transform image relative orientations into the same local coordinate system, with **GCPBascule**;
- measure image coordinates for all GCP, with **SaisieAppuisPredic**;
- transform image relative orientations into the local coordinate system, with **GCPBascule**;
- run the bundle adjustment, with **Campari**;
- transform back relative orientations into an appropriate coordinate system, with **ChgSysCo**;
- compute a rectified image, with **Tarama**;
- make the matching with **Malt**;
- generate the ortho image with **Tawny**;

The file **CmdAbbey.txt** contains all the commands needed to process these data.

## 4.2.2 Computing tie points and relative orientations

### 4.2.2.1 Tie points

As usual, we want to compute matches between all pairs of calibration data set. This is done by:

```
Tapioca MulScale "Abbey-IMG_.*.jpg" 200 800
```

### 4.2.2.2 Relative orientation

Then we want to make a first calibration with a subset of the whole data, and use this calibration as an initial value to the global relative orientation of all images. This is done by:

```
Tapas RadialBasic "Abbey-IMG_(0248|0247|0249|0238|0239|0240).jpg" Out=Calib
Tapas RadialBasic "Abbey-.*.jpg" InCal=Calib Out=All-Rel
```

We can verify that relative orientation was successful by checking the "Residu Liaison Moyens" (root mean square error) value that should be around 0.5 pixel. We can also check visually the result of orientation running **AperiCloud**, described in 3.9.1:

```
AperiCloud "Abbey-IMG_[0-9]*.jpg" All-Rel RGB=0
```

This will generate the **AperiCloud.ply** file containing tie points and cameras locations. We can see that cameras are on the same plane, and that the relative orientations match the flight plan:

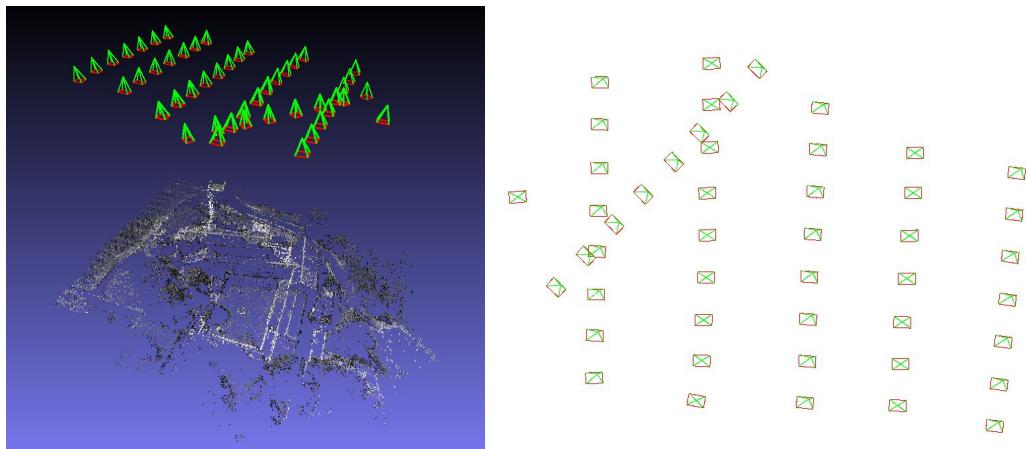


Figure 4.6: Result of relative orientation, computed with **AperiCloud**, perspective and top view.

### 4.2.3 GCP transforms

#### 4.2.3.1 Ground control point coordinates conversion

In this use case, we have got ground control points expressed in WGS84 system. We need to convert them into a local euclidean coordinate system. The important thing is that the local system is euclidean, because all the **MicMac** tools need this assumption to solve equations. Most of the cartographic coordinate systems are not euclidean systems, so we define a local tangent system, defined around a 3D point and its tangent plane, that will lead to a geometry compliant with **MicMac**'s one. This is done with the **GCPConvert** tool (detailed in 13.3.1):

```
GCPConvert "#F=N_X_Y_Z_I" F120601.txt ChSys=DegreeWGS84@SysCoRTL.xml Out=AppRTL.xml
```

#### 4.2.3.2 Ground control point image coordinates input

To add image coordinates measures, we can use the **SaisieAppuisInit** interface in Linux (detailed in 8.4.2):

```
SaisieAppuisInit "Abbey-IMG_0211.jpg" All-Rel NamePointInit.txt MesureInit.xml
```

This will create two **Xml** files **MesureInit-S2D.xml** and **MesureInit-S3D.xml**, which respectively contain images coordinates and corresponding 3D coordinates, computed by spatial resection.

#### 4.2.3.3 Bascule

Now we can transform images relative orientations, as computed with Tapas, expressed in an arbitrary coordinate system, into the local euclidean coordinate system, using 2D images coordinates measures and 3D corresponding ground control points.

```
GCPBascule "Abbey-.*jpg" All-Rel RTL-Init AppRTL.xml MesureInit-S2D.xml
```

Once the images relative orientations have been transformed back in local euclidean coordinate system, one can verify that Z coordinates for the whole data set is nearly constant, which corresponds to the data acquisition setup.

Possible error: "Not enough samples (Min 3) in cRansacBasculementRigide". It means that there is not enough points to compute a Bascule transform. You should add more points with **SaisieAppuisInit**: at least 3 GCP whose projection are known in at least 2 images are needed.

#### 4.2.3.4 Adding points with predictive interface **SaisieAppuisPredic**

When the global transform between ground control points and image relative orientations is known, we can switch to the predictive interface **SaisieAppuisPredic** which will display the remaining ground control points, loaded from the **Xml** file **AppRTL.xml**. You need to adjust points image location and validate them.

```
SaisieAppuisPredic "Abbey-.*jpg" RTL-Init AppRTL.xml MesureFinale.xml
```

#### 4.2.3.5 Bascule

Again we can transform images relative orientations, this time with a more substantial number of images measures, which will give a better transform.

```
GCPBascule "Abbey.*jpg" All-Rel RTL-Bascule AppRTL.xml MesureFinale-S2D.xml
```

### 4.2.4 Bundle adjustment with ground control points

Now we can run a constrained bundle adjustment combining ground control points and tie points, with the **Campari** command, described in 3.9.2.

```
Campari "Abbey.*.jpg" RTL-Bascule RTL-Compense GCP=[AppRTL.xml,0.1,MesureFinale-S2D.xml,0.5]
```

### 4.2.5 Post-processing

#### 4.2.5.1 Coordinate system backward transform

Then one can transform coordinates from the local euclidean coordinate system to a geographic coordinate system, and compute ortho-images which can be superimposed on vectorial maps (and *vice versa*). For example, if we want to transform our data into the sinusoidal projection, for which we have got a file `SysCoSinus90W.xml` storing the transformation parameters, the command is:

```
ChgSysCo "Abbey.*.jpg" RTL-Compense SysCoRTL.xml@SysCoSinus90W.xml Sin90
Tarama "Abbey.*.jpg" Sin90
Malt Ortho "Abbey.*.jpg" Sin90 SzW=1 AffineLast=false DefCor=0.0
Tawny Ortho-MEC-Malt/
```

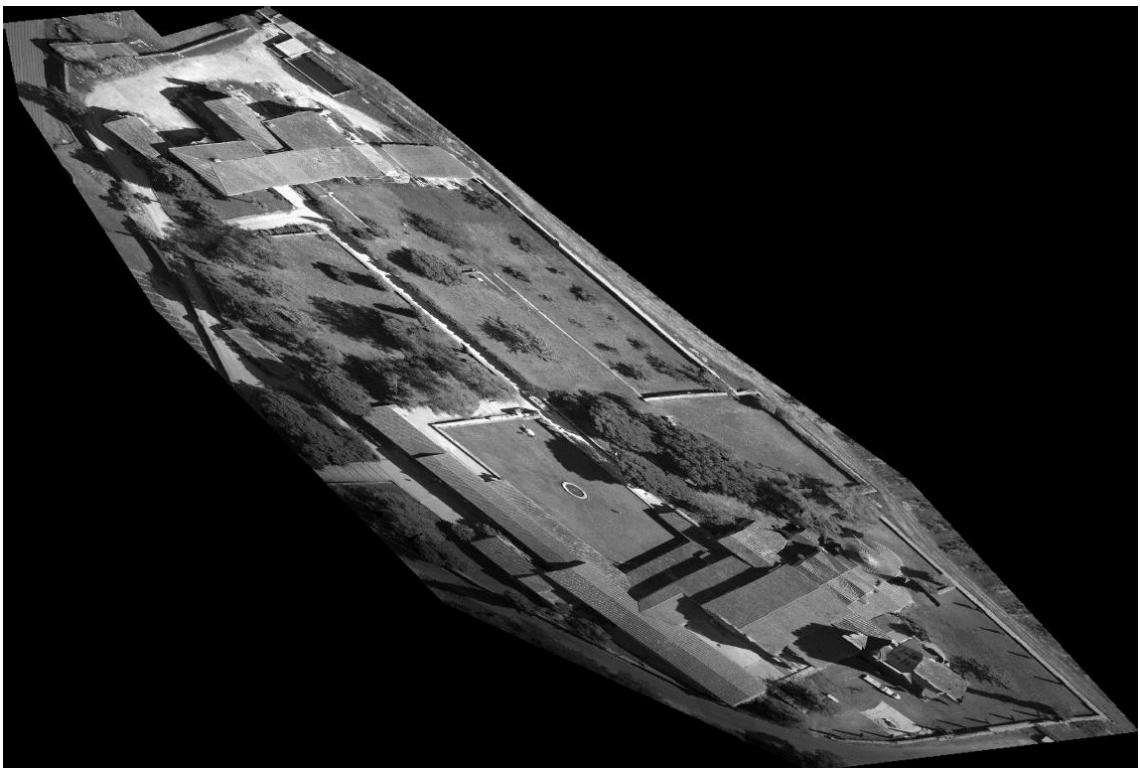


Figure 4.7: Image rectification in sinusoidal projection, with Tarama

The result is ugly, but if we have a look to the global earth mapping with sinusoidal projection, it is obvious that we cannot have a good representation at the European longitude with the sinusoidal projection.

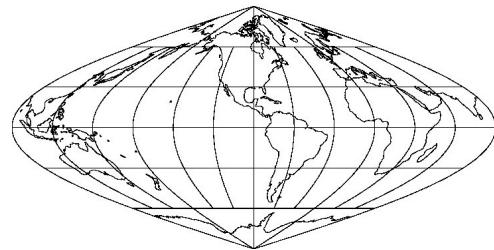


Figure 4.8: Sinusoidal projection, with Central Meridian 90 °W

What we expect would be more like the result of a projection in Lambert93 coordinate system:

```
ChgSysCo "Abbey.*.jpg" RTL-Compense SysCoRTL.xml@Lambert93 L93
```

```
Tarama "Abbey.*.jpg" L93
```

```
Malt Ortho "Abbey.*.jpg" L93 SzW=1 AffineLast=false DefCor=0.0
```

```
Tawny Ortho-MEC-Malt/
```



Figure 4.9: Image rectification in Lambert93 projection, with Tawny

## 4.3 The Grand-Leez dataset

### 4.3.1 Dataset description

The directory `UASGrandLeez/` in `micmac.data/ExempleDoc/`, contains UAS<sup>3</sup> imagery which are used to illustrate a complete workflow devoted to the production of a canopy surface model. The aerial survey was performed by the lab of Forest and Nature Management<sup>4</sup> of the University of Liege (Belgium). The image block is made up of 200 low-oblique vantage jpeg images, acquired with a Ricoh GRIII (10 Mpixels, focal length of 28 mm 35 equivalent). The flight was performed with a Gatewing X100 platform. The inertial measurement unit provides GPS position and attitude (omega, phi, kappa) of the UAS for each image frame (stored in `GPS_WPK_Grand-Leez.csv` file). In order to reduce the size of this dataset, raw images were resampled to 800 pixels width. The processing of these images can however take a few hours. The file `Documentation/FIGS/UASGrandLeez/Cmd_UAS_Grand-Leez.txt` contains all the command lines related to this processing workflow.

First, let's take a look at the images. A convenient tool to visualize multiple images in a panel is the `PanelIm` tool which was used to produce figure 4.11 and other image panels in this manual:

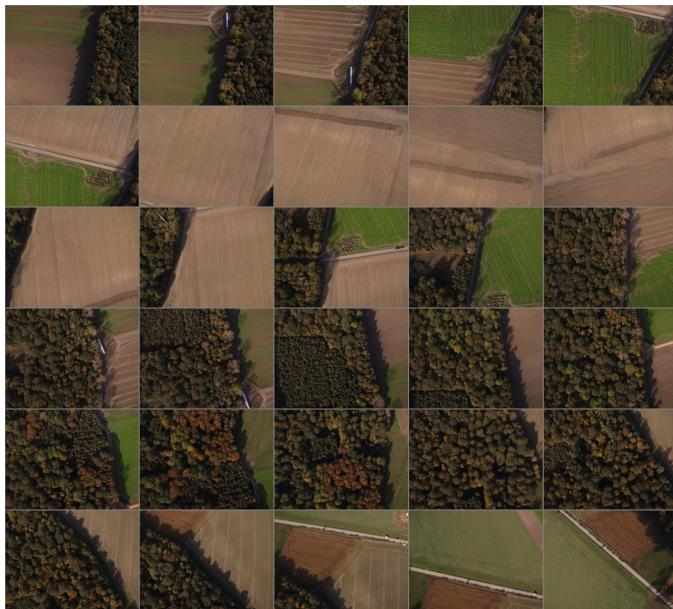


Figure 4.11: The Grand-Leez dataset

```
mm3d PanelIm ./ "R00405[0-5][0:2:4:6:8].JPG" Scale=3
```

In this example, we deal with direct georeferencing, which consist of using camera positions (or camera center) to georeference the photogrammetric model. At first, the tool `OriConvert` (section 13.3.4) is used to convert telemetry data into `MicMac` format. Telemetry data aren't exclusively used for georeferencing, but also to determine potential image pairs. The list of image pairs is then used for the computation of tie points (`Tapioca File` ...). In addition, embedded GPS data are used in a constrained bundle block adjustment in order to avoid non-linear distortions which can hinder photogrammetric measurements.

The pipeline presented here to process UAS imagery with embedded GPS data with `MicMac` is organized as follows:

1. Transform initial external orientation file (embedded GPS data) into the `MicMac` format and generate an image pairs file with `OriConvert`. In addition, latitude and longitude GPS information are projected in the Belgian Lambert 72 coordinate system;
  2. Compute image tie points with `Tapioca File`;
  3. Initialize the image block orientation with `Martini`;
  4. Determine image relative orientation, with `Tapas`;
- 
3. Unmanned Aerial System or `drone`
  4. <http://www.gembloux.ulg.ac.be/gestion-des-ressources-forestieres-et-des-milieux-naturels/>

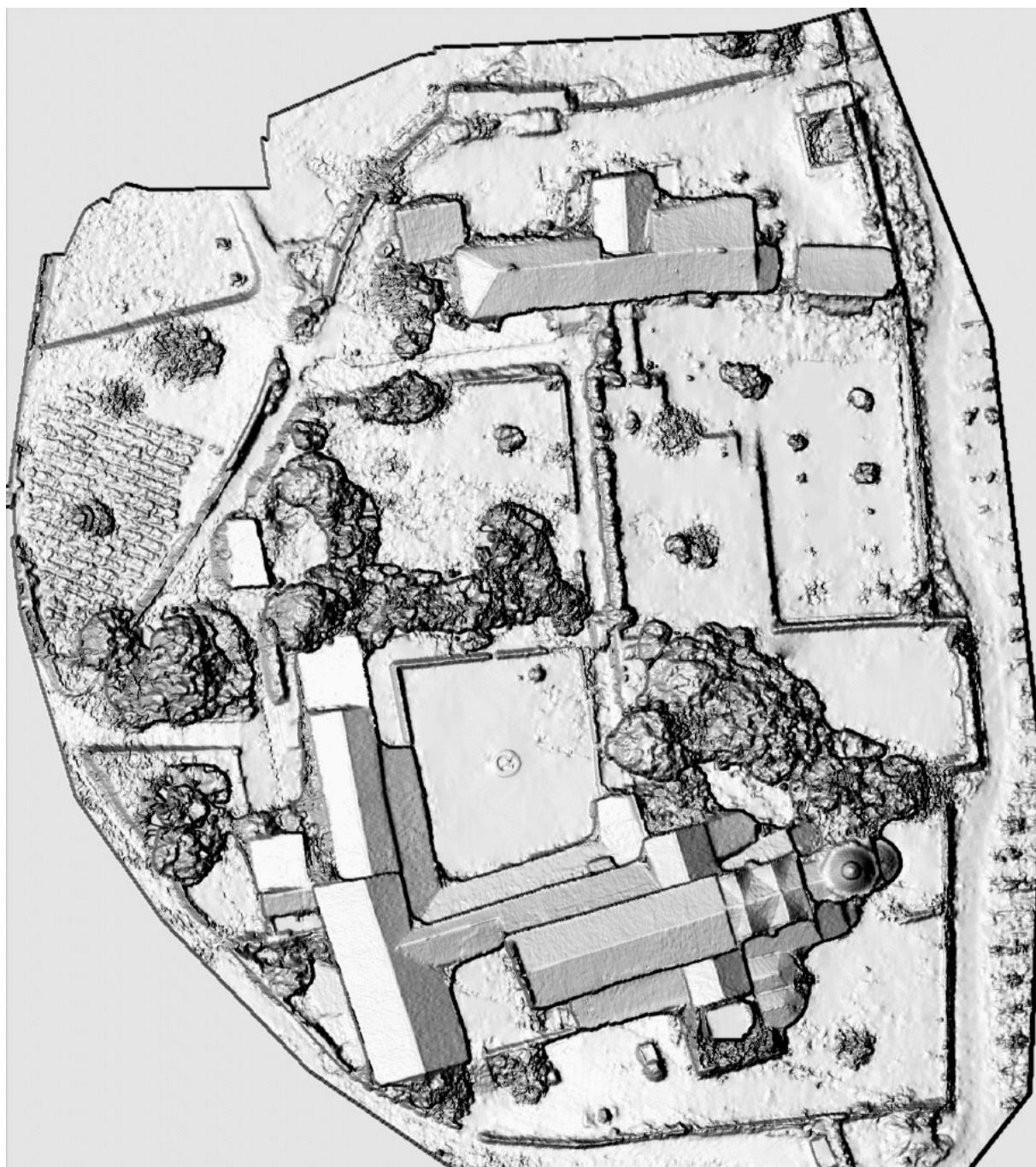


Figure 4.10: Shading in Lambert93 projection, with GrShade

5. Transform image relative orientation into absolute orientation, e.g. performing direct georeferencing, with **CenterBascule**;
6. Improve the aerotriangulated model by adding GPS information in the bundle block adjustment, with **Campari**; It results in the image orientation (*Ori-BL72-Campari*), which is used to perform the image dense matching and subsequently the image orthorectification and mosaicking. The canopy surface is characterized by many abrupt vertical changes, which are difficult to model by image matching. The dense matching is performed in *image geometry* with the *Per Image Matchings* tool **PIMs**. Thus, one depth map is computed for each image. These depth maps are then georeferenced and merged in one single digital surface model covering the entire area. The canopy surface model is then used of orthorectification and individual orthoimages are then mosaicked. The remaining of the workflow is thus as follows:
7. Compute depth map for each image with *Per Image Matching Tools* (**PIMs**);
8. Merge individual depth maps in a global Digital Surface Model and compute orthoimage with **PIMs2Mnt**;
9. Merge individual orthoimages in an orthophotomosaic with **Tawny**.

#### 4.3.2 Computing tie points and absolute orientation

Figure 4.12 illustrates the determination of the orientation for the image block. The final orientation database which is used for the dense matching process and for orthophoto generation is the folder *Ori-BL72-Campari*.

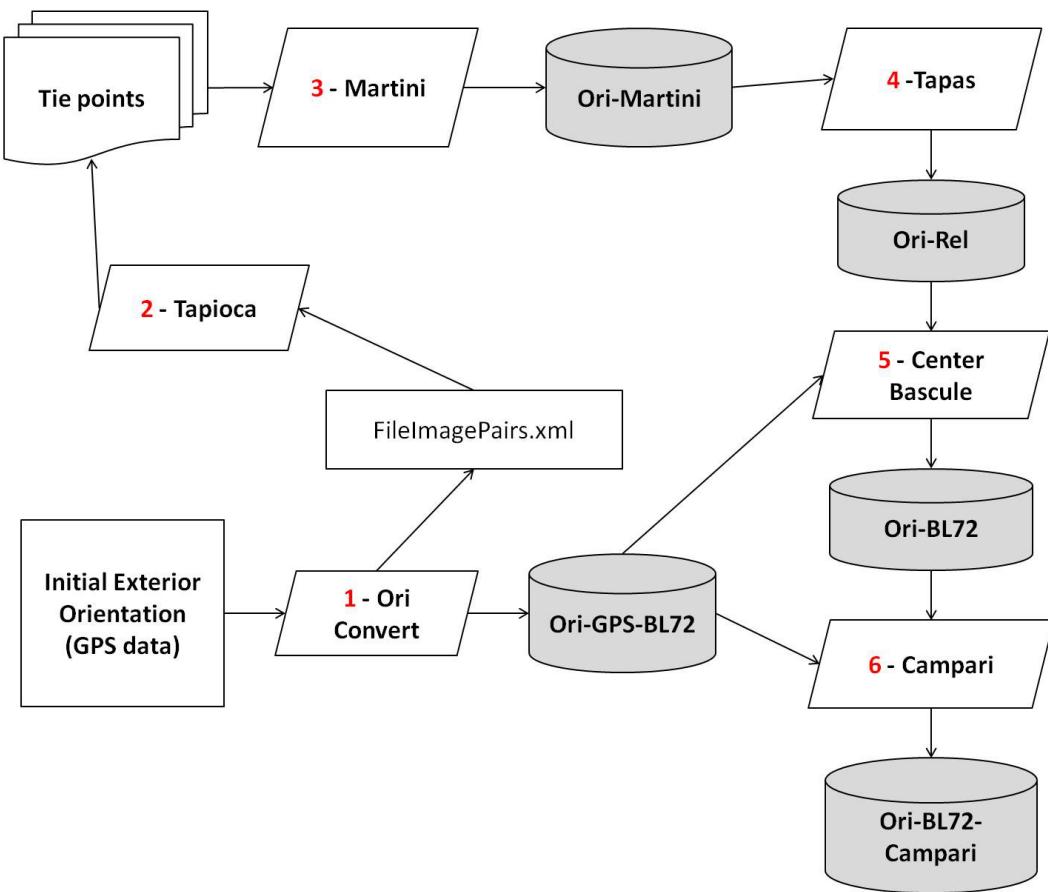


Figure 4.12: The processing chain for computing the image orientation (*Ori-BL72-Campari*). Processing steps are numbered in red.

##### 4.3.2.1 Conversion of GPS data in MicMac format

```
mm3d OriConvert OriTxtInFile GPS_WPK_Grand-Leez.csv GPS-BL72 MTD1=1\
ChSys=DegreeWGS84@SysCoBL72_EPSG31370.xml NameCple=FileImagePairs.xml
```

Note that **MicMac** uses the proj4 library to change the coordinate systems. The Belgian Lambert 72 coordinate system is defined using its "proj4 code" written in an xml file (see *SysCoBL72\_EPSG31370.xml*)

#### 4.3.2.2 Tie points

The file `FileImagePairs.xml` is used for computing tie points with `Tapioca`.

```
Tapioca File "FileImagePairs.xml" -1
```

Tie points are used as observations in the bundle adjustment (`Tapas` and `Campari`) to determine the element of image orientation (external orientation and camera calibration).

#### 4.3.2.3 Relative orientation

Initialization of the orientation for a large image block (hundreds of image) can be carried out with the `Martini` tool:

```
mm3d Martini "R.*.JPG"
AperiCloud "R.*.JPG" Martini Out=Martini-cam.ply WithPoints=0
```

As `Martini` does not account for any radial distortion of the lens, the visual inspection of the image orientation with `AperiCloud` shows large non-linear distortions. Initialization of the image orientation can also be performed successfully directly with `Tapas`, but for a large image block, the use of `Martini` is faster. The complete dataset is then aligned in a relative orientation `Rel` with the following command line:

```
Tapas RadialBasic "R.*.JPG" Out=Rel InOri=Martini
```

#### 4.3.2.4 Georeferencing

The center database `Ori-GPS-BL72` is employed to georeference the aerotriangulated model with `CenterBascule`:

```
CenterBascule "R.*.JPG" Rel GPS-BL72 BL72
```

#### 4.3.2.5 Bundle adjustment with embedded GPS data

Adding GPS information in the bundle adjustment has a positive impact on the refinement of the camera orientation, in particular on the camera calibration.

```
Campari "R.*.JPG" BL72 BL72-Campari EmGPS=[GPS-BL72,2] FocFree=1 PPFree=1
```

### 4.3.3 Dense matching and orthorectification

The digital surface model of the canopy is created with `PIMs` and `PIMs2Mnt`.

```
mm3d PIMs Forest "R00.*.JPG" BL72-Campari ZoomF=2
```

The mode `Forest` of the `PIMs` tool is appropriate for aerial images of forested zones. In this mode, a dense matching is performed independently for every pair of successive images. In the terminal, a message display the pairs that will be used for stereo image matching (in epipolar geometry):

```
Adding the following image pair: R0040571.JPG and R0040570.JPG
Adding the following image pair: R0040572.JPG and R0040571.JPG
Adding the following image pair: R0040573.JPG and R0040572.JPG
...

```

Dense matching is time consuming and generates a lot of intermediate results. Figure 4.13 illustrates the functioning of the *Per Image Matchings* approach.

Stereo depth maps are merged together with `PIM2Mnt`. Subsequently, orthorectification is performed for each image and orthoimages are stored in the directory `PIMs-ORTHO/`.

```
mm3d PIMs2Mnt Forest DoOrtho=1
```

The global digital surface model resulting from the merging of every single depth map is the raster file named `PIMs-TmpBasc/PIMs-Merged_Prof.tif`. It can be visualized and analysed in any GIS software. Eventually, orthoimages are mosaicked together with `Tawny`. Because the radiometry of the different images are quite similar (no important illumination changes during the image acquisition), no radiometric equalization is performed (`RadiomEgal=0`).

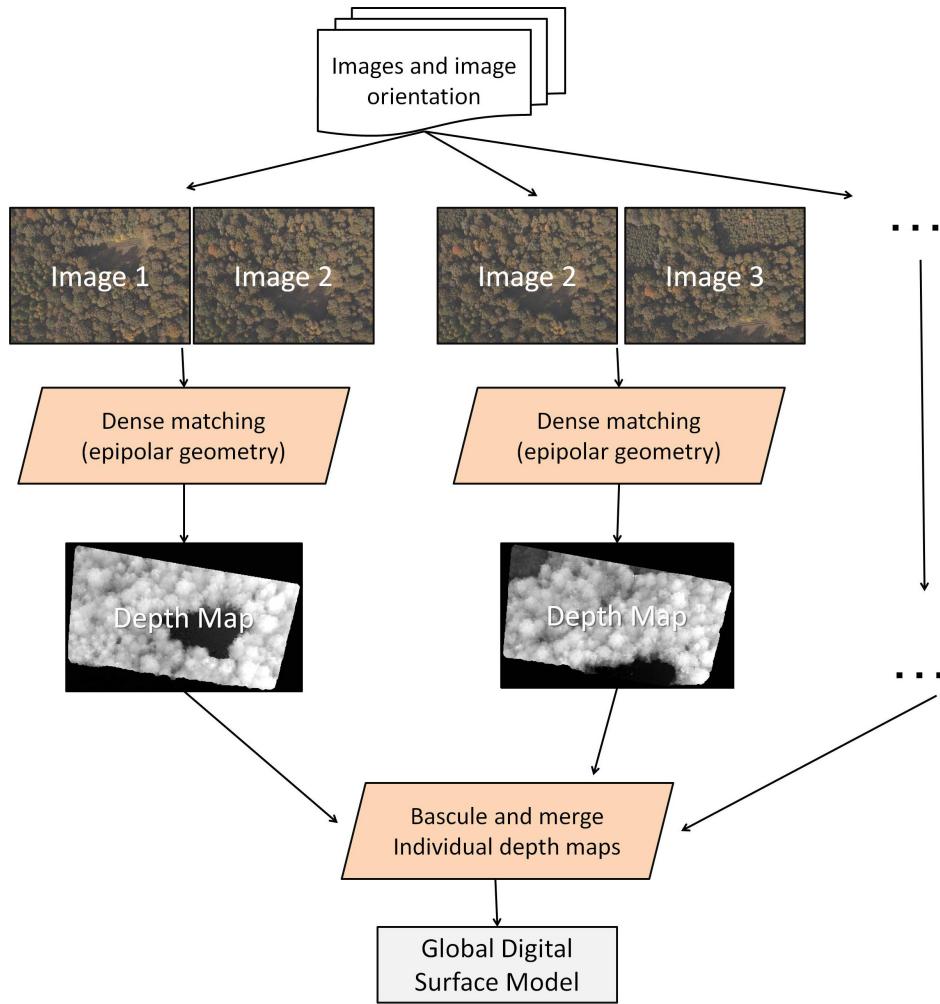


Figure 4.13: Simplified representation of the functioning of the *Per Image Matchings* approach implemented in the **PIMs Forest** tool. Image dense matching is performed for a list of image pairs, resulting in one (or more) depth map per image. These depth maps are georeferenced and merged together with the tools **PIMs2Mnt**.

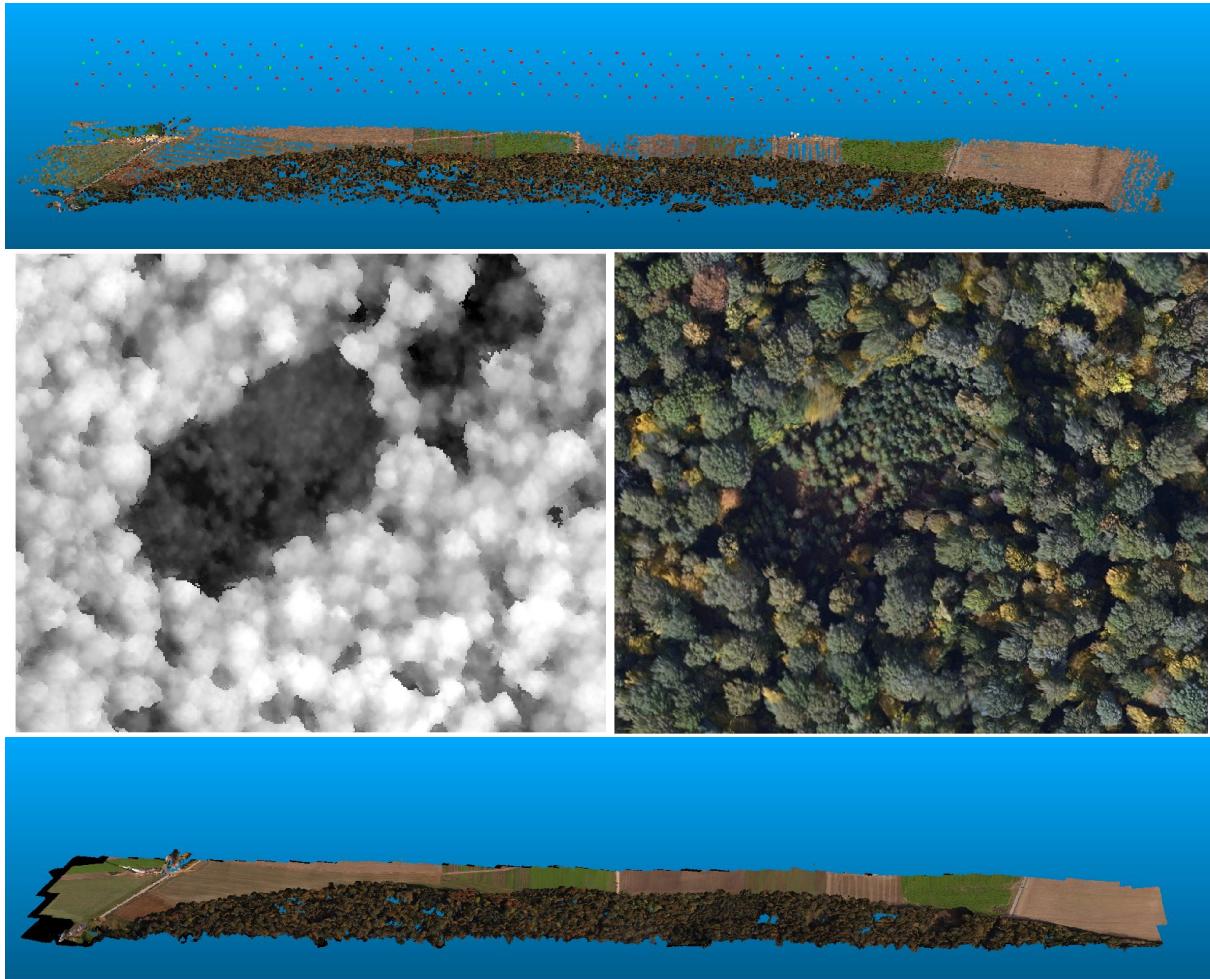


Figure 4.14: Illustration of the different results for the Grand-Leez dataset. Top: the orientation (camera poses and tie points). Middle left: zoom-in on the canopy relief. Middle right: zoom-in on the orthophotomosaic. Bottom: the colored dense 3D point cloud.

```
Tawny PIMs-ORTHO/ RadomEgal=0 Out=Orthophotomosaic.tif
```

The digital surface model and the orthophotomosaic can be combined in a colored 3D point cloud with Nuage2Ply (see figure 4.14).

```
# export the dense point cloud and colorize it with Nuage2Ply:  
Nuage2Ply "PIMs-TmpBasc/PIMs-Merged.xml" Scale=1 /  
Attr="PIMs-Ortho/Orthophotomosaic.tif" RatioAttrCarte=2 Out=CanopySurfaceModel.ply  
  
# Optionally, if meshlab is installed:  
meshlab CanopySurfaceModel.ply
```

## 4.4 GoPro Video data-set

### 4.4.1 Description of the data set

The characteristics of the acquisition are :

- Data is a video LM.mp4;
- This video was acquired with a GoPro camera mounted on a paraglider;



Figure 4.15: First image of video LM.mp4

- The target is a cliff as illustrated on figure 4.15;
  - Part of the images contains sea with flooding wave;
- The issue we have to deal with are the following :
- MicMac can process still images and not video;
  - If we extract all the images, we will have too much redundant data, as can be seen on figure 4.16 with two consecutive images in superposition;
  - The waves generate a lot of tie points (see figure 4.17) , which will be a problem for photogrammetry as they are obviously not motionless relatively to the cliff;
  - Currently with video, a lot of image are blurred (although it's not so much the case here ...);
  - There is no meta data embedded with video (at least, they disappear with the tool used to extract still images);
  - With this camera, there is a rolling shutter, so potentially each images has its own deformation;

#### 4.4.2 The commands

The file `Cmd.txt` in `Documentation/FIGS/GoProVideo` contains the commands that have been used. They are :

```
# Develop all images
ffmpeg -i LM.mp4 Im_0000_%5d_0k.png

# Add missing xif
mm3d SetExif .*png F35=20 F=4.52 Cam=GoProVideoLM

# Select approximatively 3 image / sec , preferring the sharpest one
mm3d DIV Im_0000_.*png Rate=3

# Put the unselected images in basket
mkdir POUB
mv *N1.png POUB/

# Tie points adapted to linear acquisition
Tapioca Line .*png 1000 10

# Compute a initial calibration; would not be necessary if we had already used this camera
mm3d Tapas FishEyeBasic Im_0000_000.*png Out=Calib

# Orient all the images
mm3d Tapas FishEyeBasic Im_0000_.*png InCal=Calib Out=All0
```

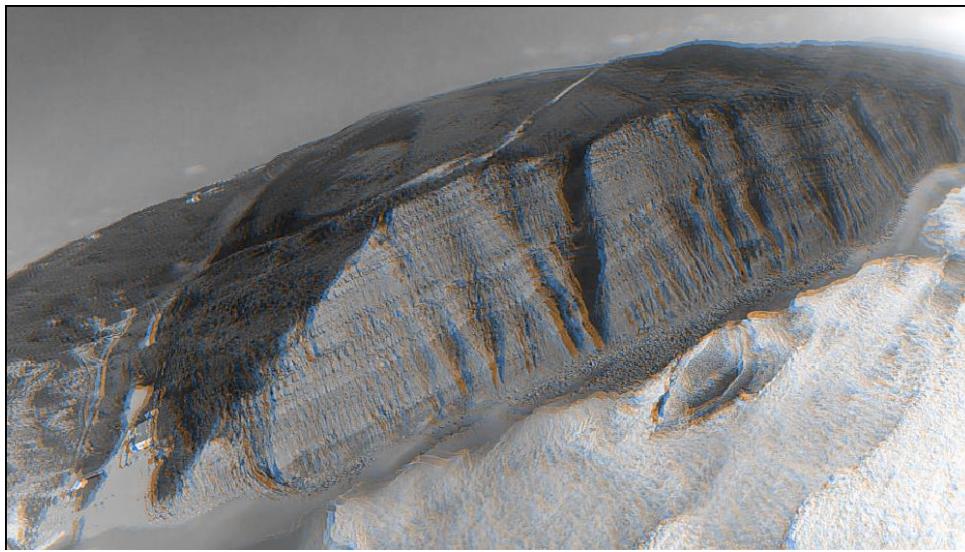


Figure 4.16: Two consecutive images of the video in superposition

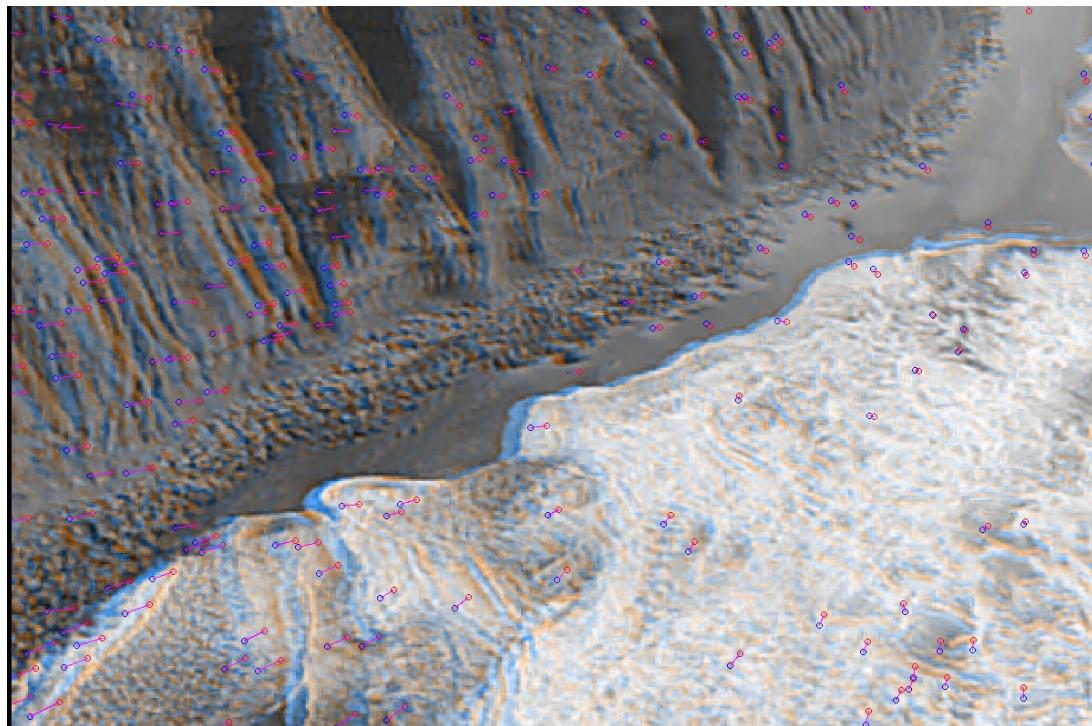


Figure 4.17: Tie points from two extracted images

```

# Generate a ply to visualize the scene
AperiCloud .*png Ori-Al10/

# Input a 2D mask that removes the sea
mm3d SaisieMasqQT AperiCloud_Al10.ply

#Filter the homologous point
mm3d HomolFilterMasq .*png OriMasq3D=Ori-Al10/

# rename homologous points, the filtered one will be seen as the default
mv Homol HomolInit
mv HomolMasqFiltered/ Homol

# Compute orientation without the sea
Tapas FishEyeBasic .*png InOri=Ori-Al10/ Out=Al11

# Free parameters
Campari .*png Al11 Al12 CPI1=1 FocFree=1 PPFree=1 AffineFree=1

# Generate the point cloud
mm3d C3DC BigMac .*png Ori-Al12/ Tuning=0 Masq3D=AperiCloud_Al12.ply ZoomF=1

```

#### 4.4.3 Some comments

##### 4.4.3.1 Developing still images with ffmpeg

The software **ffmpeg** is a free open source package, we use it to extract the still images from video. Note :

- We ask to extract *all* the images, because we want to do *a posteriori* our own selection of non-blurry images;
- To do this selection it is a requirement that the images use **ffmpeg** with the naming `Im_0000_%5d_0k.png` (well the tool is still in very prototype state);

##### 4.4.3.2 Adding missing xif with SetExif

As there is no **exif** information in the data set, we add it to avoid the use of `MicMac-LocalChantierDescripteur.xml`. Note that is important to do it at the very beginning of the process, before using any other **MicMac** tool, because after the xif will memorized in the `Tmp-MM-Dir/.xml` files

##### 4.4.3.3 Selecting sharpest images with DIV

The **DIV** command, makes selection of video images. `mm3d DIV Im_0000_.*png Rate=3` means : select approximately 3 images per second (in fact one image out of 8, assuming an initial rate 24 images per second). As some image have to be deleted, this rate is only an approximation.

If the image is selected, its name is unchanged, while "deleted" images are renamed by replacing `0k` by `N1`. As we don't want to use the deleted images, we put them in a "trash can" with the two lines `mkdir POUB` and `mv *N1.png POUB/`.

##### 4.4.3.4 Standard orientation

The three next line are quite standard **MicMac** processing :

- `Tapioca Line .*png 1000 10`, compute tie point with command adapted to a linear acquisition;
- `mm3d Tapas FishEyeBasic Im_0000_000.*png Out=Calib`, compute a first value of calibration , we use a fish-eye model adapted to this GoPro camera;
- `mm3d Tapas FishEyeBasic Im_0000_.*png InCal=Calib Out=Al10`, orient all the images, starting from the calibration
- `AperiCloud .*png Ori-Al10/` generate a ply file to visualize the scene and position of camera (see 4.19);

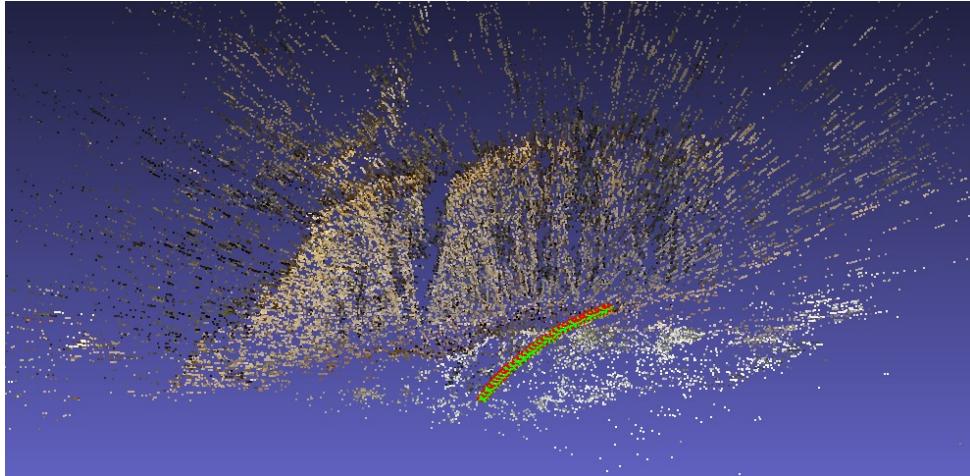


Figure 4.18: Orientation of images

#### 4.4.3.5 Seizing the waves

With such acquisition, the best option to seize the location of the wave, is to seize in 3D. The other alternative, seize them in each images, would be much more time consuming. For this we can use the `SaisieMasqQT` command, see 8.2.2.

#### 4.4.3.6 Filtering homologous points

We can now use the `HomolFilterMasq` command to select the tie points that are inside the 3d masq. We use the `OriMasq3D` option to indicate the orientation (necessary to compute by ray intersection the 3d point associated to each tie point). By default, it assume that the mask has been seized on a `AperiCloud` result, and the default name of the 3d mask is here `AperiCloud_A112.polyg3d.xml`.

We have to rename the homologous folder because by default all the MicMac command search the tie points in the folder `Homol` :

```
mv Homol HomolInit
mv HomolMasqFiltered/ Homol
```

#### 4.4.3.7 Final orientation

Then we have two command to run to have the final orientation :

- `Tapas FishEyeBasic .*png InOri=Ori-A110/ Out=A111`, here we run `Tapas` taking into account the set of tie points without the sea;
- `Campari .*png A111 A112 CPI1=1 FocFree=1 PPFree=1 AffineFree=1`, here we run `Campari` with the option that select one internal calibration by images, we free the 0 and 1 degree parameter to take into account the rolling shutter (is it sufficient ? This is another story ...).

And finally we can use the `C3DC` command to generate a point cloud, a snapshot is presented on figure fig:GoProPlyFin.

```
mm3d C3DC BigMac .*png Ori-A112/ Tuning=0 Masq3D=AperiCloud_A112.ply ZoomF=1
```

## 4.5 The satellite data set

### 4.5.1 Description of the data

A Pleiades tristereo is processed in the following. The images are part of a sample dataset disseminated by the Airbus Defence and Space and can be downloaded from <http://www.geo-airbusds.com/en/23-sample-imagery>

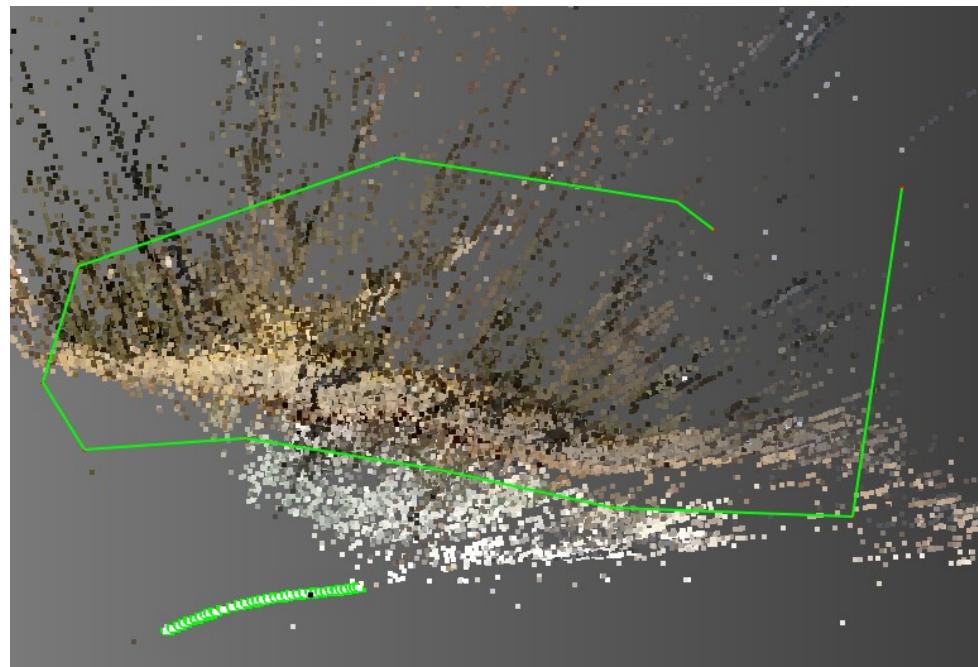


Figure 4.19: Seizing 3D masq of the cliff

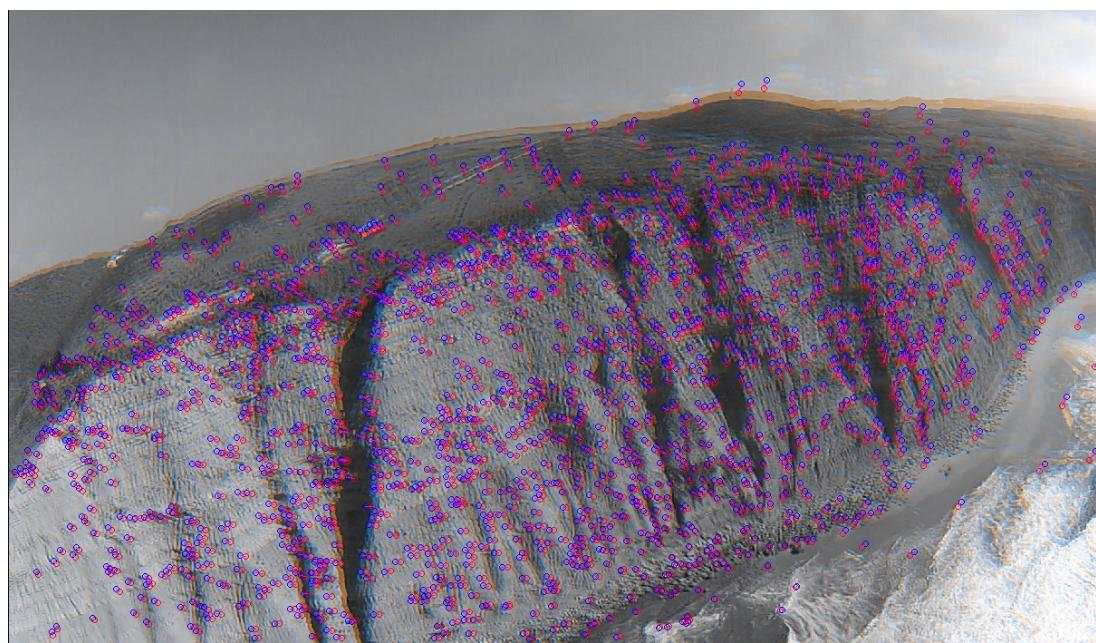


Figure 4.20: Tie points from two extracted images after selection by 3d masq

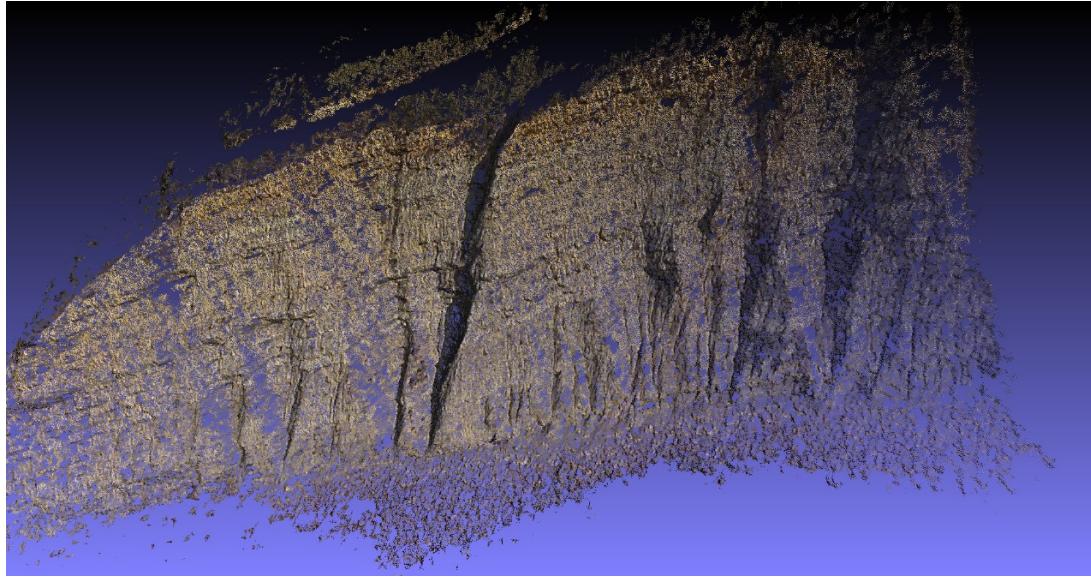


Figure 4.21: Tie points from two extracted images after selection by 3d masq

#### 4.5.2 From 2D images to 3D objects – the processing commands

Unless you work with MicMac and the Kakadu license, the original JPEG2000 images must be converted to tiff. The otb library<sup>5</sup> allows for the conversion using the command below:

```
otbcli_Convert -in image.jp2 -out image.tif uint16
```

The tie points can now be extracted from the images:

```
mm3d Tapioca All *.tif 10000
```

Next, as mentioned in Section 19.1, the input files with rational polynomial coefficients ought to be converted to a MicMac readable format, and the processing coordinate system defined:

```
mm3d Convert2GenBundle "IMG_PHR1A_P_20120225002(.*)_SEN_PRG_FC_51(.*)-001_R1C1.tif"
"RPC_PHR1A_P_20120225002\$1_SEN_PRG_FC_51\$2-001.XML" RPC-deg1 ChSys=WGS84toUTM.xml Degre=1
```

which is equivalent of independently running the same command for three images:

```
mm3d Convert2GenBundle IMG_PHR1A_P_201202250025329_SEN_PRG_FC_5110-001_R1C1.tif
RPC_PHR1A_P_201202250025329_SEN_PRG_FC_5110-001.XML RPC-deg1 ChSys=WGS84toUTM.xml Degre=1
mm3d Convert2GenBundle IMG_PHR1A_P_201202250025599_SEN_PRG_FC_5108-001_R1C1.tif
RPC_PHR1A_P_201202250025599_SEN_PRG_FC_5108-001.XML RPC-deg1 ChSys=WGS84toUTM.xml Degre=1
mm3d Convert2GenBundle IMG_PHR1A_P_201202250026276_SEN_PRG_FC_5109-001_R1C1.tif
RPC_PHR1A_P_201202250026276_SEN_PRG_FC_5109-001.XML RPC-deg1 ChSys=WGS84toUTM.xml Degre=1
```

Generally you one would prefer to use the regular expression rather than run repeatedly the same command for each image as the latter is very error-prone. The content of the coordinate system file renders:

```
<SystemeCoord>
  <BSC>
    <TypeCoord>eTC_Proj4</TypeCoord>
    <AuxStr>+proj=utm +zone=55 +south +ellps=WGS84 +datum=WGS84 +units=m +no_defs</AuxStr>
  </BSC>
</SystemeCoord>
```

Given the extracted tie points, the RPC bundle adjustment can proceed with the simplified tool **Campari** (see Subsection 3.9.2):

```
mm3d Campari *.tif RPC-deg1 RPC-deg1_adj
```

---

5. <https://www.orfeo-toolbox.org/>

Refer to section 3.9.2.2 for a more detailed description of the adjustment algorithm. Provided the results deliver satisfying residuals (in the presented case reflecting only the reprojection errors of the tie points, but more generally also determining the adherence of the data to some control information), the dense matching can be carried out. Nevertheless, it is worthwhile to verify the refined orientation between pairs of images using the `mm3d MMTTestOrient` (see Subsection 12.1.1):

```
mm3d MMTTestOrient IMG_PHR1A_P_201202250025329_SEN_PRG_FC_5110-001_R1C1.tif
IMG_PHR1A_P_201202250025599_SEN_PRG_FC_5108-001_R1C1.tif Ori-RPC-deg1_adj GB=1 ZMoy=0 ZInc=500
```

In case one would want to use the adjusted orientation in some external software, it is possible to recompute the RPCs with the command:

```
mm3d SateLib RecalRPC Ori-RPC-deg1_adj/
GB-Orientation-IMG_PHR1A_P_201202250025329_SEN_PRG_FC_5110-001_R1C1.tif.xml
```

The DSM generation is handled, again, by the simplified tool `Malt` (see Subsection 3.12.2):

```
mm3d Malt UrbanMNE .*.*tif Ori-RPC-deg1_adj ZMoy=0 ZInc=500
```

#### 4.5.3 Understanding the bundle adjustment output (Campari)

The adjustment result is stored inside the `Ori-RPC-deg1_adj` directory. Understanding the bundle adjustment message printed to the screen (and additionally stored inside a `Residus.xml` file) is already explained in subsection 6.2.7. The output directory contains files with the original RPCs (all files with the prefix `UncorExtern-`), and the corresponding files with adjusted orientation parameters (all files with the prefix `GB-`). The `GB-` files contain:

- `NameCamSsCor`, the filepath to the original RPCs;
- `NameIma`, the name of the image that the file corresponds to;
- `SysCible`, the definition of the coordinate system used in the processing (proj4 format);
- `DegreTot`, the degree of the adjustable 2D polynomial correction function;
- `Center`, the polynomial's normalizing shift (in pixels);
- `Ampl`, the polynomial's normalizing amplitude;
- `CorX`, the polynomial's normalized x-coefficients;
- `CorY`, the polynomial's normalized y-coefficients;
- `Monomes`, three values corresponding to respective polynomial terms (e.g. `<Monomes>-0.94 0 1</Monomes>` interprets as  $-0.94 \cdot x^0 \cdot y^1$ ).

To avoid numerical instabilities of the polynomial functions, the `Center` and `Ampl` parameters normalize the image space such that all observations are contained within the range  $<-1, 1>$ . The user can visualize the correction polynomial functions with

```
mm3d SateLib SATD2D Ori-RPC-deg1_adj/GB-Orientation-
IMG_PHR1A_P_201202250025329_SEN_PRG_FC_5110-001_R1C1.tif.xml
```

The tool produces images of displacements separately in x, y and combined xy directions (see Fig. 4.22), and prints to the screen the minimum/maximum values in pixels:

```
displacement in x: GMin,Gax -0.96187676315564 -0.919344454426481
displacement in y: GMin,Gay -0.542404322159884 0.234261700467604
displacement in xy: GMin,Gax 0.919404732429707 1.06767206091774
```

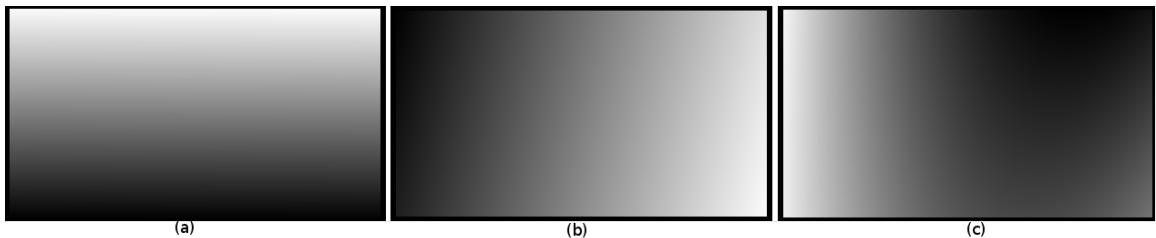


Figure 4.22: Displacements in image space caused by the correcting polynomial functions in image `IMG_PHR1A_P_201202250025329_SEN_PRG_FC_5110-001_R1C1.tif`. Displacement magnitude (a) along the x-coordinate, (b) along the y-coordinate, (c) combined along both coordinate directions.

#### 4.5.4 Understanding the bundle adjustment validation output (`MMTestOrient`)

The output of the `MMTestOrient` tool described in Section 12.1 is shown in Fig. 4.23. Using the tool `mm3d StatIm`, some basic image statistics can be obtained allowing the interpretation of the outcome:

```
mm3d StatIm GeoI-Px/Px2_Num16_DeZoom2_Geom-Im.tif [1000,1000] Sz=[8000,4000]
```

The command above calculates the mean, the standard deviation, as well as min/max parallax values over the bounding box anchored at [1000,1000], of size [8000,4000]. Because the input parallax image is at the `DeZoom=2`, rather than the full resolution, all values must be multiplied by two. The image statistics over the selected bounding box in pixels are then:

```
ZMoy=0.064 ; Sigma=0.122
ZMinMax=[-2.70 , 2.03]
```

The two most relevant statistics are: the mean transverse parallax which is very close to zero, and the sigma equal to  $\approx 0.1$  pixel. The min/max values are relief-related and occur in places of low correlation, e.g. on vegetation, water surfaces, or in places of shadows and occlusions. The magnitude of systematic pattern visible in Fig. 4.23 remains at the level of the sigma value, hence well below the adjustment precision ( 0.6 pixel). The user is encouraged to use `mm3d Vino` tool to display very big image files.

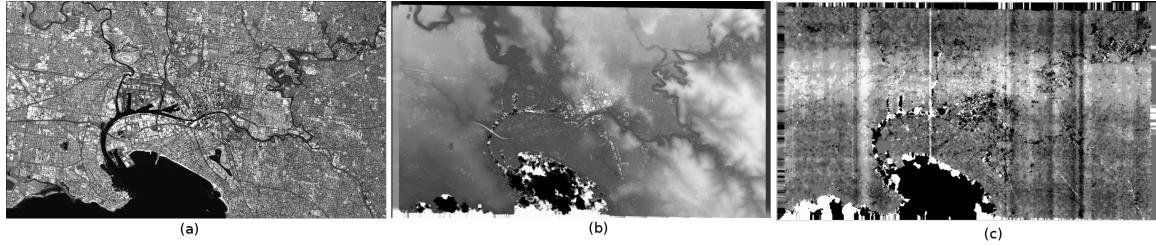


Figure 4.23: (a) A satellite image, (b) the epipolar parallax (`Px1_Num16_DeZoom2_Geom-Im.tif`), (c) the transverse parallax (`Px2_Num16_DeZoom2_Geom-Im.tif`).

## 4.6 The Viabon dataset

The directory `Viabon/` in `/micmac_data/ExempleDoc/` contains a set of data which will allow us to perform a direct-georeferencing<sup>6</sup> of images based on embedded GPS data. Below we will detail all the necessary steps to achieve maximum ground accuracy, here, in the range of 1-2 cm. First, we will compute different GPS trajectories in order to make comparisons and in a second time we will be interested in the fusion of these results with the photogrammetric processing part.

This UAS acquisition has been performed by the surveying service of **Vinci-Construction-Terrassement**<sup>7</sup>. A **DJI-F550**<sup>8</sup> hexa-copter has been used to achieve the flight. The images were acquired by the **IGN** panchromatic light camera developed at the **LOEMI**<sup>9</sup> laboratory. The GPS on-board raw measurements were acquired by the **GeoCube**, a multi-sensor geo-monitoring system developed at the same laboratory.

The file `Viabon/Pipeline-Viabon.txt` contains all command lines related to this work-flow. The data in `Viabon/` directory consist of:

- 73 nadir images in `.tif` raw format
- `15073106.obs` rinex file of rover receiver
- `00012120.150` rinex file of pivot/base station
- `ct19212z.15o` rinex file of closest RGP<sup>10</sup> network station
- `ct19212z.15n` GPS satellites navigation file

6. To be more precise, this dataset deals with Integrated Sensor Orientation (ISO) because no inertial measurements are available

7. <http://www.vinci-construction-terrassement.com/>

8. [http://dl.djicdn.com/downloads/flamewheel/en/F550\\_User\\_Manual\\_v2.0\\_en.pdf](http://dl.djicdn.com/downloads/flamewheel/en/F550_User_Manual_v2.0_en.pdf)

9. Opto-Electronics, Instrumentation and Metrology Laboratory

10. GNSS Permanent Network: <http://rgp.ign.fr/>

- `ct19212z.15g` Glonass satellites navigation file
- `igs08.atx` satellites and receiver antennas calibration values
- `CpleImg.xml` images couples for tie points computation
- `Pts_GeoC.txt` ground points coordinates
- `MesImages.xml` image measurements of ground points
- `SysCoRTL.xml` file for system coordinates transformation



Figure 4.24: The Viabon dataset

#### 4.6.1 Processing GPS data

For GPS data processing we will use `RTKLIB`<sup>11</sup> an open-source software. `RTKLIB` consists of several modules (`convbin`, `rnx2rtkp`, `rtkrcv`, ...etc). We will only use `rnx2rtkp` which will allow us to do post-processing of our data based on different positioning modes.

##### 4.6.1.1 Compile MicMac with RTKlib

To use `rnx2rtkp`, if compiling `MicMac` from sources, run `cmake` with `-DBUILD_RNX2RTKP=ON` option activated as follows:

```
cmake .. / -DBUILD_RNX2RTKP=ON
```

##### 4.6.1.2 Base station processing

First we use `GpsProc` command to compute the position of the base station which will be used to process the UAV trajectory. The data recorded by the on-board GPS receiver are single-frequency data. This implies, for optimal accuracy, having a base station within a radius of  $\sim 10$  km. A pivot station has been installed near the acquisition area (file `00012120.150`). This station is a multi-constellation dual-frequency `Novatel GNSS` receiver. The position of this pivot station is estimated relatively to a reference station of the French permanent `GNSS` network (file `ct19212z.150`).

First we estimate the position of the pivot station:

```
mm3d TestLib GpsProc './' static 00012120.150 ct19212z.15o ct19212z.15n NavSys=5
GloNavFile=ct19212z.15g Freq=11_12 AntFileRCV=igs08.atx AntFileSATs=igs08.atx
AntBType=TRM55971.00
```

First mandatory argument is the current directory. Second mandatory argument is the processing mode. Here we tell the software that we want to estimate a position of `static` measurements. Third mandatory argument

---

11. <http://www.rtklib.com/>

is rinex file of known station, here **CT19**<sup>12</sup> of French Permanent GNSS Network. Last mandatory argument contains GPS constellation satellites navigation parameters. For optional arguments, **NavSys=5** means that we use both constellations **GPS** and **Glonass** with respect to **RTKlib** conventions<sup>13</sup>. In this case, we need then to give the navigation file for Glonass constellation too using the optional argument **GloNavFile=ct19212z.15g**. Once both receivers are dual-frequency receivers, we perform the processing in both frequencies, this is specified with the optional argument **Freq=l1\_l2**. Finally, we use optional arguments to perform antennas corrections, and we specify the antenna model for **CT19** as the antenna is listed in **igs08.atx** file. Here, 3 files are created:

```
— ./rtkParamsConfigs.txt a summary of the options used by rnx2rtkp
— ./Output_static.txt the result in RTKLIB format
— ./Output_static.xml the result in an XML format for MicMac internal using
```

#### 4.6.1.3 UAV trajectories processing

All UAVs board at least a single-frequency GPS chip<sup>14</sup> that receives the L1<sup>15</sup> C/A<sup>16</sup> code signal. First, we process a trajectory based solely on this data in order to evaluate its accuracy in the case our system delivers only available code positions:

```
mm3d TestLib GpsProc './' single 15073106.obs NONE ct19212z.15n
```

Here the second mandatory argument, corresponding to positioning mode, value is **single**. This means that we process only rover code measurements. Next mandatory argument is the rinex file of raw measurements of rover receiver (**15073106.obs**). Any differential processing is done here, we give the value **NONE**. Last mandatory argument correspond to GPS constellation satellites navigations parameters. The trajectory is saved in **RTKlib** and **MicMac** respectively in file **Output\_single.txt** and **Output\_single.xml**.

Assume that the GPS module of the UAV autopilot allow us to record L1 C/A code raw data. Then, it is possible to process a trajectory in differential mode based on code data (the same data used for navigation of the UAV) to improve the accuracy of the estimated trajectory:

```
mm3d TestLib GpsProc './' dgps 15073106.obs 00012120.150 ct19212z.15n
    AntBPosType=XYZ StaPosFile=Output_static.xml
```

The positioning mode is **dgps**. Here we give as input file rinex raw measurements of pivot station (**00012120.150**). As the position has been estimated in 4.6.1.2, optional arguments **StaPosFile=Output\_static.xml** is used to give reference position of the pivot and **AntBPosType=XYZ** specifies the format of the given position.

The GPS chip embedded in the **GeoCube** is a **u-blox LEA-6T-0-001**<sup>17</sup> model. This GPS module allows recording carrier-phase raw data. As noise measurement on carrier-phase is much less important than on code measurements, we perform a differential processing based on raw phase data:

```
mm3d TestLib GpsProc './' kinematic 15073106.obs 00012120.150 ct19212z.15n
    AntBPosType=XYZ StaPosFile=Output_static.xml
```

The positioning mode value is **kinematic**. As for previous command, we give reference position of pivot station using optional arguments. Carrier-phase trajectory is stored in the files **Output\_kinematic.xml** and **Output\_kinematic.txt**.

#### 4.6.2 Computing tie points

**CpleImg.xml** file is used with the **mm3d Tapioca** command to accelerate tie points extraction. This file contains all pairs of overlapping images. We perform tie points extraction based on images sub-sampled by a factor of 3:

```
mm3d Tapioca File CpleImg.xml 1300
```

Visualize tie points between 2 images using **mm3d SEL** command:

- 
- 12. <http://rgp.ign.fr/STATIONS/#CT19>
  - 13. [http://www.rtklib.com/rtklib\\_document.htm](http://www.rtklib.com/rtklib_document.htm)
  - 14. For example u-blox NEO-7N
  - 15. 1575.42 MHz
  - 16. Coarse/Acquisition
  - 17. <https://www.u-blox.com/en/product/neolea-6t>

```
mm3d SEL './' image_002_00069.tif image_002_00070.tif KH=NB
```

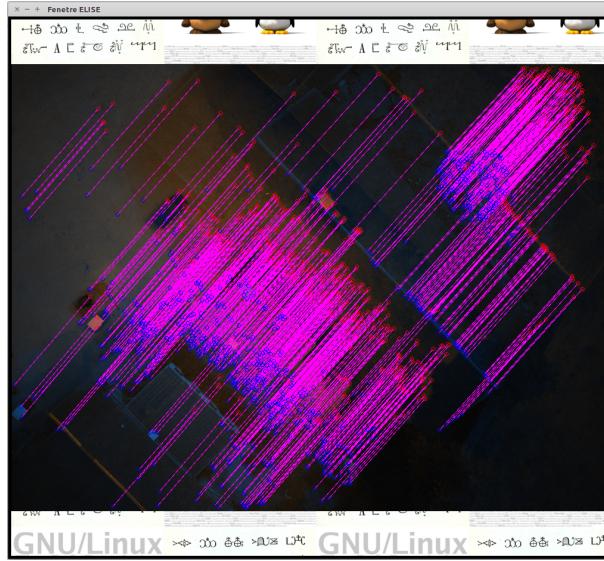


Figure 4.25: Tie points visualization

#### 4.6.2.1 Computing exterior orientation

To speed up the relative orientation computation, the images bloc is initialized using `mm3d Martini` command used for large bloc. `mm3d AperiCloud` command is used to export images estimated exterior parameters in a `.ply` file format and one can visualize it using the free and open-source software `meshlab`:

```
mm3d Martini "image_002_00*.*tif"
mm3d AperiCloud "image_002_00*.*tif" Martini
meshlab AperiCloud_Martini.ply
```

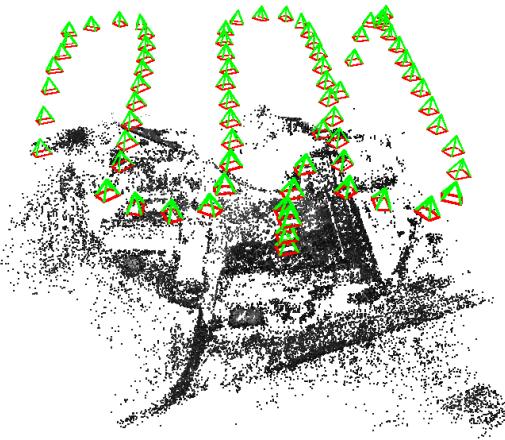


Figure 4.26: Bloc visualization

The command `mm3d Tapas` is used to perform relative orientation based on the bloc initialization computed before and we use the `RadialStd` camera model which has 8 degrees of freedom:

```
mm3d Tapas RadialStd "image_002_00*.*tif" InOri=Martini Out=All-RS
mm3d AperiCloud "image_002_00*.*tif" All-RS
meshlab AperiCloud_All-RS.ply
```

#### 4.6.2.2 GPS positions & Camera centers matching

Time synchronization of sensors (camera & GPS) was conducted in the laboratory. The electronic delay is negligible ( $\sim 0.5$  ms) and the GPS receiver is in charge of triggering the camera. This means that image centers are aligned with GPS positions. However, sampling of both trajectories is different, the camera does not have an internal clock (no time information in `exif` meta-data) and camera triggering is not dated in the GPS time-scale. The identification of corresponding positions is done by computing the best correlation score of distances ratios curves and by testing all possible time shifts using `mm3d TestLib MatchCenters`:

```
mm3d TestLib MatchCenters './ Ori-All-RS/ Output_single.xml "image_002_00*.tif"
mm3d TestLib MatchCenters './ Ori-All-RS/ Output_dgps.xml "image_002_00*.tif"
mm3d TestLib MatchCenters './ Ori-All-RS/ Output_kinematic.xml "image_002_00*.tif"
```

The output files (`Ori-Output_single.txt`, ...) gives for each image the corresponding GPS position. Then, the `mm3d OriConvert` command is used to convert the resulting file into `Ori-XXX/ .xml` format orientations folder and performing at the same time a coordinate system transformation using optional argument `ChSys`:

```
mm3d OriConvert "#F=N_X_Y_Z" Ori-Output_single.txt Nav-Code ChSys=GeoC@SysCoRTL.xml
mm3d OriConvert "#F=N_X_Y_Z" Ori-Output_dgps.txt Nav-DCode ChSys=GeoC@SysCoRTL.xml
mm3d OriConvert "#F=N_X_Y_Z" Ori-Output_kinematic.txt Nav-DPhase ChSys=GeoC@SysCoRTL.xml
```

At this step we performed different GPS trajectories calculations. We also have a set of images positions/orientations computed based on bundle block adjustment using only tie points. Also, we have for each GPS trajectory solution (absolute code, differential code and differential carrier-phase) correspondences between GPS positions and images centers that will allow us to convert the relative external orientation into absolute one.

#### 4.6.2.3 Images georeferencing

Let's first convert Ground Control Points file from `.txt` to `.xml` MicMac format using the `mm3d GCPConvert` command and performing at the same time a coordinate system transformation using the optional argument `ChSys`:

```
mm3d GCPConvert "#F=N_X_Y_Z_Ix_Iy_Iz" Pts_GeoC.txt Out=AllPts-RTL.xml
ChSys=GeoC@SysCoRTL.xml
```

Here we start by comparing raw similarity transformations using different GPS processing results performed above. First for absolute navigation using code measurements. The command `mm3d CenterBascule` is used to generate absolute orientations, here stored in the folder `Ori-Bascule-RS-Code/`. Then the command `mm3d GCPCtrl` is used to control accuracy on the ground by computing residuals on all available points here all considered as check points:

```
mm3d CenterBascule "image_002_00*.tif" Ori-All-RS/ Ori-Nav-Code/ Bascule-RS-Code
mm3d GCPCtrl "image_002_00*.tif" Ori-Bascule-RS-Code/ AllPts-RTL.xml MesImages.xml

===== ERROR MAX PTS FL =====
|| Value=140.12 for Cam=image_002_00100.tif and Pt=7 ; MoyErr=130.967
=====
== GCP STAT == Dist, Moy=1.52601 Max=1.65398
```

Here for differential code trajectory processed:

```
mm3d CenterBascule "image_002_00*.tif" Ori-All-RS/ Ori-Nav-DCode/ Bascule-RS-DCode
mm3d GCPCtrl "image_002_00*.tif" Ori-Bascule-RS-DCode/ AllPts-RTL.xml MesImages.xml

===== ERROR MAX PTS FL =====
|| Value=71.3855 for Cam=image_002_00058.tif and Pt=1 ; MoyErr=53.4976
=====
== GCP STAT == Dist, Moy=0.640848 Max=0.844784
```

Here for differential carrier-phase trajectory processed:

```

mm3d CenterBascule "image_002_00*.tif" Ori-All-RS/ Ori-Nav-Dphase/ Bascule-RS-DPhase
mm3d GCPCtrl "image_002_00*.tif" Ori-Bascule-RS-DPhase/ AllPts-RTL.xml MesImages.xml

===== ERROR MAX PTS FL =====
|| Value=40.8538 for Cam=image_002_00076.tif and Pt=4 ; MoyErr=21.3675
=====
*** GCP STAT === Dist, Moy=0.32083 Max=0.470131

```

We notice after the three georeferencing computations that the reprojection error is improved by a factor of  $\sim 2.5$  (for this dataset) when code measurements are used in differential mode (`dgps`). The best georeferencing results are obtained using the calculated trajectory based on differential carrier-phase measurements (`MoyErr`  $\sim 21$  px).

#### 4.6.2.4 Bundle Bloc Adjustment with GPS observations and lever-arm offset

Here we perform, with the `mm3d Campari` command, a heterogeneous compensation using tie points and GPS camera positions estimated using GPS observations. In addition, here we take into account the fact that camera optical center and GPS antenna phase center are separated by a vector called the lever-arm offset using the optional argument `GpsLa`.

For bundle block adjustment using C/A code positions, planimetric uncertainty is fixed to 3 m and vertical component uncertainty is fixed to 5 m in the optional argument `EmGPS`:

```

mm3d Campari "image_002_00*.tif" Ori-Bascule-RS-Code/ Compense-RS-Code-La
EmGPS=[Ori-Nav-Code/,3,5] GpsLa=[0,0,0]

mm3d GCPCtrl "image_002_00*.tif" Ori-Compense-RS-Code-La/ AllPts-RTL.xml MesImages.xml

LA: [0.263412,0.034836,-2.03753]

===== ERROR MAX PTS FL =====
|| Value=212.782 for Cam=image_002_00110.tif and Pt=10 ; MoyErr=195.637
=====
*** GCP STAT === Dist, Moy=2.89733 Max=2.96082

```

Here using as embedded GPS trajectory the one computed based on differential code measurements, where planimetric uncertainty is fixed to 0.8 m and vertical component uncertainty is fixed to 1 m :

```

mm3d Campari "image_002_00*.tif" Ori-Bascule-RS-DCode/ Compense-RS-DCode-La
EmGPS=[Ori-Nav-DCode/,0.8,1] GpsLa=[0,0,0]

mm3d GCPCtrl "image_002_00*.tif" Ori-Compense-RS-DCode-La/ AllPts-RTL.xml MesImages.xml

LA: [0.254068,0.106455,-2.40184]

===== ERROR MAX PTS FL =====
|| Value=82.723 for Cam=image_002_00093.tif and Pt=9 ; MoyErr=72.9078
=====
*** GCP STAT === Dist, Moy=1.29547 Max=1.43259

```

Here using differential carrier-phase measurements estimated GPS trajectory where planimetric uncertainty is fixed to 1.5 cm and vertical component uncertainty is fixed to 2.5 cm:

```

mm3d Campari "image_002_00*.tif" Ori-Bascule-RS-DPhase/ Compense-RS-DPhase-La
EmGPS=[Ori-Nav-DPhase/,0.015,0.025] GpsLa=[0,0,0]

mm3d GCPCtrl "image_002_00*.tif" Ori-Compense-RS-DPhase-La/ AllPts-RTL.xml MesImages.xml

LA: [0.10641,-0.0544187,-0.462974]

===== ERROR MAX PTS FL =====
|| Value=18.6601 for Cam=image_002_00074.tif and Pt=3 ; MoyErr=12.5342
=====
*** GCP STAT === Dist, Moy=0.282267 Max=0.309341

```

We note here that the bundle block adjustment using all available observations improves the accuracy for the last GPS trajectory (the most accurate computed on carrier-phase measurements) while for trajectories based on code measurements residuals on check points are more important compared to the results of a similarity estimation without any compensation 4.6.2.3. This is due to the fact that the high uncertainty on code estimated trajectories strongly impacts the estimation of lever-arm offset during the compensation.

#### 4.6.2.5 Advanced internal camera model

Here we use a high degree distortion polynomial function. While the `RadialStd` model used above contains 3 polynomial coefficients ( $r^3, r^5, r^7$ ), the `Four15x2` contains 7 polynomial coefficients ( $r^3, \dots, r^{15}$ ). The optional argument `DegRadMax=3` means that we stop at the third polynomial coefficient. The optional argument `DegGen=0` means that we are not taking into account XY systematism for now. This strategy is used (several steps) to initialize internal calibration. From this section we will only use the results of the calculated GPS trajectory based on carrier-phase measurement in order not to overload the tutorial.

```

mm3d Tapas Four15x2 "image_002_00.*.tif" DegGen=0 DegRadMax=3 Out=Calib-Four
mm3d CenterBascule "image_002_00.*.tif" Ori-Calib-Four/ Ori-Nav-DPhase/ Bascule-CF-DPhase
mm3d GCPCtrl "image_002_00.*.tif" Ori-Bascule-CF-DPhase/ AllPts-RTL.xml MesImages.xml

```

===== ERRROR MAX PTS FL =====

|| Value=19.4863 for Cam=image\_002\_00076.tif and Pt=4 ; MoyErr=9.30796

==== GCP STAT === Dist, Moy=0.11394 Max=0.221325

```
EmGPS=[Ori-Nav-DPhase/,0.015,0.025] GpsLa=[0,0,0]
```

```
mm3d GCPCtrl "image_002_00*.tif" Or
```

LA: [-0.0959945, -0.0505342, -0.360517]

EBERBACH, MAIL, BTG, EU

Value=11.8484 for Cam=image\_002\_00074.tif

= GCP STAT === Dist, Moy=0.209958 Max

Here we add general parameters of degree 2.

```
mm3d Tapas Four15x2 "image_002_00*.*tif" InOri=Calib-Four DegGen=2 Out=All-F15  
mm3d CenterBascule "image_002_00*.*tif" Ori=All-F15/ Ori=Nav-DPhase/ Bascule=AF15-DPhase
```

Digitized by srujanika@gmail.com

===== ERROR MAX PTS FL =====

mm3d Campari "image\_002\_00\*.\*tif" Ori-Bascule-AF15-DPhase/ Compense-AF15-DPhase-La

```
mm3d GCPCtrl "image_002_00*.tif" Ori-Compense-AF15-DPhase-La/ AllPts-RTL.xml
```

LA: [0.0949466,-0.

LA: [0.0949466, -0.0561756, -0.261616]

LA: [0.0949466,-0.0561756,-0.261616]

===== ERROR

-----

= GCP STAT === Dist, Moy=0.163896 Max=0.178663

Here we add a general polynomial model. Only additional distortion is estimated to avoid over parametrized problems.

```
mm3d Tapas AddPolyDeg7 "image_002_00*.tif" InOri=All-F15 Out=All-F15-AddP7
mm3d CenterBascule "image_002_00*.tif" Ori=All-F15-AddP7/ Ori-Nav-DPhase/
    Bascule-AF15P7-DPhase

mm3d GCPCtrl "image_002_00*.tif" Ori-Bascule-AF15P7-DPhase/ AllPts-RTL.xml MesImages.xml

=====
|| Value=7.1342 for Cam=image_002_00081.tif and Pt=4 ; MoyErr=5.69632
=====
== GCP STAT == Dist, Moy=0.0932433 Max=0.106558

%%%%%%%%%%%%%%%
mm3d Campari "image_002_00*.tif" Ori-Bascule-AF15P7-DPhase/ Compense-AF15P7-DPhase-La
    EmGPS=[Ori-Nav-DPhase/,0.015,0.025] GpsLa=[0,0,0]

mm3d GCPCtrl "image_002_00*.tif" Ori-Compense-AF15P7-DPhase-La/ AllPts-RTL.xml
    MesImages.xml

LA: [0.094229,-0.0559327,-0.239925]

=====
|| Value=8.45862 for Cam=image_002_00071.tif and Pt=6 ; MoyErr=6.2671
=====
== GCP STAT == Dist, Moy=0.138502 Max=0.153681
```

We perform the same processing by releasing the focal and the principal point as parameters to be reestimated using optional arguments FocFree & PPFree. We keep the best exterior orientations after compensation which is Ori-Compense-AF15P7-DPhase-La/ performing  $\sim 6$  px mean reprojection error. Since the camera model here is quite complex with a large number of parameters, it is not reliable to release all parameters during the compensation.

```
mm3d Campari "image_002_00*.tif" Ori-Bascule-AF15P7-DPhase/ Compense-AF15P7-DPhase-La
    EmGPS=[Ori-Nav-DPhase/,0.015,0.025] GpsLa=[0,0,0] FocFree=true PPFree=true

mm3d GCPCtrl "image_002_00*.tif" Ori-Compense-AF15P7-DPhase-La/ AllPts-RTL.xml MesImages.xml

LA: [0.0931686,-0.0576498,-0.148083]

=====
|| Value=7.36595 for Cam=image_002_00071.tif and Pt=6 ; MoyErr=5.67418
=====
== GCP STAT == Dist, Moy=0.106254 Max=0.120522
```

#### 4.6.2.6 Integrated Sensor Orientation using embedded GPS and 1 GCP

We perform the same processing by releasing the same internal parameters as above and introducing a constraint using 1 GCP using the optional argument GCP. We start by splitting the file containing all ground points (AllPts-RTL.xml) into 2 files, one containing only 1 point to be used in the bundle block adjustement and the remaining points will be used as check points.

```
mm3d TestLib SplitPts ./ AllPts-RTL.xml GCPs=[10] OutGCPs=GCP_LA_Calib-RTL.xml
    OutCPs=CPs_LA_Calib-RTL.xml
```

Then we perfom the bundle block adjustement:

```
mm3d Campari "image_002_00*.tif" Ori-Bascule-AF15P7-DPhase/ Compense-AF15P7-DPhase-La
    EmGPS=[Ori-Nav-DPhase/,0.015,0.025] GpsLa=[0,0,0]
    FocFree=true PPFree=true GCP=[GCP_LA_Calib-RTL.xml,0.1,MesImages.xml,0.5]
```

```
mm3d GCPCtrl "image_002_00*.*tif" Ori-Compense-AF15P7-DPhase-La/ CPs_LA_Calib-RTL.xml
MesImages.xml

LA: [0.0943014,-0.0574927,-0.146457]

=====
|| Value=1.84504 for Cam=image_002_00121.tif and Pt=5 ; MoyErr=0.933554
=====
== GCP STAT == Dist, Moy=0.0124384 Max=0.0270012
```

The value 0.1 is a multiplicative factor of the uncertainty field already given in the file `GCP_LA_Calib-RTL.xml` and whose value is fixed to 1 cm for planimetric components and 2 cm for vertical component. As the number of tie points is more important in the compensation, one should sometimes try to give more weight to external measurement, specially when its number is very low, as here, we have only one GCP measurement. The value of 0.1 means that weight of this measurement is 10 times more important in order to constraint the parameters estimation during the compensation. (One can check that with a value of 1 instead of 0.1 the accuracy on check points is two times worse). We show here that with no prior calibration and with a cheap GPS receiver, it is possible to achieve with only one single ground control point an absolute georeferencing of camera poses with an accuracy of  $\sim 1\text{ px}$ .

#### 4.6.2.7 Classical GCPs indirect georeferencing

We perform here a classical conversion of relative camera poses into absolute ones using (a reduced number) ground control points. We start by splitting gound points into ground control points and check points. (We need at least 3 ground control points).

```
mm3d TestLib SplitPts ./ AllPts-RTL.xml GCPs=[1,4,10,15,17] OutGCPs=Reduced_GCPs-RTL.xml
OutCPs=Reduced_CPs-RTL.xml
```

Here we perfom the similarity transformation using the `mm3d GCPBascule` command:

```
mm3d GCPBascule "image_002_00*.*tif" Ori-All-F15-AddP7/ Basc-GCPs-F15-AddP7
Reduced_GCPs-RTL.xml MesImages.xml

mm3d GCPCtrl "image_002_00*.*tif" Ori-Basc-GCPs-F15-AddP7/
Reduced_CPs-RTL.xml MesImages.xml

=====
|| Value=1.01048 for Cam=image_002_00071.tif and Pt=6 ; MoyErr=0.722451
=====
== GCP STAT == Dist, Moy=0.00770411 Max=0.0182981
```

We perform a bundle block adjustment using reduced ground control points in the compensation and releasing some internal parameters of the camera.

```
mm3d Campari "image_002_00*.*tif" Basc-GCPs-F15-AddP7 Compense-GCPs-F15-AddP7
GCP=[Reduced_GCPs-RTL.xml,1,MesImages.xml,0.5] FocFree=true PPFree=true

mm3d GCPCtrl "image_002_00*.*tif" Ori-Compense-GCPs-F15-AddP7/
Reduced_CPs-RTL.xml MesImages.xml

=====
|| Value=0.945775 for Cam=image_002_00071.tif and Pt=6 ; MoyErr=0.706563
=====
== GCP STAT == Dist, Moy=0.00779557 Max=0.0187995
```

#### 4.6.3 Dense Matching and Orthorectification

First, we export optimal dataset of camera poses, here using GPS measurements, in a .ply format using the `mm3d AperiCloud` command.

```
mm3d AperiCloud "image_002_00*.*tif" Ori-Compense-AF15P7-DPhase-La/
```

Then, with the `mm3d SaisieMasqQT` command, we draw a 3d polygon in order to limit the area of matching.

```
mm3d SaisieMasqQT AperiCloud_Compense-AF15P7-DPhase-La.ply
```

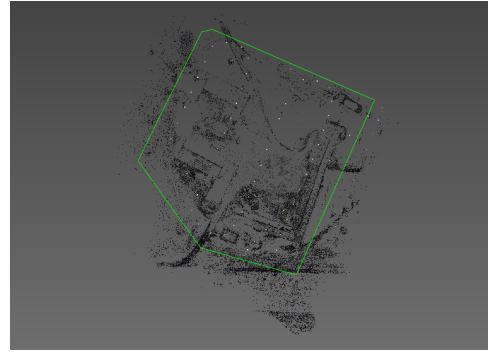


Figure 4.27: Drawing a mask

The `mm3d PIMs` command is used to generate the digital surface model using the mode `QuickMac`. The optional arguments `Masq3D` and `FilePair` are used to speed up the processing time cost.

```
mm3d PIMs QuickMac "image_002_00*.tif" Ori=Compense-AF15P7-DPhase-La/
    Masq3D=AperiCloud_Compense-AF15P7-DPhase-La_polyg3d.xml
    Out=NuageGps.ply FilePair=CpleImgs.xml
```

We use `mm3d PIMs2Mn` command to merge the result of the stereo depth maps computed above. The optional argument `DoOrtho` is used to generate individual orthoimages.

```
mm3d PIMs2Mnt QuickMac DoOrtho=1
```

The orthomosaic image is generated using the `mm3d Tawny` command without performing any radiometric equalization.

```
Tawny PIMs-ORTHO/ RadiomEgal=false
```

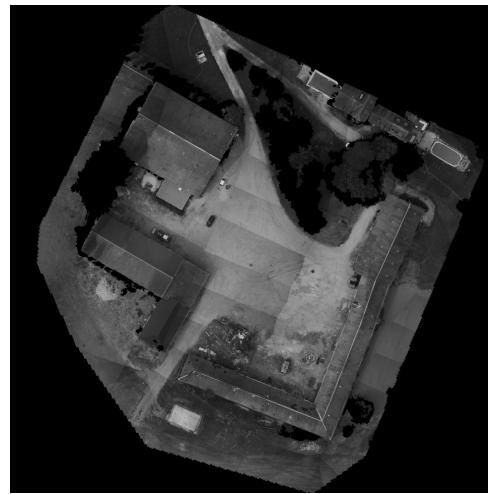


Figure 4.28: Orthoimage

The shading image of depth map is computed using the `mm3d Grshade` command.

```
mm3d Grshade PIMs-TmpBasc/PIMs-Merged_Prof.tif Out=Shading.tif ModeOmbre=IgnE
```

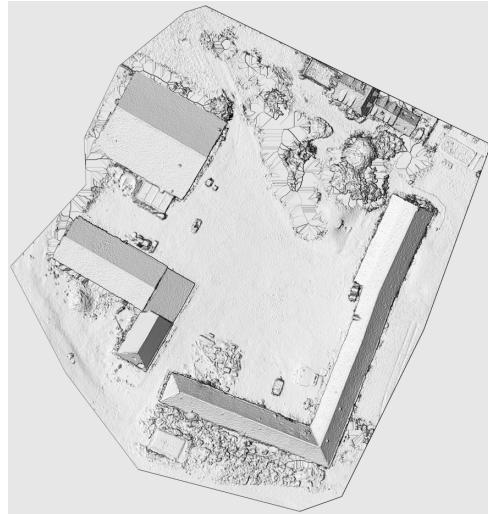


Figure 4.29: Shading image

To convert the depth map into a hypsometric representation we use the `mm3d to8Bits` command.

```
mm3d to8Bits PIMs-TmpBasc/PIMs-Merged_Prof.tif Out=Hypso.tif Circ=1
```

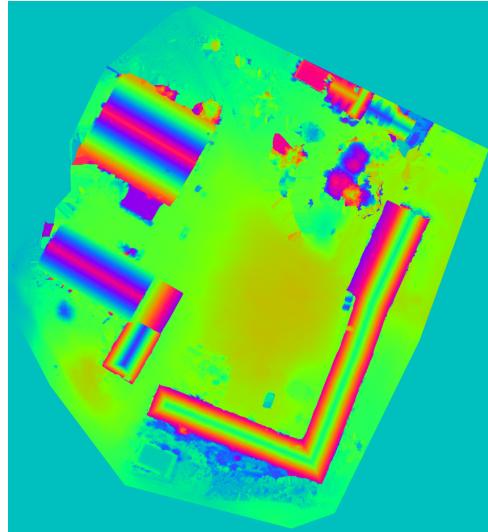


Figure 4.30: Hypsometric image

The command `mm3d Nuage2Ply` is used to export a dense points cloud.

```
mm3d Nuage2Ply PIMs-TmpBasc/PIMs-Merged.xml Scale=1 Attr=PIMs-ORTHO/Orthophotomosaic.tif
RatioAttrCarte=2 Out=GpsNuage.ply
```

Visualization of the 3d points cloud using `meshlab`.

```
meshlab GpsNuage.ply
```

# Chapter 5

# A Quick Overview of Matching

This chapter will give a global overview of the tools, and the concept they use. This chapter is restricted to matching, an overview of orientation is given in chapter 6. It is not a formal or complete description, this will be done in further chapters; here there are essentially examples and comments.

## 5.1 Installing the Tools

### 5.1.1 svn Extraction - obsolete (see 3.2.1)

All that you need to install the tools and run the example can be found on the site <http://www.micmac.ign.fr/>. Clik on Téléchargement to go to the download page. To have an up to date version, you will need to install the subversion tools. To get the micmac softwares, type:

```
svn co http://www.micmac.ign.fr/svn/micmac/trunk micmac
```

### 5.1.2 Compilation

The `Readme.txt` contains the list of instructions necessary to install the software. If you compile from source code, after

`hg clone` it may be a bit long (15 min on a mono processor computer) as it will require full compilation.

Follow exactly the directive of `Readme.txt`, including directory creation and the required `touch` actions. `touch` is necessary to break some tricky circular dependencies in the `Makefile`.

After the installation, there should exist on `/usr/local/bin/` a file `MicMacConfig.xml`, that describes some installation configuration, it should look like :

```
<?xml version="1.0" ?>
<MicMacConfiguration>
  <DirInstall>/home/mpd/micmac/</DirInstall>
  <NbProcess>2</NbProcess>
</MicMacConfiguration>
```

### 5.1.3 Some Important Directories

Starting from the `micmac` directory, there are some important directories to know:

- `include/XML_GEN/`, where you will find the formal XML specification that will be described in chapter 10;
- `Documentation/DocMicMac/`, where you will find this documentation and others documents related to these tools;
- `bin/`, will contain the executable command generated after the different `make`

If you are in software development and interested to take a look at the code, here are others directories:

- `src`, all the C++ code needed to generate the `EIGELib`, this is a general image library I developed before MicMac, all the tools are using this lib for basic manipulation;
- `include/`, all the header files needed by src files for `EIGE`

### 5.1.4 Installing the Examples

You can download several examples (many of them are obsolete ...) by typing:

```
svn co http://www.micmac.ign.fr/svn/micmac_data/trunk micmac_data
```

However, it may be a bit long, depending of the bandwidth. If you do not need the interface for now and only want the example of this documentation, you can type :

```
svn co http://www.micmac.ign.fr/svn/micmac_data/trunk/ExempleDoc/ micmac_data/ExempleDoc/
```

### 5.1.5 Verification and Global Vision of the Main Tools

Once the installation is completed<sup>1</sup>, go into the micmac directory, and take a look at the file `boudha_test.sh`. It contains 3 command lines:

```
"$BIN_DIR/Tapioca" MulScale "$CHANT_DIR/IMG_[0-9]{4}.tif" 300 -1 ExpTxt=1
"$BIN_DIR/Apero" "$CHANT_DIR/Apero-5.xml"
"$BIN_DIR/MICMAC" "$CHANT_DIR/Param-6-Ter.xml"
```

This is a typical simple execution of the pipeline to build 3D models from images:

- the first line `Tapioca` is the command line for generating tie points from a set of images; this tool is described in 3.3;
- the second line `Apero` is the command line for generating orientations; it is described in 6;
- the third line `MICMAC` is the command line for generating dense matching from oriented images; it is described in this chapter;

Practically, the ordering of different phases will always be:

1. Tie Points
2. Orientation
3. Dense Matching.

However, in this quick overview, we present it in the reverse order because it is dense matching that justifies orientation, and it is orientation that justifies tie points.

Being still in the micmac directory, type:

```
source boudha_test.sh
```

You will see a lot of text on the terminal. Do not stop the process, it will produce data, that will be used in the rest of the examples. If everything is correctly installed, it will be over after 15 minutes (or less, according to your computer). Check the directory:

```
micmac_data/ExempleDoc/Boudha/MEC-6-Im/
```

You must have 3 files having a `.ply` extension. You can take a look at these files using a free software such as `MeshLab`.

## 5.2 A MicMac Example Using Simplest Parametrization

All the data needed in the following part are in the directory `micmac_data/ExempleDoc/Boudha/`.

### 5.2.1 Epipolar Geometry

For this first set of examples, to make it simple, we will treat a pair of images that have been resampled in epipolar geometry. These images are `Epi-Left.tif` and `Epi-Right.tif`. The figure 5.1 shows a quick view of these two images.

According to the definition of epipolar, for each point  $x, y$  of `Epi-Left.tif` its homologous point, in image `Epi-Right.tif`, is located on the same line, so at position  $x', y$ . The aim of epipolar matching is, given  $x, y$ , compute the disparity  $d(x, y) = x' - x$  so that  $x + d(x, y), y$  is homologous of  $x, y$ .

Obviously, epipolar geometry is not very interesting in modern photogrammetry, however it will allow to separate in this introduction the parameters related to matching from parameters related to geometry.

MicMac uses (almost always) the normalized cross-correlation coefficient as a similarity measurement. See the chapter 22 if you forgot the meaning of this coefficient.

One of the simplest matching algorithm for computing a disparity map is the following:

- let  $Sz$  be the size of correlation window,  $D$  be the size of the disparity interval, and  $Cor(x_1, y_1, x_2, y_2, Sz)$  be the normalized cross correlation between two windows of size  $Sz$ , centered on  $x_1, y_1$  and  $x_2, y_2$ ;
- set  $d(x, y) = \text{ArgMax}_{d \in [-D, D]} Corr(x, y, x + d, y, Sz);$

### 5.2.2 Analyzing Matching Parameters

---

1. tools are built and you got data on `ExempleDoc`



Figure 5.1: The pair of epipolar images used in these examples

```

<!--
  The simplest MicMac example
-->

<ParamMICMAC>
  <Section\PriseDeVue>
    <GeomImages> eGeomImage\_EpipolairePure </GeomImages>
    <Images>
      <Im1> Epi-Left.tif </Im1>
      <Im2> Epi-Right.tif </Im2>
    </Images>
  </Section\PriseDeVue>

  <Section\Terrain>
    <IntervParalaxe>
      <Px1IncCalc> 50.0 </Px1IncCalc>
    </IntervParalaxe>
  </Section\Terrain>
  <Section\MEC>

  <EtapeMEC>
    <DeZoom> -1 </DeZoom>
    <SzW> 1 </SzW>
    <AlgoRegul> eAlgoMaxOfScore </AlgoRegul>
    <Px1Pas> 1 </Px1Pas>
    <!-- Unused but mandatory -->
    <Px1DilatAlt> 0 </Px1DilatAlt>
    <Px1DilatPlani> 0 </Px1DilatPlani>
    <Px1Regul> 0 </Px1Regul>
  </EtapeMEC>
  <EtapeMEC> <DeZoom> 1 </DeZoom> </EtapeMEC>
</Section\MEC>

  <Section\Results>
    <GeomMNT> eGeomPxBiDim </GeomMNT>
  </Section\Results>

  <Section\WorkSpace>
    <WorkDir> ThisDir </WorkDir>
    <TmpMEC> MEC-0-EPI/ </TmpMEC>
    <TmpResult> MEC-0-EPI/ </TmpResult>
    <TmpPyr> Pyram-Epi/ </TmpPyr>
  </Section\WorkSpace>
<Section\Vrac> </Section\Vrac>
</ParamMICMAC>

```

The file `Param-0-Epi.xml` contains parameters for MicMac to execute the previous simple algorithm. Some comments:

- the most encompassing tag is `ParamMICMAC`<sup>2</sup>, there must be only one; MicMac loads and interprets all the information contained in this tag;
- the first level tag `Section_PriseDeVue`<sup>3</sup> contains parameters about image acquisition, the name of images are in tags `Im1` and `Im2` and the tag `GeomImages` whose value `eGeomImage_EpipolairePure` means that the image have been sampled in epipolar geometry;
- the tag `Section_Terrain`<sup>4</sup> contains information about the terrain, in other geometry it may be the interval of  $Z$  and the desired footprint; here, we are in epipolar geometry, so the "terrain" is just the image space,  $x$  and  $y$  are pixel of master image and the only required information is the disparity interval specified in the tag `IntervParalaxe`;
- the tag `Section_MECA`<sup>5</sup> contains the information relative to the algorithmic aspect; it is detailed below;
- the tag `Section_Results` contains the information necessary to generate the desired results; here it is semantically empty, the value `eGeomPxBiDim` of `GeomMNT` being mandatory in the epipolar case;
- the tag `Section_WorkSpace` contains the information for files organization; the value `ThisDir` of tag `WorkDir` means that all the names of files are relative to the directory where the parameter file is located.

2. = shooting section  
 3. = shooting section  
 4. = Ground section  
 5. = Mise En Correspondance = Matching

The **Section\_MEC** can be very complex; MicMac has a multi-resolution approach in which several phases of matching are piped, for each step all the algorithmic parameters can be changed if necessary. However, generally this is not desired, so to limit the complexity, when the parameters are mainly constant for all phases, MicMac has the following mechanism:

- the first **Section\_MEC** is not executed, it conventionally sets the default parameters of all the following **Section\_MEC**, it is mandatory that the tag **DeZoom**, fixing the resolution, has the value  $-1$  for this first step;
- the default parameters set in this example are:
  - **SzW** sets the size of the correlation window, the value  $1$  means in fact  $[-1, 1] \times [-1, 1]$ ;
  - **AlgoRegul** specifies the possible regularization algorithm, in this simplest example, the value **eAlgoMaxOfScore** means that there is no regularization (i.e. compute arg-max);
  - **Px1Pas** specifies the quantification step of the disparity map; here, the value  $1$  means in fact that only the integer disparity is tested;
  - all the other tags are mandatory for internal reason but they have no meaning here;
- in this example, there is only one phase, and as all the interesting parameters have been set in the default phase, it contains only the resolution parameter **DeZoom**, here we work at full resolution, so *DeZoom* =  $1$ .

### 5.2.3 Running the Program

Now, type the following command:

```
bin/MICMAC /micmac_data/ExempleDoc/Boudha/Param-0-Epi.xml
```

The terminal should print a lot of idiot messages like:

```
<< Make Masq Resol 1 /home/mpd/micmac_data/ExempleDoc/Boudha/MEC-0-EPI/Masq_LeChantier_DeZoom1.tif
>> Done Masq Resol 1 /home/mpd/micmac_data/ExempleDoc/Boudha/MEC-0-EPI/Masq_LeChantier_DeZoom1.tif
>> Done Masque for /home/mpd/micmac_data/ExempleDoc/Boudha/MEC-0-EPI/Masq_LeChantier_DeZoom1.tif
...
<< Make Masque for /home/mpd/micmac_data/ExempleDoc/Boudha/MEC-0-EPI/Masq_LeChantier_DeZoom64.tif
>> Done Masque for /home/mpd/micmac_data/ExempleDoc/Boudha/MEC-0-EPI/Masq_LeChantier_DeZoom64.tif
...
<< Make Pyram for /home/mpd/micmac_data/ExempleDoc/Boudha/Epi-Right.tif
>> Done Pyram for /home/mpd/micmac_data/ExempleDoc/Boudha/Epi-Right.tif
...
<< Make Pyram for /home/mpd/micmac_data/ExempleDoc/Boudha/Pyram-Epi/Epi-Left.tifDeZoom64.tif
>> Done Pyram for /home/mpd/micmac_data/ExempleDoc/Boudha/Pyram-Epi/Epi-Left.tifDeZoom64.tif
Make Masq /home/mpd/micmac_data/ExempleDoc/Boudha/Pyram-Epi/MasqIm_Dz128_M.tif
...
----- BEGIN ETAPE, , Num = 1, DeZoomTer = 1, DeZoomIm = 1
-- BEGIN BLOC Bloc= 0, Out of 2
  Images Loaded
  Correl Calc, Begin Opt
  TCOr 3.87195 CTimeC 0.383362 TOpt 0.039567 Pts , R2 100, RN 0 Pts , R-GEN 0, Isol 0 PT 1.01e+07
  Images Loaded
...
  Correl Calc, Begin Opt
  TCOr 34.5481 CTimeC 0.388705 TOpt 0.372508 Pts , R2 100, RN 0 Pts , R-GEN 0, Isol 0 PT 8.888e+07
```

Do not worry, this is perfectly "normal", after a couple of seconds it should be over, and the prompt should appear again on your terminal. On the other hand, if something like that appears, then there is a problem:

```
-----
| the following FATAL ERROR happened (sorry)
|
| XXXXX HERE WILL BE SOME INCOMPREHENSIBLE EXPLANATION XXXX
|
-----
|   (Elise's) LOCATION :
|
| Error was detected
|   at line : 1640
|   of file : applis/MICMAC/cAppliMICMAC.cpp
-----
Bye (tape enter)
```

This message informs you that something wrong occurred. When you create your own file parameters, it may happen quite often at first ... However, if this happens with the test parameters, it will be completely abnormal, so please contact the hotline.

### 5.2.4 Analyzing the Results

Suppose the program finishes normally, you will see two new directories: **Pyram-Epi/** and **MEC-0-EPI/**, the name having been specified in the **Section\_WorkSpace**.

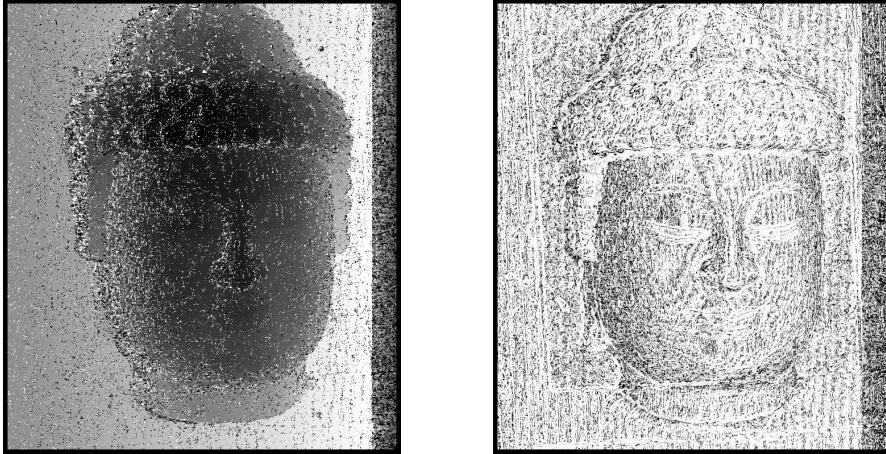


Figure 5.2: Disparity map an correlation image, dynamic has been adapted

In the directory `Pyram-Epi` there are pyramids of images used for the multi-resolution approach; we will comment this directory in 5.3.1 where multi-resolution is used.

In the `MEC-0-EPI/`, the following files exist:

- `Px1_Num1_DeZoom1_LeChantier.tif`, this is the disparity map; it is a 16 bits, signed image, so you may have some difficulty to visualize it with many basic viewers which only handle 8 bits unsigned images; in the context, the interpretation is easy, the value of pixel  $x, y$  is the disparity at  $x, y$ ; left image of figure 5.2 shows what it looks like;
- `Correl_LeChantier_Num_1.tif` contains the value of normalized correlation coefficient for the selected disparity, see figure 5.2;
- `param_LeChantier_Ori.xml` is just a copy of your parameter file, it may be useful if you work again on this data, a few months later;
- other files are not very interesting for now for such a simple test.

## 5.3 Examples Using MicMac, Algorithmic Aspect

### 5.3.1 Using Multi-resolution

The file `Param-1-Epi.xml` adds the multi-resolution aspect to the previous one. The modified or added parts from `Param-0-Epi.xml` are:

```
....  
<EtapeMEC>  
  
...  
<Px1DilatAlti> 3    </Px1DilatAlti>  
<Px1DilatPlani> 4   </Px1DilatPlani>  
<!-- Unused here but mandatory -->  
<Px1Regul> 0     </Px1Regul>  
</EtapeMEC>  
  
<EtapeMEC> <DeZoom >      8      </DeZoom> </EtapeMEC>  
<EtapeMEC> <DeZoom >      4      </DeZoom> </EtapeMEC>  
<EtapeMEC> <DeZoom >      2      </DeZoom> </EtapeMEC>  
<EtapeMEC> <DeZoom >      1      </DeZoom> </EtapeMEC>  
<EtapeMEC>  
    <DeZoom >      1      </DeZoom>  
    <Px1Pas>      0.5   </Px1Pas>  
</EtapeMEC>  
...
```

The principle of the multi-scale matching is that real homologous points are similar at every scale, while "false" homologous points that may appear, by hazard, at a certain scale will not be similar at other scales. To implement this idea, MicMac computes a pyramid of images at scale 1, 2, 4, 8..., and tries to compute a matching that selects points who are similar in all the specified scales. The directory `Pyram-Epi/` specified by the tag `TmpPyr` contains the file at different resolutions with a quite obvious name (`Epi-Left.tifDeZoom8.tif` for image

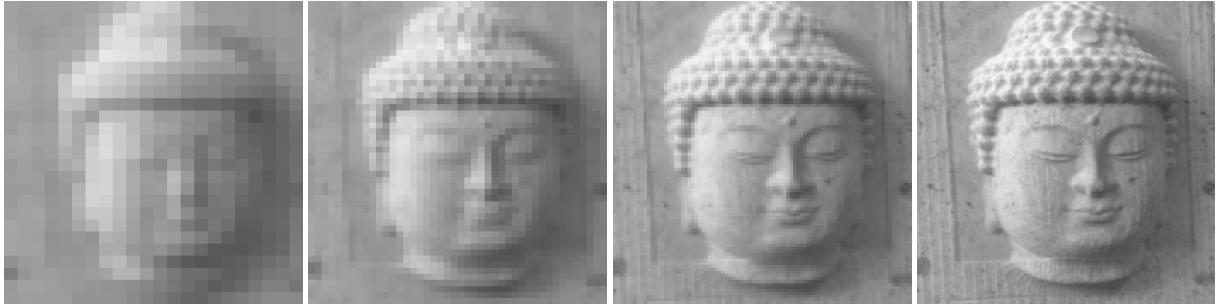


Figure 5.3: Some levels of the Buddha pyramid (the low resolution has been scaled so that all images have the same size in document)

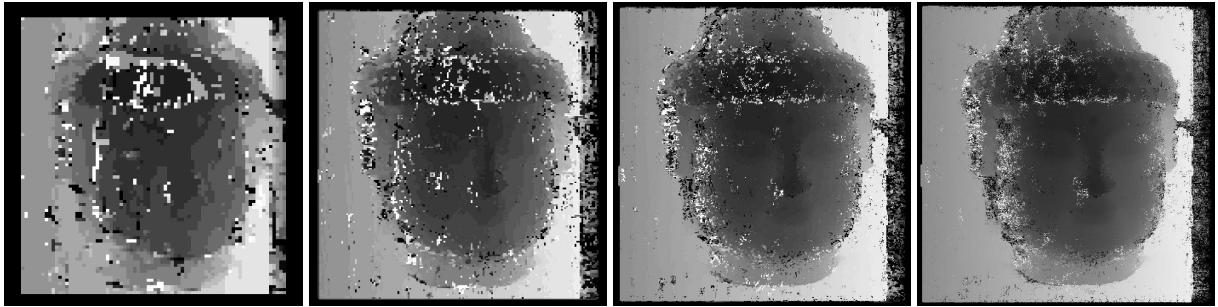


Figure 5.4: The disparity maps computed at different resolutions

`Epi-Left.tif` reduced of a factor 8). As they are floating images, you may have a problem to visualize them with most of the viewers. Figure 5.3 presents some levels of the pyramid.

MicMac does not process all the resolutions at the same time; in fact, it begins with the lower resolution (here 8). This lower resolution is treated "normally" (i.e. without multi-scaling). Then iteratively, the resolution  $2^n$  is computed, using the solution  $2^{n+1}$ : MicMac forces the solution  $2^n$  to be close to the previous  $S(2^{n+1})$ . How much  $S(2^n)$  must be close to  $S(2^{n+1})$  is controlled by the parameters `Px1DilatAlti` and `Px1DilatPlani`. How these parameters are precisely used is described in 21.2.

At each step, the computed disparity maps are saved and they can be seen in directory `MEC-1-EPI`:

- `Px1_000_DeZoom8.LeChantier.tif` is a special case, it is full of zero, and computed to initiate the iterative process;
- the first computed disparity is `Px1_Num1_DeZoom8.LeChantier.tif` where the `Num1` means first computed and `DeZoom8`; the `DeZoom` is not sufficient as identifier as there are currently different phases at the same resolution; here, for example, `Num4` and `Num5`;
- this naming is a bit tricky, like almost all automatically generated names.

Figure 5.4 shows levels of the pyramid of computed disparity maps.

If you compare the left image of figure 5.2 with the right image of figure 5.4, you can observe that with the multi-scale approach the level of noise decreases. Moreover, as this multi-scaling is implemented via multi-resolution, the computation time generally decreases very significantly.

With highly modern high quality camera and multi-stereoscopic acquisition, it is possible to have a matching precision much better than the pixel. To obtain this sub-pixel precision, a step parameter can be set with the tag `Px1Pas`. Let `Step` be the value of this parameter, MicMac will compute the disparity:

$$d(x, y) = \operatorname{ArgMax}_{d \in [-D, D]} \operatorname{Corr}(x, y, x + d * Step, y, Sz);$$

Instead of :

$$d(x, y) = \operatorname{ArgMax}_{d \in [-D, D]} \operatorname{Corr}(x, y, x + d, y, Sz);$$

Of course we need to get values of images at non integer point, so it requires an interpolation schema. In the previous example, the last phase was specified with a precision of half a pixel:

```
<EtapeMEC>
    <DeZoom >      1      </DeZoom>
    <Px1Pas>        0.5   </Px1Pas>
</EtapeMEC>
```

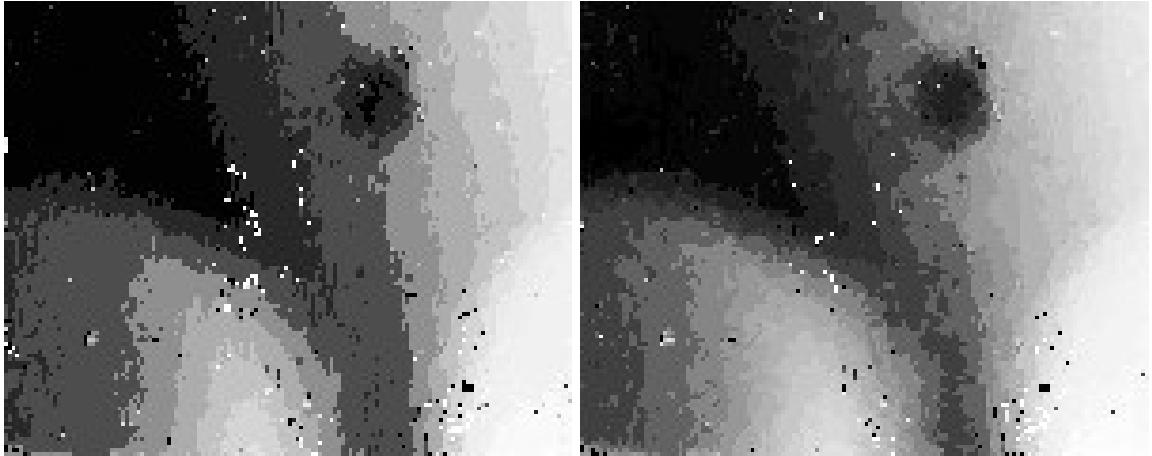


Figure 5.5: Zoom on a detail to illustrate the effect of the step parameter, left  $step = 1$  and right  $step = 0.5$

The figure 5.5 presents the effect of this sub-pixelar matching.

### 5.3.2 Using Regularization

The file `Param-2-Epi.xml` adds the regularization aspect to the previous one. The modified or added parts from `Param-0-Epi.xml` are:

```
.....
<AlgoRegul> eAlgoCoxRoy </AlgoRegul>
<Px1Regul> 0.05 </Px1Regul>
<ModeInterpolation> eInterpolMPD </ModeInterpolation>
.....
<EtapeMEC>
    <AlgoRegul> eAlgoDequant </AlgoRegul>
    <DeZoom > 1 </DeZoom>
    <Px1Pas> 1 </Px1Pas>
</EtapeMEC>
....
```

The line `<ModeInterpolation> eInterpolMPD </ModeInterpolation>` shows just how the interpolator, necessary for sub-pixel matching, is controlled.

For the regularization, MicMac uses an energetic formalism, where a functional, combining a regularity term and a image matching, is globally minimized. Mathematical details can be found in chapter 24.1. As several algorithms are proposed, the tag `<AlgoRegul>` controls which algorithm is to be chosen. Here, the line `<AlgoRegul> eAlgoCoxRoy </AlgoRegul>` corresponds to the Cox-Roy implementation of the Min-Cut/Max-Flow algorithm.

All theses algorithms have a parameter which allows to adjust the importance of regularity term relatively to the image term. The tag `Px1Regul` controls the value of this parameter. Figure 5.6 illustrates the results of regularization.

In file `Param-2-Epi.xml` the last and 6<sup>th</sup> phase contains the line `<AlgoRegul> eAlgoDequant </AlgoRegul>`. The matching algorithm used by MicMac makes a quantification of the disparity (or depth, or height ... , according to the geometry). This quantification creates jumps in the results that may be undesirable. The dequantification algorithm, specified by `eAlgoDequant`, is a post-processing phase that fixes the quantification artifacts. This algorithm is described in 24.5. The result is a floating point map. MicMac imposes the step 1.0 when using this algorithm (as the notion of step is not pertinent for a floating point result). Figure 5.6 illustrates the effect of dequantification.

The file `Param-3-Epi.xml` is similar to `Param-2-Epi.xml`, the main difference being the used regularization algorithm. It is specified `eAlgo2PrgDyn` which is a 2D generalization of dynamic programming. It is specified `eAlgoTestGPU` which is a 2D generalization of dynamic programming with using CUDA. This algorithm does not produce the exact minimum of the energy function, but a solution generally very close to the optimum. On the other hand, it is generally faster and more flexible. It has more parameters; the lines of `Param-3-Epi.xml` corresponding to a basic parametrization are:

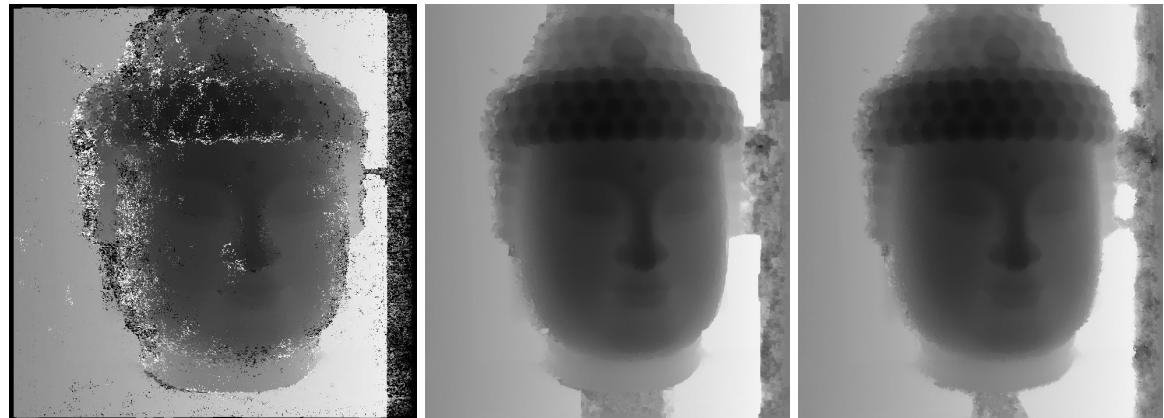


Figure 5.6: Without regularization, with Graph Cut regularization, with 2D dynamic regularization

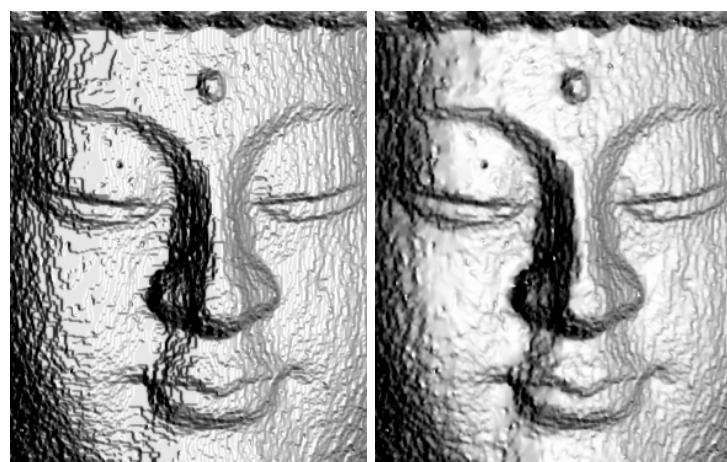


Figure 5.7: Without and with dequantification

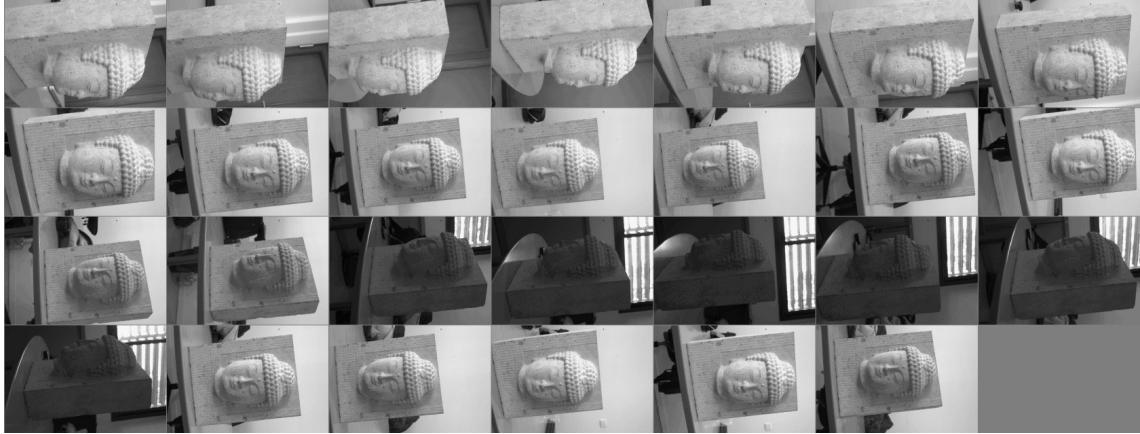


Figure 5.8: The 34 images used in the following section

```

...
<ModulationProgDyn>
  <EtapeProgDyn>
    <ModeAgreg> ePrgDAgrSomme </ModeAgreg>
    <NbDir> 7 </NbDir>
  </EtapeProgDyn>
  <Px1PenteMax> 3.0 </Px1PenteMax>
</ModulationProgDyn>
...
  
```

The middle and right images of figure 5.6 allow to compare the result of max flow algorithm and 2D dynamic programming.

In `Param-3-Epi.xml`, the line `<GenImagesCorrel > true </GenImagesCorrel>` requires to generate the correlation coefficient image for every phase (otherwise, it is generated only for the last phase). The file name is easy to understand, for example `Correl_LeChantier_Num_3.tif`.

Another innovation of `Param-3-Epi.xml` is `<ByProcess> 2 </ByProcess>` in section `Section_WorkSpace`. With this option, MicMac will have the ability to split the computation in two parallel processes. With my computer, I set `<ByProcess>` at 2, because I have only two processor, but if you have, for example 4 bi-processor, you can set it at 8. Of course, it is your responsibility to set it to a reasonable value, knowing the characteristics of your computer and other tasks that may be running at the same time.

## 5.4 Examples using MicMac, Geometric Aspect

To run the example coming in this section, you need to have executed the shell described in 5.1.5 because it generates the required orientation files.

Figure 5.8 presents a quick view of the 34 images that will be used to illustrate full 3D modelization with Apero and MicMac.

### 5.4.1 Ground Geometry

Epipolar geometry has the advantage of simplicity. However, it lacks of generality, it cannot be used for multi-stereoscopy and it imposes unnecessary resampling of the images. Moreover, generating proper epipolar images requires some preliminary orientation of the image, so potentially if you can do epipolar geometry, you will be able to do ground geometry. There are almost no case where epipolar matching is preferable.

Mathematically, the object defining the orientation of an image  $k$  is a function  $\pi_k : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ ;  $\pi_k(x, y, z) = (i, j)$  where  $(x, y, z)$  is a ground point and  $i, j$  its projection in image  $k$ . The model understood by MicMac can be:

- stenope projection, which is adapted to all current cameras; the model is  $\pi_k(x, y, z) = I(p_0(R_k((x, y, z) - C_k)))$ , where  $I$  is the intrinsic parameters (focal, principal point and distortion) and  $p_0(x, y, z) = \frac{(x, y)}{z}$ ; the imported stenope model can be in different formats such as XML coming from the Apero software and several formats used at IGN;
- different generic model (Grid model of RPC, ratio of polynomial) adapted to non physical modelization of push-broom images (satellites ...);
- the user can add its own model using a dynamic library (a bit tricky ...).

In this example we will consider only stenope model stored in the format generated by Apero. The matching in ground geometry consists in computing a height map  $Z = Z(x, y)$  so that:

- the windows centered on the  $I_k(\pi_k(x, y), Z(x, y))$  are similar;
- $Z(x, y)$  satisfies some regularity function.

All the algorithmic points studied for epipolar disparity maps, are usable for ground geometry height map.

### 5.4.2 An Example in Ground Terrain, Parameter Analysis

The file `Param-4-Ter.xml` is our first example of matching in ground geometry. It uses multi-stereoscopy matching with the five images of figure 5.9. It contains several innovations:

```
.....
<!-- M1 : Declare an association between an image and its orientation file-->
<DicoLoc>
  <KeyedNamesAssociations>
    <Calcs>
      <Direct>
        <PatternTransform> (.*) </PatternTransform>
        <CalcName> Ori-Test-5/Orientation-$1.xml </CalcName>
      </Direct>
    </Calcs>
    <Key> Loc-Key-Orient </Key>
  </KeyedNamesAssociations>
</DicoLoc>

<!-- M2 : Describe the ground zone where the matching is to be done -->
<Section_Terrain>
  <IntervAltimetrie>
    <ZIncCalc> 3.0 </ZIncCalc>
  </IntervAltimetrie>
  <Planimetrie>
    <BoxTerrain> -2.13 -2.56 3.51 5.08 </BoxTerrain>
  </Planimetrie>
</Section_Terrain>

<!-- M3 : describe the set of images to match -->
<Section_PriseDeVue >
  <GeomImages> eGeomImageOri </GeomImages>
  <Images >
    <ImPat> IMG_[0-9]{4}.tif </ImPat>
    <Filter>
      <Min> IMG_5588.tif </Min>
      <Max> IMG_5592.tif </Max>
    </Filter>
  </Images>
  <NomsGeometrieImage>
    <FCND_Mode_GeomIm>
      <FCND_GeomCalc> Loc-Key-Orient </FCND_GeomCalc>
    </FCND_Mode_GeomIm>
  </NomsGeometrieImage>
</Section_PriseDeVue>
...
<!-- M4 : Specify the output geometry -->
<Section_Results >
  <GeomMNT> eGeomMNT_Euclid </GeomMNT>
</Section_Results>
```

The first new tag, `<DicoLoc>`, is an example of how the user can describe the association between the name of an image and the name of its orientation file:

- the pair of tags `<PatternTransform>` and `<CalcName>` declares an association rule; this rule means, for example, that the orientation of `IMG_5580.tif` is located in the file `Ori-Test-5/Orientation-IMG_5580.xml`;
- this rule means, for example, that the orientation of `IMG_5580.tif` is located in the file `Ori-Test-5/Orientation-IMG_5580.tif.xml`;
- take a quick look to a file `Ori-Test-5/Orientation-IMG_*.xml` and the file `Ori-Test-5/AutoCalib.xml`; if you have some idea of what a stenope projection is, you should understand the main principle of the used format; the section A.1 gives a formal description of the format;
- to this rule it is given an identifier, here `Loc-Key-Orient`; when it is necessary to refer to this association rule, this key will be used;
- generally, the mechanism allowing the declaration directly in the MicMac parameter is to be avoided; it is preferable to use predefined rules or to declare them in the special file

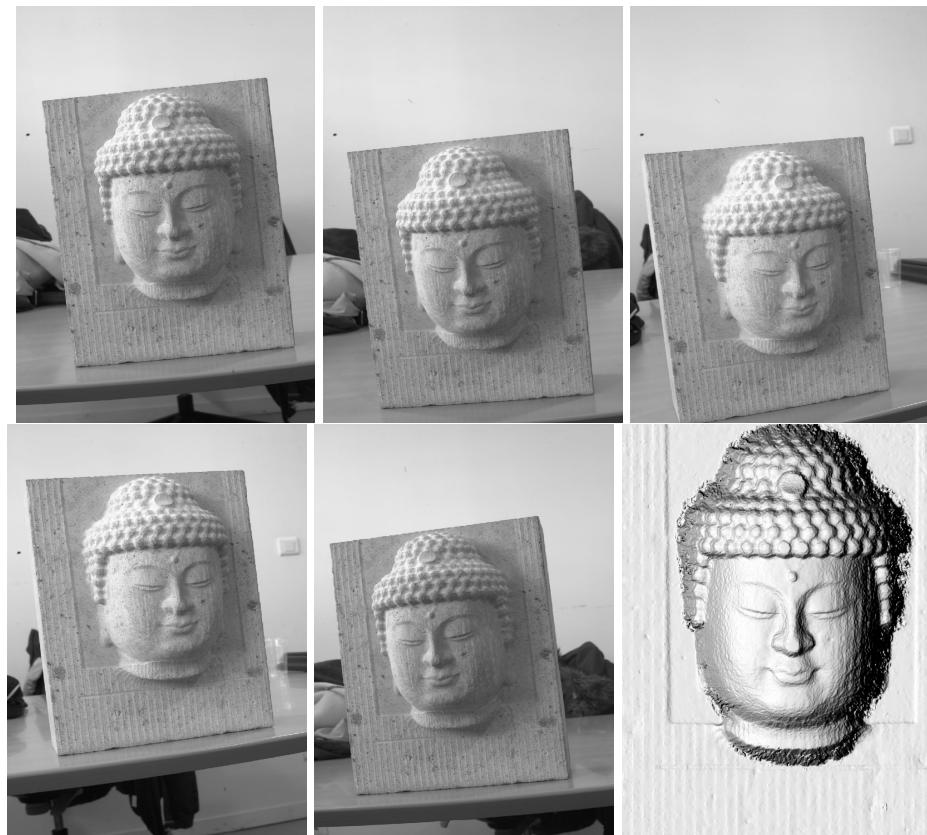


Figure 5.9: The five images used for ground geometry matching, and a shading of the DTM

`MicMac-LocalChantierDescripteur.xml` to facilitate convention sharing and maintenance (I did it here, this time, for the simplicity of reading);

Here, orientations have been computed with Apero in a purely relative way. The orientations are defined up to a global rotation, translation and scaling. An option of Apero has been used to orientate the set of images so that the background plane is globally horizontal.

The second new tag, `<Section_Terrain>`, allows the user to enter the information relative to the ground. The used options are :

- `<ZIncCalc>` specifies the amplitude of the incertitude interval that is to be explored; the central value is not specified here because the orientation file contains sufficient information to evaluate;
- `<BoxTerrain>` specifies the ground surface on which the matching is to be done; if this parameter is omitted, MicMac will automatically compute a surface which corresponds to the ground points seen at least in two images (at the average height); however, this would not be suitable here because it would include the background which we do not want to modelize.

The third innovation, `<Section_PriseDeVue>`, describes, in a manner adapted to ground geometry, the way the scene was shot:

- the tag `eGeomImageOri` specifies that the images were acquired by stenope camera<sup>6</sup>;
- the tag `ImPat` specifies the images which are to be used for matching; we want to do multi-stereo matching but not necessarily with all the images existing in the directory, so we need a way to specify the set of images; there are many ways to do it, the most current ones are:
  - like this example, by giving a general pattern and `Filter` including the name of the image, the selected image will verify  $Min \leq Name \leq Max$  according to lexicographical order;
  - by specifying a pattern using the power of regular expressions, here it would be equivalent to say `<ImPat> IMG_((558[8-9])|(559[0-2])).tif </ImPat>`;
  - as an arbitrary number of tags `ImPat` can exist, it will be alright to have `<ImPat> IMG_5588.tif </ImPat> <ImPat> IMG_5589.tif </ImPat> ...`
- the section in the tree `NomsGeometrieImage` specifies that the key `Loc-Key-Orient` declared above is to be used for getting the orientation name associated to an image.

### 5.4.3 An Example in Ground Terrain, Result Analysis

Finally, the value `eGeomMNTEuclid` of `GeomMNT` specifies that the output geometry is equal to the input geometry. This means that the resulting image is, up to translation and scaling, directly understood as a grid  $Z = f(x, y)$ . In fact, the value of the orientation file specifies the input geometry, but there are many cases where we do not want the computation to be done in this geometry<sup>7</sup>. The most current case offered by MicMac is the "ground-image" geometry presented in the example in 5.4.5. There is also the possibility of working in cylindrical geometry<sup>8</sup>.

Here we work in "Euclidean" geometry, so the result is a grid  $Z = f(x, y)$ . The bottom right image of figure 5.9 shows, in shading mode, the result obtained with this geometry. If we want to use this result as a measurement tool, we will need more than an image; as it is a grid, we need some information (meta-data) to interpret it like a 3D model which we will be able to import in another application (GIS, CAD ...).

Take a look at directory `MEC-4-Ter`. You will see that, to each image file `Z_NumXXX_DeZoomYYY_LeChantier.tif` is associated a file `Z_NumXXX_DeZoomYYY_LeChantier.xml`. For example, here is `Z_Num3_DeZoom2_LeChantier.xml`:

```
<FileOriMnt>
  <NameFileMnt>/home/mpd/micmac_data/ExempleDoc/Boudha/MEC-4-Ter/Z_Num3_DeZoom2_LeChantier.tif</NameFileMnt>
  <NameFileMasque>/home/mpd/micmac_data/ExempleDoc/Boudha/MEC-4-Ter/Masq_LeChantier_DeZoom2.tif</NameFileMasque>
  <NombrePixels>333 451</NombrePixels>
  <OriginePlani>-2.129999999999989 5.0800000000000007</OriginePlani>
  <ResolutionPlani>0.0169328251622493653 -0.0169328251622493653</ResolutionPlani>
  <OrigineAlti>0.380598155186812559</OrigineAlti>
  <ResolutionAlti>0.00846641258112469999</ResolutionAlti>
  <Geometrie>eGeomMNTEuclid</Geometrie>
</FileOriMnt>
```

The important tags are `<Origine...>` and `<Resolution...>`. Let:

- $OriginePlani = X_0, Y_0$  ;
- $OrigineAlti = Z_0$  ;
- $ResolutionPlani = \Delta_x, \Delta_y$  ;
- $ResolutionAlti = \Delta_z$  ;

6. Of course, using only XML-type files, this information is already existing in orientation files, but it is necessary for historical reasons

7. sometimes for exploitation purpose, sometimes for quality of matching

8. it would be interesting to offer the possibility to work directly in different geodetic-cartographic systems, planned it for a long time, but still to be done

Then each point  $I, J, K = K(I, J)$  of the grid defines an  $x, y, z$  point by the basic formula:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} X_0 + I * \Delta_x \\ Y_0 + J * \Delta_y \\ Z_0 + K * \Delta_z \end{pmatrix} \quad (5.1)$$

Notice that in the parameter file there is no specification of planimetric resolution. In this case, we have to let MicMac make the choice; it has selected a ground resolution equal to the average resolution of the image, this is possible because the orientation file generated by Apero contains the necessary information (like average height). For the altimetric resolution, it is implicitly specified by the parameters **ZPas**: the altimetric resolution is equal to the planimetric resolution multiplied by **ZPas**; this convention has the advantage that, for this point, the parameter files are adaptable to a wide family of configurations.

#### 5.4.4 An Example in Ground Terrain with CUDA

If you built MicMac with CUDA (see install documentation), you can speed up calculations of the dense matching and regularization.

```
<EtapeMEC>
    <DeZoom> -1 </DeZoom>
    <CorrelAdHoc>
        <SzBlocAH> 128 </SzBlocAH>
        <TypeCAH>
            <GPU_CorrelBasik> true </GPU_CorrelBasik>
        </TypeCAH>
    </CorrelAdHoc>
    <AlgoRegul> eAlgoTestGPU </AlgoRegul>
    ...
</EtapeMEC>

— SzBlocAH Size of bloc for the matching;
— GPU_CorrelBasik Use Cuda in dense matching if the value is true;
— AlgoRegul Use Cuda in regularization (with 2D generalization of dynamic programming) if the value is eAlgoTestGPU;
```

#### 5.4.5 An Example in "Ground-image" Geometry

Having the output geometry equal to the input geometry is perfectly fine for the modelization of quasi-planar objects: modelization of the earth surface seen for aerial point of view, modelization of bas-reliefs... However it is not suited for the modelization of fully 3-dimensional objects. For this kind of modelization, MicMac proposes the "image-ground" geometry in which the user can easily select a geometry adapted to each point of view. There are several variants of this geometry; in all image-ground geometry, there is a master image  $I_m$  and the  $x, y$  of this geometry represents simply the pixel  $I, J$  of  $I_m$ . We describe here the 1D-depth-of-field (1Ddof) which is optimized for well calibrated stenope camera.

The input geometry defines the correspondence between a point  $x, y, z$  of the Euclidean space and its projection in each image. To define the output geometry we must define the correspondence between a point  $A, B, C$  of the result space and its homologous in Euclidean space  $x, y, z$ , in 1Ddof:

- $A, B$  represents a pixel of the master image  $I_m$ ;
- $C$  represents the inverse of the depth of field ;
- so  $x, y, z$  is the point located on the ray emerging from  $A, B$  and located at depth<sup>9</sup>  $\frac{1}{C}$ ;
- the advantages of this geometry are:
  - like all the other ground-image geometries, the disparity map is directly superposable to the first image;
  - for a given ray, a regular sampling of  $\frac{1}{C}$  corresponds, as much as possible, to a regular sampling of the projection in all images; if  $C$  was regularly sampled, with scenes having a very high depth of field, there would be no good choice for the quantification step;
  - so this geometry has the advantage of the epipolar one, without the drawbacks.

Now we want to make a 3D model of the left face of the Buddha, using the 5 images of figure 5.10. The file **Param-5-Ter.xml** gives an example of such paramatring, there are important differences with the previous file.

```
<Section_Results>
    <GeomMNT> eGeomMNTFaisceauIm1PrCh_Px1D </GeomMNT>
</Section_Results>
```

One difference, at the end of the file, just specify the kind of geometry: the name is a bit strange... However **FaisceauIm1** means bundle of image 1; **PrCh** is a short for "Profondeur de champs" (depth of field) and **Px1D** means 1D disparity<sup>10</sup>.

9.  $\text{depth}(P) = \text{distance between the optical center and the projection of } P \text{ on the optical axis}$

10. by opposition with variants allowing 2D match for compensation of inaccurate calibration

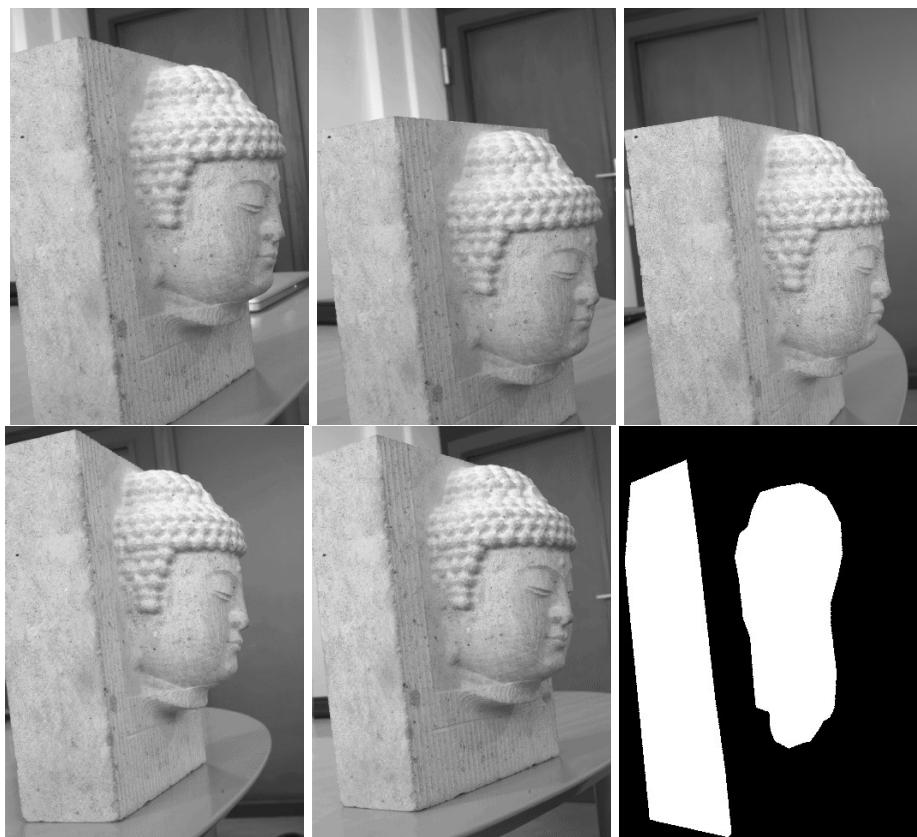


Figure 5.10: Input to matching for left face of Buddha: 5 image and a mask

```

<Section_MEC >
    <ChantierFullImage1> true </ChantierFullImage1>
...
<Section_MEC >

<Section_PriseDeVue >
    <GeomImages> eGeomImageOri </GeomImages>
    <Images >
        <Im1> IMG_5564.tif </Im1>
        <ImPat> IMG_[0-9]{4}.tif </ImPat>
        <Filter>
            <Min> IMG_5564.tif </Min>
            <Max> IMG_5568.tif </Max>
        </Filter>
    </Images>

    <NomsGeometrieImage>
        <FCND_Mode_GeomIm>
            <FCND_GeomCalc> NKS-Assoc-Im2Orient@-Test-5 </FCND_GeomCalc>
        </FCND_Mode_GeomIm>
    </NomsGeometrieImage>
</Section_PriseDeVue>
...

```

The **Section\_PriseDeVue** has two differences:

- to specify the image set we have two different tags **Im1** and **ImPat**; in this geometry, as there is a master image that plays a special role, we need a way to indicate which is the master, this is the role of tag **Im1**;
- we have no section **DicoLoc** defining a rule for computing association, instead, we have used **NKS-Assoc-Im2Orient@-Test** this is what is called a predefined parametrized key :
  - it is predefined, because the definition of the key is made in the file **include/XML\_GEN/DefautChantierDescripteur.xml** which contains a lot of predefined rules that are automatically loaded in all the tools;
  - it is parametrized, because here **NKS-Assoc-Im2Orient** is the name of a parameterizable key and **-Test-5** is the value of the parameter; in the definition of **NKS-Assoc-Im2Orient** in **XML\_GEN/DefautChantierDescripteur.xml** you will find strings like **Ori#1/Orientation-\$1.xml**, each time **#1** is encountered it will be replaced by **-Test-5**;
  - so **NKS-Assoc-Im2Orient@-Test-5** is strictly equivalent to the definition that has been given in **Loc-Key-Orient** in file **Param-4-Ter.xml**;
  - it is hoped that the parametrized rule will be sufficient to cover 95% of needs;

```

...
<Section_Terrain>
    <IntervAltimetrie>
        <!-- Mandatory but unused -->
        <ZIncCalc> 0.0 </ZIncCalc>
    </IntervAltimetrie>
    <IntervSpecialZInv >
        <MulZMin > 0.3</MulZMin>
        <MulZMax > 5 </MulZMax>
    </IntervSpecialZInv>
    <Planimetrie>
        <MasqueTerrain>
            <MT_Image> IMG_5564_Masq.tif </MT_Image>
            <MT_Xml> IMG_5564_Masq.xml </MT_Xml>
        </MasqueTerrain>
    </Planimetrie>
</Section_Terrain>

```

The **Section\_Terrain** has several differences:

- the value of **ZIncCalc**, specifying an interval of **Z**, is set to 0 , it will not be used because of the following tags **IntervSpecialZInv**;
- the tag **IntervSpecialZInv** specifies that the depth of field interval to explore is  $[0.3 * D_0, 5 * D_0]$  where  $D_0$  is the average depth (MicMac knows it from the information contained in the orientation files);
- the tag **MT\_Image** contains information for a mask of the terrain points that are to be matched, the terrain points refer to the output geometry, here the pixels of the image **IMG\_5564.tif**; figure 5.10 contains an image of the mask used here; it has been created with the tool **bin/SaisieMasq**;
- the tag **MT\_Xml** contains the name of the meta data file that georeferences the image file **IMG\_5564.Masq.tif**; this tag is important because when working on georeferenced data it may be necessary to use a mask, created on a GIS, at a step and resolution different from those chosen by MicMac; here, in the image geometry, it is not very informative (open it), however it is mandatory;

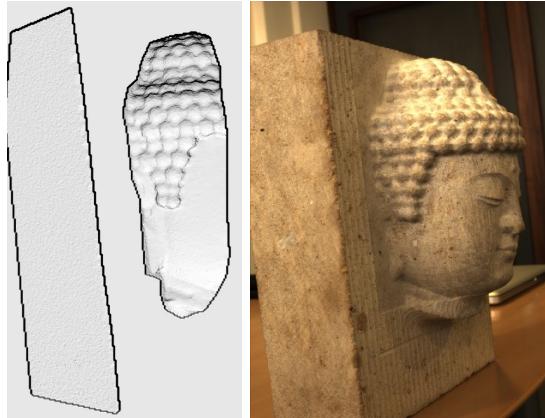


Figure 5.11: Input to Cloud generation: depth map and RGB image

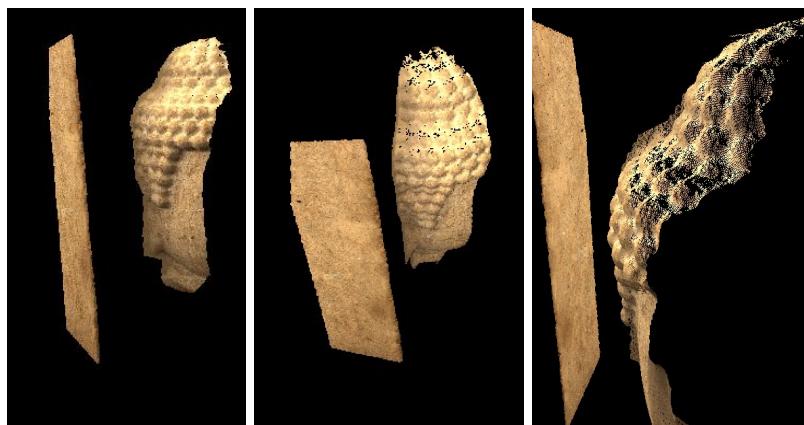


Figure 5.12: Some views of the generated point cloud

The left image of the figure 5.11 presents, in shading, the result of the depth-map you will obtain. Often, for 3D modelization, what is needed is not a depth-map but a 3D point cloud. Due to the chosen geometry, several pieces of information are necessary to generate 3D points from depth map:

- calibration of the camera (internal and external) to compute, for each pixel, its ray in Euclidean space;
- origin and steps of depth quantification;
- mask of image;
- the depth map itself.

All these pieces of information exist in separated files and, once merged, they are sufficient to generate 3D points. For a greater convenience, files merging this information are generated. In directory `MEC-5-Im/` you will find the files `NuageImProf_LeChantier_Etape_X.xml`. Take a look at these files, they contain all the information necessary to generate the point clouds. There is only the conversion to do. The tool `bin/Nuage2Ply` makes the conversion from a file `NuageImProf ...` to a point cloud in `ply` format. For example, you can type:

```
bin/Nuage2Ply ../micmac_data/ExempleDoc/Boudha/MEC-5-Im/NuageImProf_LeChantier_Etape_5.xml \
Attr=../micmac_data/ExempleDoc/Boudha/IMG_5564-RGB.tif
```

Here, the image `IMG_5564-RGB.tif` is a RGB image superposable to the master image `IMG_5564.tif`. Figure 5.11 presents a view of this image. This command generates a file `NuageImProf_LeChantier_Etape_5.ply` in the directory `MEC-5-Im/`. If you have the MeshLab tool, you will be able to visualize the cloud. Figure 5.12 presents some viewpoints generated with MeshLab.

### 5.4.6 Batching Several Computations

Now we know enough to build a complete 3D model of the Buddha from the set of images presented on figure 5.8. The Buddha is quite a simple object, so we can make the modelization with three viewpoints. We just need to run 3 times micmac with the following parameters :

- for left viewpoint `Im1=IMG_5564.tif`, `Min=IMG_5564.tif` and `MaxIMG_5568.tif`;
- for central viewpoint `Im1=IMG_5588.tif`, `Min=IMG_5588.tif` and `MaxIMG_5592.tif`;
- for right viewpoint `Im1=IMG_5581.tif`, `Min=IMG_5581.tif` and `MaxIMG_5585.tif`.

The obvious way to do it would be to run with one set of parameters, wait for the computation to be over, modify the file of parameters, wait the ... But imagine you have 30 viewpoints to generate, you do not want to spend hours in front of your computer, you want to specify once all your matchings, start the computation and concentrate on other tasks while the computer works; in other words, you need to batch the process.

If you are a computer programmer, you will be able to easily build a program that generates and runs MicMac with the good parameters; if not, MicMac offers a simple mechanism to do it. It is used in the file `Param-6-Ter.xml`.

First, the header is quite new:

```
<ParamMICMAC

  Subst="@##1"
  NameDecl="@##1"

  NumC="@5564"
  NumMin="@5564"
  NumMax="@5568"
>
```

The syntax is a bit strange, and dangerous, so you will have to follow it very carefully:

- `Subst="@##1"` means that the substitution mechanism is to activate;
- `NameDecl="@##1"` means that in the following lines, the attribute like `NumC="@XXX"` are name declarations;
- `NumC="@5564"` declares that `NumC` is a symbol, here it is given the value 5564; `NumMin="@5564"` declares a symbol `NumMin` ...
- be careful, the @ is very important, it allows the mapping substitution that will be described below.

With this syntax, each time a symbol is encountered in the file between \${ and } it will be replaced with its value; for example, `IMG_${NumC}_Masq.tif` is replaced with `IMG_5564_Masq.tif`.

A first use of symbol declaration is to make easier file editing and modification. In file `Param-5-Ter.xml` the number 5564 appears 3 times as the master image; so if you change the master image, you would have to change the three occurrences; it is easy to imagine that it can be error prone. With the symbol declaration, you only need to change the line `NumC="@5564"`.

However, this does not solve the batching problem. For this, we have to look at `Section_WorkSpace`:

```
<MapMicMac>
  <ActivateCmdMap> true </ActivateCmdMap>
  <ModeCmdMapeur>
    <CM_One> unused but mandatory </CM_One>
  </ModeCmdMapeur>

  <CMVA>
    <NV> NumC 5588 </NV> <NV> NumMin 5588 </NV> <NV> NumMax 5592 </NV>
  </CMVA>
  <CMVA>
    <NV> NumC 5581 </NV> <NV> NumMin 5581 </NV> <NV> NumMax 5585 </NV>
  </CMVA>
  <CMVA>
    <NV> NumC 5564 </NV> <NV> NumMin 5564 </NV> <NV> NumMax 5568 </NV>
  </CMVA>

</MapMicMac>
```

The `ModeCmdMapeur` is mandatory but useless here. The `ActivateCmdMap` set to `true` means that will effectively use the mechanism. Then:

- the list of elements `<CMVA>` will be mapped;
- for each `<CMVA>`, MicMac is run once;
- for each of these runs the `<NV>` elements of `CMVA` are interpreted as pair of strings, the first pair being the name of a symbol, and the second one the value given to the symbol in the call to MicMac;
- for example, in the third line, `NumC 5581` means that MicMac will be called with the value of `NumC` equal to 5581 (instead of 5564).

With this mechanism, we can now compute the three depth maps in one command. However, for now, if we want to generate point clouds, we still have to run three times `Nuage2Ply`. If you have many commands, this can be cumbersome. The `<PostProcess>` section offers an alternative:



Figure 5.13: The 3 separate point clouds

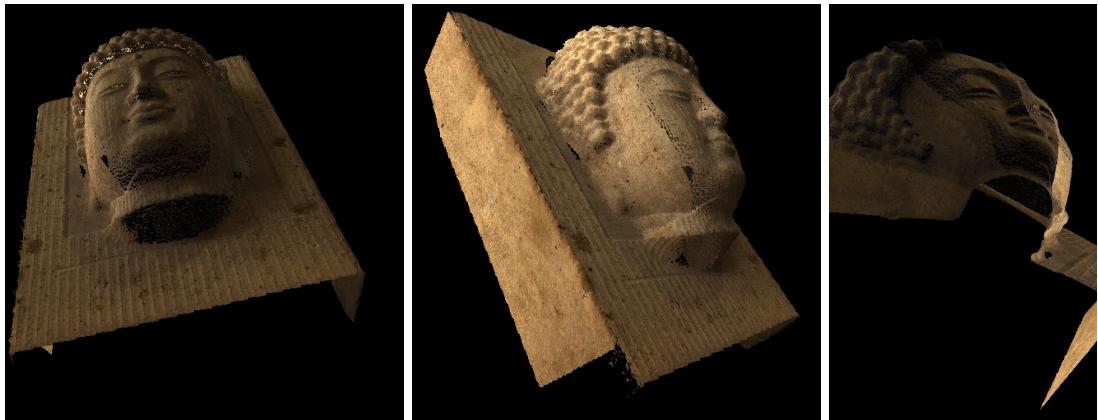


Figure 5.14: Some views of the merged point cloud

```

<PostProcess>
  <OneCmdPar>
    <OneCmdSer> echo ${ThisDir} </OneCmdSer>
    <OneCmdSer> ${MMDir}bin/Nuage2Ply ${ThisDir}MEC-6-Im/NuageImProf_Geom-Im-${NumC}_Etape_5.xml Attr=${ThisDir}IMG_${NumC}-RGB.tif </OneCmdSer>
  </OneCmdPar>
</PostProcess>

```

TO DO \${MMDIR} TO DO \${MMNbProc}

When it is present, the `<PostProcess>` is quite easy, each `<OneCmdSer>` is executed sequentially as a command line. Of course the symbol substitution has been carried out before. It is often necessary to give the absolute name of file (and not the name relative to the directory of the data set), so for this, there is a special symbol `${ThisDir}` whose value is the path of the current directory.

Once you have executed MicMac with `Param-6-Ter.xml`, you should obtain three point clouds in `ply` format. Figure 5.13 shows a separate view of each of these point clouds. As all the orientation have been computed with Apero in the same system coordinates, if the three point clouds are opened simultaneously, you will obtain a global modelization of the Buddha. Figure 5.14 shows some snapshots generated with MeshLab once they have been globally loaded.

## 5.5 Hidden part and individual ortho images generation

In the current version of `MicMac`, the formula used for multi-correlation is by default the average of cross correlation on all pairs of images (see 29.5.1.2 for variants). When there are parts of the ground that are not seen on some images, it would be more appropriate to compute the average only on images that see some part of the ground. We can compute hidden part if we know a *3D* model of the scene. Obviously, we don't have a "perfect"

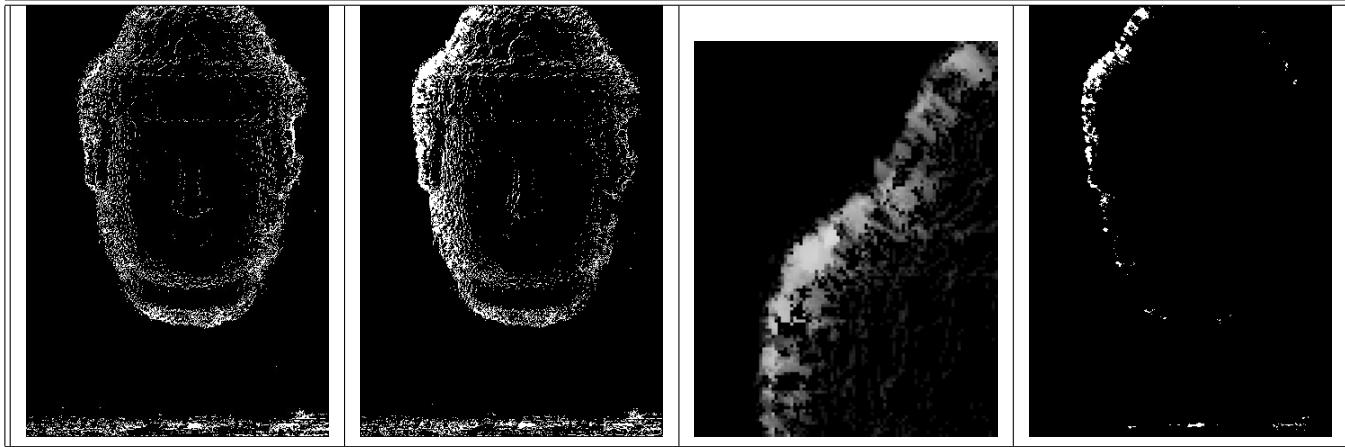


Figure 5.15: Images 1 and 2: examples of binary hidden part; Image 3: gray function of hidden part; Image 4: thresholded hidden part function

3D model<sup>11</sup> but with multi-resolution approach, there are cases where the solution of previous steps may be an acceptable approximation of the 3D for hidden part computation. The file `Param-7-Ter.xml` gives an example of how this functionality can be used in **MicMac**:

```
...
<EtapeMEC>
    <DeZoom>      8      </DeZoom>
</EtapeMEC>
<EtapeMEC>
    <DeZoom>      4      </DeZoom>
    <GenerePartiesCachees>
        <ByMkF> true </ByMkF>
        <SeuilUsePC> 3 </SeuilUsePC>
    </GenerePartiesCachees>
</EtapeMEC>
<EtapeMEC>
    <DeZoom>      2      </DeZoom>
    <GenerePartiesCachees>
        <ByMkF> true </ByMkF>
        <SeuilUsePC> 3 </SeuilUsePC>
    </GenerePartiesCachees>
</EtapeMEC>
...

```

At resolution 4 and 2, there is a `<GenerePartiesCachees>`, when this tag is encountered **MicMac** will compute<sup>12</sup> at the end of the processing step, for each of the input image, an image superposable to the DTM; this image indicates for each pixel if this pixel is visible or hidden. For example, you can see in directory `MEC-7-Ter`, that a file `MasqPC_LeChantier_Num2.IMG_5588.tif` has been created.

As the DTM is not perfect, if the result is used without further precaution, every small noise inside it can cause a small hidden part in the computation. The two right images of figure 5.15 show the mask that would result from a direct use of hidden part: there would be everywhere in the image a lot of isolated pixels supposed to be hidden;

In fact the notion of hidden part is not a binary notion, intuitively it's easy to understand that some points are weakly hidden by a relief of a few pixels, and that others are strongly hidden by high reliefs. **MicMac** computes a value that implements this intuitive notion, the third image of figure 5.15 shows in grayscale such a quantitative notion of hidden part. We can now describe the subtags of `<GenerePartiesCachees>`:

- `<SeuilUsePC>` is used as threshold for the quantitative hidden image that has been computed by **MicMac**; points over this threshold will not be used in the correlation computation; the fourth image of figure 5.15 shows the thresholded image;
- `<ByMkF>` just means that you allow **MicMac** to do parallel computing (in fact it should always be true, except when debugging);

While **MicMac** is computing the hidden part, it has in fact almost all the necessary information to compute the geometric deformation required to make ortho-images. This is the reason why the tag `<MakeOrthoParImage>`, allowing this ortho computation, is a sub-tag of `<GenerePartiesCachees>`:

11. otherwise, we would not loose our time to make work irritating correlation tools ...  
 12. using a classical Z-buffer algorithm

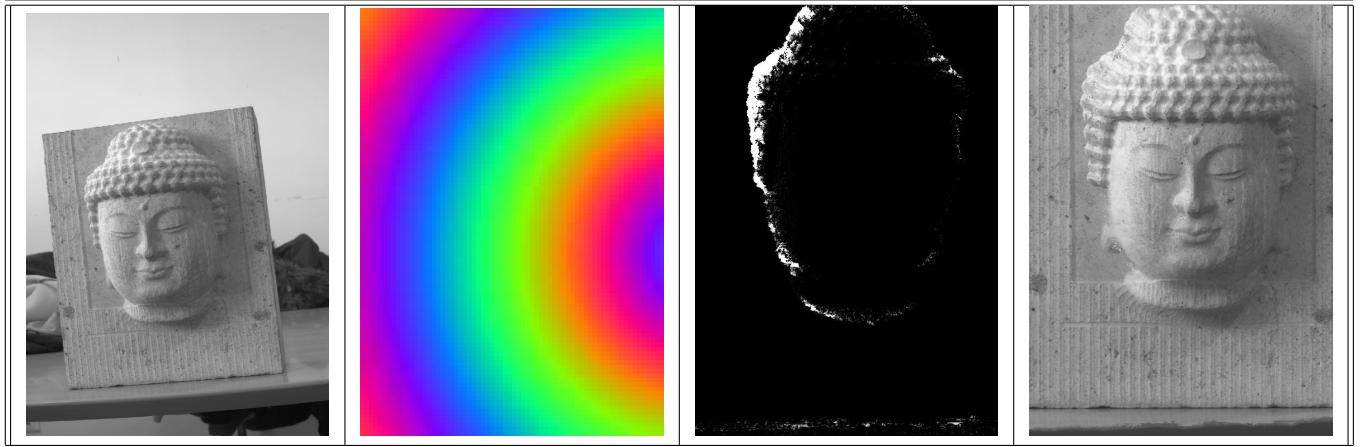


Figure 5.16: An image (5592) and the results of individual ortho generation: its incidence image, its hidden part image, its single ortho-image

```

<EtapeMEC>
    <DeZoom >      1      </DeZoom>
    <GenerePartiesCachees >
        <ByMkF> true </ByMkF>
        <SeuilUsePC> 3 </SeuilUsePC>
        <KeyCalcPC> NKS-Assoc-AddDirAndPref@ORTHO@PC_ </KeyCalcPC>
        <MakeOrthoParImage >
            <KeyCalcInput> Key-Assoc-Id </KeyCalcInput>
            <KeyCalcOutput > NKS-Assoc-AddDirAndPref@ORTHO@Ort_ </KeyCalcOutput>
            <KeyCalcIncidHor> NKS-Assoc-AddDirAndPref@ORTHO@Incid_ </KeyCalcIncidHor>
            <!-- Par rapport a la resolution des image R1 -->
            <ResolIm> 1.0 </ResolIm>
            <CalcIncAZMoy> true </CalcIncAZMoy>
        </MakeOrthoParImage>
    </GenerePartiesCachees>
</EtapeMEC>

```

Some comments on this part of the file:

- this is made at a final step of the computation, so the hidden part will not be reused in correlation, however they are computed because they are useful when creating the mosaic (see 7.2).
- the input image to ortho-rectify is specified by `<KeyCalcInput>`; here it is the same image as the correlation image, so we used the special key `Key-Assoc-Id`;
- `NKS-Assoc-AddDirAndPref` is a parametrized key that generates name under a different directory and add pref (see the definition in `DefautChantierDescripteur.xml`); it is used several times because we need to generate several files for each image;
- the generated files are:
  - the individual ortho-image, its name is specified by `<KeyCalcOutput>`;
  - the hidden part image specified by `<KeyCalcPC>`;
  - the incidence image specified by `<KeyCalcIncidHor>`; this image will be used when creating the mosaic, between the different individual ortho-images, select the best one (i.e. incident ray is closest to vertical);

The figure 5.16 shows the input and output of this ortho-image generation.

### 5.5.1 Other Options

## 5.6 2D Matching

This section will describe the possibility offered when the orientations are unprecise and the matching needs to be 2-dimensional.

### 5.6.1 Pure Image 2D Matching

Epipolare bi-dim avec Boudah

### 5.6.2 Ground Image 2D Matching



# Chapter 6

## A Quick Overview of Orientation

This chapter will give a global overview of the necessary tools to orientate a set of images.

### 6.1 General Organization of Apero

#### 6.1.1 Input and Output

Apero is a software for computing orientation, position and calibration of a set of images compatible with a set of observations, these observations being noisy and (hopefully) highly redundant. The objective is that you provide the observations, the confidence you have in these observations, and that Apero computes the most compatible solution with your observations. As the computation of the optimal solution may be complicated, you will sometimes have to help and guide Apero.

Basically, the observations can be:

- tie points; in fact, for object modelization it is current that the only observations are homologous points;
- ground points;
- observations about the position of the projection center (with GPS embedded in the camera);
- results of previous computation;

The confidence you have in the observations is communicated to Apero via weighting functions.

The output of Apero is represented by the computed values of the unknowns, which can be:

- values of orientation and position of camera;
- values of internal calibration;
- values of ground points coordinates if they are unknown;
- values of parametric surfaces (still largely undeveloped).

#### 6.1.2 General Strategy

The problem of finding the solution to orientation is classically divided into two main sub-problems:

- computation of initial values, hopefully closed to the "real" solution, with direct algorithms;
- once a reasonable solution is obtained, refine this solution by iteration of :<sup>1</sup>
  - linearization of the equations;
  - solving the redundant linearized equations by some kind of weighted least mean squares;

The first step is the most difficult because there is no known algorithm for computing directly a set of orientations compatible with tie points. There exist several algorithms for computing the relative orientation between pair of images and these algorithms have to be called many times to build step by step the global orientation of a large block of images; although there is no better known solution, this incremental approach has two consequences:

- it creates the need of a "good" ordering of the images; this order can be specified by the user, or you can let Apero use its own heuristics;
- because the direct algorithms do not use all the information, there is an accumulation of errors that can lead to divergence when the refinement step begins; to avoid this problem, we have to mix the initialization phase with the refinement phase;

The difficulty of the orientation problem is that, to our knowledge, there is no universal global solution; there is a bag of elementary solutions, and in most cases, it is possible to assemble these small pieces of solution to build a coherent puzzle. Practically, there is a lot of current cases, where the orientation can be solved with a limited

---

1. in photogrammetry, this classical second step has some particularities and it is called bundle adjustment

number of predefined strategies, and some few cases that require a very special tuning. What I tried to do when conceiving Apero was to have a tool that simultaneously gives a fine control to the user, for solving some difficult cases<sup>2</sup>, and that on the other hand offers the possibility to have predefined files for most current configurations.

The interface around Apero generally lets the user select one of the predefined configuration. Here is an example of predefined strategy :

- Unknowns: position and orientation of each image, internal calibration common to all images; initially all the internal parameters are frozen;
- Add the first image, in arbitrary position;
- Iterate:
  1. choose the best images to add by computing stability estimator on the cloud of tie points;
  2. use the tie points to compute the initial value of the orientation of new image with direct algorithms;
  3. make one round of a bundle adjustment to avoid error accumulation;
- Release one by one the internal parameters in this order : distortion coefficient, focal length, distortion center, principal point; each time a parameter is released, make a bundle adjustment round;
- Make several rounds of bundle adjustment with more and more strict weighting on the residual of projection.

## 6.2 A First Example

At first, we will see examples where the ordering is fixed by the user; although this will be quite uncommon in practice case, it is an occasion to understand what it is done by Apero in the ordering process.

### 6.2.1 Introduction

The file `Apero-0.xml` is our very first example of using Apero. To keep it simple we have only two input images. We suppose the camera is already calibrated, and we try to compute the orientation of the second image relatively to the first. At the end we output the result in **XML files**. As in all relative orientation problems, the solution is undetermined up to a global rotation-translation and scaling, so we will have to take some action to avoid degeneracy. There are different ways to do it, here we will fix seven arbitrary values:

- fix the position and orientation of an arbitrary pose;
- fix the length of the base between two arbitrary poses.

The skeleton of this file is:

```
<ParamApero>
  <SectionBDD_Observation>
    ...
  </SectionBDD_Observation>

  <SectionInconnues>
    <CalibrationCameraInc> ... </CalibrationCameraInc>

    <PoseCameraInc> ... </PoseCameraInc>
    <PoseCameraInc> ... </PoseCameraInc>
  </SectionInconnues>

  <SectionChantier> </SectionChantier>
  <SectionSolveur> </SectionSolveur>

  <SectionCompensation>
    <EtapeCompensation> ... </EtapeCompensation>
    <EtapeCompensation> ... </EtapeCompensation>
    <EtapeCompensation> ...
      <SectionExport> ... </SectionExport>
    </EtapeCompensation>
  </SectionCompensation>
</ParamApero>
```

A few comments on the different sections:

- `<SectionBDD_Observation>` contains all the information relative to the observations, this is essentially a set of files Apero has to load;
  - `<SectionInconnues>` contains the declaration of the unknowns and the information necessary to their initialization;
- 
2. maybe after some testing

- `<SectionChantier>` and `<SectionSolveur>` contain some possible global fine controls of the process, essentially used for internal development;
- `SectionCompensation` contains information for controlling the optimization process, essentially weighting of equations, freezing de-freezing of unknowns and also generation of results in the included `<SectionExport>`.

### 6.2.2 Tie Points

Here, the observations are limited to a set of tie-points. The tie points we will use are located in the directory `Homol/`. Here, they have been generated using the tool `Tapioca` described in 3.3. If you take a look at the content of `Homol/` you will see that it contains many subdirectories; go, for example, in `PastisIMG_5588/`. It contains many files. Open, for example, `IMG_5589.txt`, you will see something like:

```
184.003000 196.441000 43.183200 190.975000
186.997000 222.827000 45.956300 214.604000
188.920000 255.417000 46.587800 245.838000
195.034000 340.312000 50.586800 326.498000
193.219000 246.304000 51.661800 237.330000
194.059000 258.730000 52.074900 249.161000
192.173000 196.095000 52.216800 189.605000
....
```

The structuring of the tie points is probably clear now: the file `Homol/PastisIMG_5588/IMG_5589.txt` contains the tie points between images `IMG_5588.txt` and `IMG_5589.txt`. This is an ASCII-file, each line contains a tie point in format  $x_1y_1x_2y_2$  where  $x_1y_1$  is in `IMG_5588.txt` and  $x_2y_2$  is in `IMG_5589.txt`. This structuring should make it easy, if you have a tie point generator preferred to `Tapioca`, to replace it by your own tie points.

Although this structuring is quite evident for human, it has to be precised when speaking to a computer.

### 6.2.3 Declaring the Observations

The observation section is:

```
<SectionBDD_Observation>
  <BDD_PtsLiaisons>
    <Id> Id_Pastis_Hom </Id>
    <KeySet> NKS-Set-Homol@txt </KeySet>
    <KeyAssoc> NKS-Assoc-CplIm2Hom@txt </KeyAssoc>
  </BDD_PtsLiaisons>
</SectionBDD_Observation>
```

Here, as said before, we only have tie-points observations. They are contained in tags `BDD_PtsLiaisons`. The meaning of the three tags of `BDD_PtsLiaisons` is:

- `Id` is a name, or identifier, that is given to this set of tie points, it will be used each time we will refer to this set; it is necessary, because in some special cases, you may have several sets of tie points<sup>3</sup> and you will need to indicate the set you are referring to;
- `KeySet` tells Apero which files of tie points are to be loaded, this is a key that refers to a set of files; it is detailed below;
- `KeyAssoc` describes to Apero the association that, given two image names, can be computed the tie points' file name, and the reverse association: given the tie points' file name, how the two images names can be computed; examples of names association have already been seen in 5.4.2 (local definition) and 5.4.5 (predefined parametrized key) here there are slight innovations and they are described below;

The value of `KeySet` is `NKS-Set-Homol@txt`. This is a predefined key, its definition can be found in `DefautChantierDescripteur.xml`:

```
<KeyedSetsOfNames >
  <IsParametrized> true </IsParametrized>
  <Sets>
    <PatternAccepteur> Pastis(.*)/(.*)\.(#2) </PatternAccepteur>
    <SubDir> Homol#1/ </SubDir>
    <NivSubDir> 2 </NivSubDir>
  </Sets>
  <Key> NKS-Set-Homol </Key>
</KeyedSetsOfNames>
```

This is a parametrized key. Here, the first argument, `#1`, is empty and the second argument, `#2`, is `txt`. It describes that the files are in the directory `Homol/`, in the sub-directories `Pastis.*/`, and they have `txt` as a suffix.

The value of `KeyAssoc` is `NKS-Assoc-CplIm2Hom@txt`. The definition in `DefautChantierDescripteur.xml` is:

- 
3. for example, a set computed automatically and a set created by operator
  4. directory include/XML-GEN/

```

<KeyedNamesAssociations>
  <IsParametrized> true </IsParametrized>
  <Calcs>
    <Arrite> 2 1 </Arrite>
    <Direct>
      <PatternTransform> (.*)%(.*) </PatternTransform>
      <CalcName> Homol#1/Pastis$1/$2.#2 </CalcName>
      <Separateur> % </Separateur>
    </Direct>
    <Inverse>
      <PatternTransform> Homol#1/Pastis(.*)/(.*)\.#2 </PatternTransform>
      <CalcName> $1 </CalcName>
      <CalcName> $2 </CalcName>
    </Inverse>
  </Calcs>
  <Key> NKS-Assoc-CplIm2Hom </Key>
  <SubDirAutoMake> Homol#1 </SubDirAutoMake>
  <SubDirAutoMakeRec> true </SubDirAutoMakeRec>
</KeyedNamesAssociations>

```

Although we have already seen some examples of key association, there are some interesting innovations:

- the tag **Arrite** which has the value 2, 1. This tag is ignored by the program for now, but it is useful for the user.  $S$  being the set of strings, this tag means that this key is a mapping  $S \times S \rightarrow S$ , because what we need is to transform the pair of names of images in the name of the associated tie points file;
- in **<Calcs>** there is a **<Direct>** and a **<Inverse>** part. This is because what we want here is a reversible mapping, **<Direct>** is a  $S \times S \rightarrow S$  mapping and **<Inverse>** has to be the  $S \rightarrow S \times S$  inverse mapping of **<Direct>**. Apero will need to get back the pair of images from the name of the tie points; of course, it is your responsibility to have the property that **<Inverse>** and **<Direct>** are inverse of each other;
- **<Direct>** specifies a  $S \times S \rightarrow S$  mapping. When it is used, for example, with **IMG\_5588.tif** and **IMG\_5589.tif**, the string **IMG\_5588.tif% IMG\_5589.tif** which matches the regular expression **(.\*)\.tif\%(.\*)\.tif**;
- **<Inverse>** specifies a  $S \rightarrow S \times S$  mapping. This is simply done by having several **<CalcName>**, the first **<CalcName>** specifies the first string, the second **<CalcName>** specifies the second string ...

#### 6.2.4 Declaring the Unknowns, Camera-calibration

The section **SectionInconnues** contains the declaration of the unknowns and their initialization. Here, the unknowns are camera calibration and camera poses.

In this file there is only one calibration. It is declared as:

```

<CalibrationCameraInc>
  <Name> TheKeyCalib </Name>
  <CalValueInit>
    <CalFromFileExtern>
      <NameFile> Calib-F050.xml </NameFile>
      <NameTag> CalibrationInternConique </NameTag>
    </CalFromFileExtern>
  </CalValueInit>
</CalibrationCameraInc>

```

The signification of the tags is:

- **<Name>** is an identifier that must be used when referring to this calibration<sup>5</sup>;
- **<NameFile>** is the name of the file where Apero will look for the initial value of calibration. Take a look at the file **Calib-F050.xml**, you will understand that it is a typical file for radial distortion model;
- **<NameTag>** is the name of the tag containing the calibration (may be useful, for example, if a file contains several calibrations).

#### 6.2.5 Declaring the Unknowns, Camera-poses

There are two pose declarations. The first one is:

```

<PoseCameraInc>
  <PatternName> IMG_5588.tif </PatternName>
  <CalcNameCalib> TheKeyCalib </CalcNameCalib>
  <PosValueInit>
    <PosId> ### </PosId>
  </PosValueInit>
</PoseCameraInc>

```

The signification of the tags is:

---

5. it would have been more homogeneous to call it **Id**

- **<PatternName>** is the pattern of the name of the camera. Here it creates a single camera, but more sophisticated regular expression may create multiple creations;
- **<CalcNameCalib>** defines the calibration associated to this pose, it must be an already created id;
- **<PosValueInit>** is the initialization part. It can contain different sub-tags, according to the selected initialization method;
- **<PosId>** means that we want an identity pose<sup>6</sup>; in fact, as it is the first image, the value being 100% arbitrary, identity is not a bad choice.

For the initialization of the second image, we must compute a rotation coherent with the tie points:

```

<PoseCameraInc>
    <PatternName>      IMG_5589.tif </PatternName>
    <CalcNameCalib>  TheKeyCalib </CalcNameCalib>
    <PosValueInit>
        <PoseFromLiaisons>
            <LiaisonsInit>
                <NameCam> IMG_5588.tif  </NameCam>
                <IdBD> Id_Pastis_Hom </IdBD>
            </LiaisonsInit>
        </PoseFromLiaisons>
    </PosValueInit>
</PoseCameraInc>

```

The signification of the new tags is:

- **<PoseFromLiaisons>** indicates that we initialize with tie points;
- **<LiaisonsInit>** indicates that we want to compute the orientation of this image, relatively to **IMG\_5588.tif**, the tie points being extracted from the **Id\_Pastis\_Hom** set.

## 6.2.6 Running the Compensation

### 6.2.6.1 What Is Done During the Compensation

There is no place here to explain in detail what is a bundle adjustment. We encourage the interested reader to consult a book on the topic. We just recall, quickly and approximately, the very basic necessary to understand the control mode detailed after.

The basic idea is:

- let  $R_i, C_i$  be the unknown rotations and position centers;
- let  $I_k$  be the unknown intrinsic parameters, with  $k = k(i)$ ;
- let  $P_l$  be the unknown ground points corresponding to tie points, and  $p_{l,m}$  the pixel position of  $P$  in the different image where it is seen;
- note  $\pi(R_i, C_i, I_{k(i)}, P)$  the projection function;

The orientation problem is to find  $R_i, C_i$  and  $I_k$  (and consequently  $P_l$ ) so that:

$$\forall l, m \ p_{l,m} = \pi(R_{i(l,m)}, C_{i(l,m)}, I_{k(i(l,m))}, P_l) \quad (6.1)$$

Generally the problem is redundant, and equation 6.1 cannot be satisfied completely, so we rather search for  $R_i, C_i, I_k, P_l$  minimizing the global energy:

$$res(l, m) = p_{l,m} - \pi(R_{i(l,m)}, C_{i(l,m)}, I_{k(i(l,m))}, P_l) \quad (6.2)$$

$$E = \sum_{l,m} ||res(l, m)||^2 \quad (6.3)$$

If the equations were linear, and the observations robust, it would be easy: just run a least mean squares minimization. As the problem is not linear, Apero classically linearizes the equation for you<sup>7</sup>. This is fine but:

- this means that you will need several iterations;
- there is a risk of divergence if you start too far from the solution;
- there is a risk of divergence if the system has to estimate poorly determined unknowns, so you will need a way to freeze, temporarily or not, some unknowns.

The classical problem with this quadratic expression is that it gives far too much importance to outliers. It is then much more robust to do what is called  $L_1$  minimization:

$$E = \sum_{l,m} ||res(l, m)|| \quad (6.4)$$

---

6. rotation matrix=identity, center =(0,0,0)

7. so this is a Gauss-Newton descent

As  $L_1$  minimization is hard to compute directly and very large system<sup>8</sup>, we solve a weighted least mean squares:

$$E = \sum_{l,m} \rho(res(l,m)) ||res(l,m)||^2 \quad (6.5)$$

The weighting function  $\rho$  plays a very important role, in the solution, and the convergence itself. Theoretically  $\rho(x) = \frac{1}{x}$  leads to  $L_1$ , a solution already very, very close to the solution. Practically it can lead very easily to divergence ... So it is generally preferred to have parametrized weighting functions:

$$\rho(x)_{B,\sigma} = if(x > B) 0 else \frac{1}{\sqrt{1 + (\frac{x}{\sigma})^2}} \quad (6.6)$$

The problem is to get the good value of  $B$  and  $\sigma$ ? When we are far from the solution, as the residual may be high,  $B$  and  $\sigma$  must have high values; then when we get closer and closer, they can have stricter values. The question is how do you know that you are far or close to the solution. Unfortunately, I do not know the good answer.

To conclude, the problem of choosing which parameters are to be frozen or defrozen, and which weighting function is to be used, is a delicate problem and it will be your problem. Apero will not make the choice for you, but it will offer, I hope, a fine control on all these options. This said, what is done during compensation is no more than a weighted constraint Gauss-Jordan minimization using *your* parameters for weighting and constraining.

#### 6.2.6.2 Structure of SectionCompensation

As it can be seen in file `Apero-0.xml`, the structure is the most complicated:

- the compensation is made of several `<EtapeCompensation>`, for each one you redefine completely the observations and their weighting in the `<SectionObservations>`;
- each `<EtapeCompensation>` is made of several `<IterationsCompensation>`;
- for each `<IterationsCompensation>`, Apero will run a Gauss-Jordan iteration:
  - computation of linearised of equation and accumulation in the weighted least squares system;
  - resolution of least squares system;
  - update of unknowns with previous solutions;
- for each `<IterationsCompensation>`, you have the opportunity to freeze or free the unknowns you wish.

#### 6.2.6.3 Handling the Constraints

In this example, the constraints are handled very simply, because they are set at the first iteration and never changed after. The first `<IterationsCompensation>` is:

```
<SectionContraintes>
  <ContraintesCamerasInc>
    <Val> eAllParamFiges </Val>
  </ContraintesCamerasInc>

  <ContraintesPoses>
    <NamePose> IMG_5588.tif </NamePose>
    <Val> ePoseFigee </Val>
  </ContraintesPoses>

  <ContraintesPoses>
    <NamePose> IMG_5589.tif </NamePose>
    <PoseRattachement> IMG_5588.tif </PoseRattachement>
    <Val> ePoseBaseNormee </Val>
  </ContraintesPoses>
</SectionContraintes>
```

The meaning of the constraints is:

- `<ContraintesCamerasInc>` generates constraints on calibration, by default it applies to all created calibrations; the enumerated value `eAllParamFiges` means that all parameters are frozen; this a reasonable choice because
    1. we have used an already calibrated camera for initialization;
    2. however, we cannot make robust self-calibration with two images;
  - `<ContraintesPoses>` generates constraints on poses, `<NamePose>` indicates to which pose it applies;
    - for the first image, the value `ePoseFigee` means that the pose is completely frozen; this freezes the translation and the rotation of the block (six of the seven arbitrary parameters);
- 
8. and  $L_1$  is not the best solution

- for the second image, the value `ePoseBaseNormee` means that in the evolution of `IMG_5589.tif` we freeze the length of the base to `IMG_5588.tif`; this will freeze the scale of the block (and so the seventh arbitrary constraint);

All the constraints are enumerated values, they are of type `eTypeContrainteCalibCamera`, defined in `ParamApero.xml`. Formal description of enumerated values can be found in 10.3.2.

#### 6.2.6.4 Using Weighted Observations

In each `<EtapeCompensation>` there is a `<SectionObservations>`. Here the observations are limited to one set of tie points, the structure is the same for the three `<EtapeCompensation>`. There is just the weighting that evolves. The first one is:

```
<ObsLiaisons>
  <NameRef> Id_Pastis_Hom </NameRef>
  <Pond>
    <EcartMesureIndiv> 1.0 </EcartMesureIndiv>
    <Show> eNSM_Paquet </Show>
    <EcartMax> 100 </EcartMax>
    <SigmaPond> 5 </SigmaPond>
    <ModePonderation> eL1Secured </ModePonderation>
  </Pond>
</ObsLiaisons>
```

The meaning of the tags is:

- `<Show>` controls the level of detail for the printed messages; it is an enumerated value;
- `<EcartMesureIndiv>` is useless here, but when there are several observations, possibly heterogeneous, it will allow to control the respective influence of each set of measurements, each weight being multiplied by  $\frac{1}{E_{C^2}}$ ;
- `<ModePonderation>` is an enumeration controlling which weighting function is used; the choice `eL1Secured` corresponds to the function of the equation 6.6;
- `EcartMax` and `SigmaPond` are the two parameters of equation 6.6; `EcartMax` for  $B$ , `SigmaPond` for  $\sigma$ ;

#### 6.2.6.5 Weighting of homogeneous and heterogeneous observations

In this section, we present the different ways to cope with the weighting of observations used in the bundle adjustment. Prior to the description of the functioning of weighting in `Apero` (and `Campari`), we review briefly the different categories of measurements/observations that can be used in the bundle bloc adjustment (in french, the *compensation par faisceaux*).

In many cases, the bundle block adjustment is performed based exclusively on one type of observations: **tie points**. Sets of tie points are abundant observations that are generated automatically by means of image feature descriptor, as e.g. SIFT (tool `Tapioca`). The quality of the set of tie point measurements is related to the images themselves: their quality and overlap, the complexity of the structure of the objects visible on the images, etc. For example, what may be a good tie point is a tie point located on a motionless object corner, detected on more than two images. A low quality tie point may be a tie point detected on the edge of a shadow, as the interval of time between two image shots involves a motion of the illumination source (the sun) and thus of the shadow as well. Presence of outliers in the tie point measurements is not scarce. The great advantage of tie points, which are measured on the images (image measurement), is the fact that they are numerous and automatically measured.

In addition, it is appropriate to provide to the bundle adjustment some observation about the **camera inner orientation**, i.e. the calibration. Contrary to tie points, observations related to camera calibration are not numerous, as the calibration is shared by many images. Camera calibration is crucial in photogrammetry [MPD 2011, personal communication]. There is no weighting of calibration parameters implemented in `Apero`.

For aerial acquisition, embedded GPS and possibly inertial measurement unit (IMU) can deliver an initial image exterior orientation (i.e. image position,  $X$ ,  $Y$ ,  $Z$  and image orientation *omega*, *phi*, *kappa*). Bundle adjustment process can integrated the **embedded GPS** observations ( $X$ ,  $Y$ ,  $Z$ ). These measurements may have different level of accuracy, depending on the GPS itself. There is obviously a maximum of one GPS measurement per image. These observations are not as numerous as the tie point for example (often hundreds of tie points per image), but for very large image block (hundreds or thousands of images), these measurements can be abundant. Depending on the GPS system, lever arm and boresight bias may be present, i.e. the difference in the GPS position and the camera position. In addition, badly synchronization between the GPS and the camera triggering may result in a delay (see section 13.3.4.2 for an example in Unmanned Aerial Systems photogrammetry).

The last category of observation that can be supported in the bundle adjustment implemented in `Apero` are **ground control point**. Ground control points (GCPs) are of capital importance, because they are in many cases the best way to reference accurately the model in a consistent coordinate system. GCPs are features clearly recognizable on the images for which coordinates have been measured in the real world. As GCPs are mainly used in aerial photogrammetry, we refer to as *ground measurement* the coordinate measurement of these features

in the physical world. Ground measurements of GCPs may be the planimetric and altimetric position ( $X$ ,  $Y$  and  $Z$ ) or only the planimetric or the altimetric position ( $X$  and  $Y$  or only  $Z$ ). Generally, ground measurement of GCPs are of good accuracy and the number of observations is about a dozens of GCPs. To be supported in the bundle adjustment, GCPs have additionally to be measured on the images. Each GCP have to be identified (generally manually) on at least 2 images, resulting in *image measurements* similar than tie point.

There are 3 ways to deal with weighting of measurements in **Apero**. These three techniques are cumulative and optional, offering to the user a flexible (and complicated) tool. The different way of weighting may be perceive as redundant.

The first way can be utilized either to weight observations of different categories, or to weight observations from a single category. It is based on the uncertainty of the measurement and can be controlled by means of the tag `<EcartMesureIndiv>` and more specifically for the ground measurements of the GCPs through the tag `<Incertitude>`. Each observation is multiplied by  $\frac{1}{Ec^2}$ ,  $Ec$  being the uncertainty of the measurement.  $Ec$  units are the same than the measurement: for tie point,  $Ec$  is in pixels. For embedded GPS, the uncertainty is in meter is the coordinate system is metric. This weighting has no effect if the value of  $Ec$  is equal to 1. It is possible to assign an uncertainty value for tie points, embedded GPS observations and GCPs image measurements. In addition, it is possible to assign a value of uncertainty for ground measurement of GCPs, individually (the tag `<Incertitude>` pertaining to a single GCP). For a set of embedded GPS measurements, it is currently not possible to define an uncertainty for a single embedded GPS observation: the whole set of GPS measurements share the same uncertainty. To conclude, setting uncertainty is useful for controlling the respective influence of each set of measurements, and can be utilized additionally for differentiating different level of accuracy in GCP ground measurements.

The second way to control the weighting of observation is to limit the influence of a specific category of measurement. Let's consider an example in which we use a bundle block adjustment supporting, in addition to tie points, both embedded GPS and ground control points. Considering this fictitious example, the number of observation per category is the following: tie points are the more abundant (thousands, e.g. 25k), followed by embedded GPS (hundreds, e.g. 200 images) and eventually by GCPs (dozens, e.g. 15 GCPs with a total of 75 image measurements, each GCP being visible on 5 images). Without any weighting, tie points would absolutely lead the solving of the bundle adjustment. In order to limit the impact of the set of tie points measurements, a threshold can be set, referred to as  $Nbmax$  (tag `<Nbmax>`). This threshold limit the influence of the set of  $x$  observations below the influence of a set of  $Nbmax$  observations by means of the following weighting function:

$$\frac{\left(\frac{x*Nbmax}{x+Nbmax}\right)}{x} \quad (6.7)$$

For a threshold value of  $Nbmax$  equals to 100, each tie point measurement of the set of 25 000 observation ( $x$ ) is multiplied by 0.00398 (equation 6.7, illustrated on figure 6.2.6.5). The weight of the whole set of tie points is  $\frac{x*Nbmax}{x+Nbmax}$ , either 99.6. Theirs influence will thus will be limited, given more importance to the other categories of observations (embedded GPS and GCPs). In a similar way, impact of the embedded GPS data or of the GCP image measurements can be control with the same weighting function, provided that a threshold of  $Nbmax$  is set for these observation categories. On the other hand, this is not possible to limit the number of GCP ground measurement with this weighting function. Indeed, **Apero** is right in assuming that the number of GCP ground measurement is never exceeding. In our example, there is a large number of embedded GPS measurements ( $x = 200$ ). We may want to limit their impact in a similar way, setting  $Nbmax$  to 50 for e.g. The weight of the whole set of embedded GPS observations is thus reduced to a value of 40. Weights for tie points, embedded GPS, GCP ground measurements and GCP images measurement are thus 99.6, 40, 15 and 75 respectively.

The third and last way to control the weighting of a measurement is to link the weight of an observation to its residual in the bundle block adjustment. The approach is based on the assumption that a set of observation of a specific category is made up of measurements characterized by different levels of accuracy. The goal of this weighting function is to give more weight to observations with a high level of accuracy and less weight to measurement with a low level of accuracy. In addition, a filter is implemented in order to exclude measurements that seem to be outliers or to have a very low level of accuracy. This weighting function has been previously described in this document and is illustrated on figure 6.2.6.5. On this figure, two weighting functions are compared, the weighting function *L1Secured* (equation 6.6) and *L1* ( $\rho(x) = \frac{1}{x}$ ). The behavior of *L1* for very low residual values is unappropriated, as it gives too much weight on observations presenting such a residual. This is the reason why *L1Secured* is more appropriated, but have the drawback to require the parameterization of two parameters, **EcartMax** ( $B$ ) and **SigmaPond** ( $\sigma$ ). To illustrate the functioning of the weighting function *L1Secured*, let's consider an example of a set of tie points, among them three tie points *TP1*, *TP2* and *TP3*. *L1Secured* is parameterized, for the tie points measurements, in the same way than in the figure 6.2.6.5, with **SigmaPond** equals 1 and **EcartMax** equals 3 pixels. In the bundle adjustment, for one of the numerous iteration, *TP1* has a reprojection error (residual) of 1.32 pixels. Its weight for the following iteration is then 0.604. *TP2* shows a better result of a reprojection error of 0.45 pixel. Its weight for the next iteration is thus 0.912. Regarding *TP3*, its reprojection error is quite high, about 3.3 pixels. As this residual exceeds the threshold **EcartMax**, this tie

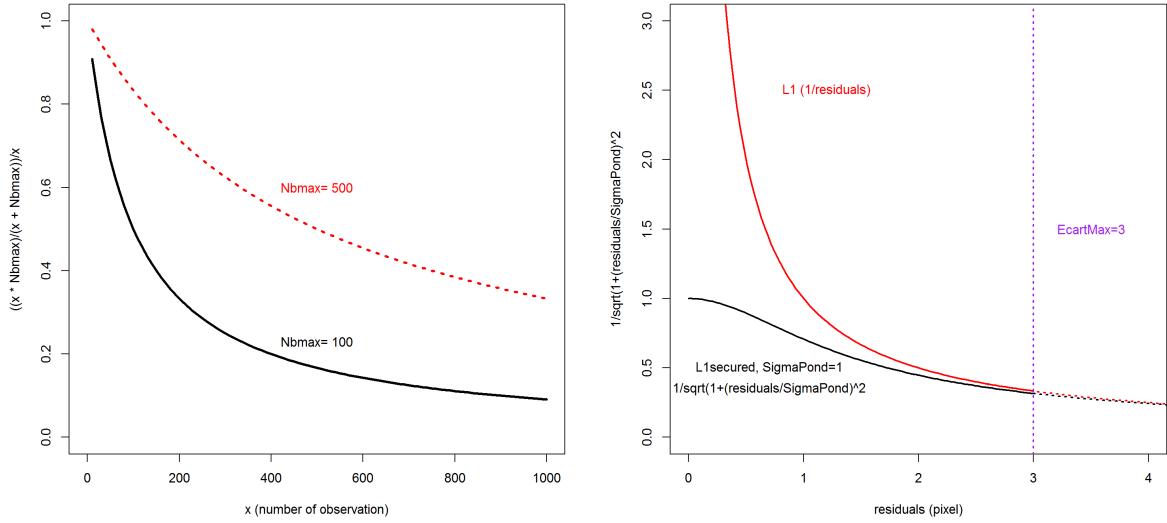


Figure 6.1: Visual representation of 2 weighting functions. Left, figure 6.2.6.5: weighting heterogeneous observation set with threshold  $Nbmax$  (equation 6.7). The goals is to limit the impact of large set of observation as tie points, in order to give more weight to other kind of observation as ground control points. Right, figure 6.2.6.5: weighting observation by their residuals with  $L1Secured$  (equation 6.6). The objective is to reduce the weight of observation presenting a high residuals (e.g. reprojection error for a tie points) and to filter the outliers.

point is considered as an outliers and will not be taken into account in the following adjustment iteration.

## 6.2.7 Understanding the Message

Run now the command line:

```
bin/Apero ../micmac_data/ExempleDoc/Boudha/Apero-0.xml
```

On the terminal, you will see a lot of messages:

```
BEGIN Pre-compile
BEGIN Load Observation
BEGIN Init Inconnues
NUM 0 FOR IMG_5588.tif
NUM 1 FOR IMG_5589.tif
BEGIN Compensation
BEGIN AMD
AMD::NB= 4
END AMD
END AMD
RES:[IMG_5588.tif] ER2 1.04921 Nn 100 Of 4335 Mul 0 Mul-NN 0 Time 0.199447
RES:[IMG_5589.tif] ER2 1.05125 Nn 100 Of 4327 Mul 0 Mul-NN 0 Time 0.164516
| | RESIDU LIAISON MOYENS = 1.05023 pour Id_Pastis_Hom
--- End Iter 0 ETAPE 0

RES:[IMG_5588.tif] ER2 0.308341 Nn 100 Of 4335 Mul 0 Mul-NN 0 Time 0.160701
RES:[IMG_5589.tif] ER2 0.314634 Nn 100 Of 4327 Mul 0 Mul-NN 0 Time 0.160834
| | RESIDU LIAISON MOYENS = 0.311503 pour Id_Pastis_Hom
--- End Iter 1 ETAPE 0

.....
RES:[IMG_5588.tif] ER2 0.124029 Nn 99.9769 Of 4335 Mul 0 Mul-NN 0 Time 0.160253
RES:[IMG_5589.tif] ER2 0.13504 Nn 99.9769 Of 4327 Mul 0 Mul-NN 0 Time 0.159912
| | RESIDU LIAISON MOYENS = 0.129651 pour Id_Pastis_Hom
--- End Iter 2 ETAPE 2
```

Until the line `END AMD` there is nothing very interesting, there are messages about initialization process, for internal purpose. The lines beginning with `RES:` may be useful for interpreting the results:

- [IMG\_5588.tif] is obviously the name of the image;
- [ER2 0.143439] is the square root of the weighted average of quadratic residuals;
- [Nn 99.9769] is the percentage of residuals that are under **EcartMax**; it should be over 95%, or else it may signify that you have got residuals just because you have thrown away high residuals!
- Of 4327 is the number of tie points;
- Mul 0 is the number of multiple points<sup>9</sup>; here, of course, with two images there are none;
- Mul-NN 0 is the number of multiple points having residuals under **EcartMax**;
- Time 0.159912 is the time of computation, internal purpose mainly.

### 6.2.8 Storing the Results

By default, Apero does not save any result. If you want the result of your computation to be saved, you must create a **<SectionExport>** inside the **EtapeCompensation**. Your report result will be executed at the end of the iterations. Here is the example of **Apero-0.xml**:

```
<SectionExport>
    <ExportPose>
        <PatternSel> (*.*) .tif </PatternSel>
        <KeyAssoc> NKS-Assoc-Im2Orient@${AeroOut} </KeyAssoc>
        <AddCalib> true </AddCalib>
        <NbVerif> 10 </NbVerif>
        <TolWhenVerif> 1e-3 </TolWhenVerif>
    </ExportPose>
</SectionExport>
```

Here we make an export only for the poses. In **ExportPose** we have:

- **<PatternSel>**, a regular expression filtering the name of the pose we want to export;
- **<KeyAssoc>**, a key referencing an association that, for an image name, will compute the name of the file where storing the orientation; the same key will be directly reusable in **MicMac** for loading;
- **<AddCalib>** indicates that, in each file, we also want to add the internal calibration.

Now take a look at the directory **..//micmac\_data/ExempleDoc/Boudha/Ori-Test-0**.

## 6.3 More Examples

### 6.3.1 Adding More Images

In the file **Apero-1.xml**, we compute the orientation of a block of 5 images. The main difference with **Apero-0.xml** is the added initialization of the three supplementary images:

```
<PoseCameraInc>
    <PatternName> IMG_559[0-2].tif </PatternName>
    <CalcNameCalib> TheKeyCalib </CalcNameCalib>
    <PosValueInit>
        <PoseFromLiaisons>
            <LiaisonsInit>
                <NameCam> IMG_5588.tif </NameCam>
                <IdBD> Id_Pastis_Hom </IdBD>
            </LiaisonsInit>
            <LiaisonsInit>
                <NameCam> IMG_5589.tif </NameCam>
                <IdBD> Id_Pastis_Hom </IdBD>
            </LiaisonsInit>
        </PoseFromLiaisons>
    </PosValueInit>
</PoseCameraInc>
```

The line **PatternName** adds three images, **IMG\_5590.tif**, **IMG\_5591.tif**, **IMG\_5592.tif**. For these images, two images are given for initialization:

- **IMG\_5588.tif**, like before, is used to compute the initial position using direct algorithm; however, with only one image, there is an ambiguity on the length of the base, which is perfectly normal for the second image, but not for the following ones;
- that is why **IMG\_5589.tif** is given, so that Apero can also resolve the base ambiguity in initialization.

As there are more than two images, the information on multiple points becomes relevant:

```
RES:[IMG_5588.tif] ER2 0.179343 Nn 99.8955 Of 5739 Mul 4434 Mul-NN 4430 Time 0.361473
RES:[IMG_5589.tif] ER2 0.175283 Nn 99.9625 Of 5333 Mul 4081 Mul-NN 4079 Time 0.339617
RES:[IMG_5590.tif] ER2 0.171174 Nn 99.9401 Of 5009 Mul 3604 Mul-NN 3602 Time 0.311397
RES:[IMG_5591.tif] ER2 0.162117 Nn 99.9807 Of 5187 Mul 3718 Mul-NN 3717 Time 0.32216
RES:[IMG_5592.tif] ER2 0.169253 Nn 99.9196 Of 4977 Mul 3827 Mul-NN 3826 Time 0.316134
```

---

9. points seen in 3 or more images

The export section is slightly different, we store the internal calibration in a separate file and, instead of copying in each external orientation files, we add a reference to this file:

```

<SectionExport>
    <ExportPose>
        <PatternSel> (*.tif </PatternSel>
        <KeyAssoc> NKS-Assoc-Im20rient@${AeroOut} </KeyAssoc>
        <AddCalib> true </AddCalib>
        <NbVerif> 10 </NbVerif>
        <TolWhenVerif> 1e-3 </TolWhenVerif>
        <FileExtern> ${OutCalib} </FileExtern>
        <FileExternIsKey> false </FileExternIsKey>

    </ExportPose>
    <ExportCalib>
        <KeyAssoc> ${OutCalib} </KeyAssoc>
        <KeyIsName> true </KeyIsName>
    </ExportCalib>

</SectionExport>

```

The new tags are:

- **FileExtern** is the name of the calibration file; for the value we use a symbol defined at the beginning of the file;
- **FileExternIsKey** means that the value **FileExtern** is the exact string to include, or else it would be used as a key for computing the value (useful when multiple calibrations are used);
- **ExportCalib** generates a calibration file, **KeyAssoc** is the name and **KeyIsName** indicates that it is not to be interpreted as a key.

Another difference is in constraint handling:

```

<ContraintesPoses>
    <NamePose> 0 </NamePose>
    <Val> ePoseFigee </Val>
</ContraintesPoses>

<ContraintesPoses>
    <NamePose> 1 </NamePose>
    <Val> ePoseBaseNormee </Val>
    <PoseRattachement> 0 </PoseRattachement>
</ContraintesPoses>

```

Instead of referencing the images by their name, they are referenced by a number (indicating their order in the initialization process). As these constraints are totally arbitrary, this can be more convenient as this part of the file stays valid whatever may be the set of images.

### 6.3.2 Computing the Internal Parameters

Until now, we have used a file **Calib-F050.xml** that contains a "good" value of internal calibration, so we have not tried to re-evaluate this value. However, it will happen very often that you do not have such a value of internal calibration, and you will need to compute it by yourself. The file **Apero-2.xml** and **Apero-2-Bis.xml** introduce how internal parameters of calibration can be re-estimated in Apero.

The file containing the initial value of calibration is **CalibInitNoDist.xml**. Take a look at this file, you will see it is easy to build such a file:

```

<ExportAPER0>
    <CalibrationInternConique>
        <KnownConv>eConvApero_DistM2C</KnownConv>
        <PP> 704 469 </PP>
        <Fx> 1956 </Fx>
        <SzIm> 1409 938 </SzIm>
        <CalibDistortion>
            <ModRad>
                <CDist> 704 469 </CDist>
            </ModRad>
        </CalibDistortion>
    </CalibrationInternConique>
</ExportAPER0>

```

Some comments:

- we choose here a radial model; this is generally a good idea when you do not have an explicit reason for selecting something more complicated;
- **SzIm** is the size of image; you can easily get it by using any image viewer;
- **PP** and **CDist** are principal points and distortion, by default initialize at the center of the image;

— F, the more complicated, you have to know (using xif meta data for example) the  $36mm$  equivalent, here it is a  $50mm$ , so just do  $1409 * \frac{50}{36} = 1956$ .

With this initialization, *Apero-2.xml* just do the same thing as *Apero-1.xml*. If you run *Apero-2.xml* you will see that, with this poor calibration, the final residuals are higher. In *Apero-2-Bis.xml*, you can see the differences in the second *EtapeCompensation*:

```
<EtapeCompensation>
...
<IterationsCompensation> </IterationsCompensation>

<IterationsCompensation>
    <SectionContraintes>
        <ContraintesCamerasInc>
            <Val> eLiberte_DR1      </Val>
        </ContraintesCamerasInc>
    </SectionContraintes>
    <Messages> LIB DR1 </Messages>
</IterationsCompensation>
<IterationsCompensation> </IterationsCompensation>

<IterationsCompensation>
    <SectionContraintes>
        <ContraintesCamerasInc>
            <Val> eLiberte_DR2      </Val>
        </ContraintesCamerasInc>
    </SectionContraintes>
    <Messages> LIB DR2 </Messages>
</IterationsCompensation>
<IterationsCompensation> </IterationsCompensation>

<IterationsCompensation>
    <SectionContraintes>
        <ContraintesCamerasInc>
            <Val> eLiberteFocale_1      </Val>
        </ContraintesCamerasInc>
    </SectionContraintes>
    <Messages> LIB FOCALE </Messages>
</IterationsCompensation>
<IterationsCompensation> </IterationsCompensation>
...
</EtapeCompensation>
....
```

- the value *eLiberte\_DR1* releases the first parameter of central distortion;
- the value *eLiberte\_DR2* releases the first and second parameters of central distortion; there exists until *eLiberte\_DR5*;
- the value *eLiberteFocale\_1* releases the focal length;
- the tag *<Messages>* generates a message on the terminal.

Run *Apero-2.xml* and *Apero-2-Bis.xml*, you will notice the significant difference in the final values of the residuals.

### 6.3.3 Automatic Image Ordering

Until now, we have orientated a limited set of images. Now, suppose we want to orientate all the 26 images of Buddha, we could use the previous mechanism specifying for each image on which image we want to base the initialization. Of course, we should do it in the right order ... It would be quickly unmanageable.

*Apero-3.xml* is a first example. We will let Apero build for us the tree of image initialization. The first image is initialized as before, the difference is in the second set:

```
<PoseCameraInc>
    <PatternName> IMG_[0-9]{4}.tif  </PatternName>
    <CalcNameCalib> TheKeyCalib </CalcNameCalib>
    <MEP_SPEC_MST>
        <Show> true </Show>
    </MEP_SPEC_MST>

    <PosValueInit>
        <PoseFromLiaisons>
            <LiaisonsInit>
                <NameCam> #### </NameCam>
                <IdBD> Id_Pastis_Hom </IdBD>
            </LiaisonsInit>
        </PoseFromLiaisons>
    </PosValueInit>
</PoseCameraInc>
```

Here we use the `IMG_[0-9]4.tif` to specify that we want to load all the gray images of Buddha. Then, we use the tag `MEP_SPEC_MST`. That means that we let Apero build automatically the initialization tree. Remark that in `LiaisonsInit`, the `NameCam` is given as dummy value, in fact it will not be used.

Run this file, you will see that it converges to a reasonable value of residuals. However, if you look at the intermediate results, you will see:

```
...
| | RESIDU LIAISON MOYENS = 2.95062 pour Id_Pastis_Hom
--- End Iter 0 ETAPE 0
...
| | RESIDU LIAISON MOYENS = 12.6271 pour Id_Pastis_Hom
--- End Iter 1 ETAPE 0
...
| | RESIDU LIAISON MOYENS = 2.41104 pour Id_Pastis_Hom
--- End Iter 2 ETAPE 0
...
```

So you can see that the process goes through a phase of very high residuals. The problem is that the direct initialization is error prone; with many images there is an accumulation of error that can lead to a solution far from the optimal one. Here it has no big consequences, but this may be a source of divergence.

To decrease this risk, Apero proposes a mechanism where you can mix bundle adjustment with initialization: each time you have made a given progress in initialization process, you ask to run the bundle adjustment. You can see the slight differences in `Apero-3-Bis.xml`:

```
...
<PoseCameraInc>
...
<InitNow> false </InitNow>
<MEP_SPEC_MST>
    <Show> true </Show>
</MEP_SPEC_MST>
...
</PoseCameraInc>
...
<SectionCompensation>
    <EtapeCompensation>
        <IterationsCompensation>
...
            <Pose2Init>
                <ProfMin> [2,4,6] </ProfMin>
                <Show> true </Show>
            </Pose2Init>
        </IterationsCompensation>
    </EtapeCompensation>
</SectionCompensation>
```

The first difference is in `<PoseCameraInc>`, the tag `<InitNow>` value is `false`; this blocks the initialization. Then in the first `IterationsCompensation` we have `<Pose2Init>`, when Apero gets this tag, it will do that:

- let us call depth of an image its distance to the first image in the initialization graph;
- `IterationsCompensation` is run once with all the images having a depth  $\leq 2$ , then once with all the images having a depth  $\leq 4$  ... then once with all the image having a depth  $\leq 10$  ... until all the images are initialized.

Run `Apero-3-Bis.xml`, you will see that the residuals do not go through a high value phase.

Another way to avoid divergence is to compute a approximate calibration with a small set of convergent images, and then use this result as initial value in the global set. Run for example `Apero-3-Ter.xml`.

## 6.4 Geo-referencing

### 6.4.1 External Initialization

Until now, we have always build the orientation "from scratch". This way is fine when you use "small" terrestrial acquisition. However, there are several cases where you would like to start with some pose, having already a given initial value:

- when managing large and complicated acquisition, you may have different resolutions of images. It will currently be a good strategy to orientate first the low resolution images and, in a second step, orientate the high resolution images starting from the already orientated images;

- when you have an instrumental system (IMU), that gives reasonable initial values.

The `Apero-4.xml` file shows how known orientations can be used as initial values:

```

...
<SectionBDD_Observation>
...
    <BDD_Orient>
        <Id> Or-Init </Id>
        <KeySet> NKS-Set-Orient@${AeroIn} </KeySet>
        <KeyAssoc> NKS-Assoc-Im20orient@${AeroIn} </KeyAssoc>
    </BDD_Orient>

</SectionBDD_Observation>

<SectionInconnues>
...
    <PoseCameraInc>
        <PatternName> IMG_[0-9]{3}[02468]\.tif </PatternName>
        <CalcNameCalib> TheKeyCalib </CalcNameCalib>
        <PosValueInit>
            <PosFromBDOrient> Or-Init </PosFromBDOrient>
        </PosValueInit>
    </PoseCameraInc>
...
</SectionInconnues>
...

```

First, the data containing the observation on initial value has to be declared for loading in the `<SectionBDD_Observation>`. This declaration is similar to many observation<sup>10</sup>:

- in `<Id>`, it is declared the identifier that will be used when referring to this data;
- `<KeySet>` contains the key describing a set of files;
- `<KeyAssoc>` contains the key describing the association between a name of an image and a name of an orientation file;
- with  `${AeroIn}` being equal to `-Test-3`, we will load results of `Apero-3.xml`.

For initializing the cameras, we use `<PosFromBDOrient>` with the corresponding Id. This example is quite artificial, because we use this initialization with the image having an even number, the odd number being initialized, as before, by algorithms. However, it shows what is the mechanism that could be used in a two step orientation with different resolutions of images.

#### 6.4.2 Scene-based Orientation

There are occasions where, although there is no "real" geo-referencing information available, one would like to compute an orientation which is coherent with some physical constraint:

- use some part of the scene, known to define a horizontal plane, to obtain an orientation where the *OZ* axis is on the "real" vertical (in fact orthogonal to this plane);
- use a line of the scene, known to be on a given direction, to obtain an orientation where the *OX* axis is pointing North;
- use an object of known size to set the scale of the model (until now, all scaling were totally arbitrary).

File `Apero-5.xml` illustrates these three operations. The first operation is `<BasculeOrientation>`. In this operation, a set of 3D points known to be coplanar are specified, a "best" plane *P* is fitted on these points, then one of the rotation-translation transforming the plane *P* to the plane *Z = 0* is applied:

```

<BasculeOrientation >
    <PatternNameEstim> IMG_5588.tif </PatternNameEstim>
    <ModeBascule>
        <BasculeLiaisonOnPlan >
            <EstimPl>
                <KeyCalculMasq> NKS-Assoc-AddPost@_MasqPlan </KeyCalculMasq>
                <IdBdl >Id_Pastis_Hom </IdBdl>
                <Pond>
                    <EcartMesureIndiv> 1.0 </EcartMesureIndiv>
                    <Show> eNSM_Paquet </Show>
                    <NbMax> 100 </NbMax>
                    <ModePonderation> eL1Secured </ModePonderation>
                    <SigmaPond> 2.0 </SigmaPond>
                    <EcartMax> 5.0 </EcartMax>
                </Pond>
            </EstimPl>
        </BasculeLiaisonOnPlan>
    </ModeBascule>
</BasculeOrientation>

```

The meaning of the tags above is the following:

- `<PatternNameEstim>` specifies the images  $I_k$  that are to be used to select the 3D points;

---

10. BDD\_PtsLiaisons has already been seen

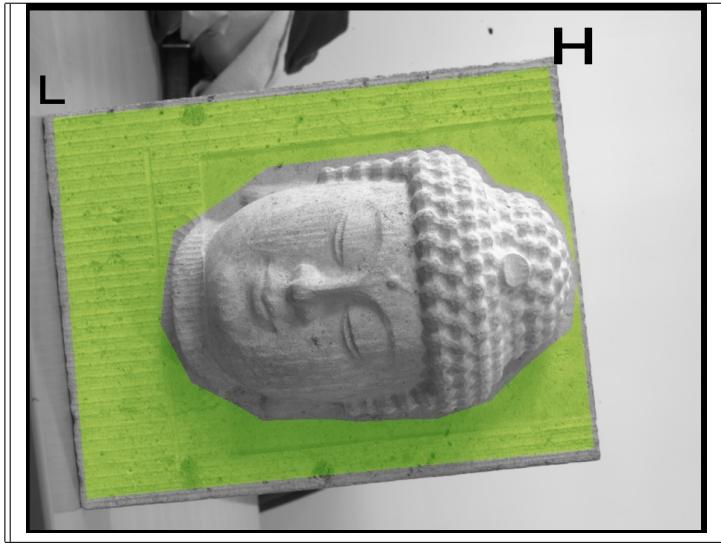


Figure 6.2: Bas Relief Plane mask used ...

- for each image  $I_k$ , KeyCalculMasq is used to compute the name of a file mask  $M_k$  (so be careful when setting <PatternNameEstim> that all corresponding masks exist);
- each tie points  $Q$  of IdBd1 is selected if, for at least one image,  $p$  pf  $Q$ ,  $p$  correponds to one of the  $I_k$ , and  $p$  is in the corresponding mask  $M_k$ ;  $Q$  is given a weighting by the reprojection ponderation given by <Pond> (of course if the weight equals 0,  $Q$  is not selected); for each selected  $Q$ , the 3D point computed by ray-intersection (using current values of pose) is added to the point cloud defining the plane.

For "small" canvas, it will be generally sufficient to have only one mask for setting the vertical. However, with linear acquisition, if you want higher precision, it may be a good idea to have at least two masks, one at the beginning and one at the end.

After <BasculeOrientation>, the vertical has a physical meaning, but the orientation is completely arbitrary inside this plane. With the tag <FixeOrientPlane>, we can set the "horizontal" orientation:

```
<FixeOrientPlane>
  <ModeFOP>
    <HorFOP>
      <VecFOH>
        <Pt> 1088 81 </Pt>
        <Im> IMG_5588.tif </Im>
      </VecFOH>
      <VecFOH>
        <Pt> 1167 779 </Pt>
        <Im> IMG_5588.tif </Im>
      </VecFOH>
    </HorFOP>
  </ModeFOP>
  <Vecteur> 1 0 </Vecteur>
</FixeOrientPlane>
```

The idea is to select a line, and to set its horizontal orientation. For the most general case, the line would have to be specified in 3D by giving two stereoscopic points. This general case will be implemented later. In the present case, the line is supposed to be horizontal<sup>11</sup> and two monoscopic points are sufficient, they are transformed into 3D points by giving them the altitude 0. With this specification, Apero will apply a rotation around the  $OZ$  axis so that  $\vec{p_1p_2}$  is aligned with  $\vec{V}$ <sup>12</sup>.

To set the arbitrary scale of the model, a 3D segment of a known size must be specified. To specify a 3D segment, two stereoscopic points must be specified:

```
<FixeEchelle>
  <ModeFE>
    <StereoFE >
      <HomFE>
        <P1>193 184 </P1>
        <Im1> IMG_5588.tif </Im1>
```

11. i.e. the points have the same  $Z$  after the <BasculeOrientation> has been run

12. with  $p_1$  and  $p_2$  given by <Pt> and  $\vec{V}$  given by <Vecteur>

```

<P2> 362 112 </P2>
<Im2> IMG_5577.tif</Im2>
</HomFE>
<HomFE>
  <P1> 262 869 </P1>
  <Im1> IMG_5588.tif </Im1>
  <P2> 473 803 </P2>
  <Im2> IMG_5577.tif</Im2>
</HomFE>
</StereoFE>
</ModeFE>
<DistVraie> 0.2 </DistVraie>
</FixeEchelle>

```

In the example above, the scaling has been set by specifying that the height of the rectangle is 20cm. For this, the upper left and lower left corner has been keyed: each `<HomFE>` is the stereoscopic specification of given ground point.

Note that, in the current state of **Apero**'s development, all these transformations are not made for metrology, they can be used for changing to an orientation having some sense, but they cannot be used during compensation phase.

### 6.4.3 Using Embedded GPS

This section treats the case where you have some direction information about the position of the camera projection center. A current case is with aerial acquisition where most systems have a GPS embedded and synchronized with the camera. Of course, for **Apero** it is just an external information about the summit, and it does not matter if it comes from GPS of any other system. For simplicity, we will call it GPS information.

You will generally use this GPS information in two steps:

- transform the purely relative orientation computed from tie points to a first geo-referenced orientation;
- use this GPS information in the compensation phase by adding a observation equation to the global system; of course, as compensation makes linearization, and that linearization requires that you are already close to the solution, you cannot use GPS for compensation before you have made the global transformation described above (that is why it is done in two steps).

First, you will have to attach a center information to each image with which you want to use this mechanism. File **Apero-6.xml** illustrates how these operations are made with **Apero**. In this file, we start from existing relative orientation (read from file), immediately make the global transformation, and then begin the compensation. The new parts are:

```

...
<BDD_Centre>
  <Id> Id-Centre </Id>
  <KeySet> NKS-Set-Orient@${BDDC} </KeySet>
  <KeyAssoc> NKS-Assoc-Im2Orient@${BDDC} </KeyAssoc>
</BDD_Centre>
...
<SectionInconnues>
  <PoseCameraInc>
...
  <IdBDCentre> Id-Centre </IdBDCentre>
...
  </PoseCameraInc>
</SectionInconnues>
...
<SectionCompensation>
  <EtapeCompensation>
    <IterationsCompensation>
      <BasculeOrientation>
        <AfterCompens> false</AfterCompens>
        <PatternNameEstim> .* </PatternNameEstim>
        <ModeBascule>
          <BasculeOnPoints>
            <BascOnCentre> </BascOnCentre>
            <ModeL2> true </ModeL2>
          </BasculeOnPoints>
        </ModeBascule>
      </BasculeOrientation>
    </IterationsCompensation>
...
  <SectionObservations>
...
  <ObsCentrePDV >
    <PatternApply> .* </PatternApply>
    <Pond>
      <EcartMesureIndiv> 1.0 </EcartMesureIndiv>

```

```

<Show> eNSM_Paquet      </Show>
      <ModePonderation> eL1Secured </ModePonderation>
    </Pond>
  </ObsCentrePDV>
...
</SectionObservations>
...
<EtapeCompensation>
</SectionCompensation>

```

The first innovation is the creation of data base of the center `<BDD_Centre>`. The mechanism for declaring a set of files, and an association between names is the same as usual; each file must contain a tag `<Centre>` containing a point in 3D, see for example `Ori-BDDC/Orientation-IMG_5564.tif.xml`.

Once the data has been loaded, it must be used to attach a center information to the images. This is done by `<IdBDCentre> Id-Centre </IdBDCentre>`. In case that not all the images have information center, the initialization will have to be split in two `<PoseCameraInc>`.

The global transformation is made by `<BasculeOrientation>` as in 6.4.2. The parameters are:

- `<PatternNameEstim>`, it indicates which image must be used. Be warned that, in the current version, an error will appear if images without attached center are selected. If this pattern does not select at least 3 images, an error will occur;
- `<AfterCompens>`, the value `false` specifies that the transformation must be made at the beginning of the `IterationsCompensation` including it. This is necessary to insure that the image are geo-referenced when the compensation begins;
- `<BascOnCentre>`, it has no argument, `Apero` knows that the center attached to the image must be used;
- `<ModeL2>`, it specifies that the global transformation must be made with least mean squares optimization.

The compensation on the center is made by `<ObsCentrePDV>`. For each selected pose, the following term is added to the global minimization:

$$\frac{p(|GPS_k - C_k|) * |GPS_k - C_k|^2}{Ecart^2} \quad (6.8)$$

Where:

- $p(|GPS_k - C_k|)$  is the usual weighting function; here with no parameters on sigma,  $p$  values 1; this is quite common if there is no outlier in GPS;
- $Ecart$  for `<EcartMesureIndiv>` allows to control the weighting between heterogeneous measurements (here, the relative importance between tie points and GPS).

Note that in this example, we do not use any constraint. This is not necessary for the poses, because the attach to GPS resolves the scale-translation-rotation ambiguities. This is not necessary for the internal calibration, because we start from a good initial value, and there is no risk to free all the parameters.

#### 6.4.4 Ground Control Points

For geo-referencing acquisition ground points are also interesting, they are generally more precise than embedded GPS and can be acquired in more general condition, on the other hand they are more expensive, time-wise, to acquire.

In this section we first describe how the information about ground point is organized in files so that `Apero` can use them, then we describe how they can be used in orientation computation.

##### 6.4.4.1 Ground Points Organization

The information about ground points will generally be stored in two files: one for the specification of the points themselves and one for their measurements in images. In the `Boudha` file take a look at `Dico-Appuis.xml` and `Mesure-Appuis.xml`. The file `Dico-Appuis.xml` is a list of declarations of ground points:

```

<?xml version="1.0" ?>
<Global>
  <DicoAppuisFlottant>
    <OneAppuisDAF>
      <Pt> 103 -645 5 </Pt>
      <NamePt>Coin-Gauche </NamePt>
      <Incertitude> 10 10 10 </Incertitude>
    </OneAppuisDAF>
...
    <OneAppuisDAF>
      <Pt> 370 -544 229 </Pt>
      <NamePt> Levre </NamePt>
      <Incertitude> 10 10 10 </Incertitude>
    </OneAppuisDAF>
  </DicoAppuisFlottant>
</Global>

```

When reading such a file, **Apero** expects to get one object **DicoAppuisFlottant** which contains a list of **<OneAppuisDAF>**. Each **<OneAppuisDAF>** specifies:

- the value **<Pt>** of the ground point;
- the value **<Incertitude>** of the uncertainty associated to each point; note that by convention, a value < 0 on one of the coordinates means that this coordinate is totally undefined;
- the value **<NamePt>**, which is an identifier; any string is OK, of course it must be unique.

The file **Mesure-Appuis.xml** contains examples of measurements on ground points:

```

<SetOfMesureAppuisFlottants>
    <MesureAppuiFlottant1Im>
        <NameIm> IMG_5576.tif </NameIm>
        <OneMesureAF1I>
            <NamePt> Coin-Gauche </NamePt>
            <PtIm> 672 218 </PtIm>
        </OneMesureAF1I>
        <OneMesureAF1I>
            <NamePt> Coin-Droite </NamePt>
            <PtIm> 731 748 </PtIm>
        </OneMesureAF1I>
    ....
    </MesureAppuiFlottant1Im>
...
</SetOfMesureAppuisFlottants>

```

The signification should be quite obvious:

- each **<MesureAppuiFlottant1Im>** contains all the measurements concerning the image **<NameIm>**;
- each measurement contains the name of the point and its position in the image.

#### 6.4.4.2 Using GCP

The file **Apero-7.xml** illustrates how GCP can be used:

```

<SectionBDD_Observation>
...
    <BDD_ObsAppuisFlottant >
        <Id> Id-Appui </Id>
        <KeySetOrPat> ^Mesure-Appuis.xml </KeySetOrPat>
    </BDD_ObsAppuisFlottant>
</SectionBDD_Observation>
<SectionInconnues>
...
    <PointFlottantInc>
        <Id> Id-Appui </Id>
        <KeySetOrPat> ^Dico-Appuis.xml$ </KeySetOrPat>
    </PointFlottantInc>
</SectionInconnues>
...
    <SectionCompensation>
        <EtapeCompensation>
            <IterationsCompensation>
                <BasculeOrientation>
                    <AfterCompens> false </AfterCompens>
                    <PatternNameEstim> .* </PatternNameEstim>
                    <ModeBascule>
                        <BasculeOnPoints>
                            <BascOnAppuis >
                                <NameRef> Id-Appui </NameRef>
                            </BascOnAppuis>
                            <ModelL2> true </ModelL2>
                        </BasculeOnPoints>
                    </ModeBascule>
                </BasculeOrientation>
            </IterationsCompensation>
        </EtapeCompensation>
    </SectionCompensation>
...
    <SectionObservations>
        <ObsAppuisFlottant>
            <NameRef> Id-Appui </NameRef>
            <PondIm>
                <EcartMesureIndiv> 1.0 </EcartMesureIndiv>
                <Show> eNSM_Paquet </Show>
                <NbMax> 100 </NbMax>
                <ModePonderation> eL1Secured </ModePonderation>
                <SigmaPond> 20.0 </SigmaPond>
                <EcartMax> 5000000.0 </EcartMax>
            </PondIm>
        </ObsAppuisFlottant>
    </SectionObservations>
</EtapeCompensation>
</SectionCompensation>
...

```

First, the GCP and their observations must be loaded:

- in the `<SectionInconnues>` they are declared as unknowns in `<PointFlottantInc>`, their initial value is contained in the file `<KeySetOrPat>`;
- in the `<SectionBDD_Observation>`, their observation in images is loaded;
- this is the value `Id=Appui` of identifier `<Id>` that makes the link between unknowns and their observations; it will be used further when referring to these GCP.

Once the GCP are loaded, they can be used, as GPS, for a global transformation or for compensation once the images are approximately geo-referenced:

- `<BascOnCentre>` is replaced with `<BascOnAppuis>`, the identifier of the GCP set being given as argument; for this operation, there must exist at least 3 GCP being measured in at least 2 images;
- `<ObsCentrePDV>` is replaced with `<ObsAppuisFlottant>`.

In the current version of **Apero**, GPS on summit and GCP cannot be mixed for the initial global transform. By the way, they can (and often, are) be mixed without inconvenient in the compensation phase.



# Chapter 7

## A Quick Overview of Other Tools

### 7.1 Developing raw and jpeg Images with Devlop

### 7.2 Making Ortho Mosaic with Porto

#### 7.2.1 Introduction

**Porto** is a tool for generating a complete mosaic from a set of single ortho images generated by **MicMac**. It is in a very basic state, a lot of improvement would be required. By the way, I think it can still be useful for several applications.

To generate the tool:

```
make -f MakeOrtho
```

**Porto** expects a parameter file like **MicMac** and **Apero**. It must contain a structure **CreateOrtho** as specified in **SuperposImage.xml**.

The **ExempleDoc/Boudha** data set contains an example of usage. Run it by typing:

```
bin/Porto ..//micmac_data/ExempleDoc/Boudha/ORTHO/Param-Porto.xml
```

#### 7.2.2 Input to Porto

The figure 5.16 presents the output of **MicMac** that is used as input to **Porto**. The input to **Porto** consists of:

- a global meta-data file specifying the geo-referencing of the DTM associated to the ortho;
- a set of individual ortho-images;
- a set of mask images specifying for each ortho which images are visible;
- a set of incidence images, specifying the priority;
- a set of XML meta-data associated to each image.

Figure 7.2.2 presents some individual ortho-images computed by **MicMac** on the Buddha data set. The figure 7.2.2 presents some details of the problems of each individual ortho-images. Figure 7.2.2 presents the incidence images.

In the example **Param-Porto.xml**, the section specifying these inputs is:

```
<SectionEntree>
  <FileMNT> ..//MEC-7-Ter/Z_Num5_DeZoom1_LeChantier.xml </FileMNT>
  <KeySetIm> NKS-Set-OfPattern@Ort_(.)\.tif </KeySetIm>
  <KeyAssocMetaData > NKS-Assoc-ChangPrefixAndExt@Ort_@tif@PC_@xml </KeyAssocMetaData>
  <KeyAssocNamePC > NKS-Assoc-ChangPrefixAndExt@Ort_@tif@PC_@tif </KeyAssocNamePC>
  <KeyAssocNameInch> NKS-Assoc-ChangPrefixAndExt@Ort_@tif@Incid_@tif </KeyAssocNameInch>
</SectionEntree>
```

The **<FileMNT>** is an XML-meta-data file containing information about the DTM produced by **MicMac**. Its tags have been described in 5.4.3. The ortho-mosaic produced by **Porto** will have the same geo-referencing as the DTM. The other arguments are keys for describing sets and associations. The functioning is quite classical now:

- **<KeySetIm>** describes the set of individual ortho images;
- **<KeyAssocMetaData>** is a key for computing the name of XML-Meta data from the name of individual ortho image;



Figure 7.1: Three of the five individual ortho images

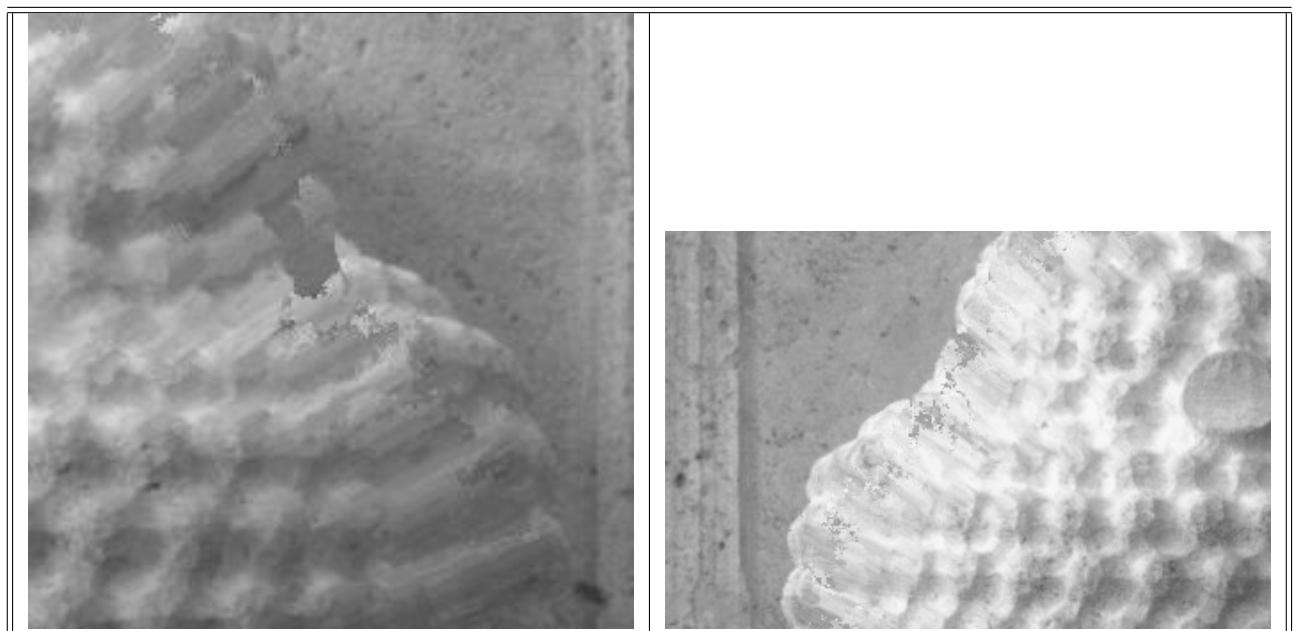


Figure 7.2: Zoom on some problems of individual ortho-image

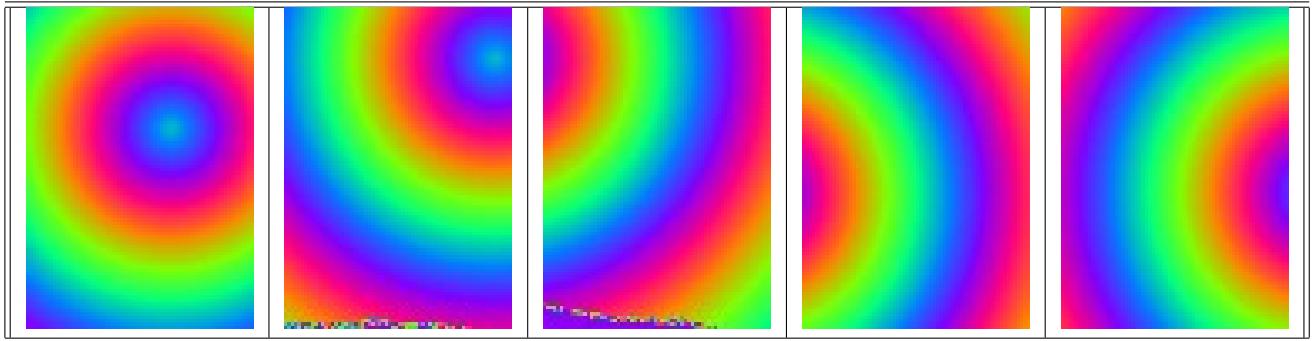


Figure 7.3: Incidence images computed for ortho priority

- <KeyAssocNamePC> is a key for computing the name of the hidden part image from the name of individual ortho image;
- <KeyAssocNameInch> is a key for computing the name of the incidence image from the name of individual ortho image;
- this is the same key, that is used with different parameters; this key allows to change the beginning (prefix) of a name and its extension.

Here is `PC_IMG_5588.xml`, an example of one of the meta-data file:

```
<MetaDataPartiesCaches>
  <Done>true</Done>
  <Offset>0 0</Offset>
  <Sz>676 960</Sz>
  <Pas>1</Pas>
  <SeuilUse>3</SeuilUse>
  <SsResolInch>10</SsResolInch>
</MetaDataPartiesCaches>
```

The important tags are:

- <Offset> and <Sz> specify the position of the individual ortho-photo on the DTM. Because, of course, in real example, the individual ortho-photo will be much smaller than the DTM or resulting mosaic, so it will be stored only a sub-rectangle of the global DTM;
- <SeuilUse> is a real value, use to threshold the images (`PC_IMG_5588.tif ...`) and defining which pixel must be used;
- <SsResolInch> is the resolution of incidence images (`Incid_IMG_5588.tif ...`) in fact as these images are very regular, it would be useless to store them at full resolution.

As this file have been generated automatically by `MicMac`, in most cases you will not need to modify it, but it is still good to understand a bit how things work, in case of problems ...

### 7.2.3 Output to Porto

Once all these inputs are known, the mosaicing algorithm is quite obvious:

- for each pixel of the output image:
  - select the unmasked image having the lowest incidence.

The parameters specifying the output are:

```
<SectionSorties>
  <SzDalle> 1000           </SzDalle>
  <SzBrd>    100            </SzBrd>
  <NameOrtho> Ortho-NonEg-Test-Redr.tif </NameOrtho>
  <NameLabels> Label-Test-Redr.tif</NameLabels>
</SectionSorties>
```

`NameOrtho` specifies the name of the ortho mosaic. `NameLabels` is an optional argument, when specified, `Porto` creates a label image indicating for each pixel which individual image is to be used for filling the geometry. Figure 7.2.3 presents the main results.



Figure 7.4: Computed label images and resulting ortho images

## 7.3 V.O.D.K.A.

### 7.3.1 Theory

The vignetting is an optical effect that results in a gradual radial drop-off in images (the corners of images are relatively darker than the center). The figure (7.5) shows an example of this effect.



Figure 7.5: Different vignetting effect

Read more about vignetting :

- <http://en.wikipedia.org/wiki/Vignetting>
- <http://fr.wikipedia.org/wiki/Vignettage>

### 7.3.2 Name and function

Vodka stands for "Vignette Of Digital Kamera Analysis".

This command estimates the vignetting effect for a set of images with the same aperture and focal length without the need of a laboratory setup (classically an integrating sphere). The vignette model that is used is an even 6th degree polynomial function centered on the middle of the image. With  $r$  the distance to the center of the image and  $\alpha$ ,  $\beta$  and  $\gamma$  the polynomial coefficients, we have :

$$V(r) = 1 + \alpha r^2 + \beta r^4 + \gamma r^6$$

The computation of the model uses a RANSAC based algorithm to solve the sets of equation of the following type (where  $G_i$  is the grey value of a tie point for the image  $i$  and  $r$  the distance to the center of the image) :

$$G_1 - G_2 = \alpha(G_2 * r_2^2 - G_1 * r_1^2) + \beta(G_2 * r_2^4 - G_1 * r_1^4) + \gamma(G_2 * r_2^6 - G_1 * r_1^6)$$

Between 9 and 30 points are randomly selected from the tie points and a solution is computed using least square matching. The solution is then awarded a score :

$$Score = \frac{P_{inliers}}{EMP}$$

Where:

- $P_{inliers}$  the percentage of the tie points complying with the model ( $\pm 2\%$ )  $\Rightarrow$  for good models, a value about 20% to 40% is expected.
- $EMP$  the mean error between the model and the points weighted by  $\min(r_1, r_2)$  to increase the importance of points away from the image center (and therefore more influenced by vignetting).

### 7.3.3 Input data

To use this command, a set of images with the same aperture and focal length, taken in a stable illumination setting is necessary. The command also requires the computation of tie points (through Tapioca).

Calling the command is done with the following command line (where ImagesPattern is the regular expression describing the set of images) :

*mm3d Vodka ImagesPattern*

Multiple datasets can be processed at once, the program would then sort the images in subsets with the same aperture and focal length and give a solution for each subset.

### 7.3.4 Output data

For each aperture/focal length combination in the input set, the commands create a floating point .tif file of the image's size named Vignette/Foc0000Dia111.tif, where 0000 is the focal length in millimeters and 111 the aperture times 10.

Corrected images can also be created if asked by the user, mostly for quality checking (see below).

### 7.3.5 Options

- *DoCor* (*bool*) toggle the creation of corrected images (Def=false)
- *InCal* (*string*) Name of folder with vignette calibration tif file (if previously computed)
- *InTxt* (*string*) True if homologous points have been exported in txt (Def=false)
- *Out* (*string*) Output folder (Default=Vignette)

### 7.3.6 How to use VODKA

Vodka is to be used to compute a vignette calibration. The best results are obtained with images where tie points are at different distances from the image center in the images that generated them, typically non-convergent images.

The output files should then be placed in a folder with other images taken with the same camera and the same aperture/focal length combination. The images that will be created in the Tmp-MM-Dir (and therefore used by every other commands) when the proper VODKA output files are present will be corrected using those files. If you want to use the VODKA results on the images used to compute it, you should place the VODKA output files in the images' directory and delete the Tmp-MM-Dir folder.

## 7.4 A.R.S.E.N.I.C

### 7.4.1 Name and function

ARSENIC stands for *Automated Radiometric Shift Equalization and Normalization for Inter-image Correction*.

This function corrects the key images used for coloring point clouds in order to have a smooth transition between sub-clouds of the same scene. It is designed to be used with the "GeomImage" correlation geometry.

**This function is not designed for the equalization of images prior to image mosaicing**

### 7.4.2 Input data

ARSENIC require a depth map for each image that will be equalized, computed with MICMAC/Malt. The dense radiometric tie point algorithm used in this program requires very well co-registered depth maps (or point clouds). Calling the command is done with the following command line (where ImagesPattern is the regular expression describing the set of images) :

*mm3d Arsenic ImagesPattern*

### 7.4.3 Output data

The output of this command is the corrected images, by default in a folder called Arsenic. These images can then be used to color a point cloud through Nuage2Ply.

### 7.4.4 Options

- *TPA* (*Tie Point Accuracy - int*) defines the precision threshold for the tie points (def=16, means  $\frac{1}{16}$  of pixel resolution)
- *ResolModel* (*int*) defines the resolution of the model to be used in the tie point computation (def=16 for DeZoom 16)
- *InVig* (*string*) defines a vignette calibration folder (if any)
- *Out* (*string*) defines the output directory
- *NbIte* (*string*) defines the number of iterations of the process (def=5)
- *ThreshDisp* (*string*) defines the disparity threshold between the tie points (Def=1.4 for 40%)

## 7.4.5 Algorithm

### 7.4.5.1 Tie point detection

In order to have a more accurate, denser and more interest-zone focused set of tie points, the tie points are extracted from the result of the dense correlation.

Every point situated in the mask of a key image is projected in 3D through the depth map generated by MICMAC/Malt, then reprojected in the other key images and finally projected again in 3D if the second projection resulted in a point in the secondary key image's mask. If the two 3D projections result in a similar point (the concept of similarity being defined through the TPA option : the maximum distance between two 3D projections that validates the points being  $PixelResolution/TPA$ ). For each validated point, the image coordinates of the point in the key image is recorded, as well as the factors  $K$  between the pixel values of each images for all channels :

$$K = (1 + G_j)/2 * G_i$$

With  $G$  a grey value,  $i$  the primary key image and  $j$  the secondary image that generated the tie point. A tie point is therefor an object with 5 values,  $Point = (X, Y, K_R, K_G, K_B)$ . The  $+1$  is a call to the initial value.

### 7.4.5.2 Equalization

In a first step, a correction factor is computed for each tie point with an inverse distance weighting and a self weighting value (the image is self influencing). For each radiometric channel of each point,  $j$ , we have the following formula ( $i$  is the tie point currently used in the interpolation):

$$Cor(TiePointj) = \sum_{i=1}^n \frac{(K_i)/2}{\sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2}}$$

This process is then iterated. A filtering system is yet to be developed to prevent radiometric outliers to push the model after too many iterations. An outlier is a point where  $K_r$ ,  $K_g$  or  $K_b$  is more than  $ThreshDisp\%$  different than the average value for the image considered.

The corrected tie points are then applied to a grid also through inverse distance weighting, the interpolated to the whole image through bilinear interpolation. The grid is computed by the formula bellow, with  $i$  the tie point index and  $(X, Y)$  the grid point's coordinates :

$$Cor(X, Y) = \sum_{i=1}^n \frac{K_i}{\sqrt{(X_i - X)^2 + (Y_i - Y)^2}}$$

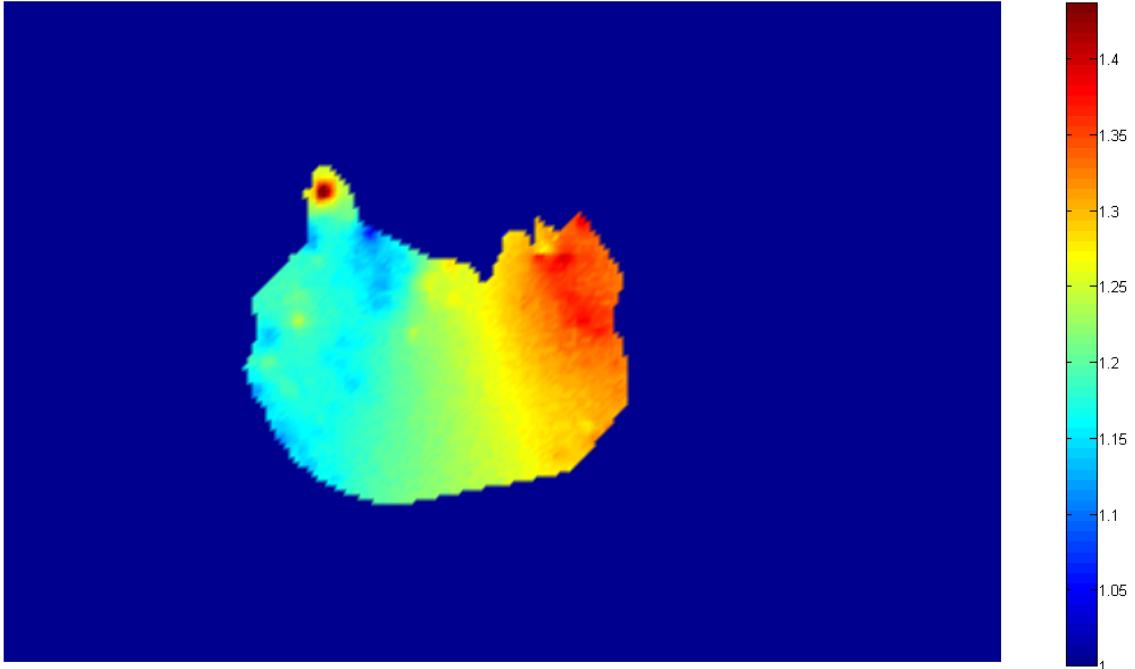


Figure 7.6: Example of a correction surface

### 7.4.6 How to use ARSENIC

Once the appropriate input data (see 7.4.2) is computed, the command can be run. The images produced in the output folder are to be used as "Attr" arguments in the "Nuage2Ply" command to produce equalized sub-point clouds.

## 7.5 Miscellaneous tools

### 7.5.1 TestLib PackHomolToPly

Tool use to display a tie points between 2 image in 3D. By combination with a mesh 3D, we can examine where tie point is founded on object surface. Inputs:

- 2 image that have homol pack - write as pattern (Ex: "image1.tif—image2.tif")
- Orientation of image
- SH : Select Homol folder of tie point. Default is "Homol/"
- color : select color in RGB od tie point. Not so necessary because with the viewer like Cloud Compare, user can change color also.

*Attention:* Output is PLY file format, store in **PlyVerify/** folder.

```
*****
*      Draw a pack of homologue in 3D PLY      *
*****
*****
* Help for Elise Arg main  *
*****
Mandatory unnamed args :
 * string :: {Pattern of images - 2 image have a pack}
 * string :: {Input Initial Orientation}
Named args :
 * [Name=SH] string :: {homol folder name - default = Homol}
 * [Name=color] vector<std::string> :: {[R,B,G] - default = [0,255,0]}
```

*Example:* mm3d MeshProjOnImg BIN\_010-011[1-4].\*.tif Ori-BasculeIGN-sec01-21/ meshSimple-Cut.ply

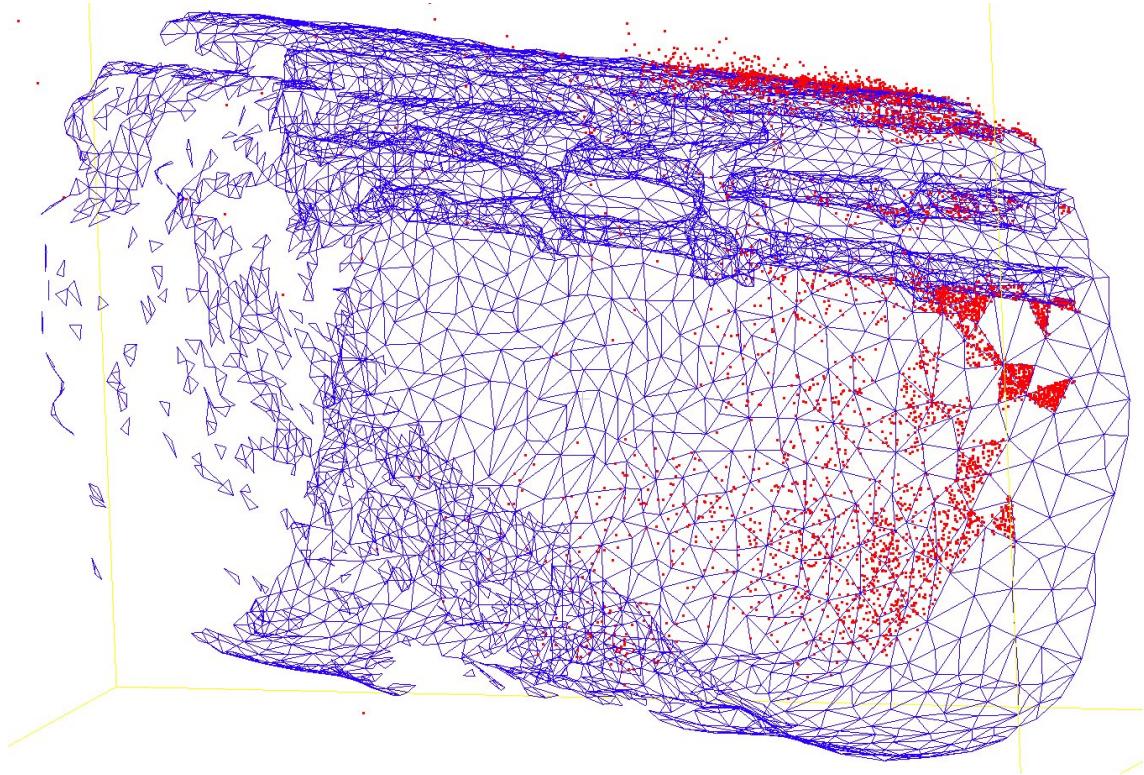


Figure 7.7: Draw tie point on mesh.

### 7.5.2 MeshProjOnImg

Tool use to back-project a mesh 3D on image 2D. Useful to specify which part of image is covered by the mesh. Inputs:

- Pattern of image to examine.
- Orientation of image
- Mesh file.
- zoomF : Zoom factor. Use to reduce image resolution to display. With image too big, display will cause program error. Default is 0.2, that's mean image is reduced to 1/5 of its size original.
- click : Draw each triangle on mesh on image by mouse click. Each click will draw 1 triangle of mesh.

*Attention:* Command will display all image of pattern at the same time, then reproject mesh on first image. You must click on first image to continue reproject mesh on second image and so on. When execute, command will read mesh file in and ask user if they like to display all the element in mesh file on the terminal. With "n", we can skip this step.

```
*****
*          Reproject mesh on specific      *
*****
*****
*  Help for Elise Arg main  *
*****
Mandatory unnamed args :
 * string :: {Pattern of images}
 * string :: {Input Initial Orientation}
 * string :: {path to mesh(.ply) file - created by Initial Ori}
Named args :
 * [Name=zoomF] REAL :: {1 -> sz origin, 0.2 -> 1/5 size - default = 0.2}
 * [Name=click] bool :: {true => draw each triangle by each click - default = false}
```

*Example:* mm3d MeshProjOnImg BIN\_010-011[1-4].\*.tif Ori-BasculeIGN-sec01-21/ meshSimple-Cut.ply

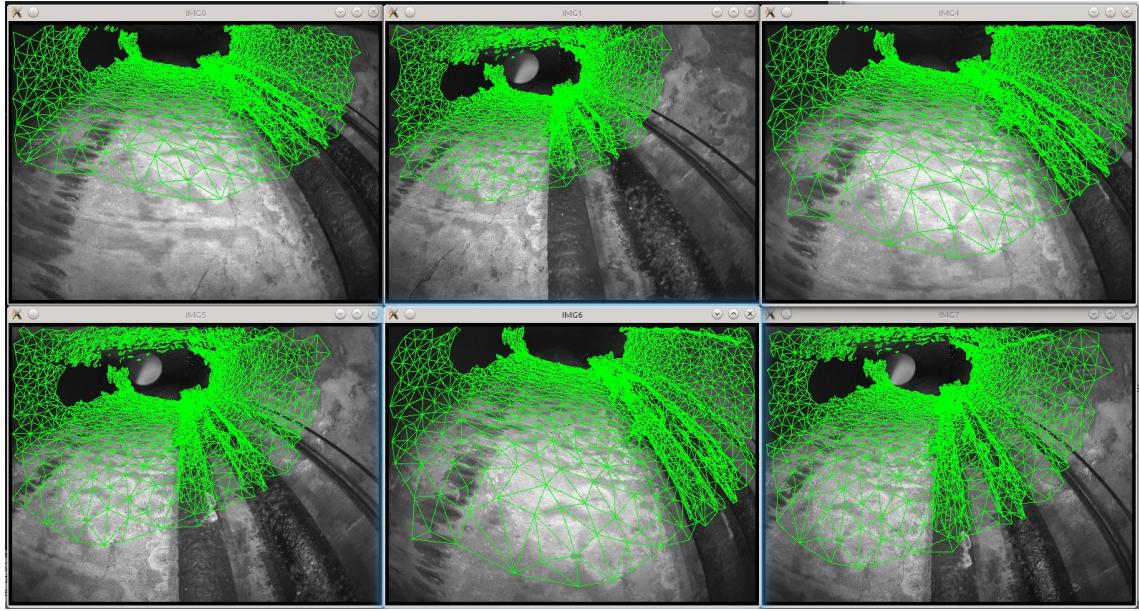


Figure 7.8: Result reproject mesh on image.

### 7.5.3 InitOriLinear

With **InitOriLinear**, you can initialize your orientation in case of your acquisition is a serie of image with a displacement linear. Tool compute a vector of displacement from a set of reference image, then using it to estimate position of other image in serie.

By using this tool to initialize orientation before compute aero with Tapas, computation speed is improved.

Inputs:

- Folder contain orientation of reference images
- Folder to output orientation file
- Pattern of image need to initialize. If your system have many camera on a bar rigide, you can give pattern correspondant with each camera, separate by " , ".
- Pattern of image use for reference. Idem if system have many camera. Order of camera must be the same.
- PatTurn : Turn image when direction of acquisition changed.(new section) (image of 1st camera)
- PatAngle : Turn angle correspondant with each turn image. Positive value for turn left, negative for turn right.
- mulF : multiplication factor to adjustment position between each section (use to spread or shorten distance between section)
- Axe : axe to turn around.

The output is orientation files initialized. Can be use as an initialized solution for **Tapas** with option **InOri**

```
mm3d InitOriLinear -help
*****
* X : Initial      *
* X : Orientation   *
* X : & Position    *
* X : For Acquisition *
* X : Linear        *
*****
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Ori folder of reference images}
 * string :: {Folder for output initialized orientation- default = Ori-InitOut}
 * string :: {Pattern of new images to orientate PatCam1, PatCam2,...}
 * string :: {Pattern of Reference Image = PatRef1, PatRef2,...}
Named args :
```

```
* [Name=PatTurn] vector<std::string> :: {Images when acquisition have turn [poseTurn1,poseTurn2...]}
* [Name=PatAngle] vector<std::string> :: {Turn angle [angle1,angle2,...] - + => turn left, - => turn right}
* [Name=mulF] vector<std::string> :: {Multiplication factor for adjustment each turn [mul1,mul2,...]}
* [Name=Axe] string :: {Which axe to calcul rotation about - default = z}
* [Name=WithIdent] bool :: {Initialize with orientation identique (default = false)}
* [Name=Plan] bool :: {Force using vector [0,0,1] to initialize (garantie all poses will be in a same plan)}
```

***Example:***

An acquisition with 3 section continue. Each section is separate by a turn. Section 1, then turn left 90 to section 2, then turn left 45 to section 3 (Fig 7.7). Acquisition is done by a system 2 camera on a bar rigide. Name image is contain an indicator of camera:

```
For camera 1: BIN_.*_image_023_.*.tif
For camera 2: BIN_.*_image_024_.*.tif
```

Two turns in acquisition:

```
Turn 1 : 90 left at image BIN_010-0120_14576019095_image_023_001_01316.thm.tif
Turn 2 : 45 left at image BIN_021-0150_14576061046_image_023_003_02935.thm.tif
```

***Step 1:*** orientate some images as a reference. Select image from 2 camera, in same shot to compute relative position between cameras and vector of displacement.

```
mm3d Tapas FishEyeBasic "BIN_010-011[1-2]_*.tif" Out=reference
```

We have an aero of reference image as show on Fig7.8.

***Step 2:*** Initialize another image. We have 2 serie correspondant with 2 camera, and we have 2 turn. By using command `InitOriLinear`:

```
mm3d InitOriLinear Ori-reference Ori-InitOut \
    "BIN_0.*-(011[4-9]|01[2-6]).*_023.*.tif,BIN_0.*-(011[4-9]|01[2-6]).*_024.*.tif" \
    "BIN_010-011[1-3].*_023.*.tif,BIN_010-011[1-3].*_024.*.tif" \
    PatTurn=[BIN_021-0128_14576060714_image_023_003_02913.thm.tif, \
        BIN_021-0148_14576061016_image_023_003_02933.thm.tif] \
    PatAngle=[90,45] mulF=[1,1] Axe=z
```

In the command, we use orientation reference in Ori-reference, we have 2 pattern correspondant with 2 camera, first is 023 and seconde is 024. Acquisition turned at pose "BIN\_021-0128\_14576060714.image\_023\_003\_02913.thm.tif" and "BIN\_021-0148\_14576061016.image\_023\_003\_02933.thm.tif" with angle correspondant is 90 and 45. Attention: number of image in each serie must be the same, and image to indicate a turn must be image of the first camera. Result is shown on Fig7.8 and Fig7.9



Figure 7.9: Plan of acquisition linear with 2 turn, 3 section

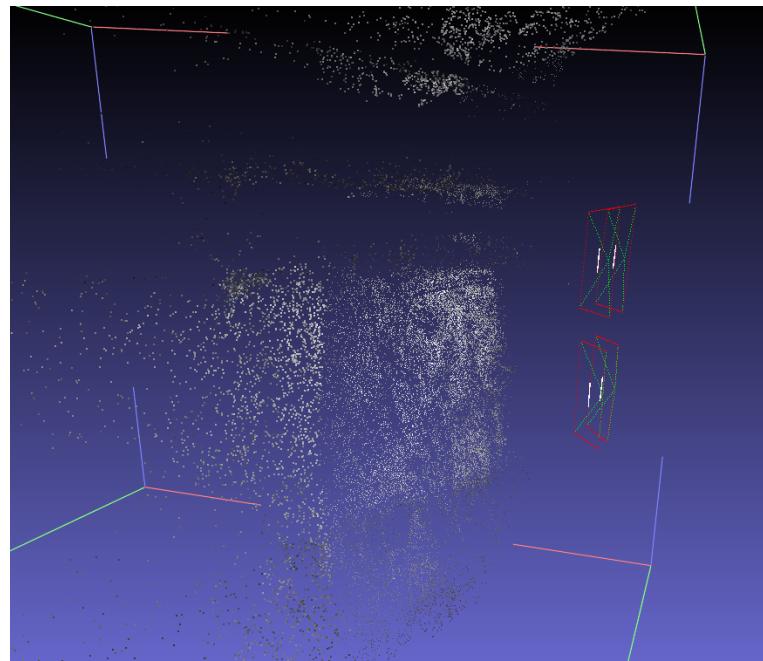


Figure 7.10: Calcul aero of reference image. System with 2 camera

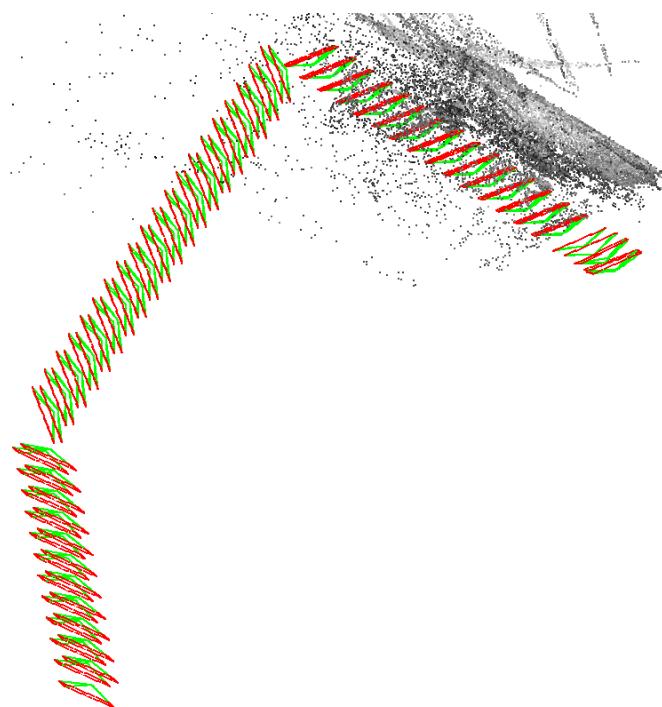


Figure 7.11: Result initialize orientation of linear acquisition

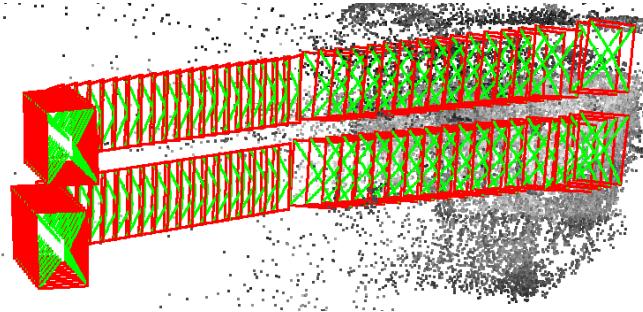


Figure 7.12: System with 2 camera

#### 7.5.4 ReprojImg

With **ReprojImg**, you can project an image into the orientation of another.

Inputs:

- Two images (reference and projected image)
- Ori of the two images
- DEM of reference image

The output is the image reprojected into reference orientation.

```
mm3d ReprojImg -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
* string :: {Orientation of reference image (xml)}
* string :: {Reference DEM filename (xml)}
* string :: {Reference image name}
* string :: {Orientation of image to reproject (xml)}
* string :: {Name of image to reproject}
Named args :
* [Name=AutoMask] string :: {AutoMask filename}
* [Name=DepthRepImage] string :: {Image to reproject DEM file (xml), def=not used}
* [Name=KeepLum] bool :: {Keep original picture luminosity (only for colorization), def=false}
```

The **KeepLum** argument is used to colorize reference picture, assuming that is green channel only.

As an example, when we have many pictures of first epoch (*epoch1.\**), we create a DEM, then we can compute the orientation of an image of another epoch (*epoch2\_a.JPG*) in the same reference, and finally we reproject it in the (*epoch1\_f.JPG*) geometry :

```
mm3d Tapioca All "epoch1...JPG" -1
mm3d Tapas RadialBasic "epoch1...JPG"
mm3d Malt GeomImage "epoch1...JPG" RadialBasic Master="epoch1_f.JPG"

mm3d Tapioca All "epoch...JPG" -1
mm3d Tapas RadialBasic "epoch...JPG" InOri=RadialBasic Out=Tout

mm3d ReprojImg Ori-Tout/Orientation-epoch1_f.JPG.xml MM-Malt-Img-epoch1_f/Z_Num8_DeZoom1_STD-MALT.tif \
epoch1_f.JPG Ori-Tout/Orientation-epoch2_a.JPG.xml epoch2_a.JPG
```

#### 7.5.5 ExtractMesure2D

The **ExtractMesure2D** command extracts only a selection of targets in a 2D measures file. Its main purpose is to split images measures into used and check targets.

```
mm3d ExtractMesure2D -help
*****
* Help for Elise Arg main *
*****
```

```
Mandatory unnamed args :
 * string :: {Input mes2D file}
 * string :: {Output mes2D file}
 * vector<std::string> :: {List of selected targets. Ex: [target1,target2])}
Named args :
```

Example:

```
mm3d ExtractMesure2D 21pts_Mesure-S2D.xml out.xml [3,203]
```

Will save in out.xml only the 2D measures for targets 3 and 203.

### 7.5.6 BasculeCamsInRepCam

The **BasculeCamsInRepCam** command express all images external parameters with respect to the camera frame of one chosen image.

```
mm3d TestLib BasculeCamsInRepCam -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Full Name (Dir+Pattern)}
 * string :: {Orientation}
 * string :: {Central camera}
 * string :: {Output}
```

Example :

```
mm3d TestLib BasculeCamsInRepCam ".*JPG" Ori-AutoCal/ IMG00004.JPG Img04
```

In **Ori-Img04/** one can check that external parametrs of image **IMG00004.JPG** are by definition equal to identity matrix.

### 7.5.7 BasculePtsInRepCam

The **BasculePtsInRepCam** command express coordinates of input points with respect to the camera frame of one chosen image.

```
mm3d TestLib BasculePtsInRepCam -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Name Camera}
 * string :: {Name GGP In}
Named args :
 * [Name=Out] string
```

Example :

```
mm3d TestLib BasculePtsInRepCam Ori-AutoCal/Orientation-IMG00004.JPG.xml App.xml
```

Output file **Basc-Orientation-IMG00004.JPG-App.xml** contains points coordinates expressed in the camera frame of **IMG00004.JPG**.

### 7.5.8 CorrLA

The **CorrLA** command computes corrected images positions with respect to a given value of lever-arm offset.

```
mm3d TestLib CorrLA -help
*****
* Help for Elise Arg main *
*****
```

```
Mandatory unnamed args :
 * string :: {Full Name (Dir+Pattern)}
 * string :: {Directory orientation}
 * Pt3dr :: {Lever-Arm value}

Named args :
 * [Name=OriOut] string :: {Output Ori Name of corrected mandatory Ori ; Def=OriName-CorrLA}
 * [Name=Ori2] string :: {Ori2 directory to apply LA correction to centers}
 * [Name=Ori2Out] string :: {Output Ori Name of corrected Ori2 ; Def=Ori2Name-CorrLA}
```

Example:

```
mm3d TestLib CorrLA ".*JPG" Ori-Basc/ [0.289,-0.043,-0.183] Ori2=Ori-Nav-GPS/
```

Sometimes, it is more accurate to compute the lever-arm correction and to apply it to a second set of images positions. Using the optional argument `Name=Ori2` one can apply corrections calculated with the orientations from the mandatory `Ori-Basc/` and generate `Ori-Nav-GPS-CorrLA/` with lever-arm corrected positions.

### 7.5.9 ExportXmlGcp2Txt

The `ExportXmlGcp2Txt` command simply convert a GCP .xml file format into a column format text file.

```
mm3d TestLib ExportXmlGcp2Txt -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Directory}
 * string :: {xml Gcps file}

Named args :
 * [Name=Out] string :: {output txt file name : def=Output.txt}
 * [Name=addInc] bool :: {export also uncertainty values : def=false}
```

Example :

```
mm3d TestLib ExportXmlGcp2Txt ./ AppAll.xml Out=AppAll-Txt.txt
```

This command can be useful for example when a GCP file is converted to .xml format using `mm3d GCPConvert` with a change of system and the user wants to recover transformed coordinates in a simple format.

### 7.5.10 SimplePredict

The `SimplePredict` command projects ground points on oriented images.

```
mm3d SimplePredict -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Pattern of images}
 * string :: {Directory orientation}
 * string :: {Ground points file}

Named args :
 * [Name=ExportPolyIGN] bool :: {Export PointeInitIm files for IGN Polygon calibration method (Def=false)}
 * [Name=PrefixeNomImageSize] INT :: {Size of PrefixeNomImage in param.txt}
```

Example :

```
mm3d SimplePredict ".*JPG" Ori-Basc/ AppAll.xml
```

The output `SimplePredict.xml` file contains (i,j) coordinates of all `AppAll.xml` points in all possible images.

### 7.5.11 Export2Ply

The `Export2Ply` command generates a .ply file containing spheres to represent points.

```
mm3d TestLib Export2Ply -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Format specification}
 * string :: {Name File of Points Coordinates}
Named args :
 * [Name=Ray] REAL :: {Plot a sphere per point}
 * [Name=NbPts] INT :: {Number of Pts / direc (Def=5, give 1000 points) only with Ray > 0}
 * [Name=Scale] INT :: {Scaling factor}
 * [Name=FixColor] Pt3di :: {Fix the color of points}
 * [Name=LastPtColor] Pt3di :: {Change color only for last point}
 * [Name=ChangeColor] INT :: {Change the color each number of points : not with FixColor}
 * [Name=Out] string :: {Default value is NameFile.ply}
 * [Name=Bin] INT :: {Generate Binary or Ascii (Def=1, Binary)}
```

Example :

```
mm3d TestLib Export2Ply "#F=N_X_Y_Z" AppAll-Txt.txt Ray=0.5 FixColor=[0,0,255]
```

This is sometimes useful for example when one wants to represent GCPs into a point cloud to clearly visualize its distribution.

### 7.5.12 CmpOri

The `CmpOri` command computes average norme differences of all external parametrs of images.

```
mm3d CmpOri -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Full Name (Dir+Pattern)}
 * string :: {Orientation 1}
 * string :: {Orientation 2}
Named args :
 * [Name=DirOri2] string :: {Orientation 2}
 * [Name=XmlG] string :: {Generate Xml}
```

Example :

```
mm3d CmpOri ".*JPG" Ori-Bascule/ Ori-Compense/ XmlG=Delta_Basc_Comp.xml
```

For example the result displayed and saved in `Delta_Basc_Comp.xml` :

```
<?xml version="1.0" ?>
<XmlTNR_TestOriReport>
    <OriName>RTL-Compense-AllPts</OriName>
    <TestOriDiff>false</TestOriDiff>
    <DistCenter>0.0397097570172935677</DistCenter>
    <DistMatrix>3.87112370850761372e-07</DistMatrix>
</XmlTNR_TestOriReport>
```

### 7.5.13 CmpCalib

The `CmpCalib` command compares two files of calibrations (in general of the same camera).

```
mm3d CmpCalib -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {First calibration file}
 * string :: {Second calibration file}
Named args :
 * [Name=Teta01] REAL
 * [Name=Teta02] REAL
 * [Name=Teta12] REAL
 * [Name=L1] INT
 * [Name=SzW] INT
 * [Name=DynV] REAL
 * [Name=Out] string :: {Result (Def=Name1_ecarts.txt)}
 * [Name=DispW] bool :: {Display window}
 * [Name=XmlG] string :: {Generate Xml}
```

Example :

```
mm3d CmpCalib Ori-Bascule/AutoCal_Foc-35000_Cam.xml Ori-Compense/AutoCal_Foc-35000_Cam.xml Out=Delta_Calib.t
```

Calibration parameters of a camera can not be directly compared, the command estimates a rotation to align the parameters. The output file `Delta_Calib.txt` contains a function which gives the differences between the two sets of calibration as function of the radius and a grid which provides planimetric vector deviation between each rays directions. Example of output :

```
----- Ecart radiaux -----
Rayon   Ecart
0.000000 0.035861
200.000000 0.094960
400.000000 0.184629
...
----- Ecart plani -----
Im.X Im.Y PhG.X Phg.Y Ec
5017.600000 3763.200000 -0.855326 -0.588127 1.038015
5017.600000 3394.560000 -0.951363 -0.529560 1.088818
5017.600000 3025.920000 -0.988994 -0.465235 1.092956
...
```

### 7.5.14 PseudoIntersect

The `PseudoIntersect` command estimates the 3D position coordinates of 2D measured points in oriented images.

```
mm3d TestLib PseudoIntersect -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Full Name (Dir+Pat)}
 * string :: {Directory of input orientation}
 * string :: {xml file of 2d points}
Named args :
 * [Name=Out] string :: {Name output file (def=3DCoords.txt)}
 * [Name=XmlOut] bool :: {Export in .xml format to use as GCP file (Def=true)}
 * [Name>Show] bool :: {Gives details on arguments (Def=true)}
```

Example :

```
mm3d TestLib PseudoIntersect ".*JPG" Ori-All/ MesuresFinales-S2D.xml
```

Each point needs to be measured at least in 2 images to compute an intersection. The output .xml file can be directly used as a GCP file with `GCPBascule` tool.



# Chapter 8

## Interactive tools

### 8.1 Generalities

### 8.2 Entering mask with SaisieMasq or SaisieMasqQT

#### 8.2.1 SaisieMasq

**SaisieMasq** is a very simple tool to edit mask images. It creates a binary mask image from a polygonal selection in the displayed image.

Typing **SaisieMasq -help**, one gets:

```
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {Name of input image}
Named args :
 * [Name=SzW] Pt2di
 * [Name=Post] string
 * [Name=Name] string :: {Name of result, default toto->toto_Masq.tif}
 * [Name=Gama] REAL
 * [Name=Attr] string
```

Meaning of args is:

- First arg: pattern specifying the images to load (can be 1 or more images) - regular expression are supported;
- optional **SzW**, def = [900, 700], size of display window;
- optional **Post**, def = "Masq" , postfix to add to output filename;
- optional **Name**, name of result;
- optional **Gama**, def = 1.0 , gama applied to images, it can help with dark images, or wide dynamics;
- optional **Attr**, text to add to postfix;

Processing is as follow:

- Click: add a point to polygon
- Shift click: close polygon and apply selection
- Ctrl + right click: delete last point
- Shift + right click + Coul : switch between add mode and remove mode
- Shift + right click + Exit : save mask image and **Xm1** file and quit

#### 8.2.2 SaisieMasqQT

**SaisieMasqQT** is the same tool as **SaisieMasq**, available on all platforms (Linux, Windows, MacOS). **SaisieMasqQT** can be run several ways. You can run command **mm3d SaisieMasqQT** and drag-and-drop files in the application window, or open files from File menu, or you can run command **mm3d SaisieMasqQT + arguments**

Example: `mm3d SaisieMasqQT IMG.tif SzW=[1200,800] Name=PLAN Gama=1.5`

You can also run command

`mm3d SaisieMasqQT -l` to load last edited file. A visual interface for argument edition is also available with command: `mm3d vSaisieMasqQT` As usual, type `mm3d SaisieMasqQT -help`, to get help message.

**SaisieMasqQT** has the same arguments as **SaisieMasq**. Some light differences with **SaisieMasq** processing workflow should be noticed: you need to draw a polygon first, and then apply an action (add to mask, remove from mask, etc.). You can get a complete list of possibles actions typing F1.

Main actions are:

- F2: display image in full screen
- Wheel roll: zoom
- Wheel click: move image
- Shift+wheel click: zoom fast
- Left click: add a point to polygon
- Right click: close polygon
- Space: Add to mask
- Suppr: Remove from mask
- Right click (close to a point): delete point
- Echap: delete polygon
- Shift + click & drag: insert point
- Ctrl+S: save mask image and Xml file
- Ctrl+Q: quit

Some parameters can be edited in the **Settings** menu: image gamma, application language, display settings, etc. These parameters are stored, and are used at next application launch, so set them once to fit your own purpose. As **SaisieMasqQT** can edit both images and 3d point clouds, application has different behavior, depending on which data you deal with. As a result, **Settings** menu, and help window dialog (F1), will change depending on the loaded data. Be careful about this!

Some special features have been added to **SaisieMasqQT**, which may differ from original **SaisieMasq**:

- modify current selection, with Shift+Click to insert a point, or Click+drag to move a point
- modify previous actions, with menu Windows>Show polygons list
- measure image distances, with Rule tool

To modify previous actions, you can undo/redo last actions with `Ctrl+Z/Shift+Ctrl+Z`, and you can edit list of previous actions, with menu **Windows>Show polygons list**: click on the actions in the list in the right window, then edit polygon (by adding or moving points), or double-click on the action name (column **Mode**) to change action. Apply changes clicking **Return**, or go in the Menu **Mask edition** and select **Confirm changes**.

Note: if you want to change application language, go in the **Settings** menu, apply changes and restart application.

### 8.3 Entering 3D mask with **SaisieMasqQT**

**SaisieMasqQT** can also be used to measure 3D mask from a point cloud. This 3D mask is useful to restrict computation to the main object. **SaisieMasqQT** allows to open ply files in a 3D view and to do a manual segmentation with a polygonal selection tool. **SaisieMasqQT** can open one or several ply files (provided that ply files have been computed in the same reference frame). The 3D mask selection with **SaisieMasqQT** is designed to work with some specific functions, such as C3DC: the idea is to do a manual segmentation, by rotating around the objet, and by drawing a polygon. Each rotation (or translation) and polygonal selection is stored into a **Xml** file which can be used with other MicMac commands.

Calling **SaisieMasqQT** can be performed in a command shell with: `mm3d SaisieMasqQT`

To open a ply file, there are 4 possibilities:

- run command with filename or pattern: `mm3d SaisieMasqQT filename.ply`
- run command `mm3d SaisieMasqQT` and drag-and-drop ply file(s) in interface
- run command `mm3d SaisieMasqQT` and use standard open file menu

- run command `mm3d SaisieMasqQT -l` will open last edited file

If input ply filename is `cloud.ply`, resulting `Xml` files are named `cloud_selectionInfo.xml` and `cloud_polyg3d.xml`, and are saved in the ply file directory. The file to be used for parameter `Masq3D` of `C3DC` command is `cloud_polyg3d.xml`.

User can mainly perform 2 actions:

- move camera around point cloud (rotate and/or translate and/or zoom)
- draw a polygon and select/deselect points inside polygon

To switch between move mode and selection mode, use F9 key.

To select a part of the point cloud, you have to draw a polygon by clicking in the 3D view. To add a point to the polygon, use left-click, to close the polygon, use right-click. To select points inside polygon, use space bar. To change point size, use +/- keys. You can hide or show axis, ball, point cloud bounding box, display point cloud in full screen (F2). Center point cloud on a vertex by double click on it: next move actions will be done around this point. Undo/redo last actions with Ctrl+Z/Shift+Ctrl+Z. You can also modify a selection, with various actions (see help: F1 or previous paragraph: 2D and 3D selection edition work the same way).

For advanced users, `Xml` file format is detailed in 32.5.

## 8.4 Entering points

### 8.4.1 Generalities

To move into an image, various solutions are proposed in the interface:

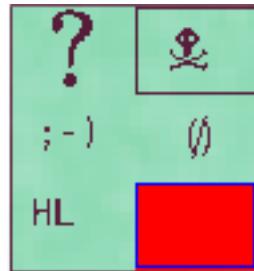
- Click on wheel + move = drag
- Shift + wheel + vertical move = quick zoom
- Shift + wheel + horizontal move = slow zoom
- Wheel roll = zoom

To input points, some menus can be displayed with these shortcuts:

- Right-click: geometry menu
- Shift + left-click: info menu
- Shift + right-click: undo menu
- Ctrl + right-click: zoom menu

#### 8.4.1.1 Geometry menu

This menu can be shown with a right-click:

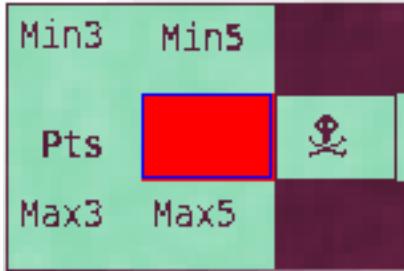


The corresponding actions are:

- ;( validate closest point;
- (/: invalidate closest point;
- ?: set point status to dubious
- skull: don't use closest point
- HL: highlight point
- empty box: escape menu (do nothing)

#### 8.4.1.2 Info menu

This menu can be shown with Shift + left-click:

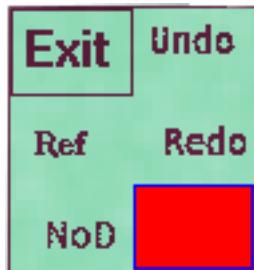


The corresponding actions are:

- Pts: select or add a name for this point
- Min3:
- Min5:
- Max3:
- Max5:
- skull: delete the point in all images (needs a confirmation)
- empty box: escape menu (do nothing)

#### 8.4.1.3 Undo menu

This menu can be shown with Shift + right-click:

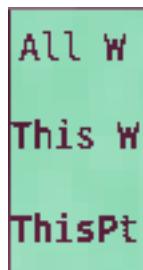


The corresponding actions are:

- Exit: quit the interface, saving `Xml` files
- Undo: undo last action
- Redo: redo last action in history
- Ref: display or not refuted points
- NoD/Ret: display or not the points names
- empty box: escape menu (do nothing)

#### 8.4.1.4 Zoom menu

This menu can be shown with Ctrl + right-click:



The three corresponding actions are:

- **All W**: full zoom in all windows, and show images where points have not been measured yet;
- **This W**: zoom only in the window where the menu has been displayed;
- **This Point**: zoom on the nearest point in all windows where the point is visible

### 8.4.2 For initial GCP with SaisieAppuisInit

This section describes **SaisieAppuisInit** the graphic interface to input 2D and 3D coordinates of ground control points.

For example with the Saint-Michel de Cuxa data set 4.2.1:

```
SaisieAppuisInit "Abbey-IMG_(021[12] | 023[3456]).jpg" All-Rel NamePointInit.txt MesureInit.xml
```

When running this command, the interface shows data set's first images, where one can point GCPs:

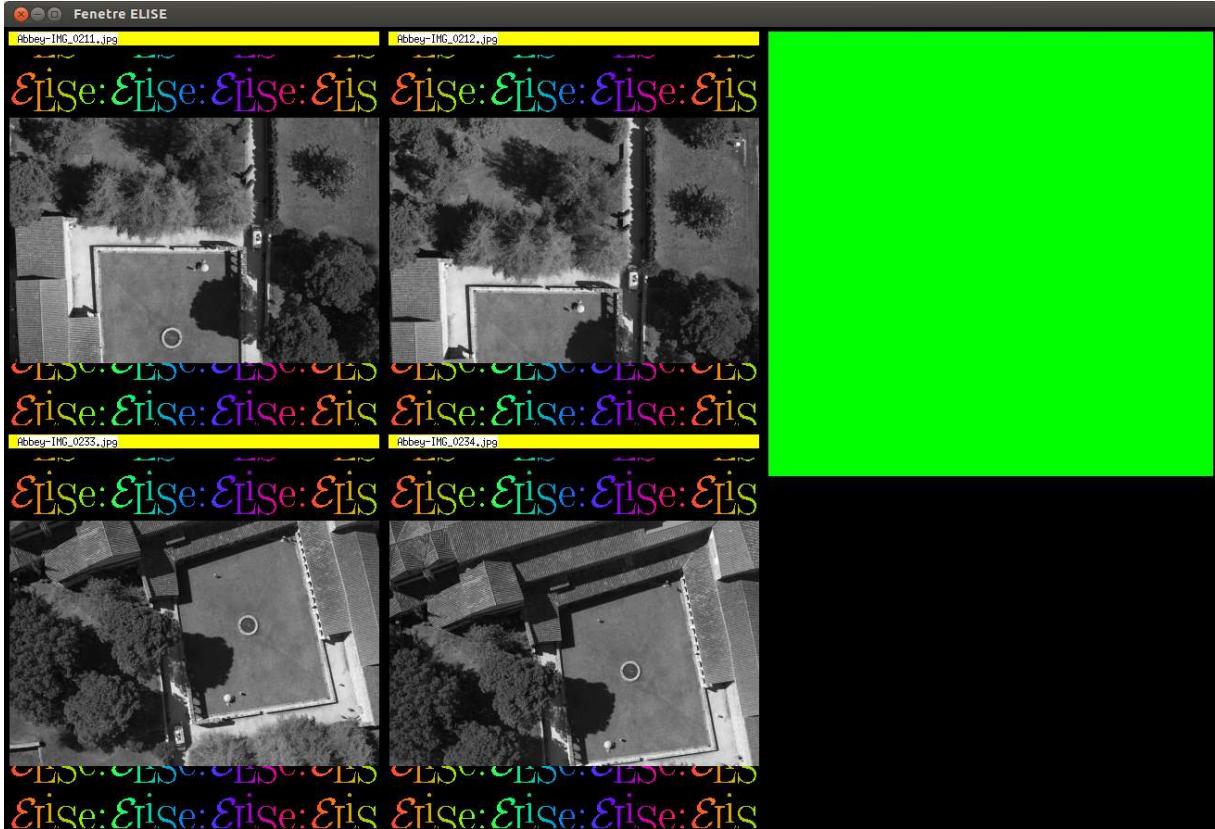


Figure 8.1: SaisieAppuis interface for ground control point selection

The general process for inputting ground control points is:

- Input a point in an image (Left-click)
- Select its name,
- Input the same point in the other images: move the yellow point and validate it with (right-clic + ; -) )
- Iterate on each point you want to add (at each iteration, it can be useful after having pointed the point in one image to zoom on this point in all the images, this can be done by (Ctrl + right-click + **This Point**)

When exiting the interface, two **Xml** files are stored, with respectively 2D and 3D coordinates of input points.

Note that if for some reason some points are missing, you can re-run the same command, and continue the input job. Points that have already been stored will be displayed, and the same process can be followed.

**SaisieAppuisInit** is available on Linux and MacOS. An equivalent tool is available on Windows, Linux and MacOS and is called with command:

```
mm3d SaisieAppuisInitQT + arguments
```

It runs with the same arguments as `SaisieAppuisInit`. For example:

```
mm3d SaisieAppuisInitQT "IMG_(023[3456]).jpg" All NamePoint.txt Mesure.xml
```

Same equivalent tools exist for `SaisieAppuisPredic` and `SaisieBasc` (ie.

```
mm3d SaisieAppuisPredicQT and mm3d SaisieBascQT
```

A visual interface for argument edition is also available with command:

```
mm3d vSaisieAppuisInitQT} or mm3d vSaisieAppuisPredicQT
```

`SaisieAppuisInitQT` displays two lists on the right side: the points list, and the images list. Points list can be clicked to choose which point to measure. You can also remove a point by clicking it in the list and press `Suppr`. You can also right-click and choose between following actions:

- Change images for selected point
- Delete selected points (multiple selection allowed)
- Validate selected points (idem)

The image lists show all available images. When a point has been measured in at least two images, the image list is displayed. Images currently displayed in the windows are highlighted in blue. The image where the cursor is moving is displayed in light orange. You can right-click and select `View images` to load corresponding images. A 3D window shows the images location, and the point measured. By default point are displayed in red ; when a point is selected, it is displayed in blue. You can drag-and-drop a ply file in this window (such as AperiCloud.ply) to check if GCP are good.

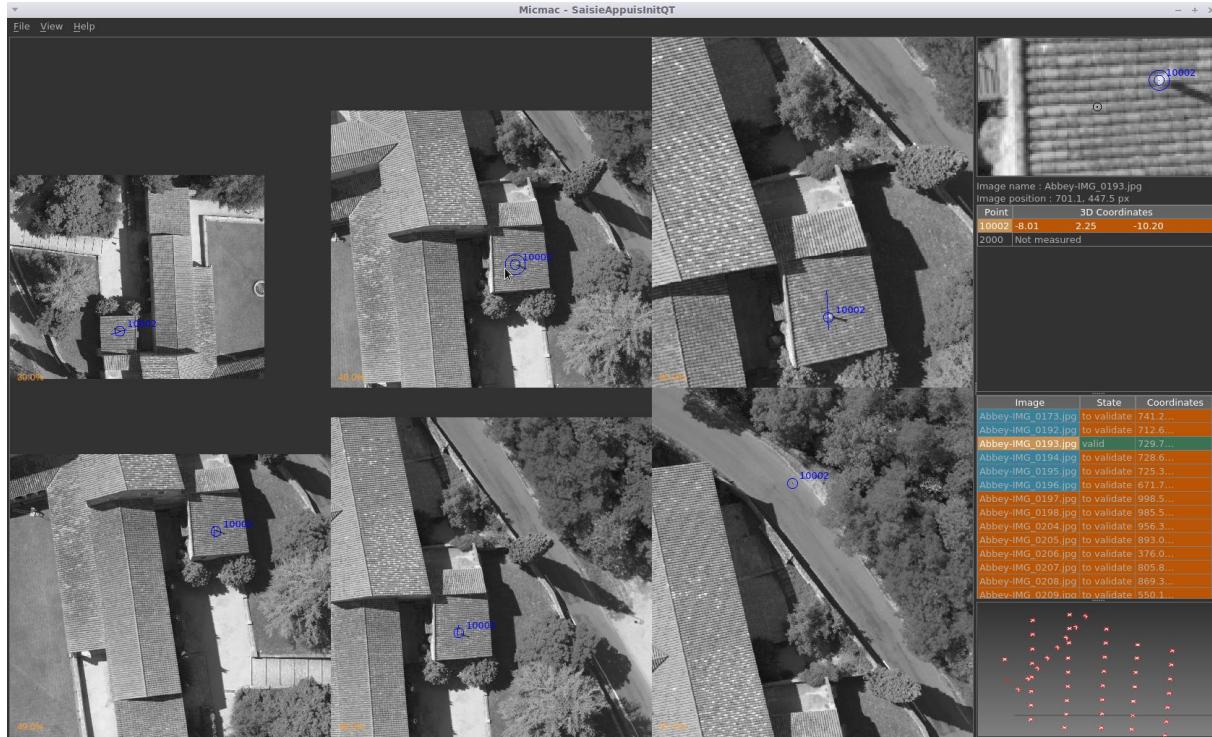


Figure 8.2: QT interface `SaisieAppuisInitQT`

### 8.4.3 For fast predictive entering GCP with `SaisieAppuisPredic`

When enough points have been selected, interface can give a prediction for each new input:

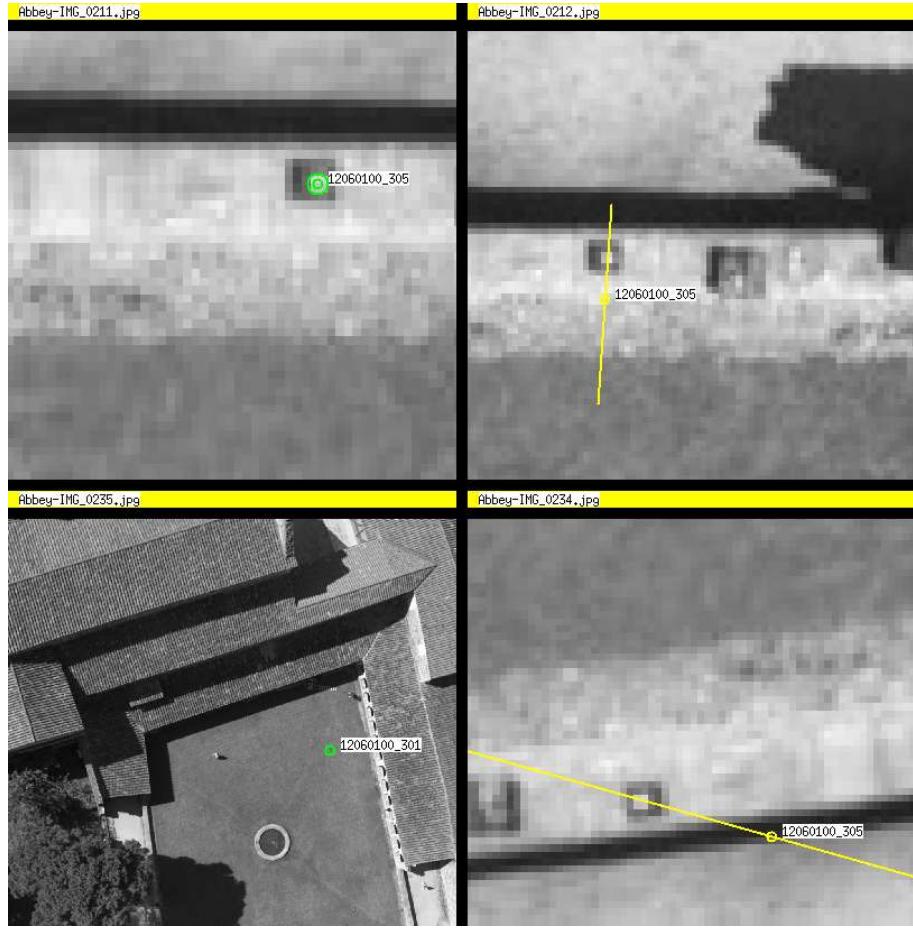


Figure 8.3: Prediction help for adding new point

#### 8.4.4 For bascule with SaisieBasc

**SaisieBasc** is a graphic interface to measure objects to be able to perform transformations such as data scaling, rigid transformation (rotation, translation).

One can size a point to set the origin of the new frame. One can size two lines:

- one to set horizontal (with two points: Line1, Line2)
- one to set scale (with two points: Ech1, Ech2)

## 8.5 Visualize Tie-points with SEL

An old and ugly tool, but it can help. To visualize tie points computed with **Tapioca** :

```
SEL ./ Face2-IMGP5331.JPG Face2-IMGP5333.JPG KH=NB
```

For creating a few set of tie points and save in XML format :

```
SEL ./ Face2-IMGP5331.JPG Face2-IMGP5333.JPG KH=S
```



# Chapter 9

## New "generation" of tools

This chapter describes some new tools, probably their documentation will be reorganized once they are completely stabilized.

### 9.1 Fully automatic dense matching

#### 9.1.1 Generalities

The `C3CD` command is the command that compute automatically a point cloud from a set of oriented images.

```
mm3d C3DC -help
Valid types for enum value:
  Ground
  Statue
  TestIGN
  QuickMac
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
  * string :: {Type in enumerated values}
  * string :: {Full Name (Dir+Pattern)}
  * string :: {Orientation}
Named args :
  * [Name=Masq3D] string :: {3D masq for point selection}
  * [Name=Out] string :: {final result (Def=C3DC.ply)}
  * [Name=SzNorm] INT :: {Sz of param for normal evaluation (<=0 if none, Def=2 mean 5x5) }
  * [Name=PlyCoul] bool :: {Colour in ply ? Def = true}
  * [Name=Tuning] bool :: {Will disappear soon ...}
  * [Name=UseGpu] bool :: {Use cuda (Def=false)}
```

The syntax is :

- type of matching in enumerated values,
- set of images to use
- orientation
- if `Masq3D` is specified, indicates a 3D masq as created with `SaisieMasqQT`;
- if `SzNorm` is specified, indicates the window size parameters for normal extraction in ply file (usefull for meshing);
- if `PlyCoul` is specified, indicates that coloring of points is required.

#### 9.1.2 Quickmac option

The `QuickMac` uses the `MMInitialModel` as matcher, which is quite fast on CPU. As example we use a dataset of 41 images of a statue, they are presented on figure 9.1.

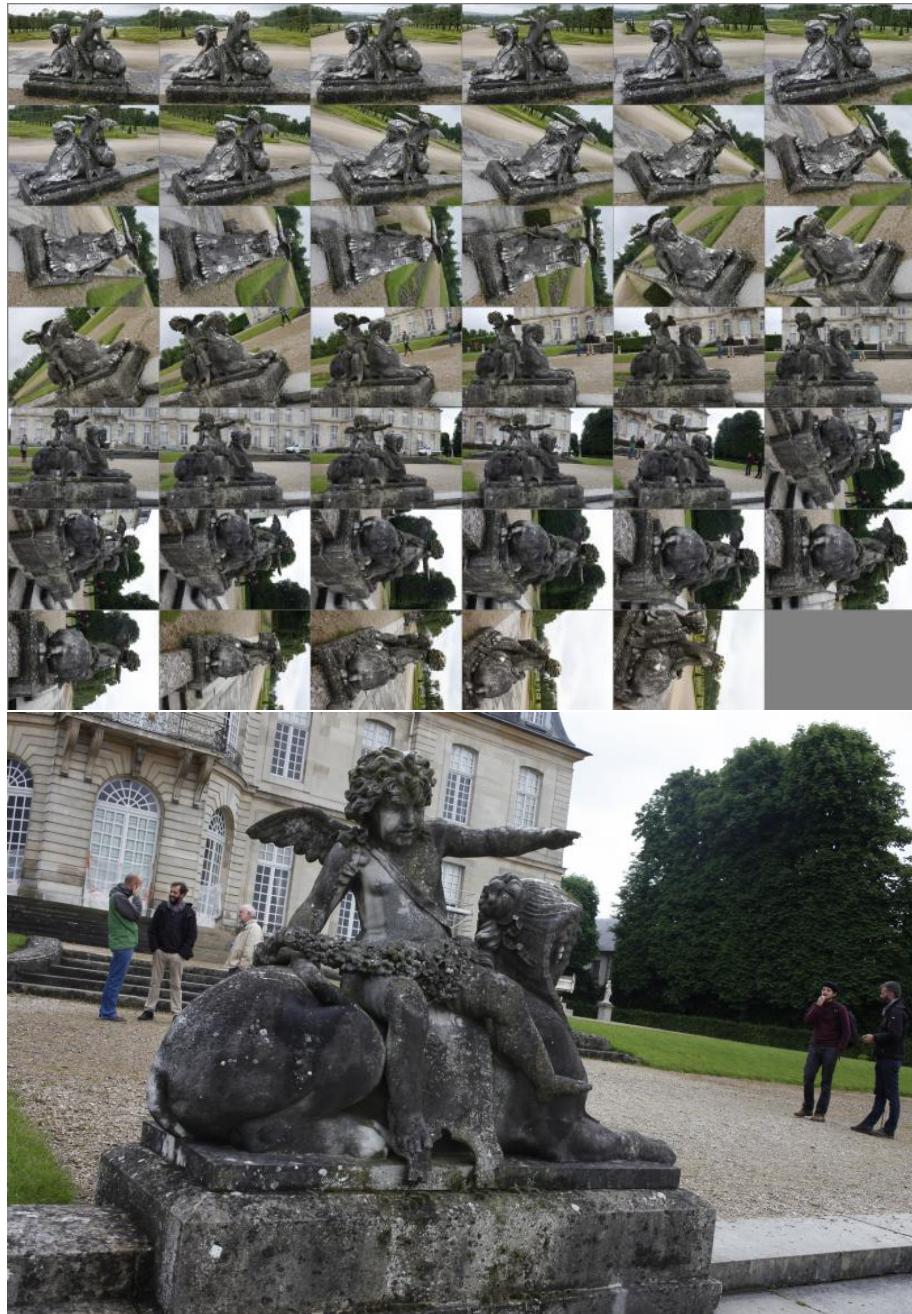


Figure 9.1: The Angel statue used for the C3DC QuickMac command

Here is a possible command using a dataset of 41 images of a statue :

```
mm3d C3DC QuickMac _MG_10.*JPG Ori-All2/ Masq3D=AperiCloud_All2_selectionInfo.xml
```

The result are presented on figure 9.2. Computation time was 12 min with a 8 processor machine.

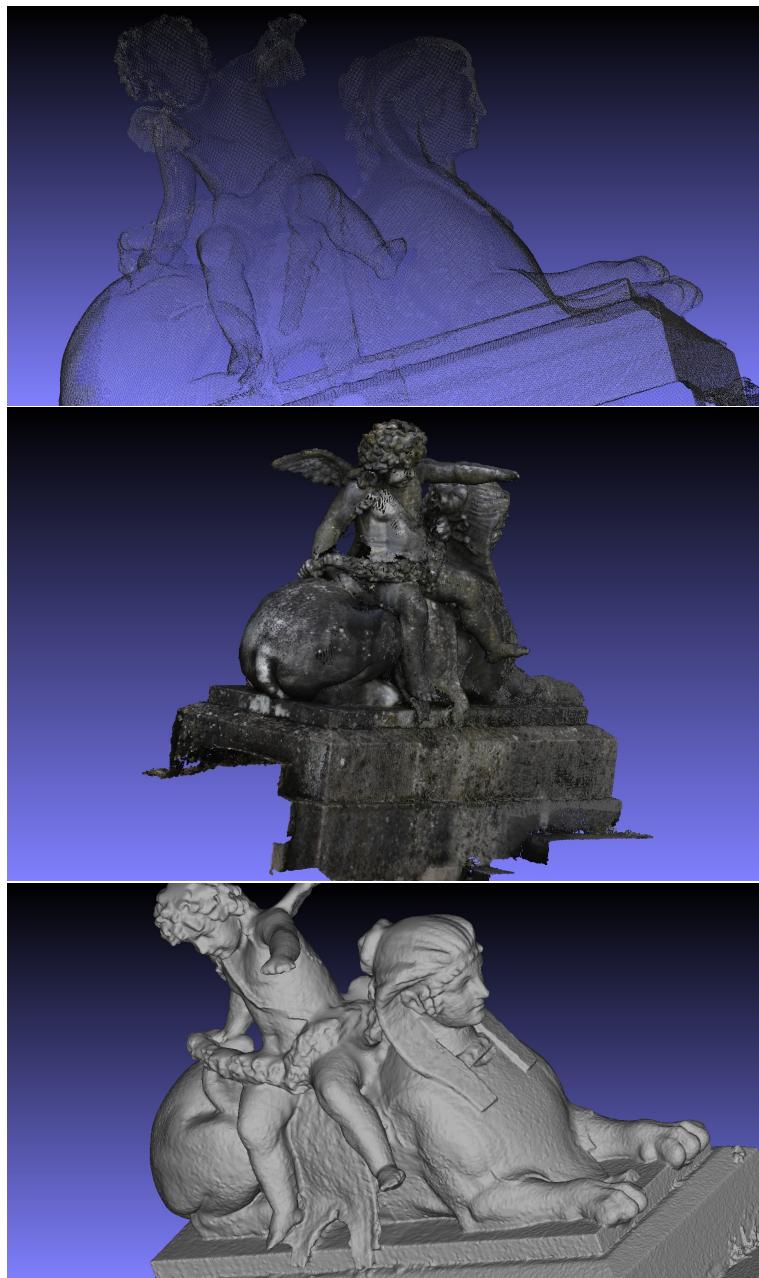


Figure 9.2: Result of C3DC QuickMac command: point cloud, coloured point cloud, meshed point cloud

## 9.2 Post-processing tools - mesh generation and texturing

### 9.2.1 Mesh generation

Note: this tool is still under development, for now, it is recommended to use it with Filter option set to false. To use this tool, if compiling from sources, run `cmake` with `BUILD_POISSON` option activated.

```
cmake -DBUILD_POISSON=ON
```

`TiPunch` command creates a mesh from a point cloud. The point cloud has to be in .ply format and has to store normal direction for each point. This command performs two steps:

- mesh generation
- mesh filtering

Mesh generation is built as a call to **PoissonRecon** binary from Misha Khazdan (for more information on M. Khazdan's code and research: <http://www.cs.jhu.edu/~misha/Code/PoissonRecon/>) It has mainly one important parameter: the depth of reconstruction. **PoissonRecon** solves the Poisson equation with a discretization of space into a voxel grid. The depth  $d$  parameter defines the size of the voxel grid, as grid is  $2^d \times 2^d \times 2^d$  voxels. As a result, a higher depth will lead to a higher level of detail in the final mesh.

As **PoissonRecon** can sometimes generate wrong mesh parts, mesh filtering is necessary to delete parts of the mesh which are too far from point cloud. Mesh filtering makes the assumption that point cloud ply has been generated using **C3DC** command. But one can also use **Nuage2Ply** (with Normale option) and **MergePly** to generate compatible point cloud. In this case, you can deactivate mesh filtering, with option **Filter=0**. To filter the mesh, depth images are used (their location is recovered from Pattern and C3DC mode). To reduce computing time, use a subset of the whole images set (typically 8 to 12 in statue configuration), by choosing the right pattern.

```
mm3d TiPunch -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Ply file}
Named args :
 * [Name=Pattern] string :: {Full Name (Dir+Pat)}
 * [Name=Out] string :: {Mesh name (def=plyName+ _mesh.ply)}
 * [Name=Bin] bool :: {Write binary ply (def=true)}
 * [Name=Depth] INT :: {Maximum reconstruction depth for {\tt PoissonRecon} (def=8)}
 * [Name=Rm] bool :: {Remove intermediary Poisson mesh (def=false)}
 * [Name=Filter] bool :: {Filter mesh with distance (def=false)}
 * [Name=Mode] string :: {C3DC mode (def=Statue)}
 * [Name=Scale] INT :: {Z-buffer downscale factor (def=2)}
 * [Name=FFB] bool :: {Filter from border (def=true)}
```

Syntax is:

- ply file, with normal direction computed for each point
- **Pattern**, needed if **Filter=true**, set of images to filter mesh (we use depth images computed by **C3DC**)
- **Out**, output mesh filename
- **Bin**, output mesh ply format (ascii or binary, true means binary)
- **Depth**, Maximum reconstruction depth for **PoissonRecon**
- **Rm**, remove output of **PoissonRecon** (mainly if **Filter=true**)
- **Filter**, do we filter mesh
- **Mode**, needed if **Filter=true**, mode of **C3DC** (needed for PIMs- directory)
- **Scale**, Z-buffer downscale factor, used for filtering (a bigger downscale factor speeds up process, but is less accurate)
- **FFB**, Filter from border: force filtering to start from mesh borders (it avoids creating holes)

### 9.2.2 Texturing the mesh

**Tequila** computes a UV texture image from a ply file, a set of images and their orientations. Ply file has to be a mesh, and can be the result of **TiPunch** (but not the direct result of **C3DC**). Here again, using the whole set of images is not necessary. Choosing a subset of the whole images is recommended (8 to 12 images can give good results, in statue mode).

**Tequila** performs following steps:

- load data
- compute zbuffers
- choose which image is best for each triangle
- filter mesh according to visibility (optional)
- graph-cut optimization (optional)
- write UV texture
- write ply file with uv texture coordinates

Choosing which image is best for each triangle can be done with three different criterions:

- best angle between triangle normal and image viewing direction (parameter **Crit=Angle**, by default, and recommended)

- best stretching of triangle projection in image (parameter Crit=Stretch)
- best acute angle of triangle projection in image (parameter Crit=AAngle)

For the angle criterion, expressed in degrees, a threshold is set to avoid using images that view a triangle with a low incidence (parameter Angle). It means that if the angle between triangle normal and image viewing direction is higher than *Angle*, the image will not be used for texturing.

**Tequila** has also two modes, which refer to texture computing strategies: *basic* and *pack*. In the *basic* mode, all images from the set are stored in the uv texture, and if necessary are downscaled. Each image is masked with the zbuffer, to store a minimum of significant information. In the *pack* mode, each image is divided in small regions, and only useful regions are packed into the uv texture, in an optimal way. This mode leads to smaller images, and gives better texture quality.

```
mm3d Tequila -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Full Name (Dir+Pat)}
 * string :: {Orientation path}
 * string :: {Ply file}
Named args :
 * [Name=Out] string :: {Textured mesh name (def=plyName+ _textured.ply)}
 * [Name=Bin] bool :: {Write binary ply (def=true)}
 * [Name=Optim] bool :: {Graph-cut optimization (def=false)}
 * [Name=Lambda] REAL :: {Lambda (def=0.1)}
 * [Name=Iter] INT :: {Optimization iteration number (def=2)}
 * [Name=Filter] bool :: {Remove border faces (def=false)}
 * [Name=Texture] string :: {Texture name (def=plyName + _UVtexture.jpg)}
 * [Name=Sz] INT :: {Texture max size (def=4096)}
 * [Name=Scale] INT :: {Z-buffer downscale factor (def=2)}
 * [Name=QUAL] INT :: {jpeg compression quality (def=70)}
 * [Name=Angle] REAL :: {Threshold angle, in degree, between triangle normal and image viewing direction (de
 * [Name=Mode] string :: {Mode (def = Pack)}
 * [Name=Crit] string :: {Texture choosing criterion (def = Angle)}
```

Relevant parameters are:

- **Angle**, threshold for maximum angle between normal and viewing direction
- **Mode**, choose between Basic and Pack (see upper)
- **Crit**, choose between Angle, Stretch, and AAngle (see upper)
- **Scale**, which allow to speed up computation (higher downscale factor leads to faster computation).
- **Sz**, which will force texture size, to conform with graphic card capacity (see GL\_MAX\_TEXTURE\_SIZE if available)
- **QUAL**, the jpeg compression quality, which allows to compact UV texture image.
- **Optim**, post-processing step, to gather neighbouring triangles with the same image texture (graph-cut algorithm, detailed below)
- **Lambda**, weighting factor for graph-cut optimization
- **Iter**, number of iteration steps for optimization

In most cases, illumination variations, BRDF variations upon directions, and surface shape will lead a simple texturing algorithm to produce artefacts in texture: two adjacent triangles can be assigned two different texture images, while only one texture image for both triangles might be better. In some rare cases (no illumination variation, etc.), these artefacts won't be visible. Also if a texture equalization is applied (process that will be included one day in the **MicMac** tools) these artefacts won't happen, or should be less visible.

To limit jumps between several texture images in adjacent triangles, an optimization can be performed as a post-processing step. This optimization is stated as a multi-label energies graph-cut. Each triangle is assigned a likelihood term (here, angle to image viewing direction or projected triangle stretching). Two adjacent triangles define a graph edge, and a coherence term is assigned to this edge (here, the difference between mean texture in each triangle).  $\lambda$  parameter (Lambda) is the weight between likelihood term and coherence term.



Figure 9.3: The Angel statue mesh textured with **Tequila** command

## 9.3 Parallelizing Apero

For now works only with linear orientation.

### 9.3.1 Parallelizing Apero

The new tool **Liquor** (for LInear QUick ORientation) accelerate the computation of orientation. The acceleration comes from two aspects:

- it uses a hierarchical building of orientation, which make the computation in  $N \log N$  instead of  $N^2$
- at the low level of the pyramid, it parallelizes the computation of subset on the several processors.

The syntax :

```
mm3d Liquor -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Full name (Dir+Pat)}
 * string :: {Calibration Dir}
Named args :
 * [Name=SzInit] INT :: {Sz of initial interval (Def=50)}
 * [Name=OverLap] REAL :: {Prop overlap (Def=0.1) }
```

An example of use with the data set of figure 9.4 :

```
mm3d Liquor CAM2_0.* Ori-Calib/
```

With these 150 images the computation time is 20min instead of 1h10min with traditional **Tapas**.

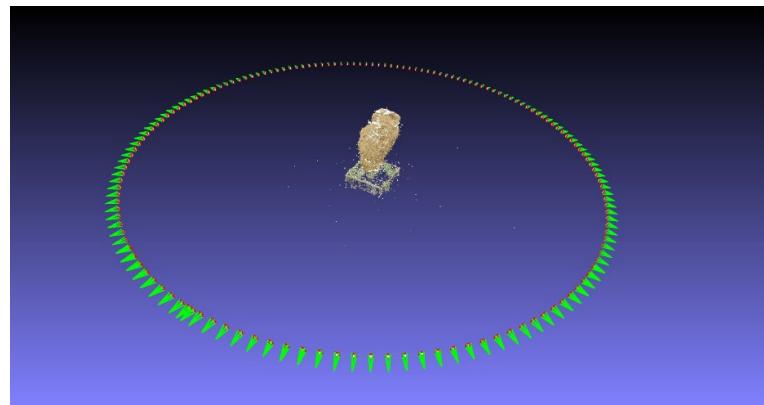


Figure 9.4: Linear acquisition used for `Liquor` command



# Chapter 10

# XML-Formal Parameter Specification

## 10.1 Introduction

### 10.1.1 General Mechanism

This chapter describes the *formal* parameter specification of "complex" tools as qualified in 1.7 (i.e. **Apero**, **MicMac**, **Porto** and **Casa** for now). The general idea is that for each of these tools:

- the user defines the parameters of the tool essentially by giving a file containing an XML-tree;
- there is, in a given file, an XML structure that describes the XML trees that are valid parameters;
- these specification trees are all located in the directory `include/XML_GEN/`;
- for tools like **Apero**, **MicMac**, there is a file (i.e. `ParamMICMAC.xml` and `ParamApero.xml`) dedicated to the specifications;
- for other tools, the specifications are distributed in files that contain several specifications (`ParamChantierPhotogram.xml` and `SuperposImage.xml`). For example, the file `SuperposImage.xml` contains the tree `<CreateOrtho>` that specifies the ortho-photo tool **Porto** and the tree `<CreateCompColoree>` that specifies the registration superposition tool **CompColoree**;
- the `MicMac-LocalChantierDescripteur.xml` (that will be described in 11) is a file that describes the common characteristics of a "project", its specification is described in `ParamChantierPhotogram.xml`.

These specifications are said formal because they specify the syntax and the typing but not the "semantic". For example, they can specify that in certain contains there must exist unique tags `<X>` and `<Y>` whose values must be valid integers, but they cannot specify that these values must respect  $X > Y$ .

### 10.1.2 Format Specification

This XML specification mechanism is important to understand because it is not only parameters specification, but also used as a format specification of most of the data used by different tools as input or output.

For example, some programs of this tools use cylinders. They must be able to read and write a cylinder on a file to communicate between them. In some occasions, the user may need to provide its own cylinder as input to one of the program. The specification of the XML encoding of cylinder is in the file `SuperposImage.xml` where the structure `XmlCylindreRevolution` is specified. In this file the user can find the line<sup>1</sup>:

```
<!-- P0, P1 : axe of the cylinder, POnCyl : a point of the cylinder  
fixing radius and origins of cylindrics coordinates -->  
  
<XmlCylindreRevolution Nb="1" Class="true" ToReference="true">  
    <!-- P0-P1 , deux points sur la droite -->  
    <P0 Nb="1" Type="Pt3dr"> </P0>  
    <P1 Nb="1" Type="Pt3dr"> </P1>  
    <POnCyl Nb="1" Type="Pt3dr"> </POnCyl>  
</XmlCylindreRevolution>
```

1. here, a cylinder is coded by three points, two for the axis and one on the cylinder fixing the radius and the cylindrical coordinates origins

With these specifications, and some experience, the user could easily understand that he can create a vertical cylinder with axis on point (10,10) of radius 5 by:

```
<XmlCylindreRevolution>
  <P0> 10 10 -1 </P0>
  <P1> 10 10 1 </P1>
  <POnCyl> 13 14 25.4 </POnCyl>
</XmlCylindreRevolution>
```

Obviously, as for the parameters, this only specifies the syntax, not the semantic. By the way, for the most simple types, the syntax and the bit of semantic encoded in the tags name is often sufficient for understanding the specification.

### 10.1.3 Command Line

For the tools using XML setting, users must specify a mandatory file name followed by any number (generally 0) additional parameters. For example, with MICMAC, the syntax will be:

```
MICMAC FILE.xml Tag0=Val0 Tag1=Val1 ... TagN=ValN
```

With:

- FILE.xml, name of an existing file containing a valid <ParamMICMAC> tree;
- N pairs (Tag<sub>k</sub>, Val<sub>k</sub>) where Tag<sub>k</sub> is the name of an XML Tag and Val<sub>k</sub> is the associated value;
- Pairs Tag-Values of command line allow the user to tune the XML structure parameter without modifying the XML file;
- this modification mechanism is useful for interactive tests; its essential utility is internal when the tools are recursively calling themselves (for parallelization).

## 10.2 Tree Matching

This section describes the tree-matching process. For presentation purpose, we suppose that MICMAC is the program it applies to, and ParamMICMAC.xml is the file. Of course, it would be exactly the same with Apero and ParamApero.xml, or all other tools using XML file parameters.

The files ParamChantierPhotogram.xml and SuperposImage.xml contain specifications that may be used by all the tools.

MICMAC uses a tree-matching notion to specify which are the files with valid parameters: the tree contained in Fichier.xml must be complementary to the tree contained in the kinship of the <ParamMICMAC> tag from ParamMICMAC.xml.

Some definitions are necessary for being able to specify what MICMAC considers a valid tree-matching:

- the notions of trees, sons and bottom-level nodes are considered to be known;
- the trees from ParamMICMAC.xml are specification trees and the ones from Fichier.xml are the real trees;
- the specification trees they all have an attribute called arity attribute.

### TO FRENCH

Un arbre effectif est appariable sur un arbre de spécification si et seulement si :

- ils ont le même nom de tag;
- chaque fils, de l'arbre effectif est appariable sur un fils de l'arbre de spécification;
- chaque fils de l'arbre de spécification est appariable sur un ensemble de N fils de l'arbre effectif, où N est compatible avec l'arité de l'arbre de spécification (cette définition implique que, par exemple, si Nb =?, l'ensemble des noeuds appariables de l'arbre effectif peut être vide);
- lorsque le noeud est terminal, le noeud de l'arbre de spécification porte un attribut Type qui impose éventuellement des contraintes sur la valeur du noeud de l'arbre effectif;

On notera que, pour l'instant du moins, l'ordre des fils n'importe pas pour définir si deux arbres sont appariables. Il est déconseillé d'utiliser cette propriété (des versions ultérieure pourront imposer une conservation de l'ordre).

Soit par exemple l'arbre de spécification suivant :

```
<UnTestParam>
  <ListTestCpleHomol Nb="*">
    <PtIm1 Type="Pt2di" Nb="1" > </PtIm1>
```

```

        <PtIm2 Type="Pt2di" Nb="1" > </PtIm2>
    </ListTestCpleHomol>
    <NomModule Nb="1" Type="std::string" > </NomModule>
    <NomAux Nb="?" Type="std::string" >     </NomAux>
</UnTestParam>
```

Les deux arbres effectifs ci dessous sont appariables sur l'exemple précédent :

```

<UnTestParam>
    <ListTestCpleHomol>
        <PtIm1> 1 2 </PtIm1>
        <PtIm2> 2 3 </PtIm2>
    </ListTestCpleHomol>
    <NomModule > UnNom </NomModule>
    <ListTestCpleHomol>
        <PtIm1> 0 0 </PtIm1>
        <PtIm2> 0 0 </PtIm2>
    </ListTestCpleHomol>
</UnTestParam>

<UnTestParam>
    <NomModule > UnNom </NomModule>
    <NomAux > UnAutreNom </NomAux>
</UnTestParam>
```

L'arbre effectif ci dessous n'est pas appariable sur l'arbre de spécification pour (au moins) les raisons suivantes

- : — dans l'arbre de spécification <UnTestParam> n'a pas de fil nommé <Toto> ;
- dans l'arbre effectif, le contenu de <PtIm1> et <PtIm2> ne correspondent pas à ce qui est attendu pour le type **Pt2di** (ce qui signifie *Point 2D entier*, et attend de lire deux valeurs entière);
- dans l'arbre de spécification <UnTestParam> a un fils nommé <NomModule>, d'arité 1 qui ne se retrouve pas dans l'arbre effectif;
- <NomAux> n'a pas la bonne arité (2 pour 0 ou 1 autorisé);

```

<UnTestParam>
    <Toto> n'a pas d'homologue </Toto>
    <ListTestCpleHomol>
        <PtIm1> 1 2 3 </PtIm1>
        <PtIm2> 2   </PtIm2>
    </ListTestCpleHomol>
    <NomAux > UnAutreNom </NomAux>
    <NomAux > UnAutreNom </NomAux>
</UnTestParam>
```

## 10.3 Types des nœuds terminaux

### 10.3.1 Types généraux

Dans l'arbre de spécification chaque noeud terminal comporte un attribut de type nommé **Type**. Cet attribut **Type** est utilisé, lors de la génération automatique de code (voir chapitre 33), pour déterminer le type de la classe C++ qui doit être utilisée pour représenter la valeur contenue dans le noeud apparié de l'arbre effectif.

Cet attribut **Type** détermine aussi si la valeur (=chaîne de caractères) contenue dans un champ est valide. Les types existants et leur pré-requis sur les valeurs sont :

- **std::string** aucune restriction sur la valeur;
- **bool** la valeur doit être dans {0, 1, *true*, *false*} (0 étant équivalent à *false* et 1 à *true*);
- **int** un seul entier;
- **double** un seul réel;

— `std::vector<double>` un nombre quelconque de réels;  
 — `std::vector<int>` un nombre quelconque d'entiers;  
 — `Pt2di` deux entiers ( $x$  et  $y$ );  
 — `Pt2dr` deux réels ( $x$  et  $y$ );  
 — `Pt3dr` trois réels ( $x$ ,  $y$  et  $z$ );  
 — `Box2dr` quatre réels ( $x_0$   $y_0$   $x_1$   $y_1$ );  
 — les types énumérés, qui sont décrits à la section suivante, et pour lesquels la valeur doit appartenir à l'ensemble des valeurs énumérées par le type;

Lorsque le type terminal est un vecteur (`int` ou `double` pour l'instant), la syntaxe est "entre crochet avec virgule comme séparateur", par exemple avec un double [1,2,3,4].

### 10.3.2 Types énumérés

Dans le fichier de spécification `ParamMICMAC.xml`, avant la définition de l'arbre de spécification `<ParamMICMAC>`, figurent la définition des types énumérés.

La définition d'un type énuméré, de nom `UnType`, et pouvant prendre les valeur `Val-1 Val-2 ... Val-N` se fait selon la syntaxe :

```
<enum Name="UnType">
  <Val-1> </Val-1>
  <Val-2> </Val-2>
  ...
  <Val-N> </Val-N>
</enum>
```

Les types énumérés n'ont pas forcément été défini dans `ParamMICMAC.xml`. Lorsqu'ils ont une utilisation en dehors de `MICMAC`, ils sont définis dans un des fichiers généraux ( `ParamChantierPhotogram.xml` ...). Par exemple le type énuméré `eModeGeomMNT` spécifie le tag `<GeomMNT>` dans le fichier `ParamMICMAC.xml`, mais il est défini dans dans `ParamChantierPhotogram.xml` (parce qu'il est utilisé plusieurs fois et dans plusieurs fichiers : `ParamChantierPhotogram.xml`, `ParamMICMAC.xml`, `SuperposImage.xml`).

## 10.4 Définition de types d'arbres

Lorsque un même type d'arbre est utilisé plusieurs fois, MicMac utilise en général un mécanisme de définition de type qui peut être ensuite référencé.

Lorsque de la définition du type d'arbre , l'attribut `ToReference` est à `true`. Ensuite, lors de l'utilisation , l'attribut `RefType` a pour valeur le nom du type à référencer.

Par exemple on pourrait avoir :

```
<Personne Nb="1" Class="true" ToReference="true">
  <Age Nb="1" Type="double">    </Age>
  <Nom Nb="1" Type="std::string">   </Nom>
</Personne>
...
<Club Nb="1">
  <Membre Nb="*" RefType="Personne"> </Membre>
  <NomClub Nb="1" Type="std::string">  </NomClub>
</Club>
...
```

Et une utilisation correcte :

```
<Club>
  <NomClub> Contributeurs MicMac </NomClub>
  <Membres> <Age> 45 </Age> <Nom> MPD </Nom> </Membres>
  <Membres> <Age> 35 </Age> <Nom> GMaillet </Nom> </Membres>
  <Membres> <Age> 32 </Age> <Nom> DBoldo </Nom> </Membres>
</Club>
```

```

<ListeNMP>
  <NMP> <Num>-1</Num> <Pi> 3.0</Pi> </NMP>

  <NMP> <Num>0</Num> <Mes>Bonjour</Mes> </NMP>
  <NMP> <Num>1</Num> <Pi Portee="Globale">3.14</Pi> </NMP>
  <NMP> <Num>2</Num> <Mes>Au Revoir</Mes> </NMP>
</ListeNMP>

```

Figure 10.1: Exemple d'utilisation du mode différentiel

Les types qui sont utilisés par plusieurs outils sont définis dans un fichier général et ensuite utilisés plusieurs fois dans les différents outils. Lorsque le fichier de définition est différent de l'utilisation, on trouvera dans le tag d'utilisation un attribut **RefFile** indiquant le fichier de définition<sup>2</sup>. Par exemple :

- le type **ChantierDescripteur** est utilisés par tous les outils un peu complexes;
- il est défini dans **ParamChantierPhotogram.xml**;
- il est utilisé dans **ParamApero.xml**, sous le tag **DicoLoc** et dans **ParamMICMAC.xml** sous le tag **DicoLoc** (mais il n'y a rien d'obligatoire à ce qu'à l'utilisation le nom soit toujours le même);

Dans **ParamApero.xml** on trouvera :

```

<DicoLoc Nb="?" RefType="ChantierDescripteur"
          RefFile="ParamChantierPhotogram.xml"
        >
</DicoLoc>

```

## 10.5 Mode "différentiel"

Cette section décrit un fonctionnement assez spécifique, qui n'est utilisé aujourd'hui que pour le paramétrage de MicMac et sur un seul de ses tag. Cependant, il est très important dans ce cas particulier.

Un nœud de l'arbre de spécification, d'arité +, peut avoir un attribut **DeltaPrec** à 1. Ce n'est le cas aujourd'hui que du nœud **<EtapeMEC>** correspondant aux étapes de la mise en correspondance, mais celui-ci joue un rôle essentiel dans le paramétrage algorithmique. Lorsque **DeltaPrec=1** (par défaut il vaut 0), l'évaluation de la liste<sup>3</sup> des valeurs de l'arbre effectif se fait en mode dit différentiel.

Le mode différentiel a été ajouté pour tenir compte des situations où, en général, la liste des valeurs est "grande" avec une forte corrélation entre les valeurs successives. C'est le cas par exemple de **<EtapeMEC>** où souvent, d'une étape à l'autre, tous les paramètres algorithmiques sont conservés et seule la résolution change. L'objectif est alors d'offrir un syntaxe, qui sans perdre de généralité, permette dans les cas les plus courants de ne spécifier que les attributs qui diffèrent de l'étape précédente. Formellement, le fonctionnement est le suivant :

- le premier élément de la liste effective ne fera pas partie du résultat utilisé par MICMAC, il constitue l'initialisation de la *valeur courante*;
- pour les éléments suivant, la valeur rajoutée est constituée de la valeur courante, modifiée des fils qui sont explicités dans la liste effective;
- les fils explicités dans la liste effective ne modifie la valeur courant que si ils ont l'attribut **Portee** qui a la valeur **= "Globale"**;

Soit par exemple l'arbre de spécification suivant :

```

<ListeNMP>
  <NMP Nb="+" DeltaPrec="1">
    <Num Nb="1" Type="int"> </Num>
    <Mes Nb="?" Type="std::string"> </Mes>
    <Pi Nb="?" Type="double"> </Pi>
  </NMP>
</ListeNMP>

```

Avec l'arbre effectif de la figure 10.1 c'est la liste de la figure 10.2 qui sera utilisée par MICMAC.  
On remarque que :

---

2. ceci n'existe pas pour les type énumérés  
3. une liste de valeur car l'arité +, indique un nombre quelconque  $\geq 1$

```
{Num=0 ; Pi=3.0 ; Mes="Bonjour";}
{Num=1 ; Pi=3.14 ;}
{Num=2 ; Pi=3.14 ; Mes="Au Revoir";}
```

Figure 10.2: Résultat de l'utilisation du mode différentiel

```
<ListeNMP>
<NMP> <Num>-1</Num> <Pi>3.0</Pi> </NMP>
<NMP> <Num>0</Num> <Pi>3.0</Pi> <Mes>Bonjour</Mes> </NMP>
<NMP> <Num>1</Num> <Pi>3.14</Pi> </NMP>
<NMP> <Num>2</Num> <Pi>3.14</Pi> <Mes>Au Revoir</Mes> </NMP>
</ListeNMP>
```

Figure 10.3: Exemple d'utilisation du mode différentiel

- la première valeur n'est pas ajoutée dans la liste mais a pour effet de donner une valeur initiale à **<Pi>**;
- la modification de **<Pi>** sur la valeur **Num=1** se propage à la valeur **Num=2** car la modification se fait avec l'attribut **Portee="Globale"**;
- la modification de **<Mes>** sur la valeur **Num=0** ne se propage pas (**<Mes>** est absent de **Num=1**);

## 10.6 Autres caractéristiques

### 10.6.1 Modification par ligne de commande

La section 10.1.3 a indiqué qu'il était possible dans certain cas de modifier la valeur des tags par la ligne de commande.

Un arbre du fichier de spécification, de nom **UnTag** est modifiable dans le fichier effectif par la ligne de commande ssi :

- c'est un noeud terminal;
- sont attribut d'arité vaut 1 ou ?;
- tous ses descendants ont un arité de 1;

Ces règles, un peu restrictives, permettent d'éviter que la modification soit ambiguë ou qu'elle conduise à définir une syntaxe trop compliquée (par exemple pour saisir des structures imbriquée). La valeur passée en ligne de commande écrase celle qui existait éventuellement dans le fichier effectif.

### 10.6.2 Valeurs par défaut

Lorsqu'un noeud de l'arbre de spécification est terminal, et que son symbole d'arité est ?, alors il peut contenir un attribut de nom **Def** indiquant la valeur par défaut qui sera donnée à ce tag s'il n'est pas spécifié par l'utilisateur.

### 10.6.3 Fichiers de paramètres générés

Dans le répertoire de mise en correspondance, MICMAC génère deux fichiers xml :

- un fichier **Fichier\_Ori.xml** qui est une simple recopie, caractère par caractère, de **Fichier.xml**; cette copie est faite pour garder une trace des paramètres avec lesquels a été lancé MICMAC;
- un fichier **Fichier\_Compl.xml**, qui est un **dump** de la structure mémoire qui a été associée à **Fichier.xml**; **Fichier\_Compl.xml** diffère de **Fichier.xml**, notamment à cause des valeurs par défaut et du mode différentiel;

Ainsi avec l'exemple de la figure 10.1, le fichier **Fichier\_Compl.xml** contiendra le texte de la figure 10.3:

# Chapter 11

## Names Convention Organization

This chapter describes how the user can specify the names convention and organization and how the different programs communicate and share the naming. To be honest, the mechanism is certainly more complex than it could and should be; the idea, when building these mechanisms, was to be able to integrate any name convention without having to rename files. This objective is not completely satisfied, but the complexity is here. It has to be assumed ...

### 11.1 General Organization

#### 11.1.1 Requirements

We suppose, in all this document, that all the initial images necessary for the process are located in the same directory. Although it may not be required for all the steps, I cannot guarantee that this can be avoided, and I am quite sure that it is a good idea to do it this way.

We will call this directory the project directory and we will refer to it as `ProjDIR`.

#### 11.1.2 The struct `ChantierDescripteur`

The XML struct describing the name convention is named `ChantierDescripteur`. Its formal specification can be found in the file `ParamChantierPhotogram.xml`. The main parts of a `ChantierDescripteur` are:

- `KeyedSetsOfNames`, it allows to describe sets of strings, and to give them a key identifier that will be used to reference them;
- `KeyedNamesAssociations`, it allows to describe string mapping, and to reference them by key; when these mappings are invertible, the user can specify the invert function (which is often required);
- `KeyedSetsOfRels`, it allows to describe relations between strings; these relations are always given between existing sets of strings (created by a `KeyedSetsOfNames`); of course, to these relations are given key identifier used to reference them.

`ChantierDescripteur` contains also many other structures, some are obsolete, others are not so often used and will be introduced at the end of the chapter.

#### 11.1.3 The Files Containing `ChantierDescripteur`

The programs using `ChantierDescripteur`<sup>1</sup> expect to find it in 3 possible locations:

- `micmac/include/XML_GEN/DefautChantierDescripteur.xml`, this file contains the predefined conventions; the conventions defined in this file are known from all the tools; the objective is that, in a short future these predefined and "standard" conventions should be sufficient in 95% of projects; it is a good idea to use these conventions when they exist; *user should never modify this file*;
- `ProjDIR/MicMac-LocalChantierDescripteur.xml`, this file contains conventions specific to the project; the name of this file must be exactly `MicMac-LocalChantierDescripteur.xml`, this is how the tools recognize this special file; the existence of this file is not mandatory and simple projects using standard conventions will omit it;
- `DicoLoc`, for some tools (including `MicMac` and `Apero`) it is possible to define conventions directly in the parameters file, this may be convenient if a convention that will be used only once is required.

---

1. almost all now

### 11.1.4 Overriding and Priority

Is was seen before that conventions are given keys (or names), overriding of key is possible but must obey certain rules:

- it is not possible to define twice a key in the same file; for example if you try to define two sets, with the same name `Key-My-Set` in `ProjDIR/MicMac-LocalChantierDescripteur.xml` you will get an error;
- it is possible to override a key between different files; the priority rule is naturally to favor the more local definition; that is:
  - definitions of `DicoLoc` override definitions of `MicMac-LocalChantierDescripteur.xml`;
  - definitions of `MicMac-LocalChantierDescripteur.xml` override definitions of `DefautChantierDescripteur.xml`.

It is sometimes useful to override in `MicMac-LocalChantierDescripteur.xml` the definition of a default key given in `DefautChantierDescripteur.xml` because this is the easiest way for parameterizing a tool. For example:

- the tool `bin/MpDcraw`, for demosaicing image, expects, in certain condition, to get for each image a "chromatic aberration calibration file";
- for a given image, `bin/MpDcraw` computes the name of chromatic calibration file using the rule associated to the key `Key-Assoc-Calib-Coul`;
- `Key-Assoc-Calib-Coul` is defined in `DefautChantierDescripteur.xml` with a rule that is convenient for my camera but that may be inappropriate for the user, so it may be override in `MicMac-LocalChantierDescripteur.xml`.

## 11.2 Regular Expression and Substitution

### 11.2.1 Regular Expression

Regular expressions are a very powerful tool for concise description of string subsets. For example the expression:

```
^.*_(..?)x[0-9]{3}.*
```

Means a string that contains a `_`, followed by 1 or 2 characters, followed by `x`, followed by 3 digits and ending by anything.

It is also a standard, in fact as current in informatics, it is a standard with a lot of variants ... The regular expressions used are the so called "posix-regular expressions". This standard has two advantages:

- there exists standard library that made the implementation easy for me;
- the user can easily find a complete description of regex. On Unix, one may type `man regex`. Of course, this is not very didactic and you may prefer to search on the web, you will get a lot of interesting pages by simply typing "regular expression" on your favorite browser. Be sure to refer to the posix page, and not the other variants who may be different (pearl regular expression is a current one).

### 11.2.2 Substitution

## 11.3 Helps tools in names manipulation

### 11.3.1 TestKey

The command `mm3d TestKey aPat` will print on the console the set of name corresponding to a pattern `aPat` (or more generally to key of subset). By default it is limited to 10 Name, the `Nb` optionale parameters can increase this default value.

### 11.3.2 TestMTD

The `TestMTD` will print the meta data as understood by MicMac :

```
mm3d TestMTD _MG_0082.CR2
FocMm 35
Foc35 34.2
Cam [Canon EOS 5D Mark II]
```

### 11.3.3 TestNameCalib

The **TestNameCalib** will print the computed name of internal calibration which by default associated an image :

```
mm3d TestNameCalib _MG_0082.CR2
./Ori-TestNameCalib/AutoCal_Foc-35000_Cam-Canon_EOS_5D_Mark_II.xml
```

## 11.4 Describing String Sets

## 11.5 Describing String Mapping

### 11.5.1 Advanced association

Sometime it may be difficult to describe a given set with patterns. The **Filter** option allow to describe more advanced feature. It can contain optionnaly :

- a Min and Max value;
- ...

In the folder **Data/Arbre**, the key **TEST-Filter** of file **MicMac-LocalChantierDescripteur.xml** give an example use. We can test :

```
m3d TestKey DSC.*jpg KeyAssoc=TEST-Filter Nb=100
Num= 0 Name=DSCF2774_L.jpg Key=ONE
Num= 1 Name=DSCF2774_R.jpg Key=ONE
Num= 2 Name=DSCF2775_L.jpg Key=ONE
Num= 3 Name=DSCF2775_R.jpg Key=ONE
Num= 4 Name=DSCF2776_L.jpg Key=TWO
Num= 5 Name=DSCF2776_R.jpg Key=TWO
Num= 6 Name=DSCF2777_L.jpg Key=TWO
Num= 7 Name=DSCF2777_R.jpg Key=TWO
NB BY RFM 8
```

## 11.6 Describing String Relations

## 11.7 Filters and In-File Definition



# Chapter 12

## Use cases for 2D Matching

The chapter covers examples of using MicMac when the matching problem is a 2 dimensionnal one. This can occure in the following situation:

- the problem is intrinsically 2 dimensional, for example in movement detection (see 12.3); this can be done with a simplified tool;
- the problem should be 1 dimensional, but the orientation parameters are unknown (see 12.2) or, at least, "very" inaccurate; at the time being, this requires a parametrization of **MICMAC** with **XML** file;
- the problem should be 1 dimensional, the orientation parameters have been computed, but for some reason, there are doubts on their accuracy and the user want to check this accuracy (see 12.1.1); this can be done with a simplified tool;

### 12.1 Checking orientation

#### 12.1.1 For Conik Orientation

In image geometry **MicMac** has "special" modes where the matching can be done taking into account a possible inaccuracy of the orientation. Although, it can be used to match badly oriented images, this is generally not a good idea (it's a better idea to understand what was wrong in orientation or acquisition and to correct it !!). However, when the user has doubts on orientation parameters, these tool can be convenient to check these orientations. In these mode :

- the matching is done in image geometry : there is a master image, and the  $X, Y$  are the pixel of this master image;
- there is *only* one secondary image;
- for each pixel of the master image, two value are computed, one represents the depths and the other represents the "transverse parallax" : it is the displacement in the direction orthogonal to the epipolar;

These mode can be fairly complex to use directly in **XML** mode, so it's generally sufficient to use the simplified tool **MMTest0rient**. The first arguments should be quite obvious from inline help (argument after **PB** are relative to satellite case, see 12.1.2) :

```
mm3d MMTest0rient -help
```

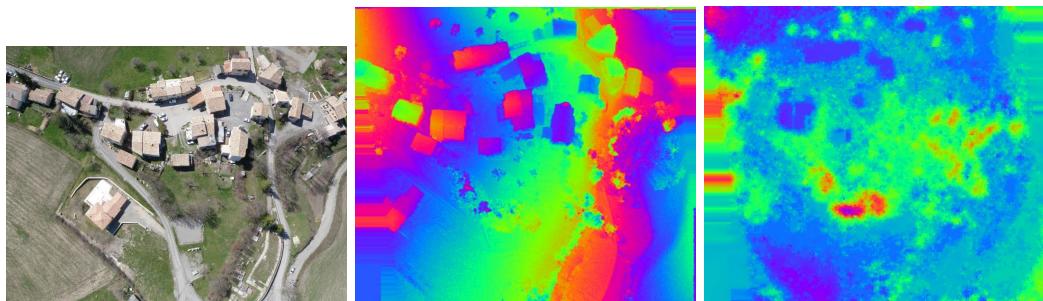


Figure 12.1: Image, depth map and transverse parallax with Draix data set (images P4090163.JPG and P4090134.JPG)

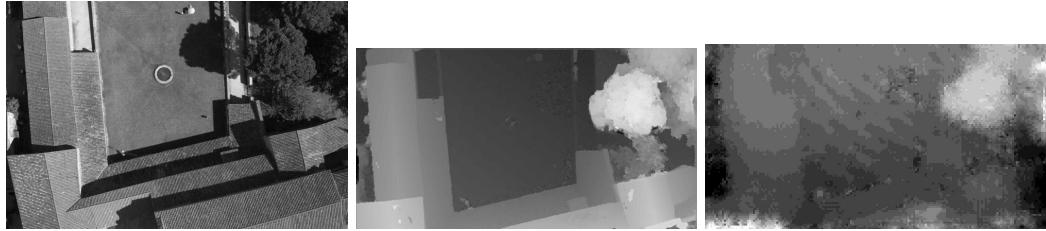


Figure 12.2: Image, depth map and transverse parallax with MiniCuxa data set (images Abbey-IMG\_0208.jpg and Abbey-IMG\_0209.jpg), the correlation between two parallax is lightly visible

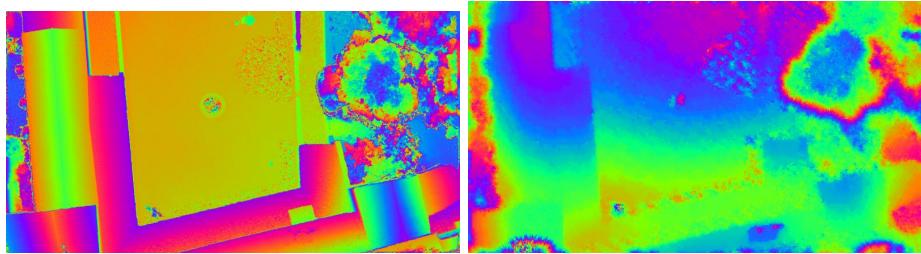


Figure 12.3: depth map and transverse parallax with full resolution Cuxa images, correlation between both is clearly visible, amplitude is  $\pm 1$  pixel.



Figure 12.4: depth map and transverse parallax with 10 cm image of Munich, acquired with a DMC, except in "noisy part" like the river, amplitude of transverse parallax is  $\pm 0.1$  pixel

```
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {First Image}
 * string :: {Second Images}
 * string :: {Orientation}
Named args :
 * [Name=Dir] string :: {Directory, Def=./}
 * [Name=Zoom0] INT :: {Zoom init, pow of 2 in [128,8], Def depend of size}
 * [Name=ZoomF] INT :: {Zoom init, pow of 2 in [4,1], Def=2}
 * [Name=PB] bool :: {Push broom sensor (GRID)}
 * [Name=GB] bool :: {Gen Bundle Mode}
 * [Name=M0ri] string :: {Mode Orientation (GRID or RTO) , Mandatory in PB}
 * [Name=ZMoy] REAL :: {Average Z, Mandatory in PB}
 * [Name=ZInc] REAL :: {Incertitude on Z, Mandatory in PB}
 * [Name>ShowCom] bool :: {Show MicMac command (tuning purpose)}
```

The result of transverse parallaxes is stored in images Px2..., the number of the last and most accurate image depends of the other parameters, so you have to check what is present on the directory **GeoI-Px**.

How can these image be used ? Basically, the idea is that with a "perfect" orientation the transverse parallax should be zero on all the image. In real life, this is more complicated, because this parallax can be noisy ( $2d$  general matching problem can be fairly ambiguous). So what is important is not only the amplitude of the transverse parallax but also its spatial analysis : is there systematism in this parallax ? Does it present low frequency movement ? Is this transverse parallax correlated to the depth map ? ... It is not so easy to make an automatic quantitative analyze of the results and the first purpose of this tool is to help human expertise in a qualitative analysis of the result. The **MMTestOrient** is illustrated on three examples (in each case with **ZoomF=1**) :

- on figure 12.1, with image from the Draix data set; in this case the transverse parallax is a bit noisy but does not show obvious systematism;
- on figure 12.2, the amplitude of transverse parallax do not seem very high, but here it is computed on reduced images, and conversely one can guess some systematism and a correlation between the depth and the transverse parallax;
- on figure 12.3, the full resolution image of Cuxa have been used (they are not in the data-set); in this case, the tool shows clearly a high systematism in the transverse parallax, if we except the noisy part like the tree<sup>1</sup> the amplitude is almost  $\pm 1$  pixel between highest and lowest value; furthermore the high correlation between two parallax maybe originated by a calibration problem, probably due to focal length;
- figure 12.4 presents an almost perfect orientation; with these 14144, 15552 coming from a *DMC* camera the amplitude of transverse parallax is  $\pm 0.1$  pixel on most of the image; the only part of the image where the amplitude is significantly higher is the river, but as can be seen on the depth image, the matching is very noisy here and the result has meaning in such part;

### 12.1.2 For Push-Broom Orientation

This tool can also be used with satellite images. Depending whether the orientation is provided in GRID/RTO or by RPCs (see Chapter 19), different input parameters are used. See the use case presented in Section 4.5 to learn how to handle the RPC input, and below find an example using the GRID/RTO:

- the third argument is interpreted as the postfix of the orientation file;
- the PB argument must be set to true;
- the M0ri must indicate the way the orientation file is stored (GRID for grid format, RTO for XML encoded RPC file);
- the ZMoy must indicate the average value of  $Z$ ;
- the ZInc must indicate the incertitude on  $Z$ ;

Here is an example (in this case, the orientation file of `./crop1.tif` is `./crop1.GRIBin`) :

```
mm3d MMTestOrient crop1.tif crop2.tif GRIBin PB=true M0ri=GRID ZMoy=0 ZInc=1000
```

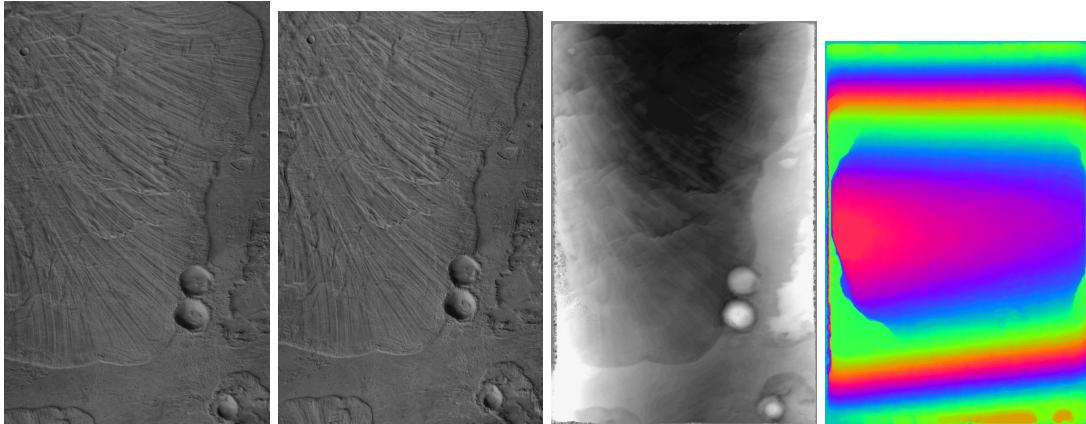


Figure 12.5: Mars data-set : the two images, the  $X$  parallax, in gray-level, and the  $Y$ -parallax in hue color

## 12.2 The Mars data-set

### 12.2.1 Description of the data set

The data can be found in the directory `Mars/` of directory `ExempleDoc/`. It consists of two stereo images acquired by sensor CTX during MRO mission on the planet Mars. In this case we do not have the physical model of the sensor, but we know that:

- the satellite is a pushbroom-satellite;
- it flights in the  $x$  direction.

### 12.2.2 Comment on the parameters

#### 12.2.2.1 Geometry

The tags controlling geometry are:

- `<GeomImages> eGeomImage_Hom_Px </GeomImages>` indicates the geometry of the acquisition, here it means that there is a principal homography  $H$ , let  $P_1 = x_1, y_1$  and  $P_2 = x_2, y_2$  be two homologous points, MicMac will compute  $U(P_1)$  and  $V(P_1)$  such that

$$P_2 = H(P_1) + (U(P_1), V(P_1)) \quad (12.1)$$

- the homography  $H$  is computed by MicMac from a set of homologous point;
- `<FCND_CalcHomFromI1I2> NKS-Assoc-CplIm2Hom@-Man@xml </FCND_CalcHomFromI1I2>` indicates where MicMac must look for the tie points (see directory `Homol-Man/`);
- `<GeomMNT> eGeomPxBiDim </GeomMNT>` indicates the geometry of restitution, the value `eGeomPxBiDim` indicates that what is computed is the pixel offset, in fact this value is mandatory when using `eGeomImage_Hom_Px`

#### 12.2.2.2 Matching

In this case, the two parallax directions have completely different meanings:

- the parallax 1 represents mainly the relief, it is expected to contain high frequencies;
- the parallax 2 represents mainly the error of the geometric model, it is expected to have low amplitude and low frequencies;

This asymmetry in the *a priori* knowledge of parallax is specified at different parts of the file :

- `<Px1IncCalc>` and `<Px2IncCalc>`, representing the global uncertainty on each parallax;
- `<Px1Regul>` and `<Px2Regul>`, representing the *a priori* knowledge of the regularity of each parallax;
- `<Px1PenteMax>` and `<Px2PenteMax>`, representing the *a priori* knowledge of the steep of each parallax;
- `<Px1Pas>` and `<Px2Pas>`, representing the discretization step (as Px2 is low frequency and low amplitude, we can compute it with higher precision);
- `<Px1DilatAlti>` and `<Px2DilatAlti>`, to gain some time, we decide not to re-estimate Px2 at the last step;

---

1. for tree the transverse parallax can be created by wind

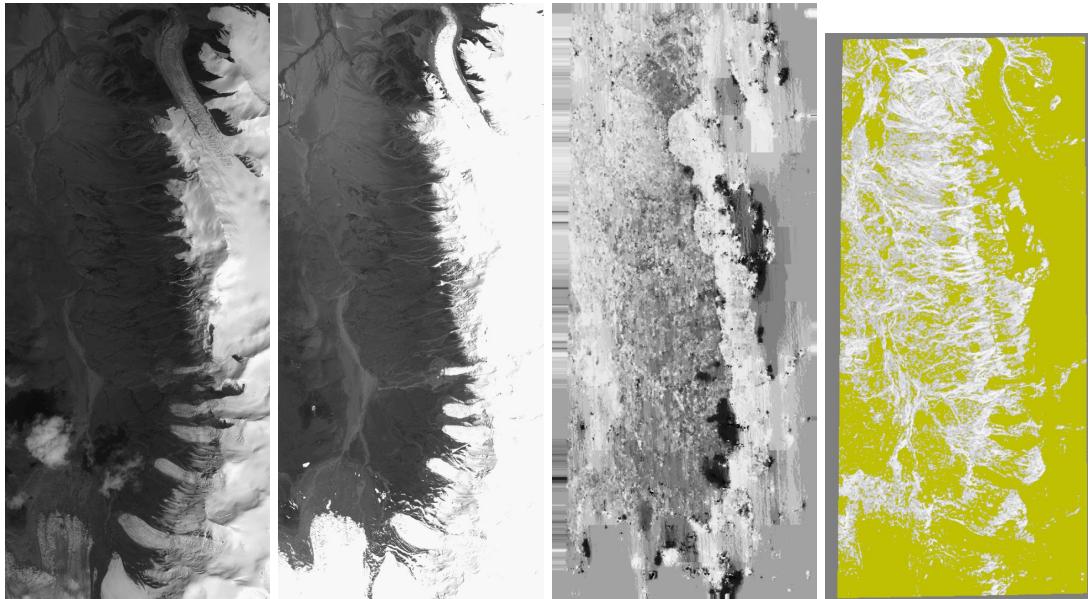


Figure 12.6: Gulya data set : the two ortho images, the  $X$ -parallax computed, and the correlation coefficient computed

### 12.2.2.3 Results

Figure 12.5 presents the two images and the results of the computed parallax. As expected:

- The Px1 contains mainly high frequency information on the relief;
- The Px2 contains mainly low frequency information on the geometry of the sensor.

## 12.3 The Gulya Earthquake Data-Set

### 12.3.1 Introduction

Since September 2011 CNES<sup>2</sup> has been funding a development for using MicMac for earthquake quantification. This development was made as a collaboration between CNES, CEA, IPGP and IGN/ENSG. The main developer of this part is Ana-Maria Rosu.

Although there exist other tools for doing this, the objective was:

- have a totally free tool, that scientists can use in open source mode;
- have a more parametrizable tool;

Although the study is not finished, the tool is now operational. The program has been tested on 3 real data set and several synthetic data set, and compared to several existing solutions working in frequency domain. From a purely subjective evaluation, these tests show that the results with MicMac are generally equivalent in quality to frequency approach and, on one of the real data-set, the results of MicMac were "better"<sup>3</sup>. One of the drawback of the dense approach of MicMac is the computation time : 15 minutes, with a 8 core computer, with the  $1600 \times 3600$  images of the Gulya data-set.

### 12.3.2 Description of the data set

The data can be found in the directory `SeismGuyla/` of directory `ExempleDoc/`. It consists of two *Spot 5* ortho photos of the same scene taken in 2002 and 2008. Between these two dates, an earthquake occurred and image matching can be used to localize the rupture and quantify the movement.

We want to use MicMac to measure very small displacements (around  $\frac{1}{10}$  pixel) in a context where the images are quite different. Figure 12.6 presents the two ortho images.

---

2. Centre National d'Etudes Spatiales, the French spatial agency

3. i.e. subjectively easier to interpret for scientist

### 12.3.3 Simplified interface

A simplified interface has been written. At the time being, it gives acces to few parameters, but it will evolve.

```
$ mm3d MM2DPosSism
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {Image 1}
 * string :: {Image 2}
Named args :
 * [Name=Masq] string :: {Masq of focus zone (def=none)}
 * [Name=Teta] REAL :: {Direction of seism if any (in radian)}
 * [Name=Exe] bool :: {Execute command , def=true (tuning purpose)}
```

An example of use :

```
mm3d MM2DPosSism 250802_ortho.tif 260608_ortho.tif Teta=1.5
```

### 12.3.4 Comment on the parameters

This section describes the "classical" interface using the XML parameters.

#### 12.3.4.1 Interpolation

Aiming at measuring very small displacements, we use a sinus cardinal interpolation :

- <ModeInterpolation> eInterpolSinCard </ModeInterpolation>
- <SzSinCard> 5.0 </SzSinCard> specifies the size of the kernel;
- <SzAppodSinCard> 5.0 </SzAppodSinCard> controls the shape of the appodization window (the general shape is a Tukey window, when SzAppodSinCard=SzSinCard, it turns to be a Hamming window);

#### 12.3.4.2 Image term

By default in MicMac, the image term is  $1 - Cor$  where  $Cor$  is the normalized cross correlation coefficient. In such data-sets, where there is a very important change locally, this can not be suitable because when there are changes of the nature (snow ...) the correlation has no signification and it is better to consider that there is no information. Three parameters are used here to control the meaning of the correlation:

- <CorrelMin> = $C^{min}$ , so that correlation bellow < $C^{min}$ > has no influence;
- <GammaCorrel> = $\gamma$ , with  $\gamma$  higher, higher is the influence of the correlation close to 1;
- <DynamiqueCorrel>=eCoeffGamma to activate the previous one ...

The following equations indicate how these parameters define the conversion from correlation to cost:

$$C_1 = \text{Max}(Cor, C^{min}), \quad (12.2)$$

$$C_2 = \frac{C_1 - C^{min}}{1 - C^{min}} \quad (12.3)$$

$$C_3 = C_2^\gamma \quad (12.4)$$

$$Cost = (1 - C_3) * (1 - C^{min}); \quad (12.5)$$

On figure 12.6, the image on the left presents the correlation coefficients. The yellow value corresponds to the threshold value (here < 0.5).

#### 12.3.4.3 Non isotropic regularization

It can happen that we have an *a priori* knowledge for favoring some direction of regularization. This can be done using in conjunction the following parameters of EtapeProgDyn:

- <NbDir>=N fixes the number of direction that will be explored;
- <Teta0>= $\theta_0$  fixes the angle of the favored direction;
- the directions that will be explored are  $\alpha_k = \theta_0 + k \frac{\pi}{N}, k \in [0, N - 1]$
- if the parameters <Px1MultRegul>= $V_1$  or <Px2MultRegul> are used, then the value of regularization in the direction  $\alpha_k$  is multiplied by  $V_1[k]$  ( $V_1$  is a vector);

In this example, we regularise more the direction close to  $\frac{\pi}{2}$ , with a weight  $\frac{1}{1+\frac{10}{N}*K}$ .

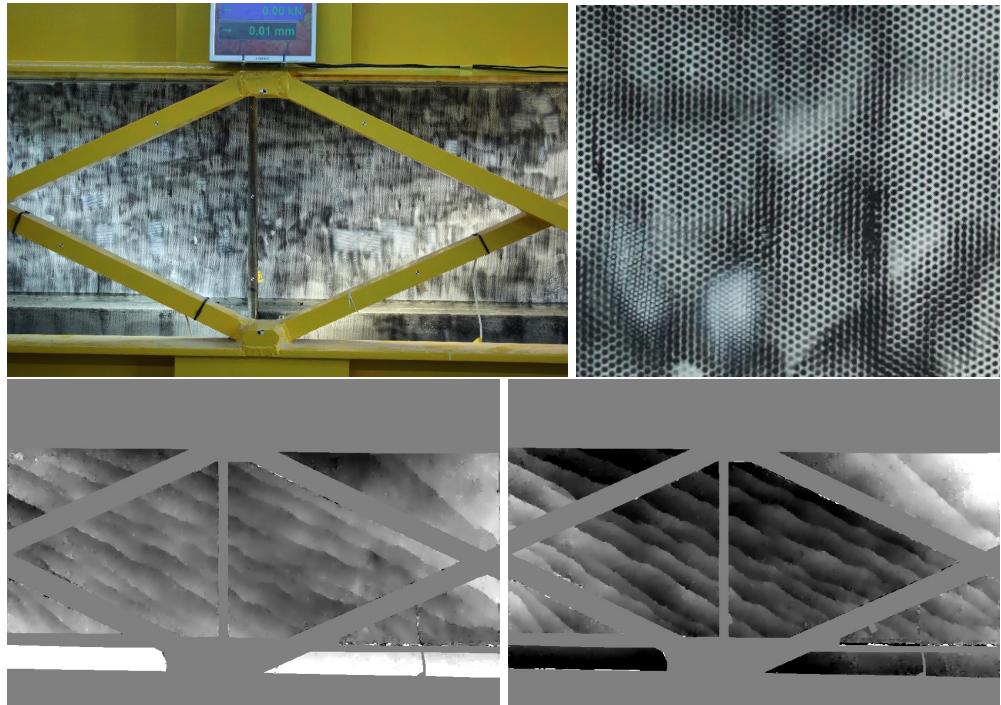


Figure 12.7: Concrete data'set, one of the image, a crop and the two displacement field

## 12.4 The Concrete Data-Set and civil engineering

### 12.4.1 Introduction

The data set **Concrete/** is an example of using MicMac for measuring very fine displacement in civil engineering. In this experiment, constraint were applied to a concrete beam, and a fix camera was used to measure the displacement during breaking phase. Figure 12.7 illustrate this data set :

- (up left) a full view of on of the two images used for this correlation;
- (up right) a zoom on a detail of this image, as it can be seen, the concrete a been painted to create an "optimal" texture for matching
- (bottom left) the x displacement (total amplitude is around  $\frac{1}{4}$  of pixel);
- (bottom right) the y displacement (total amplitude is around  $\frac{1}{4}$  of pixel).

### 12.4.2 Parametrizing

The file **Param-OneResol.xml** contain the MicMac's parametrizing used for this experience. Although the parametrizing should be quite easy to understand after March's exemple, we can do the following comments :

- the use of homographic model is well suited, it allow to model a possible translation of the camera; the 9 point used for the homography have been seized on the concrete, this way the computed displacement is exactly the deformation;
- we have not use the multi-scale approach because the quasi periodic texture uses was such that the reduced images where almost textureless at sub-resolution 8 and very aliased at resolution 4;

## 12.5 FDSC - a post-processing tool for 2D correlation results

**FDSC** (Fault Displacement Slip-Curve) is a post-processing tool developed by Ana-Maria Rosu. Like **MM2DPosSism**, it is also part of the collaboration between IPGP and IGN/ENSG, and funded by CNES through the TOSCA program.

Mainly dedicated to the geoscience community, **FDSC** computes offsets by stacking profiles across the fault on the correlation results file and at the end, draws the slip-curve which gives an overview of the displacement field.

**FDSC** can be found in the MicMac deposit (folder **fdsc/**). It is recommended to read the **readme.txt** before starting.

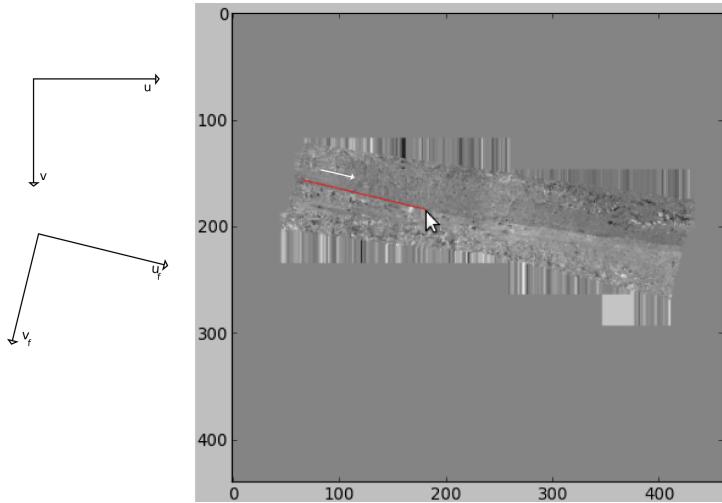


Figure 12.8: FDSC - drawing the fault trace;  $uv$  - image reference frame ( $\vec{u}$  - in epipolar direction,  $\vec{v}$  - in transverse direction);  $u_f v_f$  - fault reference frame ( $\vec{u}_f$  - parallel to the fault line,  $\vec{v}_f$  - perpendicular to the fault line)

In order to launch FDSC:

```
~/culture3d/fdsc$ ./fdsc.py
```

The FDSC's Qt interface is divided into three main blocks corresponding to the three steps of FDSC:

1. draw the fault trace
2. stack profiles across the fault
3. draw the slip-curve

### 12.5.1 Drawing the fault trace

A polyline is drawn on a parallax image file (Px1 - epipolar parallax; Px2 - transverse parallax) to describe the fault trace. The parallax image used for this step has to have the same resolution as the parallax images used when stacking, otherwise the fault trace is useless.

A file containing the points of the polyline describing the fault is saved (e.g. `trace.txt`) and later used to retrieve the fault when stacking profiles. The first drawn point of the fault trace (Fig. 12.8) is considered to be the fault origin. The drawing direction is the fault direction.

### 12.5.2 Stacking profiles

Perpendicular profiles (direction from left to right from the fault direction) are stacked to obtain the fault offsets. A stack of profiles is composed of numerous single profiles. The result is a “mean profile” where the noise is diminished and the offset trend comes out very well, making it easier to measure (see Fig. 12.9).

Parameters defining a stack:

- computing method: mean or weighted mean, median or weighted median of profiles;
- when a weighted method is chosen, the values of the correlation coefficients' image are considered as weights (these values express well the confidence in the corresponding parallax values), therefore the user must indicate the correlation coefficients' image, as well as the value of the exponent of weights.
- width : number of profiles taken into account, 1 pixel apart (it must be a odd number)
- length : length of profiles, in pixels (it must be a odd number)
- profile projection or offsets output direction: “Column”, “Line” (corresponding to  $u$  projection - only Px1 image is used and  $v$  projection respectively - only Px2 is used in stacks computation); “Parallel”, “Perpendicular” (profiles are projected in the fault parallel,  $u_f$ , and fault normal direction,  $v_f$ , respectively; both Px1 and Px2 images are needed for stacks computation).

The offsets values are saved into a file (e.g `offsets.txt++` to which a suffix is added by default depending on the chosen projection: `offsets_dirCol.txt`, `offsets_dirLine.txt`, `offsets_dirParal.txt`, `offsets_dirPerp.txt`).

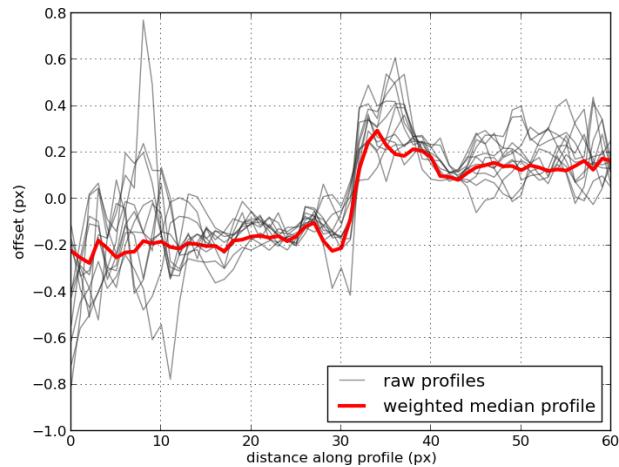


Figure 12.9: Single (raw) profiles perpendicular to the fault and a resulted stack

### 12.5.3 Drawing the slip-curve

The input file needed is the offsets output file and the slip curve will be drawn accordingly to these values.



## **Part II**

# **Reference documentation**



# Chapter 13

## Data exchange with other tools

### 13.1 Generalities

As many developers, in **MicMac/Apero**, I often could not resist to define my own format, sometime because I thought that existing format were not suited for what I needed and sometime just because I was in hurry and thought that the format question was a minor issue that could be dealt with later ... In fact, often, there were no ideal solution, because photogrammetry has such a long history and in many case it is often difficult to know what is "the" standard.

By the way, now that the user community is developing, it appears that the format issue is not a minor one. There are two chapters dealing with the format issue:

- the chapter 14 describes the internal format used by **Apero/MicMac** ; as all these format are open format and most of them in text mode, it should theoretically be sufficient to create interface at any point of the process;
- this chapter, which describes some facilities that have been developed to communicate with some existing *de facto* standard;

The directory **TestPM/** on **ExempleDoc/** contains example that will be used for illustrating this chapter.

### 13.2 Orientation's convention

Basically there are two conventions in **MicMac/Apero**, that can create difficulties for importing orientations :

- for internal orientation, all the calibration are given in pixel; although this is quite natural with digital camera, this is not the "photogrammetric tradition" which, dealing initially with analog images, rather work in millimeters;
- for external orientation, I store directly the rotation matrix which has few ambiguities<sup>1</sup> ; however many softwares use angles, the problem with angles being that almost each software has its own convention ...

This section describes how calibration in millimeters and some orientation in angles can be directly imported in **Apero/MicMac**. If you have data that cannot be imported with the facilities described here, I may be open to add the facilities in **MicMac/Apero** if I am convinced that I can do it easily and that it will be usefull to others; concretely it means that at least the following conditions should apply :

- you have some document that describe formally the convention/format;
- you have a reasonably small data-set (image+meta data) that can be used to validate the import/export;
- these data can be distributed, with acknowledgment, as open data on the **MicMac/Apero** site;

#### 13.2.1 Internal orientation

For using angle in external orientation, you can use the **<CodageAngulaire>** option under the **<ParamRotation>**. On **Ori-OMApero/** from data set **TestPM/** we can open for example the file **Orientation-DSC.6443.jpg.xml** :

```
ExportAPERO>
    <OrientationConique>
    ...
    <FileInterne> Ori-OMApero/AutoCal350.xml</FileInterne>
```

---

1. even if internally I use 3 angles in optimization step

```

<RelativeNameFI>true</RelativeNameFI>
<Externe>
...
    <Centre> 852.7267 374.238 663.2607 </Centre>
    <ParamRotation>
        <CodageAngulaire> -1.824705 48.7837 90.50795 </CodageAngulaire>
    </ParamRotation>
</Externe>
<ConvOri>
    <KnownConv> eConvAngPhotoMDegre </KnownConv>
</ConvOri>
</OrientationConique>
</ExportAPERO>

```

Inside the `<CodageAngulaire>` tag, there are three angles. As there are many conventions for coding rotation by three angles, a very important part is :

- `<KnownConv> eConvAngPhotoMDegre </KnownConv>`

Here this tag means that the angles are coded using the photo modeler software convention. The convention that have been recently tested and worked correctly are :

- `eConvAngPhotoMDegre` works for `PhotoModeler` and `Bingo`;
- `eConvAngPhotoMGrade` same as previous with angle in grade;

There exist convention that I have not tested for a long time, and I do not have dataset to check if they work. You can give it a try, if it works perfect, else you can contact me:

- `eConvAngErdas`
- `eConvAngErdas_Grade`
- `eConvAngLPSDegree`

### 13.2.2 External orientation

When you use a calibration in mm, and do not want to "translate" by hand the values in pixel, the optional `<OrIntGlob>`, of `<CalibrationInternConique>` allows to add an affinity to the internal orientation. In some way it is redundant with the `<OrIntImaM2C>` described in 14.1.2; however the `<OrIntImaM2C>` must be defined for each image and is rather convenient when dealing with the scanning of analog image. In the file `Ori-OMApero/AutoCal350.xml` of data set `TestPM/` we find the import of a calibration that was made with `PhotoModeler` :

```

<ExportAPERO>
    <CalibrationInternConique>
        <KnownConv>eConvApero_DistM2C</KnownConv>
        <PP>12.0347079589749999 8.02237118679700068</PP>
        <F>37.7672246925810029</F>
        <SzIm>3872 2592</SzIm>
        <OrIntGlob>
            <Affinite>
                <I00>0 0</I00>
                <V10>0.0061983471000000035 0</V10>
                <V01>0 0.0061983471000000035</V01>
            </Affinite>
            <C2M>true</C2M>
        </OrIntGlob>
        <CalibDistortion>
            <ModPhgrStd>
                <RadialePart>
                    <CDist>12.0347079589749999 8.02237118679700068</CDist>
                    <CoeffDist>-6.64705e-05</CoeffDist>
                    <CoeffDist>6.0366999999999873e-08</CoeffDist>
                    <CoeffDist>-6.1999999999999798e-11</CoeffDist>
                </RadialePart>
                <P1>1.3197199999999996e-05</P1>
                <P2>8.0770700000000058e-06</P2>
            </ModPhgrStd>
        </CalibDistortion>
    </CalibrationInternConique>
</ExportAPERO>

```

```

<b1>0</b1>
<b2>0</b2>
</ModPhgrStd>
</CalibDistortion>
</CalibrationInternConique>
</ExportAPERO>

```

The tag `<C2M>` mean that the affinity is given from camera coordinate to word coordinate. For example here , to transform a pixel  $x,y$  in milimeter we use  $x * 0.0061\dots, y * 0.0061\dots$ . You can check the files `AutoCal350-V0.xml` and `AutoCal350-V1.xml`, they define exactly the same calibration, the first with `C2M=true` and the second `C2M=false`. The files `AutoCal350-V2.xml` and `AutoCal350-V3.xml` define also the same calibration than `AutoCal350-V0.xml`, using the tag `<I00>`, they correspond to the case where `<CDist>` and `<PP>` are defined relatively to the center of sensor (instead of  $(0,0)$ ).

The following commands were used to test the import of orientation :

```

Tapioca All ".*jpg" 1200
AperiCloud ".*jpg" OMapero
Malt GeomImage "DSC_64(56|57|43).jpg" OMapero Master=DSC_6457.jpg

```

## 13.3 Conversion tools

These section describes some conversion tools written to transform datas from some text format to the `Xml` format used in `Apero/MicMac`.

### 13.3.1 Ground Control Point Conversion: GCPConvert

The command `GCPConvert` is used to:

- transform a set of ground control points from most text format to `MicMac's Xml` format
- transform the ground control points into an euclidean coordinate system, suitable for `MicMac`.

#### 13.3.1.1 File format conversion with GCPConvert

Consider the file `CP3D.txt` of directory `TestPM/`, here are two lines extracted :

```

157      233.28   144.03    103.05  0.00332    0.0034    0.0039
158      317.011  -0.00000   0.0000  0.0053     0.0060    0.0071
...

```

The format should be quite obvious to human, each line contains the name of point, then  $X,Y,Z$  , then accuracy on  $X,Y,Z$ . However, we have to specify this format to the computer. One way to do it, is to add first line in the file that specifies the format. This is done in the file `CP3D_Format.txt`, the beginning is :

```

#F= N X Y Z  Ix Iy Iz
157      233.28   144.03    103.05  0.00332    0.0034    0.0039
158      317.011  -0.00000   0.0000  0.0053     0.0060    0.0071
...

```

The first line must be interpreted like :

- the first character `#` means that all line beginning by a `#` will be a comment;
- the two characters `F=` mean that this is really a format specification;
- `N` means the first string of each line is the name of the point;
- `X Y Z` means that this strings number 2, 3 and 4 are the coordinates;
- `Ix Iy Iz` means that this strings number 5, 6 and 7 are the accuracy;

Here is another example with the `app` format used in some IGN's process :

```

#F= N S X Y Z
300      3        94.208685      658.506787      42.39556
301      3        95.323427      656.409116      43.502239
302      3        97.008135      654.424482      45.084237
...

```

In this case the `S` means that there is a string that won't be interpreted. It can also be seen that the accuracy is not mandatory. Once the file has been modified, the following command can be used:

```
GCPConvert AppInFile CP3D_Format.txt
```

For the first arg, the key word `AppInFile` means that the format specification is given at the first line of the file. If you don't want to modify the file it is possible to give the format specification directly on the command line. If the first argument is not a known keyword, then it will try to interpret it as a format specification. The syntax is a bit ugly because it is not possible to give white space in the shell, so they must be replaced by `_`. Here is an example:

```
GCPConvert "#F=N_X_Y_Z_Ix_Iy_Iz" CP3D.txt
```

As usual, to have the full syntax:

```
GCPConvert -help
Valid Types for enum value:
  AppEgels
  AppGeoCub
  AppInFile
  AppXML
*****
* Help for Elise Arg main *
*****
Unnamed args :
  * string :: {Format specification}
  * string :: {GCP File}
Named args :
  * [Name=Out] string :: {Xml Out File}
  * [Name=ChSys] string :: {Change coordinate file}
```

For the format arg, its values can be:

- `AppEgels`, the format is `#F= N S X Y Z;`
- `AppGeoCub`, the format is `#F= N X Y Z;`
- `AppInFile`, format is in the file, as seen above;
- `AppXML`, format is already `MicMac's Xml` format (do nothing);
- any other value, it is a format specification, as seen above;

The meaning of the other arguments is:

- first arg, contains the format specification, as seen above;
- second arg, is the name of the file containing the GCP data;
- third optional arg `Out`, is the name of the xml output file;
- optional arg `ChSys`, is the specification for coordinate system transform;

The optional arg `ChSys` can describe a file for coordinate change as described in 14.5

### 13.3.2 Orientations conversion for PMVS: Apero2PMVS

Thanks to Luc Girod, we have got a tool which converts orientations generated with Apero (or Tapas) into PMVS orientations and builds the whole repository structure for PMVS. It also corrects images from distortion, as PMVS needs undistorted images as input. Here is an example for calling the `Apero2PMVS` command:

```
mm3d Apero2PMVS "myDirectory\IMG_[0-9]{4}.tif" RadialExtended
```

To get the general syntax, one can type:

```
mm3d Apero2PMVS -help
*****
* Help for Elise Arg main *
*****
Unnamed args :
  * string :: {Images' name pattern}
  * string :: {Orientation name}
```

The two mandatory arguments are:

- first arg, the pattern for images' name for which orientation has to be converted;
- second arg, the orientation name (where orientation `xml` files are stored, for example `Ori-RadialExtended`);

NB: it is quite obvious that orientations stored in the `Ori-RadialExtended` folder should match the images' name pattern.

`Apero2PMVS` builds the needed folders for PMVS in a folder with `pmvs-` as suffix, followed by the orientation name (for example `pmvs-RadialExtended`):

- the `visualize` folder contains images corrected from distortion, computed with the `DRUNK` tool, and renamed with the PMVS convention,
- the `txt` folder contains orientation files,
- the `models` folder is empty, it is the folder where PMVS will store output models.

### 13.3.2.1 Distortion removing with DRUNK

The `DRUNK` tool, also written by Luc Girod, allows to remove distortion from images, knowing internal orientations. This tool is called by `Apero2PMVS` to prepare data in the PMVS way, but one can call it alone. The `DRUNK` command can be called the same way as `Apero2PMVS`:

```
mm3d Drunk "myDirectory\IMG_[0-9]{4}.tif" RadialExtended
```

To get the general syntax, as always:

```
mm3d Drunk -help
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {Images Pattern}
 * string :: {Orientation name}
Named args :
 * [Name=Out] string :: {Output folder (end with /) and/or prefix (end with another char)}
 * [Name=Talk] bool :: {Turn on-off commentaries}
```

The two mandatory arguments are:

- first arg, the pattern for images' name for which distortion has to be removed;
- second arg, the orientation name (where orientation `xml` files are stored, for example `Ori-RadialExtended`)

The two optional arguments are:

- first arg, is understood as the output folder name, if it ends with `\`, or as a prefix which will be followed by the orientation name, if it ends with another char;
- second arg, a boolean to allow or avoid commentaries

### 13.3.3 Apero2NVM

Orientations conversion for Computer Vision Based software : `Apero2NVM`

`Apero2NVM` is a tool which takes in input the orientations generated with `Apero` accompanied with its dataset. The Output is a file `.nvm` which is directly usable in input of the dense-matching part of four photogrammetric tool : VisualSFM, MVE, SURE, MeshRecon. This file `.nvm` includes mainly a bloc of orientation data and a sparse cloud. The sparse cloud understands not only its 3 D coordinate but also its feature measurements on the picture, which is also associated with an image index. Are also exported the dataset removed from the distortion and the shift among the image centre and the principal point.

Here is an example for calling the `Apero2NVM` command: `mm3d Apero2NVM "..*.jpg" RadialStd ExpTxt=1 ExpApeCloud=1`

In order to display the help type :

```
mm3d Apero2NVM -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Images Pattern}
```

```
* string :: {Orientation name}
Named args :
* [Name=Nom] string :: {NVM file name}
* [Name=Out] string :: {Output folder (end with /)}
* [Name=ExpTxt] bool :: {Point in txt format ? (Def=false)}
* [Name=ExpApeCloud] bool :: {Exporte Ply? (Def=false)}
* [Name=ExpTiePt] bool :: {Export list of Tie Points uncorrected of the distortion ?(Def=false)}
* [Name=KpImCen] bool :: {Dont add a little rotation for pass from Image Centre to PP ?(To be right fix LibPP=0)}
```

The two mandatory arguments are:

- first arg, the pattern for images name for which orientation has to be converted;
  - second arg, the orientation name (where orientation xml files are stored, for example Ori-RadialStd).
- Six optional argument can be used which allow the user to get more results in the desired folder.
- first argument, name of the .nvm file of Output, by default is Ori.nvm;
  - second argument is the name of the folder of Output, a char is expected with a / to the end, by default it's data/ ;
  - third argument regard the extention of the list of tie point measurement located in every Homol/PastisImageName.. folder, this extension has to be chosen in the Tapioca step. Here it's a simple boolean, 0 for .dat file and 1 for .txt.
  - fourth arguments let the user obtain the sparse cloud in .ply, for the colour a medium gey is settle. In comparison the classic AperiCloud, the camera are not represented but all the redundancy of the tie points has been removed;
  - fith argument is for getting the list of tie point measurement and associated image index in the original image coordinate system;
  - sixth argument allows the user to get the result of the conversion keeping in consideration only the distorsion, the shift between image centre and principal point is simply ignored, in this case the final image are only undistorted.

In the file .nvm the coordinate of the feature are substracted from the image centre, based on the final corrected dataset. By default the list based on the undistorted image system is exported. Nota bene that the user can obtain undistorted images directly using the DRUNK module. Is also exported a list of straight line (3 coordinates for the optical center and 3 coordinates for the director vector) of every feature and the list of its 3D coordinate.

### 13.3.4 Embedded GPS Conversion: OriConvert

The tool **OriConvert** is a versatile command used to:

- transform embedded GPS data from text format to **MicMac**'s **Xml** orientation format.
- transform the GPS coordinate system, potentially into an euclidean coordinate system.
- generate image pattern for selecting a sample of the image block.
- compute relative speed of each camera in order to determine and correct GPS systematic error (delay).
- importing external orientation from others software: **to come**.

#### 13.3.4.1 File format conversion with OriConvert

GPS and attitude extracted from telemetry logs are generally structured as followed:

```
image latitude longitude altitude yaw pitch roll
R0040438.JPG 50.5860992029 4.7957755452 375.046 319.9 8.2 -2.1
R0040439.JPG 50.5864719060 4.7953921650 376.604 319.4 10.1 3.6
...
```

In this example (from the UAS Grand-Leez data set, file **GPS\_WPK\_Grand-Leez.csv**), column titles are specified on the first line. Nevertheless, **MicMac** has its own convention regarding column title. We have to add columns specs as explained in previous section (13.3.1) but with the symbols *K*, *W*, *P* standing for kappa, omega and phi.

```
#F=N Y X Z K W P
#
#image latitude longitude altitude yaw pitch roll
R0040438.JPG 50.5860992029 4.7957755452 375.046 319.9 8.2 -2.1
R0040439.JPG 50.5864719060 4.7953921650 376.604 319.4 10.1 3.6
...
```

Once the file has been modified, the following command can be used:

```
mm3d OriConvert OriTxtInFile GPS_WPK_Grand-Leez.csv Nav-Brut NameCple=FileImagesNeighbour.xml
```

Which will produce an orientation database named **Nav-Brut** (for raw-navigation) containing the database of the center, this is the position of each camera during the shooting. For the first arg, **OriTxtInFile** means that the format specification is given at the first line of the file. If the optional argument **NameCple** is used, image pairs file will be generated and stored in a xml file (here **FileImagesNeighbour.xml**). One may require to transform the orientation into a euclidean coordinate system, which is achieved by using the optional argument **ChSys** with the appropriate file **SysCoRTL.xml** that specified the locally tangent repair (as presented in section 14.5):

```
mm3d OriConvert OriTxtInFile GPS_WPK_Grand-Leez.csv Nav-Brut-RTL ChSys=DegreeWGS84@SysCoRTL.xml
```

As usual, to display the succinct help, type **mm3d OriConvert -help**:

```
mm3d OriConvert -help
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string :: {Format specification}
 * string :: {Orientation File}
 * string :: {Targeted orientation}
Named args :
 * [Name=ChSys] string :: {Change coordinate file}
 ...
 * [Name=ImC] string :: {Image "Center" for computing AltiSol}
 * [Name=NbImC] INT :: {Number of neigboor around Image "Center" (Def=50)}
 ...
 * [Name=CalcV] bool :: {Calcul speed (def = false)}
 * [Name=Delay] REAL :: {Delay to take into account after speed estimate}
 ...
 * [Name=NameCple] string :: {Name of XML file to save couples}
 ...
 * [Name=MTD1] bool :: {Compute Metadata only for first image (tuning)}
 ...
```

Here is the meaning of some of the optional arguments:

- **ChSys**, enable to change (and define) the coordinate system;
- **ImC**, is the name of the image which will be considered as the central image of the sub-block;
- **NbImC**, is the number of neighbour images of **ImC** that will be selected and gathered in a image pattern **PATC**, referring to a sub-block potentially used for determining the delay;
- **CalcV**, based on the camera trajectory analysis, enable the computation of the relative speed of the platform during the shooting;
- **Delay**, once the eventual delay is determined, its will be used through this argument for correcting GPS data;
- **NameCple**, is the name of the file containing the images pairs that is used for the computation of tie points;
- **MTD1**, indicate if image metadata has to be extracted only from one image's exifs (appropriate if only one sensor and one focal length).

#### 13.3.4.2 Taking into account a GPS delay with **OriConvert**

GPS data of small platform as mini-UAS may possibly be marred by a systematic error which follow flight trajectory, due among other to the lap of time between GPS position recording by the autopilot system (at the exact time of camera triggering) and the image shot (a few time after camera triggering). Not considering this delay may of course impact the accuracy of the georeferencing. In particular, the scale of the model will be underestimated. Correction for this delay may be performed by means of the joint use of **OriConvert** and **CenterBascule**. The delay is determined with **CenterBascule** by a modified bascule tool that solves the delay in addition to the 7 parameters of the global transformation. An orientation is so required and may be obtained with these commands:

```
mm3d OriConvert OriTxtInFile GPS_WPK_Grand-Leez.csv Nav-Brut-RTL MTD1=1\
ChSys=DegreeWGS84@SysCoRTL.xml NameCple=FileImagesNeighbour.xml CalcV=1
```

As delay determination use trajectory information, the argument `CalcV` is set to 1, which means that relative speed of the camera will be computed. The relative time is defined a the time require for the camera to move from one pose to the next pose. Image pairs file is subsequently used in the classic pipeline:

```
Tapioca File "FileImagesNeighbour.xml" -1
Tapas RadialBasic "R.*.JPG" Out=All-Rel
```

Then, delay is determined with `CenterBascule` using the options `CalcV`:

```
CenterBascule "R.*.JPG" All-Rel Nav-Brut-RTL tmp CalcV=1
```

The resulting orientation is not interesting, so it is named `tmp` and is subsequently send to the bin. The result of `CenterBascule` is located somewhere in terminal messages and normally looks like that:

```
....
END AMD
delay init :: -0.0854304
Basc-Residual R0040439.JPG [4.12465,-8.48416,60.1676]
....
```

The value of the delay is eventually utilized back again in `OriConvert` by means of the optional argument `Delay`:

```
mm3d OriConvert OriTxtInFile GPS_WPK_Grand-Leez.csv Nav-adjusted-RTL \
ChSys=DegreeWGS84@SysCoRTL.xml MTD1=1 Delay=-0.0854304
```

It is important to notice that the orientation `Nav-adjusted-RTL` is different than `Nav-Brut`, hopefully, and this can be visualized by using the commands `grep Centre Ori-Nav-Brut-RTL/*` and `grep Centre Ori-Nav-adjusted-RTL/*`. The orientation `Nav-adjusted-RTL` is subsequently used for georeferencing a project by the means of `CenterBascule`:

```
CenterBascule "R.*.JPG" All-Rel Nav-adjusted-RTL All-RTL
```

In this example, the image pairs file used in `Tapioca` has been generated on the basis of the uncorrected embedded GPS data. In some cases, the delay may be very important, due either to inappropriate GPS position extraction from telemetry logs, high platform speed (strong wind) or very small base (high overlap combined with low altitude), and the images pairs determination may be strongly affected by this delay. In such specific cases, considering that there may have too many images for generating tie points with `Tapioca MulScale`, one could compute the delay on a sub-block of images. Images pattern of a sub-block may be generated with `OriConvert`.

### 13.3.4.3 Selection of a image sub-block with `OriConvert`

When dealing with numerous images, it is often appropriate to select a sample of the data set, e.g. for camera calibration. The options `ImC` and `NumC` enable the generation of a image pattern corresponding to a sample of `NumC` images centered on the central image `ImC`.

```
mm3d OriConvert OriTxtInFile GPS_WPK_Grand-Leez.csv Nav-Brut-RTL ImC=R0040536.JPG NbImC=25 \
ChSys=DegreeWGS84@SysCoRTL.xml
```

In the terminal, just before the end of the processing, a message containing the sub-block image pattern will be delivered:

```
....
PATC = R0040536.JPG|R0040537.JPG|R0040535.JPG|R0040578.JPG|R0040498.JPG|R0040499.JPG|
R0040579.JPG|R0040538.JPG|R0040577.JPG|R0040534.JPG|R0040497.JPG|R0040500.JPG|R0040580.JPG|
|R0040456.JPG|R0040616.JPG|R0040576.JPG|R0040496.JPG|R0040617.JPG|R0040455.JPG|R0040457.JPG|
|R0040615.JPG|R0040539.JPG|R0040501.JPG|R0040581.JPG|R0040533.JPG
....
```

This pattern may be utilized advantageously with `Tapiocas` and `Tapas` for example. Attention must be paid when using `OriConvert` for the selection of a sub-block that coordinate system is an euclidean coordinate system, as well as for the image pairs file generation.

### 13.3.5 Extracting Gps from exif

Often the Gps information is not in separate files but directly embeded in the exif metadat. The tools `XifGps2Xml` and `XifGps2Txt` allow to do extract this information and convert it to texte or xml file.

#### 13.3.5.1 Extracting Gps from exif with `XifGps2Xml`

For example, with `mm3d XifGps2Xml .*jpg Test` :

- for each image, containing gps data in exif, a file is created containing the gps information in xml micmac format;
- for example for `Image100.jpg`, `Ori-Test/Orientation-Image100.jpg.xml` is created; in xml micmac format;
- the coordinate system is a local tangent system, with origin at centre of images;
- the file `RTLFromExif.xml` contains the definition of this system in MicMac format;

```
mm3d XifGps2Xml
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Full Name}
 * string :: {Orientation}
Named args :
 * [Name=DoRTL] bool :: {Do Local Tangent RTL (def=true)}
 * [Name=RTL] string :: {Name RTL}
 * [Name=SysCo] string :: {System of coordinates, by default RTL created (RTLFromExif.xml)}
 * [Name=DefZ] REAL
```

Some options :

- `DefZ` will allow to specify the altitude value , not implemanted for now;
- `SysCo` allow to change the coordinate system;

#### 13.3.5.2 Extracting Gps from exif with `XifGps2Txt`

For example, with `mm3d XifGps2Txt .*jpg Test` file `GpsCoordinatesFromExif.txt` is created in standard txt format :

```
mm> cat GpsCoordinatesFromExif.txt
2016-04-02_12-22-07.jpg 1.908783 47.902767 161.000000
2016-04-02_12-22-18.jpg 1.908758 47.902861 161.000000
2016-04-02_12-22-29.jpg 1.908717 47.902964 159.000000
2016-04-02_12-22-56.jpg 1.908556 47.902828 154.000000
2016-04-02_12-23-07.jpg 1.908506 47.902789 157.000000
2016-04-02_12-23-12.jpg 1.908511 47.902722 157.000000
...
```

The default export coordinate is `WGS84_deg` :

```
mm3d XifGps2Txt
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Full Name}
Named args :
 * [Name=OutTxtFile] string :: {Def file created : 'GpsCoordinatesFromExif.txt' }
 * [Name=Sys] string :: {System to express output coordinates : WGS84_deg/WGS84_rad/GeoC ; Def=WGS84_deg}
 * [Name=DefZ] REAL
```

### 13.3.6 Exporting external orientation to Omega-Phi-Kapa

The tool `OriExport` can convert MicMac external orientation to the *de facto* standard codification using omega-phi-kapa.

```
mm3d OriExport
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Full Directory (Dir+Pattern)}
 * string :: {Results}
Named args :
 * [Name=AddF] bool :: {Add format as first line of header, def= false}
 * [Name=ModeExp] string :: {Mode export, def=WPK (Omega Phi Kapa)}
```

For now it's quite basic and all the options are not implemented. However, it should solve the majority of problem relative to exporting results in classical photogrammetric softwares.

An example with Cuxha data set :

```
mm3d OriExport Ori-All-Rel/Orientation-Abbey-IMG_034.*.jpg.xml res.txt
```

Will generate the file `res.txt` containinig :

```
Abbey-IMG_0340.jpg -4.304443 11.785803 136.229854 -5.491274 2.702560 -0.004106
Abbey-IMG_0341.jpg -3.775959 11.249636 137.040260 -6.109496 2.042527 0.097497
Abbey-IMG_0342.jpg -3.849398 11.231276 137.533559 -6.707432 1.351133 0.224315
Abbey-IMG_0343.jpg -3.921196 11.302498 137.899618 -7.334180 0.668316 0.362218
```

Matrix R gives rotation terms to compute parameters in matrix encoding with respect to Omega-Phi-Kappa angles given by the tool `OriExport`:

$$R = \begin{bmatrix} \cos(\phi) * \cos(\kappa) & \cos(\phi) * \sin(\kappa) & -\sin(\phi) \\ \cos(\omega) * \sin(\kappa) + \sin(\omega) * \sin(\phi) * \cos(\kappa) & -\cos(\omega) * \cos(\kappa) + \sin(\omega) * \sin(\phi) * \sin(\kappa) & \sin(\omega) * \cos(\phi) \\ \sin(\omega) * \sin(\kappa) - \cos(\omega) * \sin(\phi) * \cos(\kappa) & -\sin(\omega) * \cos(\kappa) - \cos(\omega) * \sin(\phi) * \sin(\kappa) & -\cos(\omega) * \cos(\phi) \end{bmatrix}$$

For example `OriExport` will give in degree:

- $\omega = -5.819826$
- $\phi = -7.058795$
- $\kappa = -12.262634$

The corresponding matrix encoding using  $R$  is:

```
<ParamRotation>
  <CodageMatr>
    <L1>0.96977798578237427 -0.210783330505758815 0.122887790140630643</L1>
    <L2>-0.199121821850641506 -0.974794184828703614 -0.100631989382226852</L2>
    <L3>0.141001849092942777 0.0731210284736428379 -0.987305319416100224</L3>
  </CodageMatr>
</ParamRotation>
```

# Chapter 14

## Geo Localisation formats

Chapter in telegraphic style.

The directory `FileSamples/`, at the same place that this documentation (`micmac/Documentation/DocMicMac/`), contains sample files that will illustrate the file description.

### 14.1 Overview of conic orientation specification

#### 14.1.1 Generalities

These sections describe how orientations are coded into XML-file. It is limited to conic images as created by Apero and used by MicMac. MicMac can read other format, especially for satellite images, these format are external format and description are to be found in external documentations.

To transform a ground point  $P_g$  to an image point  $I_k^m$  of image k we use the following formula :

$$I_k^m = \Pi_k(P_g) = \mathbb{J}(\mathcal{I}(\pi(R_k(P_g - O_k)))) \quad (14.1)$$

$$\pi(x, y, z) = \frac{(x, y)}{z} \quad (14.2)$$

$$P_c = R_k(P_g - O_k) \quad (14.3)$$

With :

- $\mathbb{J}$  being the internal orientation, *not to be confound with internal calibration  $\mathcal{I}$* , it can be used when dealing with analog image for modelizing the transformation between the scanner and the paper, or when using several cropped or scaled images acquired by the same camera ; most user will ignore it because they use digital camera with all images at the same resolution;
- $\mathcal{I}$  being the intrinsic calibration, it modelize the mapping between sphere of directions and plane of the sensor, it depends of the camera lenses and, classically is parametrized by focal length, principal point and distortion;
- $\pi$  being the canonic projection transforming a point, in the camera coordinates, to ray direction in the case of conic projection;
- $R_k, O_k$  being the external orientation of the camera having taken image  $k$ , it contains a projection center  $O_k$  and an orientation matrix  $R_k$ .

The file `FileSamples/Orientation-00.xml` contains an example of orientation file. The main sections of this file are :

```
<OrientationConique>
    <OrIntImaM2C> ... </OrIntImaM2C>
    <TypeProj>eProjStenope</TypeProj>
    <FileInterne>Calib-00.xml</FileInterne>
    <RelativeNameFI>true</RelativeNameFI>
    <Externe> ... </Externe>
    <Verif> .... </Verif>
    <ConvOri>
        <KnownConv>eConvApero_DistM2C</KnownConv>
    </ConvOri>
</OrientationConique>
```

The meaning of the different sections is:

- `<OrIntImaM2C>` contains the data for the interior orientation  $\mathbb{J}$ ;

- **<TypeProj>** contains an enumerated value, specifying the kind of projection;
- **<FileInterne>** is the name of the file containing the data specifying the intrinsic calibration  $\mathcal{I}$ ; it is also possible to directly embed this intrinsic calibration directly in the orientation file of each image;
- **<Externe>** contains the data for the external orientation  $R_k, O_k$  ;
- **<Verif>** does not contain any usefull data to compute the function  $\Pi_k$ , it contains optional data allowing programs to check that they interpret correctly previous data;
- **<ConvOri>** contains data to specify some of the convention for storing previous data (for example, the unity of angles when they are used for storing rotations);

### 14.1.2 Internal orientation

The internal orientation  $\mathbb{J}$  is used to represent easily the scaling, cropping, rotation of images; it can be used for example when dealing with analog images where the mapping between the "paper" and the scanner (extracted from fiducial mark) has to be represented. An affinity is sufficient for all this operation; let's note the fied of **<OrIntImaM2C>** this way :

```
<OrIntImaM2C>
  <I00>u0 v0</I00>
  <V10>ux vx</V10>
  <V01>uy vy </V01>
</OrIntImaM2C>
```

The definition of  $\mathbb{J}$  is then :

$$\mathbb{J} \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + U * \begin{pmatrix} u_x \\ v_x \end{pmatrix} + V * \begin{pmatrix} u_y \\ v_y \end{pmatrix} \quad (14.4)$$

Note, for example, that if  $\mathbb{J}$  is used to modelize interior orientation of a scanned image, input of  $\mathbb{J}$  is in millimeters and output is in pixel. In most case  $\mathbb{J}$  will be unused with digital camera and it will stay at its default value representing identity :

```
<OrIntImaM2C>
  <I00> 0 0 </I00>
  <V10> 1 0 </V10>
  <V01> 0 1 </V01>
</OrIntImaM2C>
```

When  $\mathbb{J}$  is used, it is partly or totaly redundant with the internal calibration; so it is never compensated in Apero : it has a fixed value that is used as preprocessing to all the images measurement (i.e.  $(U, V)$  is replaced by  $\mathbb{J}^{-1}(U, V)$ ).

### 14.1.3 Kind of projection

The kind of projection, specified by **<TypeProj>**, can take the following enumerated value :

- **eProjStenope**, specifying a stenope camera with projection function defined by formula 14.2;
- **eProjOrthographique**, specifying an orthographic camera with projection function defined formula 14.5.

$$\pi(x, y, z) = (x, y) \quad (14.5)$$

In current version, only **eProjStenope** is understood by MicMac and Apero. The Orthographic camera are used to export, in a unified way, the result of matching when MicMac is used in ground geometry.

### 14.1.4 External orientation

The structure of external orientation is the following :

```
<Externe>
  <AltiSol>Z </AltiSol>
  <Profondeur>Depth </Profondeur>
  <KnownConv>eConvApero_DistM2C</KnownConv>
  <Centre> Cx Cy Cz</Centre>
  <ParamRotation>
    <CodageMatr>
      <L1> A B C </L1>
```

```
        <L2> D E F </L2>
        <L3> G H I </L3>
    </CodageMatr>
</ParamRotation>
</Externe>
```

From physical point of view,  ${}^t(C_x C_y C_z)$  can be interpreted as the center of projection and, for example,  ${}^t(CFI)$  can be interpreted as the direction of optical axis. From a more formal point of view :

$$P_g = M * P_c + C = \begin{pmatrix} A & B & C \\ D & E & F \\ G & H & I \end{pmatrix} P_c + \begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix} \quad (14.6)$$

Or, equivalently,  $M$  being orthogonal ( ${}^t M = M^{-1}$ ) :

$$P_c = \begin{pmatrix} A & D & G \\ B & E & H \\ C & F & I \end{pmatrix} (P_g - \begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix}) \quad (14.7)$$

When they exist, the value `<AltSol>` and `<Profondeur>` represent a rough estimation of the 3d structure of the scene. They are generated by Apero, using tie points, and used by MicMac to determine automatically the default central value of the computed depth map :

- if it exists, `<AltSol>` must contain the average of  $z$ ; it will be used by MicMac in ground geometry;
  - if it exists, `<Profondeur>` must contain the average of depth of field, it will be used by MicMac in image geometry;

### 14.1.5 Intrinsic Calibration

The intrinsic calibration  $\mathcal{I}$  is given by the formula :

$$\mathcal{I} \begin{pmatrix} U \\ V \end{pmatrix} = D \left( \begin{pmatrix} PP_x \\ PP_y \end{pmatrix} + F * \begin{pmatrix} U \\ V \end{pmatrix} \right) \quad (14.8)$$

Where  ${}^t(PP_x, PP_y)$  is the principal point,  $F$  is the focal and  $D$  is the distortion function. There exists in Apero, many options for distortion, that are described in next sections.

The XML-structure for storing the intrinsic calibration is, in the simple case :

```
<CalibrationInternConique>
    <KnownConv>eConvApero_DistM2C</KnownConv>
    <PP> PPx PPy </PP>
    <F> Focal</F>
    <SzIm> SzImx SzImy </SzIm>
    <CalibDistortion>
        ....
    </CalibDistortion>
</CalibrationInternConique>
```

The file `FilesSamples/Calib-00.xml` contains a very basic example of intrinsic calibration (for a radial model).

### 14.1.6 The Verif Section

The <Verif> section can be used to check that the coherence between the programs having generated the orientation and the program using it. The structure is :

```
<Verif>
    <Tol>0.00100000000000000002</Tol>
    <ShowMes>true</ShowMes>
    <Appuis>
        <Num>0</Num>
        <Im> U0 V0</Im>
        <Ter>X0 Y0 Z0 </Ter>
    </Appuis>
    <Appuis>
        <Num>1</Num>
```

```

<Im> U1 V1</Im>
<Ter>X1 Y1 Z1 </Ter>
</Appuis>
...
</Verif>

```

The **<Appuis>** should verify the equation :

$$\forall i \quad \Pi_k {}^t(X_i, Y_i, Z_i) = {}^t(U_i, V_i) \quad (14.9)$$

By default **Apero** generates 10 checking point **Appuis**, note that the ground point **<Ter>** are randomly generated points that are absolutely not related to the real 3d structure of the scene. In all **EISE**'s program importing orientation and containing **<Verif>** structure, it is checked that equation 14.9 is satisfied with a tolerance given by **<To1>**. When it is not the case, an error occurs.

For a developer importing orientation from **Apero** in its programs, it would be a good idea to check the equation 14.9 on existing **<Appuis>** to ensure that all the data have been correctly interpreted. Similarly, for a programmer, exporting orientation to **MicMac** or other programs, it would be a good idea to add **<Appuis>** datas to ensure that his data are correctly interpreted by **MicMac**.

## 14.2 Distortion specification

### 14.2.1 Generalities

#### 14.2.1.1 Composition of distortions

The distortion used in **Apero-MicMac**, is the composition of several elementary distortions. An elementary distortion belongs to a predefined parametric model: radial, decentric, polynomial, fish-eye, brown ... We write:

$$D = d_1 \circ d_2 \cdots \circ d_N \quad (14.10)$$

In majority of configuration, there will be only one elementary distortion :  $N = 1$ . A possible use of composition of basic distortion is:

- use a physical model, with few parameters, to modelize the principal part of distortion (for example, a radial model);
- use a polynomial model, with more parameters, to modelize the remaining systematism; having modelized the main part of distortion allow to restrain the degree of polynomial distortion.

The XML-Structure encoding the distortion is

```

<CalibrationInternConique>
  ...
  <SzIm> SzImx SzImy </SzIm>
  <CalibDistortion> <!-- d1 --> </CalibDistortion>
  <CalibDistortion> <!-- d2 --> </CalibDistortion>
  ....
  <CalibDistortion> <!-- dN --> </CalibDistortion>
</CalibrationInternConique>

```

The file **FileSamples/Calib-1.xml** gives an example of distortion coded by composition of two basic distortions: a radial and a polynomial. Note that the last distortion stored in the file is the first applied when computing  $\Pi_k$  in the direction ground  $\rightarrow$  camera.

Like with internal orientation, the composition of several distortion would be highly redundant if they were all optimized simultaneously. In **Apero**, when there are several distortions, the  $N - 1$  first one are fixed and they are used as preprocessing to all the images measurement :

- $(U, V)$  is replaced by  $(d_1 \circ d_2 \circ \dots \circ d_{N-1})(U, V)$ .

#### 14.2.1.2 Structure of basic distortion

The XML-structure for representing a basic distortion is an "union" of the possible different types:

- **<ModRad>**, a XML-structure specialized for the radial distortion;
- **<ModPhgrStd>**, a XML-structure implementing the fraser model [Fraser C. 97];
- **<ModGrid>**, a XML-structure for describing distortion as grids, these are dense grid, conceived for a quick computation once the distortion is known; they cannot be used in **Apero** (a finite-element like model of grid, usable in compensation, used to be implemented, and will probably be offered again);

- `<ModUnif>`, a XML-structure for describing many different analytic models, the difference with `<ModRad>` and `<ModPhgrStd>` is that the XML representation is independant of the model, the semantic being implicit and entirely coded by a type-tag; this makes the work easier for the implementer (myself ...) and allows to offer to users much more models; the drawback is the obscurity for other developers aiming at decoding the XML-representation ...
- `<ModNoDist>` for representing no distortion; this is the simplest mode, but paradoxically, not fully supported now; so contact me if you need it.

The XML formal description of this structure can be found in `ParamChantierPhotogram.xml` :

```
<CalibDistortion Nb="+" UnionType="true" Container="std::vector" ToReference="true">
    <ModNoDist Nb="?">
        <Inutile Nb="?" Type="std::string"> </Inutile>
    </ModNoDist>
    <ModRad Nb="?" RefType="CalibrationInterneRadiale"> </ModRad>
    <ModPhgrStd Nb="?" RefType="CalibrationInternePghrStd"> </ModPhgrStd>
    <ModUnif Nb="?" RefType="CalibrationInterneUnif"> </ModUnif>
    <ModGrid Nb="?" RefType="CalibrationInterneGrid"> </ModGrid>
</CalibDistortion>
```

This specifies that each occurence of `<CalibDistortion>`, must have exactly one "son", which can be or `<ModNoDist>`, or `<ModRad>`, or ...

### 14.2.2 Radial Model

The specification of a radial distorsion is :

```
<CalibrationInterneRadiale Nb="1" Class="true" ToReference="true">
    <CDist Nb="1" Type="Pt2dr"> </CDist>
    <CoeffDist Nb="*" Container="std::vector" Type="double"> </CoeffDist>
    <PPaEqPPs Nb="?" Type="bool" Def="false"> </PPaEqPPs>
</CalibrationInterneRadiale>
```

If for example, the following XML structure is specified :

```
<CalibrationInterneRadiale>
    <CDist> Cx Cy </CDist>
    <CoeffDist> R3 </CoeffDist>
    <CoeffDist> R5 </CoeffDist>
    <CoeffDist> R7 </CoeffDist>
</CalibrationInterneRadiale>
```

It corresponds to :

$$D^R \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} C_x \\ C_y \end{pmatrix} + (1 + R_3\rho^2 + R_5\rho^4 + R_7\rho^6) \begin{pmatrix} d_u \\ d_v \end{pmatrix} \quad (14.11)$$

$$d_u = U - C_x \quad d_v = V - C_y \quad \rho^2 = d_u^2 + d_v^2 \quad (14.12)$$

If the boolean optional value `PPaEqPPs` is set to `true`, then the center of distorsion will be constrained to be equal to principal point in all the bundle adjustment.

### 14.2.3 Photogrammetric Standard Model

The specification of a photogrammetric standard model distorsion is :

```
<CalibrationInternePghrStd Nb="1" Class="true" ToReference="true">
    <RadialePart Nb="1" RefType="CalibrationInterneRadiale"> </RadialePart>
    <P1 Nb="?" Type="double" Def="0.0"> </P1>
    <P2 Nb="?" Type="double" Def="0.0"> </P2>
    <b1 Nb="?" Type="double" Def="0.0"> </b1>
    <b2 Nb="?" Type="double" Def="0.0"> </b2>
</CalibrationInternePghrStd>
```

It contains a radial distortion `<RadialePart>`, an affine part  $b_1, b_2$  and a decentric part  $P_1, P_2$  (see A.2.2.3 for a justification of analytical form) :

$$D^P \begin{pmatrix} U \\ V \end{pmatrix} = D^R \begin{pmatrix} U \\ V \end{pmatrix} + P_1 * \begin{pmatrix} 2d_u^2 + \rho^2 \\ 2d_u d_v \end{pmatrix} + P_2 * \begin{pmatrix} 2d_u d_v \\ 2d_v^2 + \rho^2 \end{pmatrix} + b_1 * \begin{pmatrix} d_u \\ 0 \end{pmatrix} + b_2 * \begin{pmatrix} d_v \\ 0 \end{pmatrix} \quad (14.13)$$

Note that there are only 2 affine coefficients; that is because with the focal and the pure rotation (intrinsic calibration being determined up to a 3D rotation, see A.2.3 ) these 2 coefficients are sufficient to have a base of affine function.

#### 14.2.4 Grids model

Later ...

### 14.3 Unified distorsion models

#### 14.3.1 Generalities

The specification of unified calibration model, is given in `ParamChantierPhotogram.xml` :

```
<CalibrationInterneUnif Nb="1" Class="true" ToReference="true">
    <TypeModele Nb="1" Type="eModelesCalibUnif"> </TypeModele>
    <Params Nb="*" Type="double" Container="std::vector"> </Params>
    <Etats Nb="*" Type="double" Container="std::vector"> </Etats>
</CalibrationInterneUnif>
```

It contains an enumerated value, specifying the type of model, and list of real value that contain the parameters of the model; the parameters are interpreted relatively to the type :

- the type is specified by `<TypeModele>`, which must be one the `eModelesCalibUnif`;
- the `<Etats>` and `<Params>` model are both real values;
- there are generally few `<Etats>` values, between 1 and 3; they are not optimized during `Apero` ; they are used as normalisation values so that coordinates are roughly centered on 0,0 and have an amplitude of unity;
- the `<Params>` values can be numerous (up to 66 now); for a given model, a fixed number are required by the programm, so the omitted parameters are given the default value 0; for most models, when all `<Params>` equal 0, distortion is equal to identity;

Many models are subset of polynoms, while the fish-eye models are combinations of a priori model and polynoms.

Type	Max Degree	NbEtat	NbParam	Pure Polynom
<code>eModelePolyDeg2</code>	2	3	6	yes
<code>eModelePolyDeg3</code>	3	3	14	yes
<code>eModelePolyDeg4</code>	4	3	24	yes
<code>eModelePolyDeg5</code>	5	3	36	yes
<code>eModelePolyDeg6</code>	6	3	50	yes
<code>eModelePolyDeg7</code>	7	3	66	yes
<code>eModeleEbner</code>	4	1	12	yes
<code>eModeleDCBrown</code>	5	1	14	yes
<code>eModele_FishEye_10_5_5</code>	10	1	50	No
<code>eModele_EquiSolid_FishEye_10_5_5</code>	10	1	50	No

#### 14.3.2 Unified Polynomial models

A general polynomial distortion, can be specified by setting `<TypeModele>` to a value `<eModelePolyDeg2>, <eModelePolyDeg3> ... <eModelePolyDeg7>`.

There are three states values that are used like normalisation coefficients, let's note :

- $S = Etats[0]$ ;
- $C_x = Etats[1]$ ;
- $C_y = Etats[2]$ ;

Generaly,  $S$  is approximately equal to the focal lentgh, and  $C_x, C_y$  are set to the center of images. Let's note  $N$  the normalisation function :

$$N \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} \frac{U-C_x}{S} \\ \frac{V-C_y}{S} \end{pmatrix} \quad (14.14)$$

The <Params> value defines a polynomial function  $P$  on the normalized coordinate, this means that the final distortion will be :

$$D = N^{-1} \circ P \circ N \quad (14.15)$$

For the degree over 3, it's quite easy, the set of generating polynom contains all the possible monomials. For the degree below 2, it's a bit more complicated because:

- we don't want degree 0 monomials, who would be redundant with principal point;
- for degree 1 monomials we already have the focal length and the pure rotation that define a 2 dimensional family, so we want a 2 dimensional complementary basis;
- for degree 2 monomials we have the "tilt rotation" that defines function which have limited development of degree 2 (see equation 14.16).

$$\begin{pmatrix} \frac{x}{1-x} \\ \frac{y}{1-x} \end{pmatrix} \approx \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} x^2 \\ xy \end{pmatrix} \quad (14.16)$$

So inside the 12-dimensional space of polynoms of degree 2 in  $xy$  we use the 6 first parameters to define a 6 dimensional subset :

$$P_2 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} p_1x + p_2y - 2p_3x^2 + p_4xy + p_5y^2 \\ -p_1y + p_2x + p_3xy - 2p_4y^2 + p_6x^2 \end{pmatrix} \quad (14.17)$$

The interpretation of coefficient after degree 2 is more obvious, for example :

$$P_3 \begin{pmatrix} x \\ y \end{pmatrix} = P_2 \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} p_7x^3 + p_8x^2y + p_9xy^2 + p_{10}y^3 \\ p_{11}x^3 + p_{12}x^2y + p_{13}xy^2 + p_{14}y^3 \end{pmatrix} \quad (14.18)$$

### 14.3.3 Brown's and Ebner's model

This was the first models I implemented with unified model. I am not so sure that they should be very useful now, but they exist and are somewhat considered as reference in a part of photogrammetric community ...

For ebners model, there is one <Etat>, that should be approximately equal to the base in image space, note  $B = Etat[0]$  :

$$B_2 = \frac{2}{3}B^2, \quad U_2 = U^2 - B_2, \quad V_2 = V^2 - B_2 \quad (14.19)$$

$$D_E \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} U \\ V \end{pmatrix} + \begin{pmatrix} p_1U + p_2V - 2p_3U_2 + p_4UV + p_5V_2 + p_7UV_2 + p_9VU_2 + p_{11}U_2V_2 \\ -p_1V + p_2U + p_3UV - 2p_4V_2 + p_6U_2 + p_8VU_2 + p_{10}UV_2 + p_{12}U_2V_2 \end{pmatrix} \quad (14.20)$$

For brown model, there is one <Etat>, that should be approximately equal to focal lenght, note  $F = Etat[0]$

$$\rho^2 = U^2 + V^2 \quad (14.21)$$

$$D_B \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} U \\ V \end{pmatrix} + \begin{pmatrix} p_1U + p_2V + p_3UV + p_4V^2 + p_5U^2V + p_6UV^2 + p_7U^2V^2 + p_{13}\frac{U}{F}U^2V^2 + p_{14}U\rho^2 \\ p_8UV + p_9U^2 + p_{10}U^2V + p_{11}UV^2 + p_{12}U^2V^2 + p_{13}\frac{V}{F}U^2V^2 + p_{14}V\rho^2 \end{pmatrix} \quad (14.22)$$

### 14.3.4 Fish eye models

By far the most complex ...

With a fish-eye, there is an opening of almost 180 degree, so a polynomial would not be suited because the distortion has to map, in the finite sensor plane, points that are almost at infinity.

A fish eye, of focal length  $F$ , is first defined by an approximate physical model that described the radial mapping function  $\phi$  from directions to sensor plane

$$R \begin{pmatrix} \sin\theta \sin\omega \\ \sin\theta \cos\omega \\ \cos\theta \end{pmatrix} \rightarrow F * \phi(\theta) \begin{pmatrix} \sin\omega \\ \cos\omega \end{pmatrix} \quad (14.23)$$

Two models are supported now :

- $\phi(\theta) = \theta$  for equilinear fisheye, by far the most frequent;
- $\phi(\theta) = 2\sin(\frac{\theta}{2})$  for equilisolid fisheye (never met them concretely !);

If  $C_X, C_Y$  is the distortion center, and  $F_0$  the focal length, the approximate distortion model is then :

$$R = \sqrt{(U - C_x)^2 + (V - C_y)^2} \quad (14.24)$$

$$D_A \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} C_x \\ C_y \end{pmatrix} + \frac{F_0}{R} * \phi(\text{atan}(\frac{R}{F_0})) \begin{pmatrix} U - C_x \\ V - C_y \end{pmatrix} \quad (14.25)$$

Of course this theoretical model is only an approximation, and it has to be corrected by parametric model; we have to add polynomial terms, and for evident stability reasons, we prefer that this polynoms operate on finites quantities, so the additional parameters are operating on  $D_A$ . When I implemented these models, I thought fish-eye were always poorly designed, so there are (much too) many additionnal parameters in my fish eye model :

- 10 radial parameters ( $(R^3, R^5, \dots, R^{21})$ ), pratically 5 seems always sufficient;
- 10 radial decentric parameters for the 5 first term of equations A.23 and A.24 of chapter A.2.2.3; pratically 0 or 1 seem sufficient;
- general polynomial up to degree 5, with many "whole" due to existing other parameters; practically degree 1 seems sufficient.

Finally, the XML-implementation is :

- `Estat[1] = F0;`
- `Params[1] = Cx;`
- `Params[2] = Cy;`
- `Params[3] = R3 ... Params[7] = R11`
- `Params[13] = P1 ... Params[14] = P2`
- `Params[23] = l1 ... Params[24] = l2`

$$A = \frac{U - C_x}{F_0}, \quad B = \frac{V - C_y}{F_0}, \quad R = \sqrt{A^2 + B^2} \quad (14.26)$$

$$\lambda = \frac{\phi(\text{atan}(R))}{R}, \quad a = \lambda A, \quad b = \lambda B, \quad \rho = \sqrt{a^2 + b^2} \quad (14.27)$$

$$D_{pol} \begin{pmatrix} a \\ b \end{pmatrix} = (1 + R_3\rho^2 + R_5\rho^4 \dots) \begin{pmatrix} a \\ b \end{pmatrix} + \begin{pmatrix} P_1(\rho^2 + 2a^2) + 2P_2ab \\ 2P_1ab + P_2(\rho^2 + 2b^2) \end{pmatrix} + \begin{pmatrix} l_1a + l_2b \\ l_2a \end{pmatrix} \quad (14.28)$$

And finally the distortion :

$$D \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} C_x \\ C_y \end{pmatrix} + F_0 D_{pol} \begin{pmatrix} a \\ b \end{pmatrix} \quad (14.29)$$

#### 14.3.5 The tag <RayonUtile>

To come ...

### 14.4 The tool TestCam

The tool `bin/TestCam` is a tool that allows people who would like to import orientation from Apero, or to export orientation in MicMac, to check their understanding of the convention described in this chapter.

Run `bin/TestCam NameOrient X Y Z`, the program will load the orientation file and will show the different computation step from the ground point X Y Z to the final image point :

- `-0-CamCoord` : the point in camera corrdinate system;
- `-1-ImSsDist` : the image point before applying distortion;
- `-2-ImDist 1` : previous image point after applying first distortion;
- `-3-ImDist N` : previous image point after applying optional complementary distorsion;
- `-4-ImFinale` : previous image point after applying internal orientation.

An example to chek external orientation :

```
TestCam TestOri-1.xml 10 0 1
---PGround  = [10,0,1]
-0-CamCoord = [0.773873,-0.0328935,-0.632486]
-1-ImSsDist = [276.457,1052.01]
-2-ImDist 1 = [276.457,1052.01]
-3-ImDist N = [276.457,1052.01]
-4-ImFinale = [276.457,1052.01]
```

An example to check polynomial distortion and internal orientation :

```
TestCam TestOri-2.xml 0.1 0 1
---PGround = [0.1,0,1]
-O-CamCoord = [0.1,0,1]
-1-ImSsDist = [1600,1000]
-2-ImDist 1 = [1610,1000]
-3-ImDist N = [1610,1000]
-4-ImFinale = [3220,1000]
```

## 14.5 Coordinate system

### 14.5.1 Generalities

A coordinate system describes a mapping between its coordinates and the geo-centric ( Greenwich origin ??) system, considered as "the" reference.

The abstract C++ class is `cSysCoord`. Interface specification in `include/general/ptxd.h`. Implementation files in `src/util/cSysCoor.cpp`. Interface:

- `Pt3dr ToGeoC(const Pt3dr &)` const
- `Pt3dr FromGeoC(const Pt3dr &)` const
- `Pt3dr DdgEnMetre()` const (*ordre de grandeur* in french), used as a rough estimation of the size, in meter, of each coordinates;

Existing implemented systems, are:

- `GeoC`
- `WGS84`
- `RTL`
- `Polynomial`

### 14.5.2 XML codage

#### 14.5.2.1 Generalities

- specification in file `ParamChantierPhotogram.xml`
- the class `SystemeCoord` contains the data necessary to create a C++ object `cSysCoord`
- a `SystemeCoord` is made of several `BasicSystemeCoord` (one in the simplest case);
- the first `BasicSystemeCoord` defines the coordinate system, the possible following `BasicSystemeCoord` are arguments used to define this system;

A `BasicSystemeCoord` is made from :

- a `TypeCoord` field , of type `eTypeCoord`;
- auxiliary vectors of values : `AuxR` for doubles, `AuxI` for integers, `AuxStr` for strings, `AuxRUnite` for unities ; the number and semantic of these datas is varying according to the `TypeCoord`;
- the optional boolean value `ByFile`, meaning that the system is defined in an exterior file;

The enumerated possible values of a `eTypeCoord` are :

- `eTC_WGS84`;
- `eTC_GeoCentr`
- `eTC_RTL`
- `eTC_Polyn`
- `eTC_Unknown`

Obviously, the set of possible values may grow in the future.

#### 14.5.2.2 Geocentric

A geocentric coordinate system, defined by `eTC_GeoCentr`, requires no argument.

#### 14.5.2.3 eTC\_WGS84

A WGS84 coordinate system, defined by `eTC_WGS84`, requires no argument.

#### 14.5.2.4 Exterior file coordinate system

It is often convenient to define once a coordinate system in a file, and to use it several times. In this case, for the XML-structure :

- **ByFile** must be true ;
- there must exist one **AuxStr** containing the name of the file, this file must contain a **SystemeCoord** ;
- the **TypeCoord** being redundant must, or be equal to **eTC\_Unknown** or be equal to the value specified in the file (for coherence reason, as they are redundant).

#### 14.5.2.5 Locally tangent repair

A locally tangent repair, specified by **eTC\_RTL** must contain :

- three values **AuxR** containing the origin of the repair;
- optional **AuxRUnite** values, specifying the angular unities in which the origin is given;

If the first **BasicSystemeCoord** of a **SystemeCoord** is of type **eTC\_RTL**, it must contain a second **BasicSystemeCoord** indicating the coordinate system in which the origin is given.

#### 14.5.2.6 Polynomial coordinate system

Sometimes it is convenient to use a coordinate system, that is known by a set of example, the analytic formula being unknown. In this case, it can be stored as a polynomial transformation between a known coordinate system and the unknown system.

A polynomial coordinate system, specified by **eTC\_Polyn**, is stored this way in XML format :

- the first **BasicSystemeCoord** stores the polynomial transformation, and the second store the known coordinate system;
- it contains three polynoms  $P_x, P_y, P_z$  for direct mapping and three polynoms for inverts mapping; this polynoms works on "normalized" coordinates, the normalization parameters are stored in **AuxR** after the polynom coefficient;
- the degree of the polynom are specified by **AuxI** (there are 9 **AuxI**)

### 14.6 Tools for processing trajectory and coordinate systems

Essentially tools for transformation of some txt format to "my" XML format.

#### 14.6.1 SysCoordPolyn

To create a polynomial coordinate system from a set of known pair of coordinate between the target system and an existing system. For example :

```
SysCoordPolyn applic/XML-Patton/Mumu/UTM/Tab-Appr_UTM.txt toto.xml [4,4,1] [0,0,1]
```

The file **Tab-Appr\_UTM.txt** contains lines :

```
...
12 -2.424957 -0.381650 43.445598    712886.613    7580475.496    43.446
13 -2.422346 -0.382029 58.718846    728315.975    7577855.955    58.719
14 -2.426408 -0.380253 1176.786013   704407.121    7589451.512    1176.786
...
```

Each line has the structure  $IdXYZABC$ , where  $XYZ$  is a point in a known coordinate  $S$ ,  $ABC$  are the coordinates in the system we want to learn. Let  $X'Y'Z'$  be the coordinates of this point in another known system  $S'$ , this program will compute a polynom such that  $Pol(A, B, C) = (X', Y', Z')$ . It can be interesting to have sometimes  $S \neq S'$ ; for example suppose we know  $XYZ$  in geocentric, and we want to learn a UTM system, it may be cleverer to have  $X'Y'Z'$  in WGS84 because the polynomial fitting will be much easier.

By the way, for now we necessarily have  $S = S' = WGS84$ , but could be changed easily.

Of course, the best way would be that Apero/MicMac knows all the possible coordinate systems. Well, for now I am not sure that I want to be linked to libs like Proj4: may creates dependancies and installation problems. To be discussed ... However for now there is this possibility of transforming any system into a polynomial representing if you can generate a set of learning pair.

### 14.6.2 The TrAJ2 command

A tool for converting some basic trajectography format, and ground point, all in txt, in XML format for MicMac/Apero.

```
make -f TrAJ2_Make
```

Param\_Traj\_AJ in file SuperposImage.xml. Several examples in `applis/XML-Patton/Mumu/`  
Sections :

### 14.6.3 Trajectory preprocessing

#### 14.6.3.1 The tool SplitBande

```
/media/MyPassport/Helico-MAP/Lapalliere/Aprem/
```

```
SplitBande ./ "0.*.NEF" Num0=100 NbDig=3 Exe=1
```

To recover band-structure, from time meta-data. when affordable.

#### 14.6.3.2 The tool BoreSightInit



# Chapter 15

## Advanced Tie Points

### 15.1 Changing default detector in XML\_User/

There exist different implementation of Sift detector and Ann matchor in MicMac. The newest one offer more option, while the oldest have been more tested ...

To change the default behaviour you must edit your `MM-Environment.xml` in the folder `include/XML_User/`. For example :

```
<MMUserEnvironment>
  <TiePDetect> mm3d:Digeo </TiePDetect>
  <TiePMatch > mm3d:Ann </TiePMatch>
</MMUserEnvironment>
```

Note that the `mm3d:Digeo` implementation of SIFT detector offers several advantages :

1. faster, specially in the gaussian computation;
2. you can use the `NoMax` and `NoMin` options in `Tapioca`, to supress the Min (or Max) in sift detection. This divides by 2 the number of tie points, while conserving the same multiple tie point ratio (as at 99.99...% a max is never a good homolog of a min).

The default tie point detector is `mm3d:Sift`.

### 15.2 Filtering tie points in HomolFilterMasq

This command can be used when you have the necessary spatial information to retrieve false tie points.

The command `HomolFilterMasq` can do some filtering on tie points. The masking process can be purerly in image geometry or can be done in some ground geometry.

```
mm3d HomolFilterMasq
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Full name (Dir+Pat)}
Named args :
 * [Name=PostPlan] string :: {Post to plan, Def : toto ->toto_Masq.tif like with SaisieMasq}
 * [Name=GlobalMasq] string :: {Global Masq to add to all image}
 * [Name=KeyCalculMasq] string :: {For tuning masq per image}
 * [Name=KeyEquivNoMasq] string :: {When given if KENM(i1)==KENM(i2), don't masq}
 * [Name=Resol] REAL :: {Sub Resolution for masq storing, Def=10}
 * [Name=ANM] bool :: {Accept no mask, def = true if MasqGlob and false else}
 * [Name=ExpTxt] bool :: {Ascii format for in and out, def=false}
 * [Name=PostIn] string :: {Post for Input dir Hom, Def=}
 * [Name=PostOut] string :: {Post for Output dir Hom, Def=MasqFiltered}
 * [Name=OriMasq3D] string :: {Orientation for Masq 3D}
 * [Name=Masq3D] string :: {File of Masq3D, Def=AperiCloud_${OriMasq3D}.ply}
 * [Name=SelecTer] Pt2dr :: {[Per,Prop] Period of tiling on ground selection, Prop=propotion of selected}
 * [Name=DistId] REAL :: {Supress pair such that d(P1,P2) < DistId, def unused}
```

The main option are :

1. **PostPlan** for example set **PostPlan=titi** if there is masq per image and for each image **Image.tif** the masq if **Image\_Masq.titi.tif**, by default will generate an error if this image does not exist, set **ANM=true** if you know that non existing images are normal;
2. **GlobalMasq** if masq common to all images exist (for example with fiducial marks);
3. **KeyCalculMasq**, sometime you may have many images and a few masq, each masq being applicable for a group of images; use this option with much be a computation key described in **MicMac-LocalChantierDescripteur.xml**
4. **Masq3D**, a file for 3D masq as seized by **SaisieMasqQT**, the orientation **OriMasq3D** must be initialized;
5. **SelecTer**, can be used to decrease the number of tie points while maintaining the proportion of multiplicity; if **SelecTer=[Per,Prop]**, then in each tile of size  $S = Per * Resol$  in the ground coordinate<sup>1</sup> the point are selected in the subtile of size  $S * \sqrt{Prop}$
6. **DistId** supress the pair of point  $P_1, P_2$  such that  $d(P_1, P_2) < DistId$ , for example, this can be useful when the acquisition was made using a turn table to automatically supress the point on the background;

### 15.3 Merging Tie point from multiple view with HomolMergePDVUnik

This command correspond to rather special case, when you have a set of camera that do not move (or form a rigid block) and the scene is moving. For example :

- there is three fixed camera  $A, B, C$ ;
- at time 1, 2, 3, 4 someone is moving in front of the camera and the images  $A_1, A_2, \dots, C_3, C_4$  were acquired ;

It is not possible to make some standard photogrammetric processing on  $A_1, A_2, \dots, C_3, C_4$  as the scene is not static. By the way if we knew the pose  $P_A, P_B, P_C$  of the camera, then all the homologous points  $(A_1, B_1), (A_2, B_2) \dots (A_4, B_4)$  would be compatible with  $P_A$  and  $P_B$ , which means that we can merge this tie points in a unique file that can be used to estimate  $P_A$  and  $P_B$ ; and the same with  $A, C$  and  $B, C$ .

The command **HomolMergePDVUnik** does this merging, in fact you can consider that the tie-points obtained are more or less resulting are some kind of merging from the different scene.

```
mm3d HomolMergePDVUnik
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
* string :: {Full name (Dir+Pat)}
* string :: {Dir of external point}
Named args :
* [Name=PostIn] string :: {Post for Input dir Hom, Def=}
* [Name=PostOut] string :: {Post for Output dir Hom, Def=MasqFusion}
* [Name=ExpTxt] bool :: {Ascii format for in and out, def=false}
* [Name=DirN] vector<std::string> :: {Supplementary dirs 2 merge}
```

### 15.4 Tie points on low contrast images usign SFS in MicMac-LocalChantierDesc

The current implementation of SIFT++ used in MicMac is not fully invariant to scaling/translation in radiometry. This may be a problem in case of acquisitions having a good SNR but with low contrast in the scene; in this case, thanks to good SNR there is potential information to get tie points, but as this information is assimilated to noise, it cannot be extracted.

To overcome this problem, it is possible to require that MicMac computes some contrast enhancement on images before computing SIFT points. Although this method is not optimal (it would be better to modify the SIFT++ Kernel), it has the advantage of existing...

The figure 15.1 presents an image without enhancement, in its original form, and the same image after enhancement. The figure 15.2 presents the detected tie points; we notice that the spatial density of tie points is much higher on enhanced image.

Of course, the enhanced images are fairly artificial, as it can be seen on figure 15.3 that presents a full image before and after enhancement. So if this option is activated, the enhanced images are used only for the tie points steps (which are developed as specific "hidden files" in folder **Tmp-MM-Dir**).

---

1. *Resol* being the average ground resolution

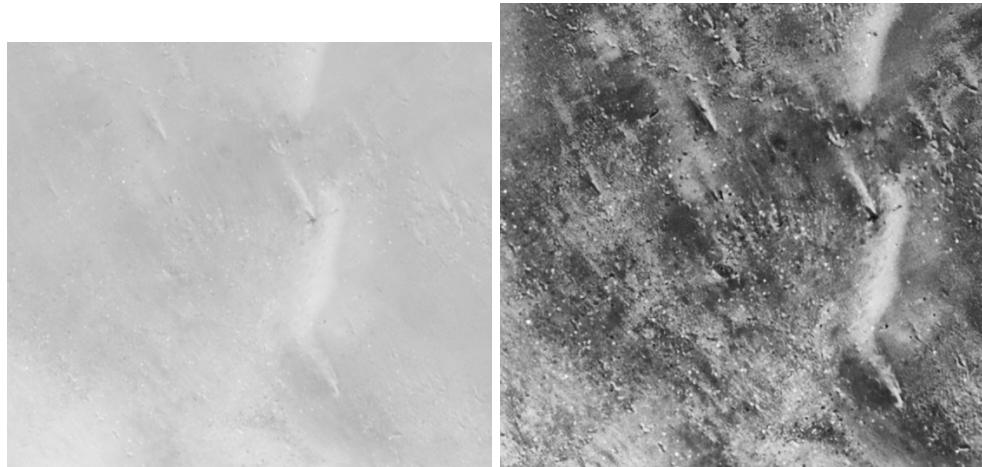


Figure 15.1: Detail of image before and after enhancement

To activate this option, the `NKS-Assoc-SFS` must be changed in the `MicMac-LocalChantierDescripteur.xml`. It must return `SFS` instead of the default value `NONE`. For example:

```
<KeyedNamesAssociations>
  <Calcs>
    <Arrite> 1 1 </Arrite>
    <Direct>
      <PatternTransform> .* </PatternTransform>
      <CalcName> SFS </CalcName>
    </Direct>
  </Calcs>
  <Key> NKS-Assoc-SFS </Key>
</KeyedNamesAssociations>
```

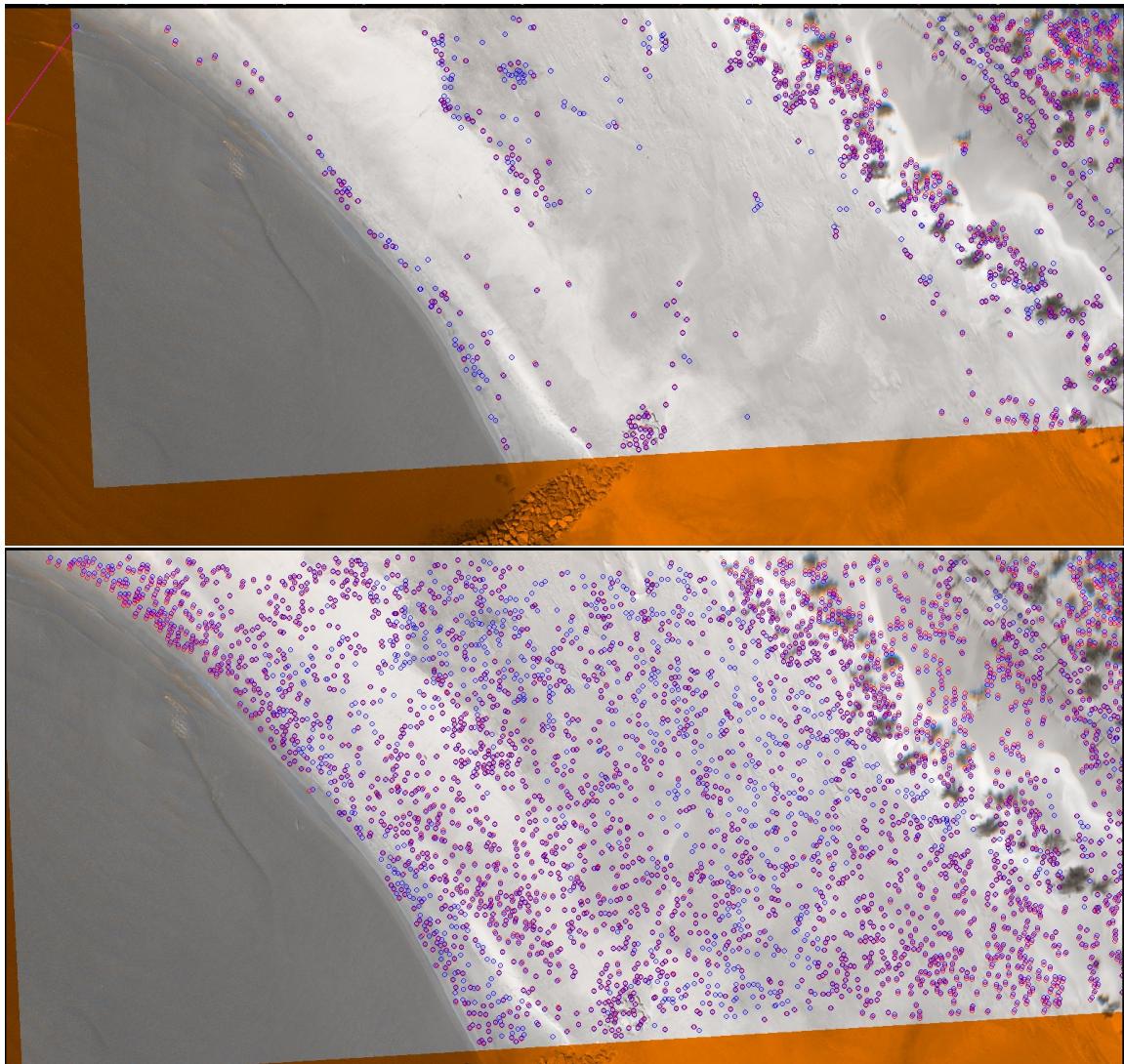


Figure 15.2: Tie points before and after enhancement



Figure 15.3: Global images before and after enhancement

### 15.4.1 Alternative syntax @SFS

It's also possible to use enhanced tie point, without `MicMac-LocalChantierDescripteur.xml`, it suffice to add `@SFS` at the `Tapioca` command.

## 15.5 Tie point reduction in RedTieP

This command can be used to reduce the number of tie points generated by *Tapioca*. Currently it requires to format the *Homol* folder into the Martini format, so before executing *RedTieP* one has to execute the tool *NO\_AllOri2Im* (obviously, after running *Tapioca* to compute the tie-points):

The command `RedTieP` keeps the tie-points that are present in a higher number of images and that guarantee a good distribution in the pixel-space of the images.

```
mm3d RedTieP
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Pattern of images}
Named args :
 * [Name=NumPointsX] INT :: {Target number of tie points between 2 images in x axis of
   image space, def=4}
 * [Name=NumPointsY] INT :: {Target number of tie points between 2 images in y axis of
   image space, def=4}
 * [Name=SubCommandIndex] INT :: {Internal use}
 * [Name=ExpSubCom] bool :: {Export the subcommands instead of executing them, def=false}
 * [Name=ExpTxt] bool :: {Export homol point in Ascii, def=false}
 * [Name=SortByNum] bool :: {Sort images by number of tie points, determining the order
   in which the subcommands are executed, def=0
   (sort by file name)}
 * [Name=Desc] bool :: {Use descending order in the sorting of images, def=0 (ascending)}
```

The main options are :

1. The first option is mandatory and it is the pattern to be used to select the images.
2. `NumPointsX` and `NumPointsY` represent the target number of tie-points between an image pair (specified as `numY`, i.e. `numPointsTarget=numX*numY`).
3. `SortByNum` to select if the sorting by file name or by number of tie-points (default is to sort by file name). This sorting has effect on the order in the processing workflow, the first image in the sorted list is the first one that has its tie-points reduced.
4. `Desc` to indicate if use ascending or descending order when sorting the images to decide (default is ascending).
5. `ExpTxt` is a boolean indicating if reduced tie-point are dumped in binary or ASCII format (default is binary).
6. `ExpSubCom` is used to export the commands to perform the tie-point reduction pipeline without actually executing it. This is used to execute the pipeline with an external parallelization tool (see section below)

### 15.5.1 Algorithm description

- Select the images from the pattern defined by the user.
- Sort the images. By default by file name in ascending order (user can choose to sort by number of tie-points, and/or to use descending order).
- Execute a set of tasks, one for each image. Each task executes various steps:
  - Define a master image, the image driving the tie-point reduction in this task.
  - Find the related images. The related images are images that share tie-points with the master image.
  - Load the tie-points shared between the master image and each of the related images. If a related image was a master image in a earlier executed task the algorithm uses the list of tie-points produced in that task (instead of the original list of tie-points as provided by *Tapioca*).
  - Perform the topological merging of tie-points into multi-tie-points. A multi-tie-point stores in how many images a related tie-point is present (multiplicity) and its positions in those images.
  - For all the images including the master and the related images:

- Create a grid that divides the image pixel-space.
- Fill in the grid with the loaded multi-tie-points.
- For each cell of the master image grid:
  - Sort the multi-tie-points according to multiplicity.
  - Attempt to remove all the multi-tie-points in the grid cell except the one with higher multiplicity. A multi-tie-point can be deleted if it meets the following conditions:
    - Condition 1: It is not present in a related image that was master in an earlier executed task.
    - For each related image where the multi-tie-point has a tie-point: (Condition 2) there is at least another tie-point in the current master grid cell that is also shared with the related image, and (Condition 3) there is at least another tie-point in the grid cell of the related image.
  - Store the tie-points which have not been marked as deleteable.

### 15.5.2 Parallelization

The *RedTieP* executes the sets of tasks that perform the tie-point reduction using a single process and in sequential order. However, these tasks could be parallelized if the parallelization schema guarantees that there are never two tasks accessing the same set of tie-points (i.e. accessing the same files). Each tasks has mutual exclusion with some other tasks. In order to parallelize them we use a workflow execution engine called Noodles (<https://github.com/NLeSC/noodles>). A script that uses Noodles can be found in *scripts/noodles.exe\_parallel.py*.

In order to run *RedTieP* and parallelize it with Noodles use:

```
mm3d RedTieP {Pattern of images} ExpSubCom=1
python {MicMac path}/scripts/noodles_exe_parallel.py -j {num. threads} subcommands.json
```

To install Noodles Python 3.5 is required. We recommend downloading and installing Anaconda (<https://www.continuum.io/>). Then set an environment with Python 3.5, activate it and download and install Noodles:

```
git clone https://github.com/NLeSC/noodles.git
cd noodles
git checkout devel
pip install .
```

## 15.6 Global and order-agnostic tie point reduction with Schnaps

The command **Schnaps** is used to clean and reduce tie points before any orientation, and without needing any order in the pictures. Its limitation is the user memory: it can't be used if computer RAM is lower than Homol directory size.

```
mm3d Schnaps
Schnaps : reduction of homologue points in image geometry
S trict
C hoice of
H omologous
N eglecting
A ccumulations on
P articular
S pots
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
  * string :: {Pattern of images}
Named args :
  * [Name=HomolIn] string :: {Input Homol directory suffix (without "Homol")}
  * [Name=NbWin] INT :: {Minimal homol points in each picture (default: 1000)}
  * [Name=HomolOut] string :: {Output Homol directory suffix (default: _mini)}
  * [Name=ExpTxt] bool :: {Ascii format for in and out, def=false}
  * [Name=VeryStrict] bool :: {Be very strict with homols (remove any suspect), def=false}
  * [Name=PoubelleName] string :: {Where to write suspicious pictures names, def="Schnaps_poubelle.txt"}
```

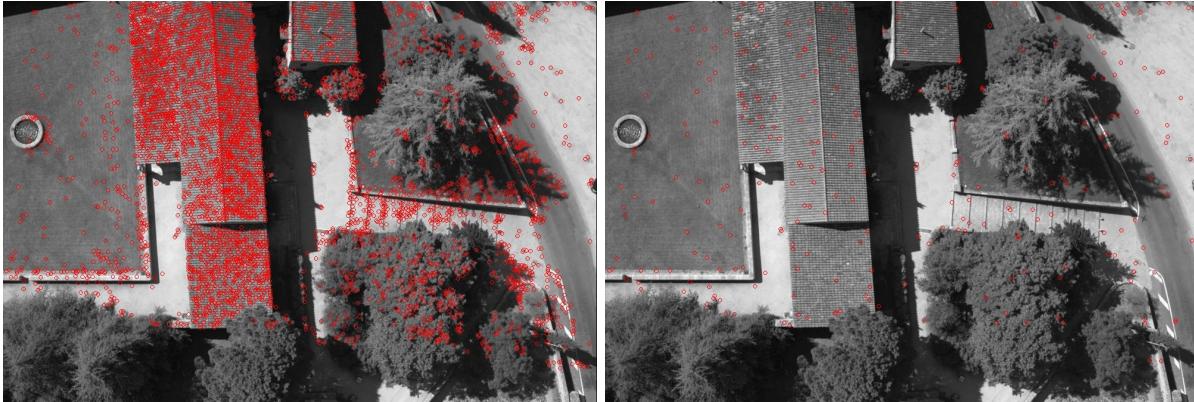


Figure 15.4: Tie points before and after Schnaps with 100 sub-windows

You can choose the Homol directory suffix for input and output. By default it uses “Homol” and creates a “Homol\_mini” directory for output.

The number of windows gives a clue about the number of tie points you will get. You may get less points if there are no tie point in every window, and you may get more points if they have a great multiplicity.

The suspicious Homol points will be removed (then detecting bad loops of tie points), and the missing pairs will be completed (closing the loops).

### 15.6.1 Algorithm

**Schnaps** computes a collection of Homol points, each recording its coordinates in every picture it appears. When something is not coherent, the Homol point is discarded.

It then selects the best Homol point (multiplicity) in each sub-window of each picture. When an Homol point is selected, it appears in every picture it is seen in.

The tie points packs are then created using every selected Homol and every combination of pictures. This may create new links between pictures (mostly if you used **Tapiocal Line** with a low number of adjacent image).

The **VeryStrict** option makes **Schnaps** check that every Homol has been seen in every couple Tapioca tested.

### 15.6.2 Output

**Schnaps** will create a new directory with filtered tie points. It will also evaluate the “useful” area of each picture, and give a list of bad pictures (less than 25% of the area used) in a file called **Schnaps\_pictures\_poubelle.txt**. You may use it to clear your pictures list before **Tapas**.

Then computing orientations, filtered tie points may give bigger residuals since they are less redundant, but bascule tests show that the geometry is better after filtering.

The orientation computation may also be faster if the number of points decreased, and **Schnaps** also allows you to earn time with tighter **Tapiocal Line**, since it will fill the links between pictures.

## 15.7 Tie point reduction , with **OriRedTieP** and **Ratafia**

### 15.7.1 Generalities

As the problem of tie points reduction has been very active, the command **OriRedTieP** and **Ratafia** are alternative solution to **RedTieP** and **Schnaps**; it is expected that all these options correspond to complementary requirements and, as there is for now no deep comparison between them, the user is invited to test which one best comply with his particular problem.

For the two last one, **Ratafia** is a general command while **OriRedTieP** is specialized for the quasi-vertical case (typically aerial or UAV) and it is expected to be a bit more efficient in time and quality. Also, the difference between the two is tiny and it is more for historical reasons that these two commands co-exist.

As with all tie-point reduction methods, the general objective is to limit the number of tie-point while maintaining a good photogrammetric distribution. This rather general objective leads to the (somewhat contradictory) specifications :

- select a minimal subset of tie points ;

- for each image, the subset of points where the image is present must have an homogeneous distribution (at least no hole, the notion of "hole" being controlled by a threshold on distance);
- similarly, for each pair of images, the subset of points where the pair is present must have no hole;
- as far as possible, the point with high degree of multiplicity must be privileged (they give more "strength" to the photogrammetric canvas);
- if there is any means to evaluate the quality of tie points based on geometry, then privilege the points of good quality.

The algorithm for the two commands are pretty much the same. First we give a detailed description of `OriRedTieP`, and after we describe the few points that are specific to `Ratafia`.

### 15.7.2 Tie point reduction , quasi-vertical case with `OriRedTieP`

#### 15.7.2.1 Algorithm

The command `OriRedTieP` treats the problem of tie points reduction in the particular case where the acquisition is quasi-vertical (generally applicable to UAV acquisition). It requires that a global orientation has been computed with the `Martini` command, as this orientation will be used both for computing the spatial distribution of tie points and evaluate the quality of selected tie points (based on the reprojection precision). As `Martini` is memory efficient it can be executed with almost arbitrary big data and there is no vicious circle.

To have a spatial distribution, for each tie point, the bundle intersection  $P^{Gr}$  is computed with the given orientation. The density of the tie point after reduction will be controlled with a parameter  $D_{Mul}$  which will be more or less the average distance between  $P^{Gr}$  of selected tie points. Also `OriRedTieP` use only the  $X, Y$  coordinate of  $P^{Gr}$  and this is why it is only suited for quasi-vertical acquisition<sup>2</sup>

For its computation `OriRedTieP` needs to evaluate the quality of each tie point, and it uses the following formula :

$$Qual = NbP * \frac{1}{1 + (\frac{R}{R_m * Th_R})^2} * (\frac{1}{2} + \frac{NbI}{NbI_0}) \quad (15.1)$$

Where :

- $NbP$  is the number of pairs of images (for example for a point with multiplicity 4, its value is between 3 and 6); this term privileges multiplicity of tie points;
- $R$  is the residual of reprojection and  $R_m$  is the median of this residual on the data,  $Th_R$  is a threshold (its default value is 2); this term penalizes high residual;
- $NbI$  is the number of images that are still uncovered (see below the definition, initially no image is covered), and  $NbI_0$  is the number of initial images; this term take into account the fact that once a tie point is partly covered, its potential contribution to the (photogrammetric) strength of the block decreases.

Finally, the tie points selection algorithm goes as follows :

- extract the tie point  $P$ , not entirely covered, with the best quality (if none, end);
- add  $P$  to the set of selected tie points , then for all points  $Q$  located within the distance of  $D_{Mul}$  from the point  $P$ 
  - for all images of  $Q$ , that are also in  $P$ , mark these images as covered;
  - if all images of  $Q$  are covered, remove  $Q$  ( $Q$  is considered as no longer useful if for each image it contains, inside a disc of ray  $D_{Mul}$ , there exists a selected tie point containing that image);
  - else, update the quality of  $Q$ , according to formula 15.1 to take into account the fact that the number of uncovered images has decreased.

This computation is done relatively fast, as a spatial indexe is used to extract the points in a given disc, and a heap is used extract the tie points with highest quality.

#### 15.7.2.2 "Von Gruber" point

The previous algorithm guarantees that for each image, it has selected tie points with "no hole". But it does not give the same warranty for a pair of images which may be a problem for the photogrammetric strength of a bloc. The so named Von-Gruber<sup>3</sup> points aim to fill this gap. A second analysis of the tie point is made , the algorithm being close but slightly different from the previous one :

- all the pair  $I_1, I_2$  of images are considered one after the other;
- in the current step only tie point containing  $I_1$  and  $I_2$  are considered, let  $S_{i1,i2}$  be this set;

---

2. this may evolve later, but is due to the fact that now in the MicMac library there is quad-tree and no octree

3. typically if the number of such points where 6 their optimal distribution would be those of the Von-Gruber point in "classical" photogrammetry

- let  $VG_{i1,i2}$  be the set of tie point to be selected for  $I_1, I_2$ ;  $VG_{i1,i2}$  is initialized with the points selected already in the previous steps and containing  $I_1$  and  $I_2$  (i.e these point can come from global algorithm in 15.7.2.1, or from previous iteration, with other pairs, of these Von Gruber points).

Let :

$$D_{i1,i2}^{VG}(Q) = \min_{P \in VG_{i1,i2}} d(P, Q) \quad (15.2)$$

We define the quality of potentiel point  $Q$  by :

$$Qual^{VG}(Q) = \frac{D_{i1,i2}^{VG}(Q)}{1 + 2 * \frac{R}{R_m}} \quad (15.3)$$

This formula means that we want to add the tie points with the highest distance from the already selected points (filling the "biggest hole") but also penalise points with high residual. The algorithm is then :

- let  $D_{max}^{VG}$  be the max of  $D_{i1,i2}^{VG}(Q)$  for  $Q \in S_{i1,i2}$  ;
- let  $D_{Mul}^{VG}$  be a threshold distance, by default  $D_{Mul}^{VG} = 2 * D_{Mul}$  (where  $D_{Mul}$  is the distance used for in 15.7.2.1);
- while  $D_{max}^{VG} < D_{Mul}^{VG}$  add to  $VG_{i1,i2}$  the point maximizing  $Qual^{VG}$ .

### 15.7.2.3 The command

Before executing `OriRedTieP`, `Martini` must have been executed with the same option for the calibration, for example :

```
mm3d Martini DSC01.*JPG
mm3d OriRedTieP DSC01.*JPG
```

Or :

```
mm3d Martini DSC01.*JPG OriCalib=Ori-Calib/
mm3d OriRedTieP DSC01.*JPG OriCalib=Ori-Calib/
```

As usual to know the parameters :

```
mm3d OriRedTieP -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Pattern of images}
Named args :
 * [Name=OriCalib] string :: {Calibration folder if any}
 * [Name=Prec2P] REAL :: {Threshold of precision for 2 Points}
 * [Name=KBox] INT :: {Internal use}
 * [Name=SzTile] INT :: {Size of Tiles in Pixel Def=2000}
 * [Name=DistPMul] REAL :: {Typical dist between pmult Def=200.000000}
 * [Name=MVG] REAL :: {Multiplier VonGruber}
 * [Name=Paral] bool :: {Do it paral, def=true}
 * [Name=VerifNM] bool :: {(Internal) Verification of Virtual Name Manager}
```

The signification of the main parameters are :

- mandatory parameter the pattern of images (must be a subset of the one used with `Martini`).
- `OriCalib` must be coherent with the parameter name used in `Martini`;
- `DistPMul` controls the density of points per image and it's the average distance between the tie points; there is a waranty that for all images and the initial tie points, there exists a selected tie point at a distance inferior to `DistPMul`
- `MVG` controls the density of points per pair; for a given pair  $I_1, I_2$  and `DistPMul`, the images will have a given density, but it does not give any guarantee of "goodness" on the points belonging to  $I_1, I_2$ , which may be a problem for the robustness; let  $D' = MVG * DistPMul$ , for each pair  $I_1, I_2$  there is a guarantee that for each tie points of  $I_1, I_2$ , there exists a selected tie point of  $I_1, I_2$  at a distance inferioir to  $D'$ ;

#### 15.7.2.4 Memory issue and parallelization

As the aim of **OriRedTieP** is to be able to process big data set with standard memory, obviously the tie points cannot be all loaded simultaneously. As ground coordinate can be computed, it is easy to separate the problem in small tiles which are computed independantly and this is what is done.

Also, one benefit of this tiling is that the tiles being independant, the computation can be done in parallel and this is what is done.

### 15.7.3 Tie point reduction , general case with Ratafia

#### 15.7.3.1 The algorithm

The tool **Ratafia** can deal with any kind of acquisition. The main difference with **OriRedTieP** is that the spatial indexation (computation of distance) is done in image geometry. This has the following consequences :

- the current computation is done with a "master" image  $I_0$ ;
- in this current computation only the tie points that contains  $I_0$  are considered;
- the position of a tie-point, used for distance computation, is the value of the point in  $I_0$ ;
- to assure a complete coverage of the acquisition, each image at a certain point will have to be the "master" image;
- as soon as the first image is processed, some tie points are selected, and it must have an influence on the computation of next images (as some points are already "covered" by the already selected tie points);
- so basically, the programm cannot be parallelized naively, as the result of each selected master image may influence the selection on remaining images;
- however when two images  $I_1$  and  $I_2$  have no common points, there is no contradiction to execute the computation in parallel (their tie points are completely separated; remember : for example when computing  $I_1$  only the points containing  $I_1$  are considered).

The algorithm used by **Ratafia** is the following :

- create a partition of the images  $P = \{S_1, S_2, \dots, S_N\}$  such that  $\forall i, j, n$  with  $I_i \in S_n$  and  $I_j \in S_n$  we have  $TieP(I_i, I_j) = \emptyset$ ; in fact this condition is not fixed strictly, and it is accepted that the number of tie points may be equal to a very few percentage of the points;
- sequentially process each element of the partition  $S_1, S_2, \dots$ , for each element  $S_k$  do:
  - let  $S_k = \{I_1, I_2, \dots\}$ , for each element  $I_k$  do in parallel :
    - read the already computed tie points, and add them to initially selected tie points;
    - consider only multiple tie points linked to  $I_k$ , and use  $I_k$  coordinates for indexation;
    - then do the same computation as in 15.7.2.1 ,
    - save only the tie points that were added at this step.

There is also a slight difference with **OriRedTieP**, as there is no need for a global 3d geometry. Moreover, as the tool **Martini** (that otherwise furnishes the global geometry) is still in progress, the evaluation of the residual is not made on a global orientation, but on the relative orientation between the pair of images. This is theoretically not as good, because intersection of all images 2 by 2 is not equivalent to a global intersection, but the difference is tiny. Nonetheless, when **Martini** will be completely finished, there will be an option to use the 3d geometry.

#### 15.7.3.2 The command line

Before executing **Ratafia**, compute the relative orientation between pairs of images with the command **TestLib NO\_AllOri2Im** (first step of **Martini**). Remember to execute the command with the same calibration option, for example :

```
mm3d TestLib NO_AllOri2Im DSC01.*JPG
mm3d Ratafia DSC01.*JPG
```

Or :

```
mm3d TestLib NO_AllOri2Im DSC01.*JPG OriCalib=Ori-Calib/
mm3d Ratafia DSC01.*JPG OriCalib=Ori-Calib/
```

Also, runing **Martini** instead of **TestLib NO\_AllOri2Im** would have worked perfectly, but the global orientation would not have been used. As usual to know the parameters :

```
mm3d Ratafia -help
*****
* Help for Elise Arg main *
*****
```

```
Mandatory unnamed args :
 * string :: {Pattern of images}
Named args :
 * [Name=OriCalib] string :: {Calibration folder if any}
 * [Name=LevelOr] INT :: {Level Or, 0=None,1=Pair,2=Glob, (Def=1)}
 * [Name=NbP] INT :: {Nb Process, def use all}
 * [Name=RecMax] REAL :: {Max overlap acceptable in two parallelly processed images}
 * [Name>ShowP] bool :: {Show Partition (def=false)}
 * [Name=SzPixDec] REAL :: {Sz of decoupe in pixel}
 * [Name=TEO] bool :: {Test Execution OriRedTieP ()}
 * [Name=Out] string :: {Folder dest => Def=-Ratafia}
 * [Name=DistPMul] REAL :: {Average dist}
 * [Name=MVG] REAL :: {Multiplier VonGruber, Def=2.000000}
 * [Name=Paral] bool :: {Do it in parallel}
 * [Name=DCA] bool :: {Do Complete Arc (Def=false)}
 * [Name=UseP] bool :: {Use prec to avoid redundancy (Def=true), tuning only}
```

The meaning of main parameters is:

- mandatory parameter the pattern of images (must be a subset of the one used with `TestLib NO_AllOri2Im`).
- `OriCalib` must be coherent with the parameter of same name used in `TestLib NO_AllOri2Im`;
- `NbP` number of processes on which the parallelization is done;
- `RecMax` proportion of common tie point where images are considered disconnected ( $\text{def} = \frac{1}{100}$ );
- `DistPMul` average distance in pixel between tie points;
- `DCA` Do Complete Arc, if this option is active an attempt will be made to complete the incomplete tie points; for example, with a triple tie point comming from the fusion of  $(I_1, p_1, I_2, p_2)$  and  $(I_2, p_2, I_3, p_3)$ ; during the export, the  $(I_1, p_1, I_3, p_3)$  will be added (i) if not already present, and if (ii) the geometric quality is sufficient (based on relative orientation); default value is false as it seems not to improve any accuracy;

`Ratafia` output some messages :

```
mm3d Ratafia DSC01.*JPG OriCalib=Ori-Calib/
=====
Done 0 Part on 8 =====
===== Done 1 Part on 8 =====
===== Done 2 Part on 8 =====
===== Done 3 Part on 8 =====
===== Done 4 Part on 8 =====
===== Done 5 Part on 8 =====
===== Done 6 Part on 8 =====
===== Done 7 Part on 8 =====
----- NbP=1246 -----
For mulplicity 2 %=28.3307 N=353 D=1
For mulplicity 3 %=25.1204 N=313 D=2.59425
For mulplicity 4 %=15.7303 N=196 D=4.62245
For mulplicity 5 %=10.7544 N=134 D=7.36567
For mulplicity 6 %=9.55056 N=119 D=10.6975
For mulplicity 7 %=5.939 N=74 D=14.5676
For mulplicity 8 %=3.29053 N=41 D=17.7805
For mulplicity 9 %=1.28411 N=16 D=20.875
*****
*
*      R-eduction      *
*      A-utomatic of   *
*      T-ie points      *
*      A-iming to get   *
*      F-aster          *
*      I-mage           *
*      A-erotriangulation*
*      *
*****
The first series of message is just an indication of the computation. The second series is some statistic on the distribution of tie points :
```

- let me comment For multiplicity 3 % = 25.1204 N=313 D=2.59425 :
- there is 313 point of multiplicity 3, they represent 25.1% the average of number of couple is 2.59 (the value being theoretically between 2 and 3).



# Chapter 16

## Advanced orientation

### 16.1 Creating a calibration unknown by image

#### 16.1.1 When is it necessary?

It sometimes happens that each image, or each group of images, has been acquired with a different set of internal parameters. Here are possible cases:

- when images were acquired with autofocus, which creates variation of the focal length (in macro photo, the focal length when focus is at  $\infty$  is half the focal length when the focus is at image ratio of 1 : 1);
- when the image stabilizer is free, this creates (at least) variation of principal point;
- when images were acquired with variable zoom.

From a photogrammetric point of view, these cases must be avoided as much as possible; however there are times when the user has no choice.

#### 16.1.2 Examples

The directory `applis/XML-Patron/Oiseau-Margot/` contains XML files that were used to process images acquired with macro lenses. The files `New-Apero1.xml` to `New-Apero6-ExportDirPlanMM.xml` contain examples on real cases of the different mechanisms shown here. By the way, these examples may be a bit complex because they were made before key standardization.

See especially the examples `Apero-ExCalibPerIm-1.xml` and `Apero-ExCalibPerIm-2.xml` on `MurSaintMartin`, that have been added after writing this section; they are more realistic and contain some comments at the beginning that should be sufficient.

#### 16.1.3 How to create unknowns

The optional section `<CalibPerPose>`, under `<CalibrationCameraInc>`, allows to handle these cases. It contains a mandatory args `<KeyPose2Cal>`:

- it must contain a string  $K$  which describes a key association, (see 11.5 for `<KeyedNamesAssociations>`);
- two images  $I_1$  and  $I_2$  will share the same internal parameters, if and only if  $K(I_1) = K(I_2)$ ;
- for example, if  $K$  is the identity key, a new calibration will be created for each new image;
- for each of these calibrations, the identifier will be  $K(I)$  and not, as usual, the tag `<Name>`; this is necessary because elsewhere different internal calibration would have the same identifier;
- this identifier is used when it is necessary to refer to a set of internal calibration (for example when applying constraint only to a subset of the existing calibration);

#### 16.1.4 Saving results with variable calibrations

The most current case for using these mechanisms is when there is one calibration per image. In this case, the easiest way for handling the results is to simply save the internal calibration with the external calibration; this is, by default, what Apero does in `<ExportPose>`, see `Apero-0.xml` in 6.2.8.

For more complicated cases, an `<ExportCalib>` section will have to be used with the following tags:

- `<PatternSel>` to specify to which calib it applies; the selection is made on the identifier (here  $K(I)$ );
- `<KeyAssoc>` to specify how to compute a file name from the identifier;
- `<KeyIsName>` at `false`<sup>1</sup> meaning that `<KeyAssoc>` is a key and not an absolute name;

---

1. which is, however, the default value

### 16.1.5 Loading initial values with variable calibrations

When variable internal calibration is used a first time, the different calibration can be initialized with the same value. This value can be read as usual in the `<NameFile>` of `<CalValueInit>`. See an example in `applis/XML-Patron/Oiseau-Margot/New-Apero1.xml`,

There is other cases where it will be needed to initialize these calibrations from variable values; for example, values that have been computed and saved by a previous running of Apero. In this case, a `KeyedNamesAssociations` will be used to specify the association between *the name of the pose* and the file where the initial value is to be read; this key is specified in the optional `<KeyInitFromPose>`; see example in `applis/XML-Patron/Oiseau-Margot/New-Apero3.xml`.

### 16.1.6 Examples with group of poses

Sometimes we do not want to create a calibration for each image, but a calibration for each group of images. This can occur when we know that the parameters affecting the calibration (zoom, focus) have changed, but only a few times, and we are able to specify which groups of images share the same parameters. This can also occur when we have used several versions of the same camera with the same focal length (so not distinguishable with the procedure used in `Tapas`).

See `Apero-ExCalibPerGROUPIM-1.xml` and `Apero-ExCalibPerGROUPIM-2.xml` on `MurSaintMartin` data set, it illustrates how this can be done. The file contains some comments.

The files `Apero-ExCalibPerIm-1.xml` and `Apero-ExCalibPerIm-2.xml` on `MurSaintMartin` have been added and contain also examples for calibration per images which are probably easier to understand than `applis/XML-Patron/Oiseau-Margot`.

### 16.1.7 Enforcing a smooth evolution

Someting, it may be usefull to have a calibration per image but to also enforce that this calibration evolve slowly. This may be the case, for example, if the variation of focal lenght is due to thermal evolution.

This is possible with option `ContrCalCamCons` of `Campari`, for now the calibration must be of type `ModelePolyDeg0` or `ModelePolyDeg1`<sup>2</sup>.

The file `Cmd2.txt` in `Documentation/NEW-DATA/Rigid-Block` contains an example. The two last command are :

```
Tapas AddPolyDeg1 DSCF.*jpg InOri=Ori-AllRel/
```

```
Campari DSCF.*jpg Ori-AddPolyDeg1/ TestSmoothEvolv CPI1=true ContrCalCamCons=[Loc-Assoc-Im2Block,2] FocFree=1 PPFre
```

Some comments :

- we first use `Tapas AddPolyDeg1` to make a 1 degree polynomial model;
- in `Campari` we use the `CPI1=true` to make one calibration per image (else is would be useless);
- we use also `FocFree=1 PPFre=1` to free focal and principal point (else again the option would be useless)
- the option `ContrCalCamCons` contains two value, first is a key (here `Loc-Assoc-Im2Block`) , second is sigma (here  $\sigma = 2$  meaning we expect the focal to evolve  $\pm 2\text{pix}$  from calibration to calibration);

The meaning of `Loc-Assoc-Im2Block` is :

- it must return for each name of image, two value `Time` and `Grp`;
- the value `Grp` indicate the group of image, the constraint will be applied inside image of the same `Grp`;
- the value `Time` indicate an order, the constraint will be applied between pair of consecutive camera regarding this order;
- here the key that describe the rigid block works perfectly for what we want to do.

The meaning of  $\sigma$  is :

- let  $C_1$  and  $C_2$  be two succesive calibration
- for a pixel  $P$  let  $N_k(P)$  be the direction of emerging ray for camera  $C_k$
- we add to the bundle adjustment minimisation function a regularisation equation  $R_{1,2}$  as writes 16.1
- $\iint_C$  mean integral on the whole sensor;
- $\sigma$  is expressed in pixel.

$$R_{1,2} = \frac{\iint_C |N_1(P) - N_2(P)|^2}{\iint_C \sigma^2} \quad (16.1)$$

`Campari` has printed an additionnal message :

---

2. This can be easily generalized, but not sure it is usefull

```
ContCamConseq= 9.69226e-06 for DSCF3297_L.jpg
ContCamConseq= 2.19443e-05 for DSCF3297_R.jpg
ContCamConseq= 2.0443e-05 for DSCF3298_L.jpg
ContCamConseq= 2.14662e-05 for DSCF3298_R.jpg
```

These values are the residual of formula  $R_{1,2}$ . Here they are very low, in fact we can check the computed value give almost the same focals :

```
grep "<F>" Ori-TestSmoothEvolv/Orientation-DSCF329*
Ori-TestSmoothEvolv/Orientation-DSCF3297_L.jpg.xml: <F>4332.52026468588519</F>
Ori-TestSmoothEvolv/Orientation-DSCF3297_R.jpg.xml: <F>4374.50135394254175</F>
Ori-TestSmoothEvolv/Orientation-DSCF3298_L.jpg.xml: <F>4332.52029293327632</F>
Ori-TestSmoothEvolv/Orientation-DSCF3298_R.jpg.xml: <F>4374.50128551130183</F>
Ori-TestSmoothEvolv/Orientation-DSCF3299_L.jpg.xml: <F>4332.52035901077033</F>
Ori-TestSmoothEvolv/Orientation-DSCF3299_R.jpg.xml: <F>4374.50121606732682</F>
```

## 16.2 Database of existing calibration

### 16.2.1 General points

### 16.3 Auxiliary exports

#### 16.3.1 Generating point clouds with <ExportNuage>

### 16.4 Using scanned analog images

#### 16.4.1 Dealing with internal orientation

Scanned analog images are important in many applications, as they represent a valuable source of information for studying phenomena on long periods of time. From the photogrammetric point of view, the main difference between scanned images and digital camera is that for each images there is a specific transformation between the photo and the scanner. This transformation can be computed when there exist fiducial marks on the camera. This section presents how this can be done with **Apero/MicMac**.

The directory **DemoScanned/** contains a data set that illustrates these features. It contains 5 images that are a simulation of scanned images: the images have been randomly rotated and scaled, simulating the interior orientation of the scanner; before the rotation, 8 fiducial marks have been added. Figure 16.1 illustrates this data set.

There are two slight differences in data processing between such data sets and "classical" digital images:

- the position of fiducial mark on images and on the camera has to be indicated;
- the calibration will not be expressed in pixel, but in the same unit as the position of the reference fiducial marks (generally mm).

On **DemoScanned/**, the directory **Ori-InterneScan/** contains all the information about the fiducial marks. It works like this:

- each file contains a structure, a type **<MesureAppuiFlottant1Im>** which describes a list of named points (here the names are P0, P1, ...; this is the same structure as the one used for image measurement of GCP, as seen in 6.4.4.1);
- there is a file **MeasuresCamera.xml** that contains the position of the fiducial marks on the camera;
- for each image XXX, there is a file **MeasuresIm-XXX.xml** that contains the position of the marks on the image; when this file does not exist, the image is considered to be a "classical" digital image that will be processed as usual;
- if required<sup>3</sup>, it is possible to change the association between an image and the two files: position in camera and image; for this, you must change the value of **Key-Assoc-STD-Orientation-Interne**<sup>4</sup> in your **MicMac-LocalChantierDescripteur.xml**

Here, the file **MeasuresCamera.xml** contains the positions of fiducial marks in mm, all the calibration must then be have the same unit and be in the same frame than these marks. For technical reasons, this point of reference must be the upper left corner and not the center. As **Apero/MicMac** cannot deal correctly with default calibration in mm, we have to give an initial value in file **Ori-CalibInit**. Some comments on this file:

- the size of image **<SzIm>** is also in mm (as the normalization focal **<Etats>** and all parameters).

3. for example if several analog camera are used in the same bundle

4. see in include/XML-GEN/DefautChantierDescripteur.xml the default value

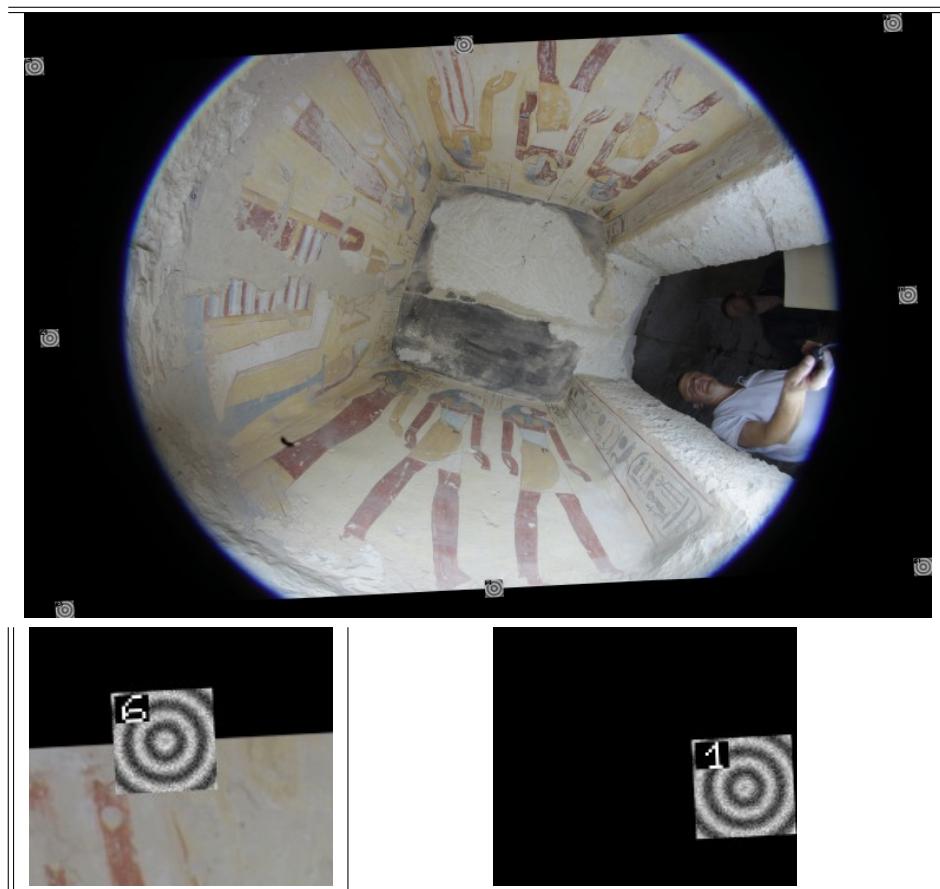


Figure 16.1: Simulation of fiducial marks: an image and two marks

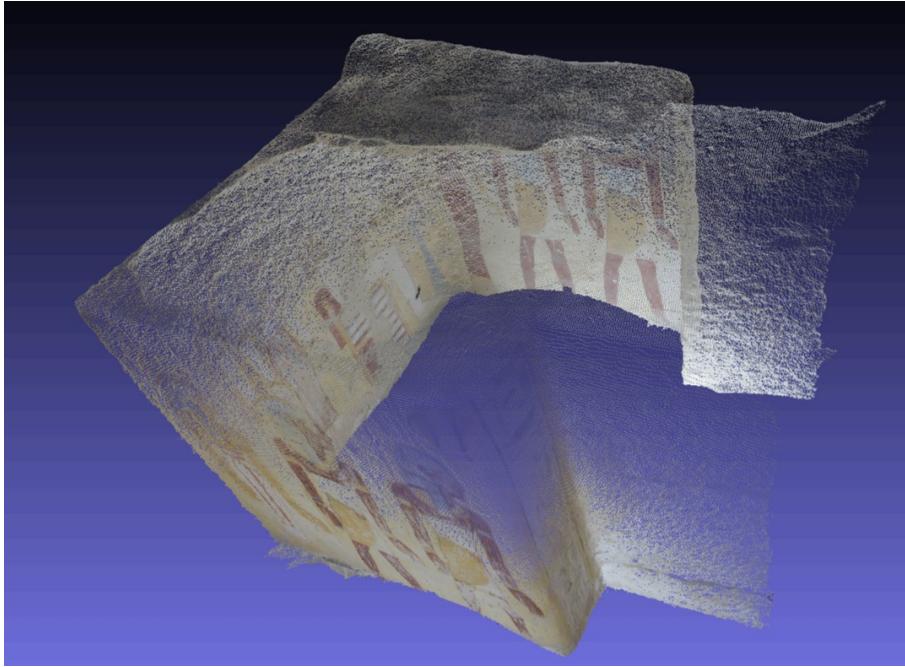


Figure 16.2: Point cloud with simulation of scanned images

- the optional `<ScannedAnalogik>` is set to true, required because it will indicate to `Apero` to be "tolerant" if tie points are detected out of the bounding box  $[0, 0]x[24, 36]$ ;

The file `ExCmd.txt` contains a possible processing of the data:

- `Tapioca All ".*jpg" 1200`, as usual ...
- `Tapas FishEyeBasic ".*jpg" Out=Ori1 InCal=CalibInit PropDiag=0.68`
  - it is necessary to indicate the calibration in `CalibInit` in *mm* because `Apero` would not built it correctly;
  - no need to indicate situation of fiducial mark, the def value of `Key-Assoc-STD-Orientation-Interne` will force `Apero` to look for them at the right place;
  - `PropDiag=0.68`, because it is a hemispheric fisheye;
- `Tapas FishEyeBasic ".*jpg" InOri=Ori1 Out=Ori2 PropDiag=0.68`
  - just to check that `Tapas` can be iterated in this configuration
- `Malt GeomImage ".*jpg" Ori2 Master=IMG_5693_Out.jpg Spherik=true SzW=2`
  - the `Spherik=true` is well adapted to the scene, in this geometry `MicMac` computes the depth  $R = f(i, j)$  where  $R$  is the distance to the master image center<sup>5</sup>;
- `Nuage2Ply MM-Malt-Img-IMG_5693_Out/NuageImProf-STD-MALT_Etape_8.xml Attr=IMG_5693_Out.jpg Scale=2`
  - usual generation of point cloud in ply format (figure 16.2);

If you take a look at orientation files, you will see that they are self sufficient for matching: the `<OrIntImaM2C>` section contains the affinity between scanner and image computed from the fiducial marks:

```

<OrientationConique>
...
<OrIntImaM2C>
  <I00>2753.06179948014324 1771.95961861625119</I00>
  <V10>-72.5948450058228758 4.58183503308403939</V10>
  <V01>-4.5818350330840607 -72.5948450058228332</V01>
</OrIntImaM2C>
...
</OrientationConique>

```

5. i.e. rectification is made on sphere

### 16.4.2 Semi-automatic fiducial mark input with Kugelhupf

Kugelhupf (Klics Ubuesques Grandement Evites, Lent, Hasardeux mais Utilisable pour Points Fiduciaux) is a tool for scanned images. It is used to automatically find fiducial marks on the images.

If the fiducial marks are almost on the same position on each picture, it is necessary to point them on one image and Kugelhupf will find the fiducial marks on the others.

Syntax of Kugelhupf:

```
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Pattern of scanned images}
 * string :: {2d fiducial points of an image}
Named args :
 * [Name=TargetHalfSize] INT :: {Target half size in pixels (Def=64)}
 * [Name=SearchIncertitude] INT :: {Search incertitude in pixels (Def=5)}
 * [Name=SearchStep] REAL :: {Search step in pixels (Def=0.5)}
```

Call example: mm3d Kugelhupf "1987\_FR4074.\*.tif" Ori-InterneScan/MeasuresIm-202.tif.xml SearchIncertitude=10

The output is xml files in **Ori-InterneScan** for every picture where the automatic search was successful. If at least one point was not found for an image, the xml file is not created. Kugelhupf only works on picture that have no xml file.

This is useful to make successive calls to Kugelhupf with different search incertitudes :

```
mm3d Kugelhupf "1987_*.tif" Ori-InterneScan/MeasuresIm-202.tif.xml SearchIncertitude=10
mm3d Kugelhupf "1987_*.tif" Ori-InterneScan/MeasuresIm-202.tif.xml SearchIncertitude=25
```

The first call will be fast and will have a result only with pictures with close fiducial points, and the second call will be slower but only on the worst pictures.

### 16.4.3 FFT variant with FFTKugelhupf

The FFTKugelhupf tool is a variant of Kugelhupf; the "philosophy" and interface is the same than Kugelhupf : it assumes that the mark can be retrieved from a "master" image that gives the shape and approximate position of each mark. When the interval of research is important FFTKugelhupf can be significatively faster due its use of fast fourrier transform for initial guess and multi resolution for more accurate positionning. However the two tool are of interest as there is no much testing of FFTKugelhupf which may fail more frequently when the mark are very small.

We can test it with the data set DemoScanned of 16.4.1.

```
mm3d FFTKugelhupf "IMG_569[4-7]_Out.jpg" Test-5963.xml Masq=NONE
ESIDU = 94.3549 for IMG_5697_Out.jpg
RESIDU = 0.269043 for IMG_5696_Out.jpg
RESIDU = 300.811 for IMG_5694_Out.jpg
RESIDU = 144.772 for IMG_5695_Out.jpg
```

Note :

- RESIDU = 0.269043 is the value of the residual computed by adapting an affinity between the initial mark and the detected mark; it is a good indicator of the quality of the match;
- here with this unrealistic data set, the results are rather poor with only one image with all good matches;
- with a data set more realistic as illustrated in figure 16.3 , we obtain currently 100% of images with residual better than one pixel;
- the default interval are quite high 500 pixel for the incertitude and 150 for the target half size;

```
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Pattern of scanned images}
 * string :: {2d fiducial points of an image}
Named args :
 * [Name=TargetHalfSize] Pt2di :: {Target half size in pixels (Def=150)}
 * [Name=Masq] string :: {Masq extension for ref image, Def=Fid, NONE if unused}
```

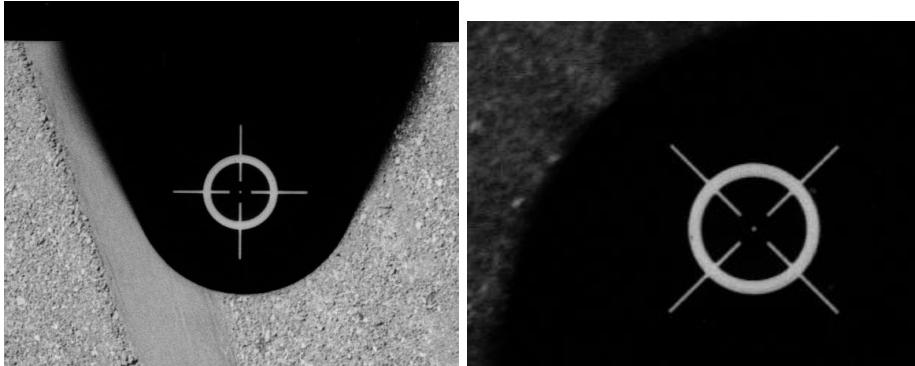


Figure 16.3: Real Fiducial mark

```
* [Name=SearchIncertitude] INT :: {Def=500}
* [Name=SzFFT] INT :: {Sz of initial fft research, power of recomended, Def=256 or 128 depending other}
```

Two parameters are specific to **FFTKugelhupf** :

- **Masq** , if it exist it must be a mask superposable to the master image , the correlation is then restricted to this mask
- **SzFFT** size of the initial reduced image on which computation is made using fast fourrier transform; with all other parameters set to their default values, it is set to 128 which makes a resolution decimation of 10;

#### 16.4.4 Resampling images with ReSampFid

This section describes an approach to analog image processing different from the one described in 16.4.1. In 16.4.1 the image are unchanged and an internal orientation is computed. Here conversely the image are resampled in a geometry where all the mark are superposable, then the resampled images can be used as "standard" images acquired by digital camera.

On one hand, the approach of 16.4.1 is theoretically slightly better as it avoid one resampling of the image. On the other hand, the approach described here is simpler for the implementation as once the first resampling is done, there no more special case to deal with. Practically the approach of 16.4.1 has led to severally tricky bug and, and it is highly recommended to use the approach described here as we probably will not have the man power to correct the next problem that may/will occur with the internal orientation approach.

```
mm3d ReSampFid -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Pattern image}
 * REAL :: {Resolution}
Named args :
 * [Name=BoxCh] Box2dr :: {Box in Chambre (generally in mm)}
 * [Name=Kern] INT :: {Kernel of interpol,0 Bilin, 1 Bicub, other SinC (fix size of apodisation window), De
```

Some comments :

- The value for fiducial mark position, on image and in the chambre, are searched in **Ori-InterneScan/**, or more exactly on the file described by the key **Key-Assoc-STD-Orientation-Interne**;
- the first parameter is the pattern of image to resample, if there is more than one, the process is done in parallel;
- the second parameter is the resampling resolution; for example if the fiducial mark are expressed in mm , and the image where scanned at 20 micron, a reasonable value could be 0.02;
- if the **BoxCh** is omitted, a default value will be computed using the englobing box of the fiducial marks;
- the image are renamed using the rule **toto.tif** → **OIS-Reech\_toto.tif** , with image name begining by **OIS-Reech**, the default rules ignorate the fiducial marks, so these images can be used as is; if user want to rename the resampled images, then he will need to hide the **Ori-InterneScan/** folder;

## 16.5 Adjustment with lines

### 16.5.1 Introduction

This section describes the case where user wants to adjust orientation using correspondence between  $3d$  points and image lines. It's rather specific, and was added in the context of georeferencing aerial photographs with a  $3d$  data base of roads<sup>6</sup>. Formally we have:

- a set of  $2d$  line  $L_k$  in images. Let  $I_k$  be the corresponding image and  $\pi_k$  be the projection function that goes from ground coordinates to  $I_k$ ;
- for each  $3d$  point a set of point  $p_{k,i}$  and we "know" that  $\forall k, i : \pi_k(p_{k,i}) \in L_k$
- let  $D(p, L)$  be the distance between point  $p$  and line  $L$ .

Mathematically we want to add the cost to the global minimization:

$$\sum_{k,i} D^2(\pi_k(p_{k,i}), L_k) \quad (16.2)$$

### 16.5.2 Data set

The folder `Documentation/NEW-DATA/CompensOnLine/` contains data that illustrates how this can be done in `Apero`. These are completely artificial (simulated) data which were generated for developing and testing this feature: in this example it will be possible to cancel completely the equation 16.2 which will obviously not be the case with real data.

To run the data set extracted from the mercurial server, one first needs to compute the tie points with:

- `mm3d Tapioca All Abbey-IMG_020.*jpg 1200`

To test all features, this data set has been processed as if it was the scans of analog images (see `Ori-InterneScan/`), but it also works with digital images. As it can be seen in `mm3d-LogFile.txt`, the orientation has been computed with the command:

- `mm3d Tapas AutoCal Abbey-IMG_020.*jpg InCal=Ori-CalibInitAnalogn/ Out=Rel`

This toy example is made from 4 small images on the same strip. Of course the data are completely artificial:

- when using the "real" orientation (i.e `Ori-Rel/`) the projection of point on lines is perfect;
- there exist data for each image;
- for each images the data is sufficient to compute its orientation;

### 16.5.3 Organization of information

Basically, the information is structured the same way as for GCP in 6.4.4.1, 3.10.1.2 and 3.9.2 :

- there is a file for storing the  $3d$  point, in this example `MesurePointGround.xml`, it is the same structure that for GCP, and the adjustment of this point can mix measurement of  $3d$  in ground,  $2d$  points in images and  $2d$  lines in images;
- there is a file for storing  $2d$  images line coordinates and the points that are supposed to project on these lines, in this example `MesureLineImage.xml`;
- in `Apero` these files are loaded, linked and then used for adjustment.

The structure of `MesureLineImage.xml` should be quite obvious:

- it must contain a global structure `<SetOfMesureSegDr>`;
- the `<SetOfMesureSegDr>` is a list of `<MesureAppuiSegDr1Im>`, each one storing the observation related to one image;
- a `MesureAppuiSegDr1Im` contains the name of the image `<NameIm>`, and list a of `<OneMesureSegDr>`, each one storing the information related to a line;
- a `<OneMesureSegDr>` contains exactly two  $2d$  points `<Pt1Im>` and `<Pt2Im>` that store the geometry of the line and a list of `<NamePt>` that contains the name of the  $3d$  points; there can be any number  $N \geq 1$  of `<NamePt>`, even if in this example we have  $N = 2$  everywhere<sup>7</sup>.

### 16.5.4 Example `Apero-2-DroiteStatique.xml`

It illustrates file loading. In this example, we only load the observation and check that the projection is perfect (because data are simulated). It's pretty much the same than 6.4.4.1 :

- `BDD_ObsAppuisFlottant` load the line information, in a data base name `Id-Appui`; the name of the file containing the line is stored in `<KeySetSegDroite>`;
- `PointFlottantInc` load the  $3d$  points information, in the same data base `Id-Appui`;

6. I am not sure to see if there exist other examples of application

7. which will probably be the standard case when the  $3d$  points come from  $3d$  lines

— ObsAppuisFlottant add the observation of the data base Id-Appui to the adjustment.  
As in this example the data are all frozen<sup>8</sup>, we just print the value of the observed distance, which turns to b 0 up to the rounding error. We obtain the error for each point, and the maximum of error :

```
...
ErrMax = 1.65784e-11 For I=Abbey-IMG_0205.jpg, C=P_8_B pixels
-----
===== ADD Pts P_9_A Has Gr 1 Inc [1,1,1]
--NamePt P_9_A Ec Estim-Ter [0,0,0]           Dist =0 ground units
Inc = [1,1,1]PdsIm = [1,1]
Ecart Estim-Faisceaux 16.2788
  ErrMoy 2.71776e-13 pixels SP=2
  ErrMax = 2.71776e-13 For I=Abbey-IMG_0204.jpg, C=P_9_A pixels
-----
===== ADD Pts P_9_B Has Gr 1 Inc [1,1,1]
--NamePt P_9_B Ec Estim-Ter [0,0,0]           Dist =0 ground units
Inc = [1,1,1]PdsIm = [1,1]
Ecart Estim-Faisceaux 16.1819
  ErrMoy 6.65003e-12 pixels SP=2
  ErrMax = 8.11932e-12 For I=Abbey-IMG_0205.jpg, C=P_9_B pixels
-----
=====
===== ERROR MAX PTS FL =====
|| Value=4.56508e-10 for Cam=Abbey-IMG_0204.jpg and Pt=P_2_A
=====
```

### 16.5.5 Example Apero-3-DroiteEvolv.xml

In this example, we want to check that with "perfect" data, the adjustment on line is sufficient to compute the "perfect" orientation as long as we are reasonably initialized. It's pretty much the same than 16.5.4, except that we start from start from orientation that have been modified and do not freeze the orientation.

```
...
=====
===== ERROR MAX PTS FL =====
|| Value=16.9249 for Cam=Abbey-IMG_0207.jpg and Pt=P_13_A
=====
--- End Iter 0 ETAPE 0
...
...
=====
===== ERROR MAX PTS FL =====
|| Value=0.00335622 for Cam=Abbey-IMG_0207.jpg and Pt=P_13_A
=====
--- End Iter 1 ETAPE 0
...
...
=====
===== ERROR MAX PTS FL =====
|| Value=8.87999e-09 for Cam=Abbey-IMG_0204.jpg and Pt=P_14_B
=====
--- End Iter 2 ETAPE 0
...
...
=====
===== ERROR MAX PTS FL =====
|| Value=3.17978e-10 for Cam=Abbey-IMG_0204.jpg and Pt=P_2_A
=====
--- End Iter 3 ETAPE 0
...
...
=====
===== ERROR MAX PTS FL =====
|| Value=3.17775e-10 for Cam=Abbey-IMG_0204.jpg and Pt=P_2_A
=====
```

8. see ePoseFigee and eAllParamFiges

--- End Iter 4 ETAPE 0

At the end, the orientation are exported in `Ori-Check3/`, and it can be seen that they are almost identical to `Ori-Rel`

### 16.5.6 Example `Apero-4-CompensMixte.xml` and `Apero-5-CompensAll.xml`

In this example, we check that, as we would do in real case, it is possible to adjust simultaneously GCP-line with other measurement. Also we use a reduced set of line-point (with `MesureLineImageIncompl.xml`), in this set there is lines only for the two central images.

In `Apero-4-CompensMixte.xml` we check that using simultaneously tie point and "GCP-line", we can recover the orientation of the extreme images (where there is no GCP-line).

`Apero-5-CompensAll.xml` had a minor modification, we see that in `<BDD_ObsAppuisFlottant >` we have

- `<KeySetSegDroite>` as before , to make adjustment on line;
  - an additional `<KeySetOrPat>` to make adjustment of 2d points like in 6.4.4.1;
- So in this example, the 3d point `P_1_A` will be adjusted simultaneously on:
- 2d point measure stored in `MesurePointImagePart.xml`;
  - 2d line measure stored in `MesureLineImage.xml`;
  - 3d point measure stored in `MesurePointGround.xml`.

## 16.6 Recent evolution in Tapas and other orientation tools

The evolution described here, are now integrated in the "standard" `Tapas` command. When necessary<sup>9</sup>, it is possible to get back to the previous behavior by using the `OldTapas` command.

### 16.6.1 Viscosity & Levenberg Marquardt stuff

Adding viscosity, also named more pedantically using Levenberg Marquardt algorithm, is a classical way to avoid problem due to ill conditioning system in energy minimization algorithm. In theory the only drawback is that it slow down the speed of convergence.

However, in Tapas, the default value of viscosity was badly dosed, and it some configuration , particularly UAV acquisition<sup>10</sup> lead to stop the bundle adjustment before the convergence . This is illustrated in figure 16.4. To change this fact, the comportment of Tapas/NewTapas has evolved this way :

- the initial default value of viscosity on center and rotation has been divided by 10;
- conversely the constraint to solve the arbitrary ambiguity has bee reactivated (i.e the first image is frozen, and the length of the base between two first images is frozen)
- a small initial viscosity has been added on internal parameters;
- in the third `<EtapeCompensation>` the viscosity is continuously decreasing;
- in the fourth step of `<EtapeCompensation>` the viscosity is highly strongly decreasing and the bundle does not stop before a test of convergence is satisfied (the test verify that with an accuracy of  $1E - 10$  the 3d point are identical after two consecutive steps).

As this new version can increase significantly the computation, the option `RefineAll=false` allow to limit the number of iteration (by a factor around 2), of course it has a risk of non convergence ...

### 16.6.2 Additional distortion

The kernel of Apero has two option that can be useful when attempting to modelize finely camera distortion:

- model with high degree of freedom
- possibility to define a distortion as composition of several distortion (as described in 14.2.1.1).

These option are now partly accessible in Tapas.

There is two family of distortion with high degree of freedom :

- high radial distortion made from a polynomial radial distortion and 6 parameters for degree 2 general polynoms; these polynoms are available in Tapas via the `Four7x2`, `Four11x2`, `Four15x2`, `Four19x2` and in Apero with `eModeleRadFour7x2`, ..., `eModeleRadFour19x2`, the `Four19x2` modelize the radial distortion with a polynom  $r_3\rho^3 + \dots + r_{19}\rho^{19}$ ; also I am not sure it maybe necessary to use until  $\rho^{19}$ , I observe that with modern camera using sophisticated aspherical lenses, it may be insufficient to use the so called  $r_3r_5r_7$  model;

9. for example because "new" `Tapas` has slower convergence

10. because it's big data set and the "loop is not closed"

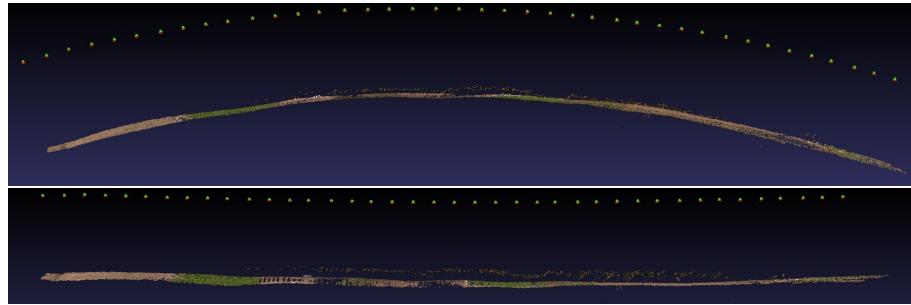


Figure 16.4: Result with old and new version of Tapas, with first convergence is visibly not achieved

- general polynomial model, i.e  $D_N(X, Y) \sum (A_{i,j}x^i y^j, B_{i,j}x^i y^j)$  where  $i + j \leq N$ , there accessible in **Apero** a unified model **eModelePolyDeg2**, ... **eModelePolyDeg7**, for example, **eModelePolyDeg7** has 66 parameters, because  $66 = 8 * 9 - 6$ , the  $-6$  comes because there 6 polynoms already modeled by focal, principal point and rotations;

The first one is accessible directly in **Tapas**, not the second. But both are accessible as additional distortion. In fact, it would be generally a bad idea to try to estimate directly the 66 parameters of a **eModelePolyDeg7**, it's preferable to estimate first a model with physical meaning and few parameters, and then to estimate the high degree polynom as a modification to this physical model. This can be done with :

- **AddFour7x2**, ... **AddFour15x2** for high degree radial models;
- **AddPolyDeg0**, ... **AddPolyDeg7** for high degree general models;

A possible example of use :

```
"mm3d" "NewTapas" "Four15x2" "R.*JPG" "DegGen=2"
"mm3d" "NewTapas" "AddPolyDeg7" "R.*JPG" "InOri=Ori-Four15x2/"
```

The first call is classical, just remark the **DegGen=2** because by default only 1 degree general parameter of **Four15x2** is free. In the second call we start from the first orientation/calibration and add a 7 degree general polynom. Note that, as with **AutoCal** et **Figeé** all the calibration must have an initial value when using the additional mode.

Also note that only the additional distortion will be optimized (else the problem would be far over parameterized). The result in **Ori-AddPolyDeg7/AutoCal60.xml** looks like :

```
<ExportAPER>
  <CalibrationInternConique>
    <KnownConv>eConvApero_DistM2C</KnownConv>
    <PP>389.315403456829813 291.190537211471565</PP>
    <F>653.34645310282724</F>
    <SzIm>800 600</SzIm>
    <CalibDistortion>
      <ModUnif>
        <TypeModele>eModeleRadFour15x2</TypeModele>
        <Params>0.000731887299586956555</Params>
      ....
        <Etats>499.9999999999943</Etats>
        <Etats>399.9999999999943</Etats>
        <Etats>300</Etats>
      </ModUnif>
    </CalibDistortion>
    <CalibDistortion>
      <ModUnif>
        <TypeModele>eModelePolyDeg7</TypeModele>
        <Params>-0.00116060037632004101</Params>
        <Params>0.000376819315728853894</Params>
      ....
        <Params>0.15580413331949222</Params>
        <Etats>652.861598677042821</Etats>
        <Etats>391.129060939043825</Etats>
        <Etats>287.689359319814059</Etats>
      </ModUnif>
    </CalibDistortion>
  </CalibrationInternConique>
</ExportAPER>
```

### 16.6.3 Non Linear Bascule (swing)

#### 16.6.3.1 Motivation

The **GCPBascule** tool, described in 3.10.1.2, transforms a relative orientation into an absolute one, using at least 3 ground control point (GCP). The default use make the assumption that the relative orientation is "perfect" and it computes the minimum number of parameters, i.e. the seven parameter corresponding to the arbitrary 3d similitude that can be computed from tie points :

- 3 parameter for rotation;
- 3 parameter for translation;
- 1 parameter for scale.

When **MicMac** tools is used for metrology, this geo-referencing by **GCPBascule** can be insufficient because there appears non linear distortion in the relative orientation. The discussion about the origin and the quantification of these effect, is quite complex and cannot be discussed here; however, to solve it we have to input more GCP that the minimum required and to use these surplus GCP for correcting this distortion. In **MicMac** tools, there is two way to do it :

- the "classical" way is to do a compensation with high weighting on the GCP, this can be done with the simplified tool **Campari** (3.9.2 and 4.2.4);
- a non standard way is to use the redundancy of the GCP to directly estimate the non linear distortion existing between the result of "Bascule" (swing) ans the ground truth; this is what is described here;

At the time of writing these documentation, it's not clear which method is "better". The first one is more standard and seems more correct from theoretical point of view, however from experimental point of view, the second one seems more accurate.

### 16.6.3.2 Mathematical model

Let :

- let  $G_k$  be the ground coordinate of the GCP;
- let  $I_k$  be the coordinate of the GCP in relative initial model (estimate by bundle intersection);
- let  $T$  be the transformation from relative to absolute we want to estimate similitude, such that  $G_k \approx T(I_k)$  ;
- let  $S$  be the initial similitude estimation of  $T$ ;
- let  $C$  be the "small" correction we want to do compute  $T = (Id + C) \circ S$

Also, it is current that the acquisition is "linear" and that the coordinate must not be treated symmetrically. For example if the acquisition is made from a single strip, let  $O'X'Y'Z'$  be a coordinate system , centered on the acquisition, such that the image center are aligned on the  $O'X'$  axes. Concretely  $O'X'Y'Z'$  can be estimated automatically from initial  $OXYZ$  by elementary computation of inertial axes of the image center (i.e. computation of order 2 moments)<sup>11</sup>.

Basically, the model for estimate  $C$  is restricted quadratic function on of each coordinates  $X'Y'Z'$ :

- $C(X', Y', Z') = (X'^c, Y'^c, Z'^c)$ ;
- $X'^c = \sum c_{ij}^x X'^i Y'^j$
- $Y'^c = \sum c_{ij}^y X'^i Y'^j$
- $Z'^c = \sum c_{ij}^z X'^i Y'^j$

For example, suppose that the acquisition is made from a single strip, and that the errors is only on  $Z^c$ , and that this error depends only on the "main" variable  $X'$ , a possible model could be :

- $X'^c = 0$
- $Y'^c = 0$
- $Z'^c = c_{00}^z + c_{10}^z X' + c_{20}^z X'^2$

Also if we have a sufficient number of control points<sup>12</sup> and have high distortion we can select the full model.

### 16.6.3.3 Using it in **MicMac**

The command **GCPBascule** has several parameters to compute a non linear correction. Of course by default, the swing-bascule if made with the standard 7 parameters with no correction. The non linear correction is activated if the optional **PatNLD** is used. The meaning of the different parameters is then :

- **PatNLD** : define the pattern of the GGP name that will be used to estimate  $C(X', Y', Z')$ ; in the final application probably one will use "**PatNLD=.\***" to specify that all GCP; however, if one want to estimate the accuracy with GCP that are not used for the estimation, it can be convenient to specify a subset;
- **NLDegX**, **NLDegY** and **NLDegZ** specify the monoms that be used for  $X'^c, Y'^c, Z'^c$ , it a vector of strings which elements must belongs to  $\{1, X, Y, X2, XY, Y2\}$ ;
- **NLFR** : as the function  $T$  is no longer a pure similitude, the orientation of each image can no longer be exactly a rotation matrix; the parameter **NLFR** mean "Non Linear For Rotation" and control the export is done ; if :
  - **NLFR** is false, **MicMac** will export for each image the closest matrix that fit with new system, if  $X'^c, Y'^c, Z'^c$  contains non linear term there will be some error but probably very small; the matrix will be non rotation matrix which mean that there will be no longer usable compensation (but usable in matching);

11. this is exactly what is done in **MicMac**

12. 6 is the minimum, but 12 would be more reasonable

- NLFR is true, MicMac will export for each image the closest rotation that fit the new system; of course the price to pay for having true rotation is that the error will be bigger;
- NLShow : give detailed information;

#### 16.6.3.4 Example of use, and message interpretation

Here is a possible use :

```
"mm3d" "GCPBascule" ".*ARW" "Ori-AllRell-F15AddP7/" "Basc-Def-Non0" "GCP.xml" "MesFinal-S2D.xml" \
"PatNLD=(3|7|14).*" "NLDegZ=[1,X,X2]" "NLDegX=[1,X,Y]" "NLDegY=[1,X,Y]" "NLFR=false" "NLShow=true"
```

The message should look like that , first classical message of GCPBascule :

```
BEGIN Pre-compile
NEW CALIB TheKeyCalib_350
NB[10a]= 11
.....
NB[9c]= 11
BEGIN Load Observation
Pack Obs NKS-Set-Orient@-AllRell-F15AddP7 NB 180
BEGIN Init Inconnues
NUM 0 FOR 001aDSC01061.ARW
.....
NUM 179 FOR 242bDSC01005.ARW
BEGIN Compensation
BEGIN AMD
END AMD
```

Then a message remembering the monom used for  $X'^c, Y'^c, Z'^c$  :

```
MQ:X [1 X Y ]
MQ:Y [1 X Y ]
MQ:Z [1 X X2 ]
```

Then the error before and after non linear correction :

```
* 14a ErInit : 0.0723239 => ErCor : 0.0260313 DZ=-0.0254591
* 14b ErInit : 0.043778 => ErCor : 0.0135564 DZ=0.00440725
* 14c ErInit : 0.0277668 => ErCor : 0.0253775 DZ=0.0211511
13a ErInit : 0.0583578 => ErCor : 0.0456215 DZ=-0.042776
13b ErInit : 0.0361378 => ErCor : 0.0231452 DZ=-0.0176777
13c ErInit : 0.0200873 => ErCor : 0.020112 DZ=0.00742163
....
```

Let detail the message for point 14a :

- \* 14a ErInit : 0.0723239 => ErCor : 0.0260313 DZ=-0.0254591;
- the \* means that point 14a belongs to PatNLD
- 0.0723239 is the initial distance  $\|G_k - S(I_k)\|$  (after bascule-swing);
- 0.0260313 is the distance after non linear correction  $\|G_k - T(I_k)\|$  ;
- -0.0254591 is the Z value of  $\|G_k - T(I_k)\|$  (generally the most important);

#### 16.6.4 A detailed example

#### 16.6.5 Miscellaneous options to Tapas

##### 16.6.5.1 FreeCalibInit

Impose that all calibration parameters are freed at the begining. Rarely usefull ...

##### 16.6.5.2 FrozenCalibs

Same role as FrozenPoses but for internal calibration.

### 16.6.5.3 SinglePos

- A pattern (generaly one) of pose and calibs in the form [PatPose,PatCalib]. When specified :
- the RefineAll is set to false;
  - only this pose and calibration will be saved;

## 16.7 GCP : accuracy and optimal weighting

When using GCP mixed with tie points may rise the following difficulties :

- what is the accuracy of the geo-referencing ?
- What is the optimal weighting of the GCP ?

Obviously, the same GCP can be used in the optimisation process and in the accuracy measurement, else the result would be obviously biased and the "optimal" weighting would be  $\infty$ . The safe alternative is to separate the GPC used for optimisation and those used for measuring accuracy. When one has as many GCP as wanted this work perfectly well, but the problems appear when there is few GCP.

For a given weighting, the classical alternative is to proceed for estimating accuracy is to proceed this way :

- parse the GCP and alternatively each GCP is considerer ejected;
- run the bundle without the ejected GCP and memorize the accuracy on this ejected GGP;
- the global accuracy of you bundle can be estimated as the average of the accuracy of the ejected GCP.

The question remain of what is the optimal weighting ? Also there is many theoreticall consideration, my personnal opinion is that the safest way is to do it purely empirically by testing different weighting with the previous method (which I consider as safe). Of course the cost to pay, is computation time ...

This testing can be done by the **MulRTA** option of **Campari**. When used, **MulRTA** is a vector of double it used this way , let  $P_T$  and  $P_I$  be the ground and image accuracy (set by secong and fourth value of optional parameter **GCP**) , for each value  $M$  of **MulRTA**

- set the Ground and image accuracy to  $M * P_T$  and  $M * P_I$
- for each GGP  $G$  :
  - run the bundle without using  $G$  in the measurement;
  - compute the distance  $D^M G$  between ground measurement and bundle measurement;
  - memorise the value  $Ac(M) = \sum_G D^M G$  as an estimator of the accuracy with  $M$  weighting.

The results are stored in a file **SauvRTA.xml**. For example :

```
<XmlResultRTA>
  <BestMult>0.3 </BestMult>
  <BestMoyErr>0.036148</BestMoyErr>
  <RTA>
    <Mult>3</Mult>
    <MoyErr>0.07951</MoyErr>
    <OneAppui>
      <Name>P123</Name>
      <EcartFaiscTerrain>-0.1311860 -0.073748 -0.03087</EcartFaiscTerrain>
      <DistFaiscTerrain>0.1536285</DistFaiscTerrain>
      <EcartImMoy>0.473698</EcartImMoy>
      <EcartImMax>0.79826</EcartImMax>
      <NameImMax>OIS-023.tif</NameImMax>
    </OneAppui>
  ....
  </RTA>
  <RTA>
    <Mult>1</Mult>
    <MoyErr>0.04443</MoyErr>
  ....
  </RTA>
</XmlResultRTA>
```

Also the interpretation should be quite obvious, a brief comment :

- the best accuracy was reached for  $M = 0.3$ , its value was 0.036148;
- for  $M = 3$  , the estimated accuracy estimated was 0.07951;
- For  $M = 3$  , and  $G = P123$ , some detail are memorized :
  - the difference bewteen bundle estimation and ground is in <EcartFaiscTerrain>

- the norm of <EcartFaiscTerrain> is <DistFaiscTerrain>
- the image accuracy is <EcartImMoy> ;
- the worst image seizing of P123 was done on image OIS-023.tif.

## 16.8 Initial Orientation with Martini

The command **Martini**<sup>13</sup> computes initial values of orientation while aiming to solve some ressource issues of **Tapas** on memory and computation time. The principle of **Martini** is :

- compute the relative orientation of pairs and triplet;
- build a global orientation coherent with the constraints given by pairs and triplets;

Althouh the second part is still not fully satisfying, **Martini** can already be used now as it solve sometimes some orientation problem that were not solved correctly by **Tapas** (and generally faster). Also preliminar execution of **Martini** is necessary for some commands as **OriRedTieP**

```
mm3d Martini
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Image Pat}
Named args :
 * [Name=OriCalib] string :: {Orientation for calibration }
 * [Name=Exe] bool :: {Execute commands, def=true (if false, only print)}
 * [Name=SH] string :: {Prefix Homologue , Def=""}
```

The signification of parameters

- first one : standard pattern of images to orientate;
- **OriCalib** when given specify the folders were internal calibration can be found; as **Martini** will not do any adjustment of internal calibration, it is highly recommanded to use this option with a relatively good internal calibration;
- **SH** if **Martini** must be used with a folder of homologous point different from the standard **Homol/**.

The result of **Martini** are stored in an orientation folder **Ori-Martini/** when no **OriCalib**] was set and ,for example, **Ori-MartiniTOTO/** if **Martini** was used with **OriCalib=TOTO**.

## 16.9 Miscellaneous tools about calibration

### 16.9.1 ConvertCalib to Calibration conversion

Sometimes it may be necessary to export a given calibration from one model to another model. For example from Fraser terrestrial model to aerial model. Of course as the mathematicall modelisation of the camera is not the same, this conversion will generally imply some lost of accuracy. The tool **ConvertCalib** allow to do such conversion.

```
mm3d ConvertCalib
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Input Calibration}
 * string :: {Output calibration}
Named args :
 * [Name=NbXY] INT :: {Number of point of the Grid}
 * [Name=NbProf] INT :: {Number of depth}
 * [Name=DRMax] INT :: {Max degree of radial dist (def=depend Output calibration)}
 * [Name=DegGen] INT :: {Max degree of generik polynom (def=depend Output calibration)}
 * [Name=PPFree] bool :: {Principal point free (Def=true)}
 * [Name=CDFree] bool :: {Distorsion center free (def=true)}
 * [Name=FocFree] bool :: {Focal free (def=true)}
 * [Name=DecFree] bool :: {Decentrik free (def=true when applicable)}
```

---

13. MARTingale d'INITialisation

The first argument is a file containing the calibration to be converted. The second is a file that contains a model of the targeted calibration. The folder Documentation/NEW-DATA/DocConvertCalib contains data to test this tool :

- `Aerial.xml` a calibration with aerial model (PPA and PPS separated);
- `Fraser-Affine.xml` a calibration with Fraser model (PPA and PPS merged, decentric distortion);

For example, if we want to produce a conversion to Fraser model, without affine distortion and a one parameter of radial distortion, we can run :

```
mm3d ConvertCalib Aerial.xml Fraser-Affine.xml DRMax=1 DegGen=0
....
===== ERROR MAX PTS FL =====
|| Value=0.145258 for Cam=Fraser-Affine.xml and Pt=Pt_0_0_1 ; MoyErr=0.0623394
=====
--- End Iter 16 ETAPE 0
```

The average accuracy will be 0.063 pixel, it is measured as the average error re-projection of synthetic 3d points.

### 16.9.2 Genepi to generate artificial perfect 2D-3D points

It generates a set of 3D points and their images projection for a given camera.

This tool was used for internal checking. Not sure it will be used very often, maybe sometime to export MicMac's orientation in other format when no better solution is available or also useful when preparing data sets for students.

```
mm3d Genepi _MG_008.*.CR2 Ori-AllFix/ -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Image}
 * string :: {Orientation}
Named args :
 * [Name=NbXY] INT :: {Number of point of the Grid}
 * [Name=NbProf] INT :: {Number of depth}
```

The first parameters define the pattern of images and the second the orientation. The optional parameters control the density of points.

### 16.9.3 Init11P, space resection for uncalibrated camera

The space resection computes the pose of a camera from a set of 3d point and their corresponding image projection. The `Init11P` deals with the case where the calibration is unknown. In this case the following parameters are :

- center of camera (3 parameters);
- orientation of camera (3 parameters);
- focal and principal point (3 parameters);
- skew and ratio  $\frac{x}{y}$  (2 parameters).

At least 6 projections are required to compute these 11 parameters.

```
mm3d Init11P
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Name File for GCP}
 * string :: {Name File for Image Measures}
Named args :
 * [Name=FM] bool :: {Fraser Mode, use all affine parameters (def=false)}
 * [Name=Filter] string :: {Filter for Image (Def=.*)}
```

The mandatory parameters are :

- the file for GCP, containing a `DicoAppuisFlottant` structure ;
- the file for image measurement containing a `SetOfMesureAppuisFlottants` structure.

For all the images contained in the image file an orientation is created in the folder `Ori-11Param/`. If the optional FM is set to true, the 11 parameters are exported as a Fraser camera. If it is set to false, the skew and xy ratio are forced to 0 and 1, and the result is exported as a radial camera (with no distorsion); however, 6 are required as forcing skew to 0 and xy ratio to 1 is done *a posteriori*.

The folder `Ori-11ParamComp/` also contains the result of a compensation done using the previous value. They may be slighlty different and theoretically more accurate, specifically when the `FM=false` option.

#### 16.9.4 Aspro, space resection for calibrated camera

The tool `Aspro` allows to orientate a set of images with known internal calibration from existing 3d points and their corresponding image projection.

```
mm3d Aspro -help
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Name File for images}
 * string :: {Name File for input calibration}
 * string :: {Name File for GCP}
 * string :: {Name File for Image Measures}
```

The parameters should be quite intuitive, an example of use :

```
mm3d Aspro "_MG_008[0-3].CR2" Ori-AllFix/ TestOPA-S3D.xml TestOPA-S2D.xml
```

The resulting orientation are stored in `Ori-Aspro/`. Note that `MicMac` must be abble to find the internal calibration with name respecting its convention in the folder of calibration (`Ori-AllFix/` here). As the naming of `MicMac` may be not obviuous, the tool described in 11.3.3 can be useful.

## 16.10 Rigid Block Compensation

### 16.10.1 Introduction

#### 16.10.1.1 Mathematics

These functionnalities are used when the camera form a rigid block, which mean that the relative position of a set of camera do not change during time (or changes are small, whatever it means).

Supose we have a sets of camera  $Cam_A, Cam_B, Cam_C \dots$ . Let  $P_{A,k}$  be the pose of image acquired by camera A at time k (idem  $P_{B,k} \dots$ ). The rigidity hypothesis means that :

$$\forall k, k', \alpha, \beta : P_{\alpha,k}^{-1} P_{\beta,k} = P_{\alpha,k'}^{-1} P_{\beta,k'} \quad (16.3)$$

If we decompose the pose  $P_{A,k} = (C_{A,k}, R_{A,k})$ ,  $C_{A,k}$  being the center and  $R_{A,k}$  the rotation matrix, equation 16.3 writes :

$$0 = R_{\alpha,k}^{-1} R_{\beta,k} - R_{\alpha,k'}^{-1} R_{\beta,k'} = \Delta_{\omega}(\alpha, \beta, k, k') \quad (16.4)$$

$$0 = (-C_{\alpha,k} + R_{\alpha,k}^{-1} * C_{\beta,k}) - (-C_{\alpha,k'} + R_{\alpha,k'}^{-1} * C_{\beta,k'}) = \Delta_{Tr}(\alpha, \beta, k, k') \quad (16.5)$$

Sometime it may be convenient to introduce the, may it be known or unknown, the global pose calibration of the block  $Q_A, Q_B \dots$ ; the equations 16.4, 16.5 become with  $Q_A = (c_A, r_A), Q_B = \dots$ :

$$0 = R_{\alpha,k}^{-1} R_{\beta,k} - r_{\alpha}^{-1} r_{\beta} = \Delta_{\omega}^g(\alpha, \beta, k) \quad (16.6)$$

$$0 = (-C_{\alpha,k} + R_{\alpha,k}^{-1} * C_{\beta,k}) - (-c_{\alpha} + r_{\alpha}^{-1} * c_{\beta}) = \Delta_{Tr}^g(\alpha, \beta, k) \quad (16.7)$$



Figure 16.5: The amateur Fuji stereo camera used for creating the rigid data set

### 16.10.1.2 Data set

The data set to illustrate are located in the folder Documentation/NEW-DATA/Rigid-Block it was acquired using a Fuji stereo camera as the one shown on figure 16.5. The data set contains :

- three files to MPO format, this the format used by Fuji to store pair of images of the stereo camera;
- the two files containing measurement of GCP (AllGCP-RTL.xml and MesuresFinales-S2D.xml)
- the folder Ori-Calib/ containing calibration of the images; in fact to limit the size, the dataset is limited to three images and it would not have been possible to do any valuable self calibration;
- the "classical" file MicMac-LocalChantierDescripteur.xml which contain will be detailed later;
- a file Cmd.txt containg the command that can be batched.

### 16.10.1.3 Preprocessing and camera naming

Each MPO files contains 2 jpg images, the first command extract these images :

```
mm3d SplitMPO DSCF329.*MPO
ls DSCF329*.jpg
DSCF3297_L.jpg DSCF3298_L.jpg DSCF3299_L.jpg
DSCF3297_R.jpg DSCF3298_R.jpg DSCF3299_R.jpg
```

All the images DSCF...\_L.jpg correspond the left camera and DSCF...\_R.jpg to the right. As left and right images correspond to different camera, it is important that MicMac create and recognize different calibration files. By default, the images having the same focal and same camera model indicated in the xif, there may be some conflict. To avoid this, it is possible to define a user specific identifier that will be added to the camera name, this is done by redifing the key NKS-Assoc-StdIdAdditionnelCam:

```
<KeyedNamesAssociations>
  <Calcs>
    <Arrite> 1 1 </Arrite>
    <Direct>
      <PatternTransform> DSCF([0-9]{4})_(.)\.(.).jpg </PatternTransform>
      <CalcName> Fuji-$2 </CalcName>
    </Direct>
  </Calcs>
  <Key> NKS-Assoc-StdIdAdditionnelCam </Key>
</KeyedNamesAssociations>
```

With this key, we assure that right and left camera will have a different identifier (and also, conversely, that this identifier do not depend of image number). We can test this by using the command TestNameCalib :

```
mm3d TestNameCalib DSCF3297_L.jpg
./Ori-TestNameCalib/AutoCal_Foc-6300_Cam-FinePix_REAL_3D_W1Fuji-L.xml
```

```
mm3d TestNameCalib DSCF3297_R.jpg
./Ori-TestNameCalib/AutoCal_Foc-6300_Cam-FinePix_REAL_3D_W1Fuji-R.xml
```

#### 16.10.1.4 Standard MicMac Processing

The first three command are standard micmac processing :

```
Tapioca All D.*jpg 1500
Tapas Figeo D.*jpg InCal=Calib Out=AllRel
GCPBascule D.*jpg AllRel Basc AllGCP-RTL.xml MesuresFinales-S2D.xml
```

As usual :

- Tapioca to compute tie point;
- Tapas to compute the relative orientation, as said before with only 3 images we use an existing internal calibration (InCal=Calib) and maintain it frozen to its initial value (Figeo);
- then we transfer to an absolute geo reference system with GCPBascule;

#### 16.10.2 Indicating block structure

To understand the block structure and use equations like 16.3 ... 16.7, MicMac will need to compute from the name of images to which camera it belongs to and what were the images aquired during the same time. This is done by creating a single reversible key that will return 2 values, here :

```
<KeyedNamesAssociations>
  <IsParametrized> true </IsParametrized>
  <Calcs>
    <Arrite> 2 1 </Arrite>
    <Direct>
      <PatternTransform> DSCF([0-9]{4})_(.)\..jpg </PatternTransform>
      <CalcName> $1 </CalcName>
      <CalcName> $2 </CalcName>
    </Direct>
    <Inverse>
      <PatternTransform> (.*)%(.*) </PatternTransform>
      <CalcName> DSCF$1_$2.jpg </CalcName>
      <Separateur> % </Separateur>
    </Inverse>
  </Calcs>
  <Key> Loc-Assoc-Im2Block </Key>
</KeyedNamesAssociations>
```

The first values correspond to  $k, k'$  of equation 16.3 ... while the second correspond to the  $A, B \dots$ . Also it shoul be pretty obvious, here are some example of result of this key :

- DSCF3297\_R.jpg  $\Rightarrow 3297 \times R$ ;
- DSCF3297\_L.jpg  $\Rightarrow 3297 \times L$ ;
- DSCF3298\_R.jpg  $\Rightarrow 3298 \times R$ ;
- ...

#### 16.10.3 Block estimation

The first step is to estimate the, supposed fixe, position of camera relatively to each others. For this we need to estimate the pose  $Q_A, Q_B \dots$ . As obviously, everything is undeterminade up to a global roto-translation, we have to fix an arbitraty constraint, for example  $Q_A = Id$ , where  $A$  is the first camera (MicMac use alphabetic order to select this arbitraty reference). Then with this constraint, using equation 16.6 and 16.7 , for any camera  $\beta$ , knowing the pose at one time  $k$  is sufficient to estimate  $r_\beta$  and  $c_\beta$ . Generally we have several  $k$ , and we can estimate a more accurate valure by simply averaging the estimation.

The command Blinis does this computation :

```
mm3d Blinis
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
```

```
* string :: {Full name (Dir+Pat)}
* string :: {Orientation in}
* string :: {Key for computing bloc structure}
* string :: {File for destination}
```

The meaning shoud be quite obvious, and an example of use :

```
mm3d Blinis DSCF329.*jpg Ori-Basc/ Loc-Assoc-Im2Block Blinis.xml
```

```
....  
=====  
EstimCurOri DSCF3297_L.jpg DSCF3297_L.jpg  
[0,0,0] -6.93889e-18 6.67869e-17 9.26854e-34  
EstimCurOri DSCF3298_L.jpg DSCF3298_L.jpg  
[0,0,0] 9.54098e-18 -1.06685e-16 3.19245e-33  
EstimCurOri DSCF3299_L.jpg DSCF3299_L.jpg  
[0,0,0] 1.38778e-17 -2.77556e-17 -5.55112e-17  
===== AVERAGE =====  
[0,0,0] tetas 5.49329e-18 -2.25514e-17 -1.85037e-17  
DispTr=0 DispMat=6.21513e-17  
=====  
EstimCurOri DSCF3297_L.jpg DSCF3297_R.jpg  
[0.0772782,0.00105963,4.59993e-05] 0.00587422 0.041968 0.00953594  
EstimCurOri DSCF3298_L.jpg DSCF3298_R.jpg  
[0.0775943,0.00110819,0.000728747] 0.00590563 0.04163 0.00881231  
EstimCurOri DSCF3299_L.jpg DSCF3299_R.jpg  
[0.0775121,0.00141723,-0.00103102] 0.00589855 0.0417358 0.00904956  
===== AVERAGE =====  
[0.0774615,0.00119502,-8.54263e-05] tetas 0.00589282 0.041778 0.0091326  
DispTr=0.000688418 DispMat=0.000271402  
--- End Iter 1 ETAPE 0
```

MicMac print for each time  $k$  and each camera  $\beta$  the value estimated of  $r_\beta$  and  $c_\beta$ . These value will be rarely usefull, also we can observe that for the first camera we have almost  $c_A = [0, 0, 0]$  and  $r_A = Id$  (the angle are in fact printed). The average value is also computed. In fact the only value printed of real insterest will generally be the dispersion : `DispTr` and `DispMat`.

The interesting result of this command rely in the file created , here `Blinis.xml`. It contains in a `xml` specification, that should be pretty obvious, the value of the estimated block calibration. In our example the file contains :

```
<StructBlockCam>  
  <KeyIm2TimeCam>Loc-Assoc-Im2Block</KeyIm2TimeCam>  
  <LiaisonsSHC>  
    <ParamOrientSHC>  
      <IdGrp>L</IdGrp>  
      <Vecteur>0 0 0</Vecteur>  
      <Rot>  
        <L1>1 -5.49329100725988965e-18 2.25514051876984922e-17</L1>  
        <L2>5.49329100725988888e-18 1 1.85037170770859382e-17</L2>  
        <L3>-2.25514051876984922e-17 -1.85037170770859382e-17 1</L3>  
        <TrueRot>true</TrueRot>  
      </Rot>  
    </ParamOrientSHC>  
    <ParamOrientSHC>  
      <IdGrp>R</IdGrp>  
      <Vecteur>0.0774615118759180987 0.00119501688228340619 -8.54262594655412071e-05</Vecteur>  
      <Rot>  
        <L1>0.999110080792062982 -0.00627395825477376507 -0.0417095181882368854</L1>  
        <L2>0.00588764369540181742 0.999938688556114119 -0.0093784209968844328</L2>  
        <L3>0.0417658007392831265 0.00912450417809996389 0.999085762741172445</L3>  
        <TrueRot>true</TrueRot>  
      </Rot>
```

```

</ParamOrientSHC>
</LiaisonsSHC>
</StructBlockCam>
```

These value are coherent with the specification of the fuji camera; the  $L$  camera being arbitrary set in identity position, we analyse the  $R$  camera :

- the base is parallel to the  $X$  axes;
- the lenght of the base is 7.7cm (approximatively the spacing between eyes of adult human);
- the two camera are approximatively parralel, with a slight convergence such that the maximal overlap occurs at 2meters.

### 16.10.4 Block compensation

The **Blinis** command does an *a posteriori* estimation of the rigid calibration, that is usefull to compute a reasonable initial value, but it does not do any compensation. The compensation can be do in **Campari** using several optional parameters.

#### 16.10.4.1 Global with no attachment to known value

The first case correspond to the following hypothesis :

- each block of camera is closed to the same value  $Q_A, Q_B \dots$ ;
- we don't have any good estimation of these value;
- how close the camera are equals to these common value is given by the a priori variance that we have to indicate  $\sigma_{Tr}^g$  and  $\sigma_\omega^g$

Then we will add to global minimization of the bundle, the term  $E(\sigma_{Tr}^g, \sigma_\omega^g)$  given by the following equation :

$$E^g(\sigma_{Tr}^g, \sigma_\omega^g) = \sum_{k,\beta} \left( \frac{\Delta_\omega^g(A, \beta, k)}{\sigma_\omega^g} \right)^2 + \left( \frac{\Delta_{Tr}^g(A, \beta, k)}{\sigma_{Tr}^g} \right)^2 \quad (16.8)$$

Note that the term is dissymetric as the first camera play a special role. This may evolve in future versions (at least optionnaly). The option **BlocGlob** of **Campari** allow to add such term to the bundle.

```
mm3d Campari -help
...
* [Name=BlocGlob] vector<std::string> :: {Param for Glob bloc compute [File,SigmaCenter,SigmaRot,?MulFinal]
* [Name=OptBlocG] vector<std::string> :: {[SigmaTr,SigmaRot]}
* [Name=BlocTimeRel] vector<std::string> :: {Param for Time Reliative bloc compute [File,SigmaCenter,Sigma}
```

The parameter **BlocGlob** is a vector that contain 3 mandatory values and 2 optionnal :

- parameter  $P_1$  is the name of calibration as created by the **Blinis** command (or also by **Campari**);
- parameter  $P_2$  and  $P_3$  correspond respectively to  $\sigma_{Tr}$  and  $\sigma_\omega$
- parameter  $P_4$  will be explained just after (default value is 1.0);
- parameter  $P_5$  is the file containing the output of new estimaded value of block calibration, its defaut value is **Out-P<sub>1</sub>**;

The role of the parameter  $P_4$  is to allow the  $\sigma_{Tr}$  and  $\sigma_\omega$  to evolve during the different iteration of the bundle. It may happen that we know that "at the end" the rigidity block is very strict, but as we are not so well initialized, we can impose it strictly at the first iteration. So the value of  $\sigma_{Tr}$  and  $\sigma_\omega$  will be :

- $P_2$  and  $P_3$  for the first iteration;
- $P_2 * P_4$  and  $P_3 * P_4$  for the last iteration;
- in between they will evolve folowing a geometric law.
- note that if we want to slow down the evolution we can increase the number of iteration with the **NbIterEnd** parameter;

Here is an example with the data set:

```
Campari "DSCF32.*jpg" Basc Cmp GCP=[AllGCP-RTL.xml,0.3,MesuresFinales-S2D.xml,0.3] BlocGlob=[Blinis.xml,1e-3
| | Residual = 0.721748
...
| | Residual = 1.0723 ;; Evol, Moy=0.00715547 ,Max=9.87406
```

```

...
=====
===== AVERAGE =====
[0,0,0] tetas 5.49329e-18 -2.25514e-17 -1.85037e-17
DispTr=2.09346e-16 DispMat=5.38957e-17
=====
===== AVERAGE =====
[0.0765272,0.00148802,-0.00165705] tetas 0.00583192 0.0416507 0.00916005
DispTr=7.49798e-10 DispMat=1.22368e-07

```

Note that here we have imposed a very stric constraint on the rigidity, the final sigma being  $1e - 7$  for translation and  $1e - 6$  for rotation. As probably this amateur camera is not completely rigid, this explain why the residual have grown from 0.72 tp 1.07.

#### 16.10.4.2 Global with attachment to known value

Sometime we want to impose that the value  $r_\beta$  and  $c_\beta$  stay close to their initial values  $r_\beta^0$  and  $c_\beta^0$ . How close the camera are equals to these common value is given by the a priori variance that we have to indicate  $\sigma_{Tr}^0$  and  $\sigma_\omega^0$ . We the add to the bundle minimization the term :

$$E^0(\sigma_{Tr}^0, \sigma_\omega^0) = \sum_{\beta} \left( \frac{(r_\beta^0 - r_\beta)}{\sigma_\omega^0} \right)^2 + \left( \frac{(c_\beta^0 - c_\beta)}{\sigma_{Tr}^0} \right)^2 \quad (16.9)$$

The attachment to inial values can be specified by the `OptBlocG` parameter :

- it contains to value ;
- $P_1$  is  $\sigma_{Tr}^0$  and  $P_2$  is  $\sigma_\omega^0$
- if these parameter both value  $-1$ , then  $r_\beta$  and  $c_\beta$  are strictly contraints to be equal to  $\sigma_{Tr}^0$  and  $\sigma_\omega^0$ ;
- if these parameter both value  $-2$ , then they are not used (may seem stupid, prepare new options).

#### 16.10.4.3 Time relative

Some time, it may happen that the block is rigid, but with a shape that evolve smoothly accross time : the block  $k$  is close to block  $k+1$  even if the final blocks can be very far from the initial one. How close the consecutive block are is given by the a priori variance  $\sigma_\omega^t$  and  $\sigma_{Tr}^t$ . This can be modelized mathematically by adding a term

$$E^t(\sigma_{Tr}^t, \sigma_\omega^t) = \sum_{k,\beta} \left( \frac{\Delta_\omega(A, \beta, k, k+1)}{\sigma_\omega^t} \right)^2 + \left( \frac{\Delta_{Tr}(A, \beta, k, k+1)}{\sigma_{Tr}^t} \right)^2 \quad (16.10)$$

This term can be add with the `BlocTimeRel` parameter, the syntax being exactly the same that with `BlocGlob`.

#### 16.10.4.4 Combination

When pertinent the option `BlocTimeRel` and `BlocGlob` can be used simultaneously, for example to modelize a global evolution linked to stay close to an initial known value.

# Chapter 17

## Advanced matching, theoretical aspect

This chapter presents some theoretical point and mathematical notation useful for a more detailed presentation of MicMac's parameters.

### 17.1 Generalities

#### 17.1.1 Geometric notations

Generally, we have  $N$  images to match, they are noted  $Im_k(u, v)_{k \in [1, N]}$ . We use the notations:

- $\mathcal{T}$ , with  $\mathcal{T} = \mathbb{R}^2$ , the "ground" space, this is the space in which the depth is computed; note that according to the selected restitution geometry, this space can be the euclidean space, or the "master" image space;
- $\mathcal{E}_{px}$ , with  $\mathcal{E}_{px} = \mathbb{R}$  or  $\mathcal{E}_{px} = \mathbb{R}^2$ , the "disparity" space<sup>1</sup>; we note  $D_{px}$  the dimension of disparity space;
- $\mathcal{I}_k$  the space of  $k^{th}$  image;

Internally, for MicMac, the geometry of images is manipulated using a function  $\pi_k$  for each image:

$$\pi_k : \mathcal{T} \otimes \mathcal{E}_{px} \rightarrow \mathcal{I}_k \quad (17.1)$$

$$(x, y, p_x) \xrightarrow{\pi_k} (u, v) \quad (17.2)$$

With  $p_x = z$  when  $\mathcal{E}_{px} = \mathbb{R}$  and  $p_x = (p_{x_1}, p_{x_2})$  when  $\mathcal{E}_{px} = \mathbb{R}^2$ . For a given  $p_x$ , the function  $\pi_k$ , considered as function from  $\mathcal{T}$  into  $\mathcal{I}_k$ , are one to one mapping, and we write  $\pi_k^{-1}$  the inverse function defined by :

$$\pi_k^{-1} : \mathcal{I}_k \otimes \mathcal{E}_{px} \rightarrow \mathcal{T} \quad (17.3)$$

$$\pi_k^{-1}(\pi_k(x, y, p_x), p_x) = (x, y) \quad (17.4)$$

The result of the matching process is a function  $\mathcal{F}_{px}$  from  $\mathcal{T}$  to  $\mathcal{E}_{px}$ :

$$\mathcal{F}_{px} : \mathcal{T} \rightarrow \mathcal{E}_{px} \quad (17.5)$$

We note  $\mathcal{F}_{px}^d, d \in [1, D_{px}]$  the components of  $\mathcal{F}_{px}$ .

#### 17.1.2 Notation for quantification

In most cases, MicMac works by discretization of spaces  $\mathcal{T}$  and  $\mathcal{E}_{px}$ (this is not the case for variational approaches, still undeveloped in MicMac). At each stage of muti resolution process (see 21.2), steps of quantification must be defined:  $\Delta^{xy}$  for  $\mathcal{T}$ , and  $\Delta_k^{px}, k \in [1, D_{px}]$  for  $\mathcal{E}_{px}$ .

For a given stage, a resolution step  $\Delta^i$  of image space is also selected ; in most cases  $\Delta^i$  is chosen to be equivalent to  $\Delta^{xy}$  (but there can be exceptions). For each used  $\Delta^i$ , MicMac computes images down sampled of a size  $\Delta^i$ , we note  $Im_{ki}^{\Delta^i}$  these images and  $\mathcal{I}_k^{\Delta^i}$  their associated spaces.

For a given quantification, we define a discrete version of  $\pi_k$  by ;

$$p_k : \mathbb{Z}^2 \otimes \mathbb{Z}^{D_{px}} \rightarrow \mathcal{I}_k^{\Delta^i} \quad (17.6)$$

---

1. according to restitution geometry , it can be "real" disparity, depth of euclidean  $Z$

$$p_k(i, j, u, v) = \frac{\pi_k(i * \Delta^{xy}, j * \Delta^{xy}, u * \Delta_1^{px}, v * \Delta_2^{px})}{\Delta^i} \quad (17.7)$$

For given steps  $\Delta^{xy}$  and  $\Delta_k^{px}$ , the objective of the matching process is to compute a table  $F_{px}$ , from  $\mathbb{Z}^2$  to  $\mathbb{Z}^{D_{px}}$ , which is a discrete version of  $\mathcal{F}_{px}$ :

$$F_{px}(i, j)^k = \frac{\mathcal{F}_{px}(i * \Delta^{xy}, j * \Delta^{xy})}{\Delta_k^{px}} \quad (17.8)$$

## 17.2 Energetic formulation and regularization

### 17.2.1 Generalities

The objective of the matching process is to compute a function  $\mathcal{F}_{px}$ , respecting some *a priori* constraints<sup>2</sup> and such that the  $Im_k(\pi_k(x, y, \mathcal{F}_{px}(x, y))), k \in [1, N]$  are similar.

Let's write :

- $\mathcal{A}(x, y, p_x) \geq 0$  a criteria measuring the local similarity between  $Im_k(\pi_k(x, y), p_x)_{k \in [1, N]}$ , with  $\mathcal{A} = 0$  when image are perfectly identical ( $\mathcal{A}$  is the image term);
- $\nabla(\mathcal{F}_{px})$  the gradient of  $\mathcal{F}_{px}$ ,  $\nabla(U) = (\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y})$
- $\|\nabla(\mathcal{F}_{px})\|^{reg}$  a norm on the gradient, which is used as a regularity criteria (it penalizes the variation of  $\mathcal{F}_{px}$ );

In energetic formulation, one tries to minimize a global energetic function  $\mathcal{E}(\mathcal{F}_{px})$  :

$$\mathcal{E}(\mathcal{F}_{px}) = \iint_{\mathcal{T}} \mathcal{A}(x, y, \mathcal{F}_{px}(x, y)) + \|\nabla(\mathcal{F}_{px})\|^{reg} \quad (17.9)$$

By default, the cost function used by MicMac are  $L_1$  :

$$\|\nabla(\mathcal{F}_{px})\|^{reg} = \alpha_1 * |\nabla(\mathcal{F}_{px}^1)| + \alpha_2 * |\nabla(\mathcal{F}_{px}^2)| \quad (17.10)$$

Note that in this equation  $\alpha_1$  is the regularization on first component of disparity and  $\alpha_2$  the regularization on the second component (usable only when  $D_{px} = 2$ ). Typically we will have :

- $\alpha_1 \approx \alpha_2$  for geometry "really" bi-dimensional (for example when matching is used for isotropic deformation measurement);
- $\alpha_1 \ll \alpha_2$  for geometry where  $\mathcal{F}_{px}^2$  is used to correct the default of geometric models (  $\mathcal{F}_{px}^2$  modelize the transverse disparity);

MicMac offers (too ?) many option for optimization:

- options on the cost function;
- options on the disparity dimension;
- options on the choice of the algorithm that will be used for optimization of the function  $\mathcal{E}$ ;

However there are several restrictions in the combination of these options. The following table sums up the main restrictions between the choice of algorithm and options:

Type	Quantification	Speed	Cost func	Disp Dim	D Opt	Optim
Prog Dyn	Oui	+++	Any	1,2	1	approximate
Cox-Roy	Oui	++	$L_1$	1	1,2	exact minimum
Differential	Non	+	$L_2$	1,2	1,2	local descent
Dequant	?	++++	?	1,2	1,2	post filtering
MaxOfScore	Oui	++++	?	1,2	1,2	?

Let's comment some columns:

- **Cost func** describes which cost functions are available for each algorithm; this refers to MicMac implementation and not theoretical possibility (for example max flow allows any convex function);
- **Disp Dim** indicates that the algorithm can handle 2 dimensional disparity;
- **Disp Opt** indicates that the algorithm makes the optimization in two dimensions;

Let's comment some line :

- **Cox-Roy** Cox-Roy is the Cox and Roy implementation of Max Flow;
- **Differential** are no longer available now (but they may be added again).

---

2. regularity

# Chapter 18

## Advanced matching, practical aspect

### 18.1 Cost function

The discrete version of the cost function is:

$$E(Z) = \sum Corr(i, j, Z(i, j)) + F(|Z(i+1, j) - Z(i, j)|) + \dots \quad (18.1)$$

The  $F$  function allows to control the a priori on the  $Z$ :

- if the desired model is smooth, a convex  $F$  can be adequate (it's better to climb a given jump by regular step);
- if the desired model has many discontinuities, a  $F$  concave can be adequate (it's better to climb a given jump in one single step);
- when there is no strong a prior, the default choice is to have  $F$  linear;

The basic form of  $F$  has two parameters  $R$  and  $Q$  :

$$F(\Delta_Z) = R|\Delta_Z| + Q\Delta_Z^2 \quad (18.2)$$

This allows to create linear and convex function. To create concavity, there exists parameters  $S$  and  $A$  such that:

$$F(\Delta_Z) = R|\Delta_Z| + Q\Delta_Z^2, |\Delta_Z| < S \quad (18.3)$$

$$F(\Delta_Z) = RS + RA(|\Delta_Z| - S) + Q\Delta_Z^2, |\Delta_Z| \geq S \quad (18.4)$$

Typically this means that when  $Z$  is over the threshold  $S$ , the slope is multiplied by  $A$ .

- **ZRegul** add a linear term  $ZRegul|\Delta_Z|$  to  $F$ ;

The MicMac tag for this parameters are:

- **<ZRegul>** for  $R$ ;
- **<ZRegul\_Quad>** for  $Q$ ;
- **<SeuilAttenZRegul>** for  $S$ ;
- **<AttenRelatifSeuilZ>** for  $A$ ;

### 18.2 Exporting the score

#### 18.2.1 Default behaviour

Malt generate an image which for each pixel  $i, j$  contains  $Corr(i, j, Z_{Opt}(i, j))$  where  $Z_{Opt}(i, j)$  is the solution computed by Malt.

Internally these value are normalized between  $-1$  and  $1$ <sup>1</sup> and they are resampled between  $0$  and  $255$ . Visualized as black and white images, the white correspond to "good" matches.

The name of the file containing this correlation maps are Correl\_STD-MALT\_Num\_XXX.tif In MicMac , it is possible to generate or not this images with the tag **<GenImagesCorrel>**.

---

1. which is "natural" when it is the centered correlation coefficient

### 18.2.2 Exporting the correlation cube

Sometime the user may want to know  $\text{Corr}(i, j, k)$  not only for  $k = Z^{\text{Opt}}(i, j)$  for all the computed value. This is possible using **GCC** with **Malt** and **GenCubeCorrel** in **MicMac**. The structure of the data is a bit more comple as, due to the multi scale approach of **MicMac**,  $\text{Corr}(i, j, k)$  is not computed for all  $k$  but only for  $k \in [Z^{\text{Min}}(i, j), Z^{\text{Max}}(i, j)]$ ; also due to the parallelization in independant process, the export is done independantly for each tile. The structure is the following :

- for each step  $k$  of the matching a folder **Cubek** is created;
- for each tile :
  - let  $X, Y$  be the origin of the tile (pixel  $(0, 0)$  of this tile will correspond to  $X, Y$  of the the global data);
  - two files **Data\_X\_Y\_ZMin.tif** and **Data\_X\_Y\_ZMax.tif** are created, these files are 16 bits signed tiff images, and contain the  $Z^{\text{Min}}(i, j)$  and  $Z^{\text{Max}}(i, j)$ ;
  - a file **Data\_X\_Y\_Cube.dat** is created, it is a raw file that contains on after each other the  $\text{Corr}(i, j, k)$ , normalized between 0 and 255 , and store on 8 bits unsigned int; the order of storage is :
    - $\text{Corr}(0, 0, Z^{\text{Min}}(0, 0)) \text{Corr}(0, 0, Z^{\text{Min}}(0, 0) + 1) \dots \text{Corr}(0, 0, Z^{\text{Max}}(0, 0) - 1)$
    - $\text{Corr}(1, 0, Z^{\text{Min}}(1, 0)) \text{Corr}(1, 0, Z^{\text{Min}}(1, 0) + 1) \dots \text{Corr}(1, 0, Z^{\text{Max}}(1, 0) - 1)$
    - ...

For user interested, the code generating the data can be found in **src/uti\_phgrm/MICMAC/cSurfaceOptimiseur.cpp**, looking for the variable **mCubeCorrel**.

# Chapter 19

## Using satellite images

The following chapter overviews the processing of satellite images data using **MicMac** tool. The content is structured into several sections,

- Section 19.1 regarding the processing of images delivered with the rational polynomial coefficients (RPCs),
- Section 19.2 regarding images that hold the GRID/RTO orientation data,
- Section 19.3 regarding the resampling of satellite imagery to epipolar geometry, and
- Section 19.4 regarding correlation of satellite images for the purpose of change detection.

### 19.1 With approximate sensor orientation – RPC bundle adjustment (recommended)

This section presents the processing chain on pushbroom sensor images when input orientation is provided in form of the RPCs. Figure 19.1 depicts the DSM generation processing workflow. The input format of the RPCs should comply with the following: Dimap v2 (tested), DigitalGlobe (tested), IKONOS/ASCII (not fully tested). Bundle adjustment observations must include tie points and optionally may include GCP data.

Having converted the RPC files to the **MicMac**-format files (see Subsection 19.1.1), the refinement of the orientation parameters is done within the **Campari** tool (2a in Figure 19.1; see also Subsection 3.9.2.2). In order to validate the internal accuracy of the retrieved new orientation parameters, the user can conduct matching in the direction perpendicular to the epipolar curve with **MMTestOrient**.

If the image pair epipolar geometry is of interest, be it for external uses or to perform the matching on, the **CreateEpip** tool will resample the images so that their corresponding image points are found in the same image rows.

The dense matching is launched through the simplified **Malt** tool. If matching in epipolar geometry shall be performed, use **mm3d MICMAC** with a suitable XML file (see a template XML file in `include/XML_MicMac/MM-Epip.xml`).

The reader is encouraged to follow a use case example included in Section 4.5.

#### 19.1.1 The RPC conversion to **MicMac**-format files

The RPC orientation parameters provided by different vendors have different file formats. Due do that, prior to the actual processing **MicMac** requires executing a data conversion step – **Convert2GenBundle**. As **MicMac** works in uniform units, the initial RPCs defined in geodetic coordinates are recomputed in a metric coordinate system specified by the user.

Typing **mm3d Convert2GenBundle**, one gets:

```
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {Name of Image}
 * string :: {Name of input Orientation File}
 * string :: {Directory of output Orientation (MyDir -> Oi-MyDir)}
Named args :
 * [Name=ChSys] string :: {Change coordinate file (MicMac XML convention)}
```

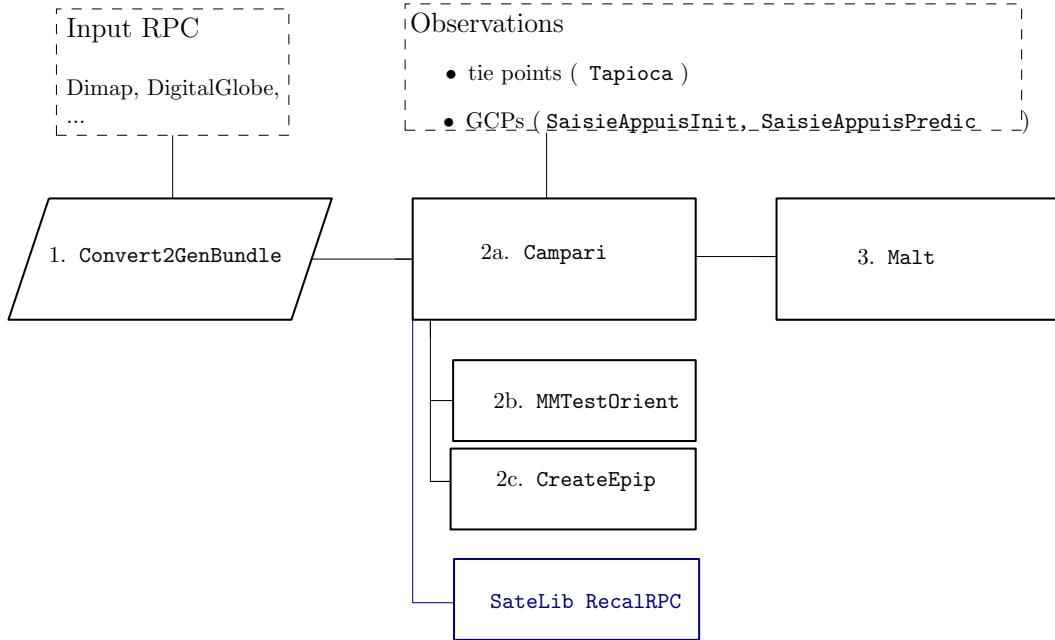


Figure 19.1: Satellite images processing workflow for DSM generation. The `MMTestOrient` is a supplementary operation aiming at evaluating the goodness of the orientation parameters. The `CreateEpip` is obligatory only in case the user wants to perform dense matching in this geometry. `SateLib RecalRPC` recalculates the original RPCs to include the adjusted corrections.

\* [Name=Degré] INT :: {Degré of polynomial correction (Def=2)}

The meaning of optional args is:

- `ChSys`, indicates the coordinate system for the processing; the files must be edited according to the `SystemeCoord` type, and contain a single `BSC` (see below),
- `Degré`, indicates the degree of the 2D polynomial adopted for orientation refinement,

Below is an example of the coordinate system file. It uses the UTM coordinate system defined over the 31. zone of the northern hemisphere, in the WGS84 datum:

```

<SystemeCoord>
    <BSC>
        <TypeCoord>eTC_Proj4</TypeCoord>
        <AuxStr>+proj=utm +zone=31 +ellps=WGS84 +datum=WGS84 +units=m +no_defs</AuxStr>
    </BSC>
</SystemeCoord>

```

## 19.2 With approximate or refined sensor orientation – the GRID/RTO processing

This section presents some tools for processing satellite images that are adapted to work on the localization GRIDs. Hence, if the input orientation is known in form of rational polynomial functions, the library will need to convert it to a localization GRID, or better, switch to Section 19.1 to work directly on the RPCs. A sub-library for importing satellite meta-data from diverse data providers is being developed and is accessible through `mm3d SateLib`.

### 19.2.1 Pleiades-Spot or DigitalGlobe very high resolution optical satellite images

Tools to convert RPC files into localization grids called `Dimap2Grid` and `DigitalGlobe2Grid` allows using VHR optical satellite images with `Malt`.

User has to provide:

- a RPC file for each image
- cartographic step (pixels)
- projection system

As initial localization is not always good enough to run a good correlation process, a refine step has been added to improve the grid.

### 19.2.1.1 Image couple

The way to get grids from an image couple is as following (example given with Dimap2Grid, valid for DigitalGlobe2Grid) :

- Dimap2Grid for each image (produces a rough grid)
- Tapioca (generates tie-points)
- RefineModel (computes affinity coefficients)
- Dimap2Grid with option refineCoef (produces accurate grid)

The command to use for this workflow are:

```
mm3d SateLib Dimap2Grid dimapFile1 imageFile1 altMin altMax nbLay nbLayers targetSyst
mm3d SateLib Dimap2Grid dimapFile2 imageFile2 altMin altMax nbLay nbLayers targetSyst
mm3d Tapioca All imagesPattern -1
mm3d SateLib RefineModel image_1.GRI image_2.GRI pts_1_2.dat meanAltitude
mm3d SateLib Dimap2Grid dimapFile2 imageFile2 altMin altMax nbLay targetSyst refineCoef=refine/refineCoef.tx
```

To know the syntax of Dimap2Grid:

```
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
* string :: {RPC Dimap file}
* string :: {Name of image (to generate appropriately named GRID file)}
* REAL :: {min altitude (ellipsoidal)}
* REAL :: {max altitude (ellipsoidal)}
* INT :: {number of layers (min 4)}
* string :: {targetSyst - target system in Proj4 format}
Named args :
* [Name=stepPixel] REAL :: {Step in pixel (Def=100pix)}
* [Name=stepCarto] REAL :: {Step in m (carto) (Def=50m)}
* [Name=refineCoef] string :: {File of Coef to refine Grid}
* [Name=Bin] bool :: {Export Grid in binaries (Def=True)}
```

DigitalGlobe2Grid is extremelly similar :

```
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
* string :: {RPB from DigitalGlobe file}
* REAL :: {min altitude (ellipsoidal)}
* REAL :: {max altitude (ellipsoidal)}
* INT :: {number of layers (min 4)}
* string :: {targetSyst - target system in Proj4 format (ex : "+proj=utm +zone=32 +north +datum=WGS84 +uni
Named args :
* [Name=stepPixel] REAL :: {Step in pixel (Def=100pix)}
* [Name=stepCarto] REAL :: {Step in m (carto) (Def=50m)}
* [Name=refineCoef] string :: {File of Coef to refine Grid}
* [Name=Bin] bool :: {Export Grid in binaries (Def=True)}
```

Dimap2Grid and DigitalGlobe2Grid generates a .GRI (and.GRIBin) file whose name is set after the image name. The first (two for Dimap2Grid) mandatory argument(s) should be quite obvious. Other arguments are:

- min and max altitude: ground min and max altitude

- number of layers: number of altitudes layers for the grid - 4 are a minimum to have good accuracy, over 10 is not profitable an increase computing time and file size
- targetSyst: target coordinate system, in the proj4 syntax
- stepPixel: grid step in image coordinates (default = 100 pixels)
- stepCarto: grid step in cartographic coordinates (default = 50m)
- refineCoef: file produced by RefineModel command to refine Grid
- Bin : if true (default), automatically calls Gri2Bin to create a GRIBin file

For example, UTM - zone 32N in the Proj4 format looks like (including the quotation marks!!) :

```
"+proj=utm +zone=32 +north +datum=WGS84 +units=m +no_defs"
```

RefineModel syntax is:

```
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {master image GRID}
 * string :: {slave image GRID}
 * string :: {Tie Points}
 * REAL :: {average altitude of the TiePoints}
```

Tie-points string is the tie-point file (.dat) computed by Tapioca and available in the Homol directory.

Once GRI files have been computed, they have to be converted to binary files (speeds up the process dramatically):

```
mm3d Gri2Bin path/file.GRI path/file.GRIBin
```

Then, correlation can be run in ground geometry, with following command:

```
mm3d Malt UrbanMNE ".*JP2" GRIBin MOri=GRID BoxTerrain=[X1,Y1,X2,Y2]
ZoomI=32 ZoomF=1 ZMoy=100 ZInc=500 NbVI=2
```

- MOri= GRID states that we work with grid files
- BoxTerrain are region of interest Lambert93 coordinates
- ZoomI=32: first step is done with images rescaled with factor 32
- ZoomF=1 : last step at full resolution
- ZMoy=100 Z mean value
- ZInc=100 uncertainty value around Z (in meter)
- NbVI=2 minimal number of visible images (by default 3 for UrbanMNE)

### 19.2.1.2 Set of Images

The way to get grids from a set of satellite images is the same as for an image couple, except for the refine stage, for which the command is (right now) slightly different:

Refine syntax is:

```
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
 * string :: {GRID files pattern}
Named args :
 * [Name=DTM] string :: {DTM file}
 * [Name=ExpRes] bool :: {Export residuals (def=false)}
```

So the command to use for this workflow looks like:

```
mm3d SateLib Refine .*GRI
```

First mandatory argument is the pattern for GRID files. Optional arguments are:

- DTM: use a DTM file (xml) to constrain depth (not supported yet)
- ExpRes: export residuals to refine/residus.txt (image coordinates and image residuals) and refine/residus-Glob.txt (ground coordinates and rms error) (default = false)

## 19.3 Epipolar geometry of a satellite image pair

to be updated

## 19.4 SAKE - Simplified tool for satellite images correlation

SAKE stands for “SAtellite Kit for Elevation”, it is a simplified tool to generate DEMs and ortho-images from sets of satellite images, developed by Ana-Maria Rosu. Like MM2DPosSism and FDSC, it is also part of the collaboration between IPGP and IGN/ENSG, and funded by CNES through the TOSCA program.

In order to see all `Sake`'s parameters:

```
mm3d Sake -help
Valid types for enum value:
DEM
OrthoIm
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
* string :: {Computation type (one of the allowed enumerated values)}
* string :: {Images' path (Directory+Pattern)}
* string :: {Orientation file extension (Def=GRI)}
Named args :
* [Name=ZMoy] REAL :: {Average value of Z (Def=1000.0)}
* [Name=ZInc] REAL :: {Initial uncertainty on Z (Def=1000.0)}
* [Name=ModeOri] string :: {Orientation type (GRID or RTO; Def=GRID)}
* [Name=Mask] string :: {Mask file}
* [Name=SzW] INT :: {Correlation window size (Def=2, equiv 5x5)}
* [Name=ZRegul] REAL :: {Regularization factor (Def=0.2)}
* [Name=ZPas] REAL :: {Quantification step (Def=0.5)}
* [Name=ZoomF] INT :: {Final zoom (Def=1)}
* [Name=BoxClip] Box2dr :: {Define computation area (Def=[0,0,1,1] means full area) relative to image}
* [Name=BoxTer] Box2dr :: {Define computation area [Xmin,Ymin,Xmax,Ymax] relative to ground}
* [Name=EZA] bool :: {Export absolute values for Z (Def=true)}
* [Name=DirMEC] string :: {Results subdirectory (Def=MEC-Sake/)}
* [Name=DirOrtho] string :: {Orthos subdirectory if OrthoIm (Def=Ortho-${DirMEC})}
* [Name=DoOrthoM] bool :: {Compute the ortho mosaic if OrthoIm (Def=false)}
* [Name=NbProc] INT :: {Number of cores used for computation (Def=MMNbProc)}
* [Name=Exe] bool :: {Execute command (Def=true)}
```

The mandatory parameters for `Sake` are:

- computation type: `DEM` (`DEM` computation) or `OrthoIm` (ortho-image computation)
- images' path (`Directory+Pattern`): indicates the directory of the images and the regular expression describing the set of images (e.g. ".../test/IMG\_[1-3].tif")
- orientation file extension: `GRI`, `GRIBin...` or `RTO`; filenames before extension must be the same as the corresponding images filenames

`Sake` has a series of optional parameters, all named, meaning that the name of the parameter must be indicated before its designated value:

- `ZMoy`: average value for Z on the area (default value = 1000.0)
- `ZInc`: initial uncertainty on Z (default value = 1000.0)
- `ModeOri`: indicates the type of orientation files (accepted values: `GRID` or `RTO`; default value = `GRID`), this parameter has to be indicated especially when using `RTO` files (`ModeOri=RTO`)
- `Mask`: computation is done only on the area contained in the mask file;
- `SzW`: correlation window size (default value = 2, which is equivalent to a window of  $5 \times 5$ )
- `ZRegul`: regularization factor (default value = 0.2)
- `ZPas`: quantification step for correlation (default value = 0.5)
- `ZoomF`: indicates the DeZoom at which the DEM will be delivered (default value = 1, meaning full resolution)
- `BoxClip`: defines the computation area  $[X_{\text{min}}, Y_{\text{min}}, X_{\text{max}}, Y_{\text{max}}]$  normalized values relative to image space (default value =  $[0,0,1,1]$ , meaning full area)

- **BoxTer**: defines the computation area [Xmin,Ymin,Xmax,Ymax] relative to ground space
- **EZA**: boolean parameter indicating if the final DEM contains the "absolute values" for Z or not (Default value = false, meaning that all Z values are given in relation to the average value for Z, **ZMoy**)
- **DirMEC**: indicates the name of the subdirectory where the matching results are to be stored
- **DirOrtho**: useful only if **OrthoIm** computation, it indicates the orthos subdirectory (by default = **Ortho-DirMEC**)
- **DoOrthoM**: useful only if **OrthoIm** computation, boolean parameter for computing the "global" ortho-image from the individual ortho-images (by default = false)
- **NbProc**: defines the number of cores used for computation (default value = total number of cores available for your computer); depending on the number of images used for DEM computation, on their size and on the RAM of your computer, it is important to give this parameter an appropriate value in order to prevent swapping

In the case of **OrthoIm** computation, a DEM is computed at the resolution indicated by **ZoomF** (by default = 2 for **OrthoIm**). Individual ortho-images, corresponding to each image, are computed at full resolution. At the end, if **DoOrthoM** parameter was set to **true**, **Sake** with **OrthoIm** provides the "global" ortho-image as a mosaic of the individual ortho-images.

Command line example for **Sake**:

```
$ mm3d Sake DEM "IMG_00[1-3].tif" GRI ZMoy=1000 ZInc=400 DirMEC="Mec-Sake-test"
```

If you prefer to use **Sake**'s graphical interface, you should first activate the **Qt** option for MicMac:

```
culture3d/build$ cmake -DWITH_QT4=ON ..
```

or

```
culture3d/build$ cmake -DWITH_QT5=ON ..
```

and then compile.

Now you can launch the Qt graphical interface for **Sake**:

```
$ mm3d vSake
```

# **Part III**

# **Algorithmic Documentation**



I think this part will not evolve very soon, as I do not have so much time and user's documentation is a priority. There are two files from conferences that may give some information `ankara2006-pierrot.pdf` and `ARCH3D-MPD-V14.pdf` before I find some time to complete this part.

MasqIsToujours1 : supprimé

MasqueTerrain

PrefixMasqImRes

MasqImageIn

ValSpecNotImage



# Chapter 20

## Généralités

### 20.1 Notations Géométriques

On est dans le contexte où l'on dispose de  $N$  images notées  $Im_k(u, v)_{k \in [1, N]}$ .

On adopte les notations suivantes :

- $\mathcal{T}$ , avec  $\mathcal{T} = \mathbb{R}^2$ , l'espace "terrain", au sens assez large, c'est l'espace dans lequel est restitué le "MNT";
- $\mathcal{E}_{px}$ , avec  $\mathcal{E}_{px} = \mathbb{R}$  ou avec  $\mathcal{E}_{px} = \mathbb{R}^2$  suivant les cas, l'espace des parallaxes; on notera  $D_{px}$  la dimension de l'espace des parallaxes;
- $\mathcal{I}_k$  l'espace de la  $k^{ième}$  image;

En interne, MICMAC voit la géométrie à travers une seule fonction  $\pi_k$  par image:

$$\pi_k : \mathcal{T} \otimes \mathcal{E}_{px} \rightarrow \mathcal{I}_k \quad (20.1)$$

$$(x, y, p_x) \xrightarrow{\pi_k} (u, v) \quad (20.2)$$

Avec  $p_x = z$  lorsque  $\mathcal{E}_{px} = \mathbb{R}$  et  $p_x = (p_{x_1}, p_{x_2})$  lorsque  $\mathcal{E}_{px} = \mathbb{R}^2$ . A  $p_x$  fixée les fonctions  $\pi_k$ , considérées comme des fonctions de  $\mathcal{T}$  dans  $\mathcal{I}_k$ , sont injectives sur leur domaine d'intérêt et on note abusivement  $\pi_k^{-1}$  la fonction "inverse" définie par :

$$\pi_k^{-1} : \mathcal{I}_k \otimes \mathcal{E}_{px} \rightarrow \mathcal{T} \quad (20.3)$$

$$\pi_k^{-1}(\pi_k(x, y, p_x), p_x) = (x, y) \quad (20.4)$$

Le résultat de la mise en correspondance est un fonction  $\mathcal{F}_{px}$  de  $\mathcal{T}$  dans  $\mathcal{E}_{px}$ :

$$\mathcal{F}_{px} : \mathcal{T} \rightarrow \mathcal{E}_{px} \quad (20.5)$$

On note  $\mathcal{F}_{px}^d, d \in [1, D_{px}]$  les différentes composantes de  $\mathcal{F}_{px}$ .

### 20.2 Discréétisation et quantification

Dans la très grande majorité des cas, MicMac fonctionne par discréétisation des espaces  $\mathcal{T}$  et  $\mathcal{E}_{px}$  (par opposition par exemple aux approches variationnelles). A chaque étape de l'approche multi-résolution (voir 21.2), sont définis des pas de discréétisation  $\Delta^{xy}$  pour  $\mathcal{T}$ , et  $\Delta_k^{px}, k \in [1, D_{px}]$  pour  $\mathcal{E}_{px}$ .

A une étape donnée, on se donne en plus un pas de discréétisation de l'espace image  $\Delta^\varepsilon$ ; dans la majorité des cas  $\Delta^\varepsilon$  est choisi de manière à ce que la résolution image obtenue soit égale à  $\Delta^{xy}$ , mais il peut y avoir des exceptions. Pour chaque pas  $\Delta^\varepsilon$  utilisé MicMac calcule des images sous-résolues d'un facteur  $\Delta^\varepsilon$ , on note  $Im_k(\frac{1}{\Delta^\varepsilon})$  ces images et  $\mathcal{I}_k * \Delta^\varepsilon$  leur espace associé.

Pour des pas de discréétisation donnés, on définit  $p_k$  la version discrète de  $\pi_k$  par<sup>1</sup>:

$$p_k : \mathbb{Z}^2 \otimes \mathbb{Z}^{D_{px}} \rightarrow \mathcal{I}_k * \Delta^\varepsilon \quad (20.6)$$

$$\check{\pi}_k(i, j, u, v) = \pi_k(i * \Delta^{xy}, j * \Delta^{xy}, u * \Delta_1^{px}, v * \Delta_2^{px}) * \Delta^\varepsilon \quad (20.7)$$

Pour des pas  $\Delta^{xy}$  et  $\Delta_k^{px}$  donnés, le travail du corrélateur est de rechercher un tableau  $F_{px}$  de  $\mathbb{Z}^2$  dans  $\mathbb{Z}^{D_{px}}$ , version discrète de  $\mathcal{F}_{px}$ :

---

1. avec modification immédiate de 20.7 lorsque  $D_{px} = 1$

$$F_{px}(i, j)^k = \frac{\mathcal{F}_{px}(i * \Delta^{xy}, j * \Delta^{xy})}{\Delta_k^{px}} \quad (20.8)$$

# Chapter 21

## Approche multi-résolution

### 21.1 Motivations

### 21.2 Modèle prédictif

Afin de limiter la combinatoire MicMac utilise une approche multi-résolution où, à chaque étape, la solution de l'étape précédente est utilisée comme un prédicteur autour duquel l'exploration se fera dans un voisinage limité.

Notons de manière générique  $X'$  ( $F_{px}'^k, \Delta'^{xy} \dots$ ) la valeur de  $X$  à l'étape précédente.

$$\rho^{xy} = \frac{\Delta'^{xy}}{\Delta_{xy}}; \rho_k^{px} = \frac{\Delta_k'^{px}}{\Delta_k^{px}} \quad (21.1)$$

On définit le prédicteur  $Pred_{px}$  par :

$$Pred_{px}(i, j)^k = F_{px}'^k \left( \frac{i, j}{\rho^{xy}} \right) * \rho_k^{px} \quad (21.2)$$

Ensuite on se donne des valeurs  $\delta_A^k$  et  $\delta_P^k$  représentant l'incertitude "altimétrique" et planimétrique. Au sens de la morpho-math, notons  $\oplus$  la dilatation et  $\ominus$  l'érosion. On définit alors  $F_{px}^{+k}$  et  $F_{px}^{-k}$  les "nappes englobantes" qui encadrent l'intervalle de recherche de  $F_{px}^k$  à l'étape courante par :

$$F_{px}^{-k} = Pred_{px}^k \ominus \delta_P^k - \delta_A^k \quad (21.3)$$

$$F_{px}^{+k} = Pred_{px}^k \oplus \delta_P^k + \delta_A^k \quad (21.4)$$

### 21.3 Noyaux utilisés pour la sous-résolution



# Chapter 22

## Mesure de ressemblance et corrélation

### 22.1 Généralité

MicMac utilise essentiellement le coefficient de corrélation normalisé centré comme critère d'attache aux données.

Le cadre général est la comparaison de vecteur de réels pondérées<sup>1</sup>, typiquement ce vecteur est constitué de l'ensemble des valeur de la vignette de corrélation et le poids vaut souvent uniformément 1. Soit  $\mathcal{E} = \mathbb{R}^N$  l'espace des vecteurs. Pour  $\lambda \in \mathbb{R}$ , lorsqu'il n'y a pas d'ambiguité, on notera  $\lambda$  le vecteur constant de  $\mathcal{E}$  tel que  $\forall k, u_k = \lambda$ . On se donne une fonction de pondération  $p_k^{ds}, k \in [1 n]$ . Pour chaque vecteur  $U$  de valeurs  $U_k, k \in [1 n]$  on pose :

$$E(U) = \frac{\sum_{k=1}^n p_k^{ds} U_k}{\sum_{k=1}^n p_k^{ds}} \quad (22.1)$$

Etant donné deux vecteur  $u_k$  et  $v_k$ , une définition classique du coefficient normalisé centré est :

$$Cor(u, v) = \frac{E(uv) - E(u)E(v)}{\sqrt{(E(u^2) - E(u)^2) * (E(v^2) - E(v)^2)}} \quad (22.2)$$

Cette définition est proche de celle utilisée pour le calcul (notamment pour les algorithmes de calculs rapides), mais pas forcément intuitive à interpréter. Pour bâtir une interprétation géométrique, on remarque d'abord que  $E(u, v)$  est un produit scalaire et on munit  $\mathcal{E}$  de la norme associée :

$$\|X\|^2 = E(X^2) \quad (22.3)$$

On note  $\mathcal{P}^0$  l'hyperplan des vecteur de moyenne nulle :

$$\mathcal{P}^0 = \{u \in \mathcal{E} / E(u) = 0\} \quad (22.4)$$

On note  $\mathcal{S}^1$  la sphère unité de  $\mathcal{E}$ :

$$\mathcal{S}^1 = \{u \in \mathcal{E} / \|u\| = 1\} \quad (22.5)$$

On note :

$$\bar{u} = u - E(u) \quad (22.6)$$

Il est immédiat que  $\bar{u} \in \mathcal{P}^0$ , et  $\bar{u}$  peut s'interpréter comme la projection orthogonale de  $u$  sur  $\mathcal{P}^0$ . On note :

$$\tilde{u} = \frac{\bar{u}}{\|\bar{u}\|} \quad (22.7)$$

Il est immédiat que  $\|\tilde{u}\| = 1$  et  $\tilde{u}$  peut s'interpréter comme la projection de  $\bar{u}$  sur  $\mathcal{S}^1$ . Une propriété intéressante de l'application  $u \rightarrow \tilde{u}$  est d'être invariante par translation et homotétrie :

$$\forall (\alpha, \beta) \in \mathbb{R}^2 \quad (\alpha + \beta * u) = \tilde{u} \quad (22.8)$$

On vérifie que le coefficient de correlation peut être défini comme :

---

1. on pourrait généraliser à des fonction réelles sur un espace probabilisé

$$\text{Corr}(u, v) = E(\tilde{u}\tilde{v}) \quad (22.9)$$

Le coefficient de corrélation peut donc être interprété au choix comme :

- le produit scalaire entre  $\tilde{u}$  et  $\tilde{v}$ ;
- le *cosinus* de l'angle entre  $\tilde{u}$  et  $\tilde{v}$ ;
- le *cosinus* de l'angle entre les projections orthogonales de  $u$  et  $v$  sur  $\mathcal{P}^0$ .

Compte tenu de  $\|\tilde{u}\| = \|\tilde{v}\| = 1$ , l'équation 22.9 peut se réécrire :

$$\text{Corr}(u, v) = 1 - \frac{\|\tilde{u} - \tilde{v}\|^2}{2} \quad (22.10)$$

Ce qui fournit une interprétation intéressante du coefficient de corrélation: c'est, à un réétalonnement près, une mesure de distance entre des échantillons normalisés de manière invariante par homothétie translation des radiométries.

## 22.2 Utilisation pour la ressemblance de deux vignettes

Soit  $I_k$  des images,  $p = (i, j)$  un point de l'espace terrain discréteisé,  $p_{x_0}$  une parallaxe fixée. On considère  $U_k^N(l, m)$  le vecteur de  $\mathbb{R}^{(2N+1)^2}$  correspondant à la "vignette" de taille  $N$  centrée en  $p$ :

$$U_k^N = (I_k(\pi_k(i + l, j + m, p_{x_0}))), (l, m) \in [-N, N]^2 \quad (22.11)$$

En général on utilise une pondération uniforme  $\forall k p_k^{ds} = 1$ , le produit scalaire  $U.V$  est alors défini par :

$$E(UV) = \frac{\sum_{k=1}^n U_k V_k}{n} \quad (22.12)$$

On définit le coefficient de corrélation de deux images  $I_1$  et  $I_2$ , au point  $p$ , avec la parallaxe  $p_{x_0}$ , sur une fenêtre de taille  $N$  par :

$$\text{Corr}_{p_{x_0}}^N[I_1, I_2](p) = \text{Corr}(U_1^N(p), U_2^N(p)) \quad (22.13)$$

## 22.3 Fenêtre de "taille 1"

La formule 22.10 montre que le coefficient de corrélation peut être exprimé à partir de la distance euclidienne sur les vecteurs normalisés en homothétie-translation sur les radiométries.

Dans le coefficient de corrélation standard, la vignette sur laquelle est faite la normalisation est la même que celle sur laquelle est calculée la norme. Cette contrainte n'a rien d'automatique, et il peut être *a priori* cohérent de faire la mesure d'écart sur des fenêtres plus petites que sur celles sur laquelle est effectuée la normalisation.

Soit  $\mathbb{I}_M^{nd}$  le vecteur de  $\mathbb{R}^{(2N+1)^2}$  qui vaut 1 ssi  $(l \leq M, m \leq M)$  et 0 sinon; on note  $\|\cdot\|_M$  la norme de  $\mathbb{R}^{(2N+1)^2}$  définie par :

$$\|\tilde{U}^N(p)\|_M^2 = E(\tilde{U}^N(p) * \mathbb{I}_M^{nd}) * \frac{1}{E(\mathbb{I}_M^{nd})} \quad (22.14)$$

A un terme de normalisation près, la norme  $\|\cdot\|_M$  est simplement l'écart mesuré sur une vignette de taille  $M$ . On définit alors le coefficient de corrélation sur une fenêtre de taille  $M/N$  par :

$$\text{Corr}_{p_{x_0}}^{M/N}[I_1, I_2](p) = 1 - \frac{\|\tilde{U}_1^N(p) - \tilde{U}_2^N(p)\|_M^2}{2} \quad (22.15)$$

On peut éventuellement utiliser  $\text{Corr}^{M/N}$  avec  $M = 1$ , d'où le nom de la section. Le coefficient  $\text{Corr}^{M/N}$  n'est clairement plus compris entre -1 et 1.

## 22.4 "Fenêtre exponentielle"

### 22.4.1 Principe des fenêtres à pondération variable

L'utilisation de fenêtre carrée avec une pondération uniforme est souvent considéré comme le choix naturel. Il n'a cependant rien d'obligatoire et il peut même souvent être plus judicieux d'avoir des fenêtres de taille plus grande et une pondération qui décroît en fonction de la distance au "pixel central".

On peut par exemple envisager une pondération en "chapeau chinois", soit par exemple avec des fenêtres de taille  $N$  :

$$p_{(x,y)}^{ds} = |N - x||N - y| \quad (22.16)$$

On peut aussi envisager des fenêtres gaussiennes, de support infini:

$$p_{(x,y)}^{ds} = e^{-\frac{x^2+y^2}{\sigma^2}} \quad (22.17)$$

$$E(UV) = \frac{\iint UV e^{-\frac{x^2+y^2}{\sigma^2}}}{C^{ste}} \quad (22.18)$$

MicMac offre la possibilité de choisir des fenêtres exponentielles qui ont l'avantage d'être à support infini (pas de troncature arbitraire) tout en pouvant être calculées rapidement :

$$E(UV) = \frac{\iint UV a^{-|x|} b^{-|y|}}{C^{ste}} \quad (22.19)$$

### 22.4.2 Equivalence des tailles de fenêtre

En vrac pour l'instant.

Comment choisir un paramètre d'atténuation exponentiel pour avoir des fenêtres, plus ou moins, équivalentes à des fenêtres carrées d'une certaine taille. L'idée est de définir la taille "généralisée" comme l'espérance de  $|X|$  (ou l'écart type  $\sqrt{E(X^2)}$ ). On peut faire le calcul sur des séries discrètes ou continues.

Avec des fenêtres carrées :

$$C_2(N) = \frac{\sum_{k=-N}^{k=N} k^2}{\sum_{k=-N}^{k=N} 1} = \frac{N * (N + 1)}{3} \quad (22.20)$$

$$C'_2(N) = \frac{\int_{-N-\frac{1}{2}}^{N+\frac{1}{2}} X^2}{\int_{-N-\frac{1}{2}}^{N+\frac{1}{2}} 1} = \frac{(n + 1)^2}{3} \quad (22.21)$$

$$E_2(a) = \frac{\sum_{k=-\infty}^{k=+\infty} a^{-|k|} k^2}{\sum_{k=-\infty}^{k=+\infty} a^{-|k|}} = \frac{2}{\log(a)^2} \quad (22.22)$$

Si on itère  $K$  le filtre exponentiel, on obtient un filtre équivalent dont l'écart type est  $K$  fois plus grand. Donc finalement :

$$a_k = e^{-\frac{\sqrt{6k}}{N+1}} \quad (22.23)$$

Avec des fenêtres exp :

## 22.5 Multi-corrélation

Lorsque l'on a plus de 2 images, on souhaite définir un coefficient de corrélation multi-images qui soit un prolongement naturel du coefficient à 2 images.

Soit  $N$  le nombre d'images, on note  $U_1 \dots U_N$  les vecteurs à comparer. Un premier cas où il est facile de définir un coefficient naturel est celui où il existe une image jouant un rôle privilégié, dite image maîtresse. Supposons que ce soit l'image 1, on pose :

$$\text{Corr}^M(U_1)(U_2, \dots, U_N) = \frac{1}{N - 1} \sum_{k=2}^N \text{Corr}(U_1, U_k) \quad (22.24)$$

Un cas encore plus courant est celui où toutes les images jouent un rôle symétrique. Une définition naturelle du coefficient de corrélation est de faire une moyenne sur tous les couples possibles:

$$\text{Corr}^S(U_1, \dots, U_N) = \frac{2}{N * (N - 1)} \sum_{1 \leq i < j \leq N} \text{Corr}(U_i, U_j) \quad (22.25)$$

L'inconvénient apparent de la formule 22.25 est d'avoir un coût de calcul en  $O(N^2)$ . En fait, nous allons voir que le calcul peut facilement se faire en  $O(N)$ . On remarque d'abord que, d'après la formule 22.10, il est équivalent de calculer :

$$\sum_{1 \leq i < j \leq N} \|\tilde{U}_i - \tilde{U}_j\|^2 \quad (22.26)$$

En symétrisant et rajoutant les termes nuls  $\|\tilde{U}_i - \tilde{U}_i\|$ , il est toujours équivalent de calculer :

$$S = \sum_{i=1}^N \sum_{j=1}^N \|\tilde{U}_i - \tilde{U}_j\|^2 \quad (22.27)$$

On définit le centre de gravité  $\Omega$  des  $\tilde{U}_i$  (calculable en temps linéaire) par :

$$\Omega = \frac{\sum_{i=1}^N \tilde{U}_i}{N} \quad (22.28)$$

On développe ensuite :

$$S = \sum_{i=1}^N \sum_{j=1}^N \|(\tilde{U}_i - \Omega) + (\Omega - \tilde{U}_j)\|^2 \quad (22.29)$$

$$S = \sum_{i=1}^N \sum_{j=1}^N (\|(\tilde{U}_i - \Omega)\|^2 + \|(\Omega - \tilde{U}_j)\|^2) + 2 \sum_{i=1}^N (\tilde{U}_i - \Omega) \cdot \sum_{j=1}^N (\Omega - \tilde{U}_j)^2 \quad (22.30)$$

$$S = 2N * \sum_{i=1}^N \|(\tilde{U}_i - \Omega)\|^2 \quad (22.31)$$

## 22.6 Interpolation

## Chapter 23

# Algorithmes de filtrages rapides



# Chapter 24

## Approches énergétiques et régularisation

### 24.1 Généralités

Le travail d'un corrélateur consiste à calculer une fonction  $\mathcal{F}_{px}$ , possédant certaines caractéristiques *a priori* (de régularité) et telle que les  $Im_k(\pi_k(x, y, \mathcal{F}_{px}(x, y))), k \in [1, N]$  se ressemblent.

On note :

- $\mathcal{A}(x, y, p_x) \geq 0$  un critère qui mesure la ressemblance locale entre les  $Im_k(\pi_k(x, y), p_x)_{k \in [1, N]}$ , avec  $\mathcal{A} = 0$  quand les images se ressemblent parfaitement (critère d'attache aux données);
- $\nabla(\mathcal{F}_{px})$  le gradient de  $\mathcal{F}_{px}$ ,  $\nabla(U) = (\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y})$
- $\|\nabla(\mathcal{F}_{px})\|^{reg}$  une norme sur le gradient, qui est utilisé comme critère de régularisation (pénalisation des variations de  $\mathcal{F}_{px}$ );

L'approche énergétique consiste à chercher une solution  $\mathcal{F}_{px}$  qui minimise une fonctionnelle d'énergie  $\mathcal{E}(\mathcal{F}_{px})$  :

$$\mathcal{E}(\mathcal{F}_{px}) = \iint_{\mathcal{T}} \mathcal{A}(x, y, \mathcal{F}_{px}(x, y)) + \|\nabla(\mathcal{F}_{px})\|^{reg} \quad (24.1)$$

Pour les problèmes relevant de l'optimisation combinatoire MicMac utilise souvent une norme  $L_1$  :

$$\|\nabla(\mathcal{F}_{px})\|^{reg} = \alpha_1 * |\nabla(\mathcal{F}_{px}^1)| + \alpha_2 * |\nabla(\mathcal{F}_{px}^2)| \quad (24.2)$$

Le(s) paramètre(s)  $\alpha$  permet de pondérer l'importance relative de l'*a priori* et de l'attache aux données. Lorsque  $D_{px} = 2$ , on a typiquement  $\alpha_1 \approx \alpha_2$  pour les géométries "vraiment" bi-dimensionnelles et  $\alpha_1 \ll \alpha_2$  pour les géométries où  $\mathcal{F}_{px}^2$  sert à modéliser des défauts de mise en place.

Dans certains cas MicMac peut utiliser aussi des normes  $L_2$ . En programmation dynamique on ne fait intervenir que les dérivées premières. Pour les algorithmes variationnels, il est aussi possibles d'introduire les dérivées secondes.

Type	Quant	Rapidité	Der	DPax	D Opt	Optim
Prog Dyn	Oui	+++	[0-1]	1,2	1	approchée
Cox-Roy	Oui	++	[0-0]	1	1,2	exacte
Differentiel	Non	+	[1-2]	1,2	1,2	locale
Dequant	?	++++	[0-0]	1,2	1,2	filtrage
MaxOfScore	Oui	++++	[0-0]	1,2	1,2	aucune

### 24.2 Programmation dynamique

La programmation dynamique se limite au cas où l'espace de départ est ordonné (donc de dimension 1) et l'espace d'arrivée est quantifié. Ces limitations prises en compte, c'est une méthode de portée assez générale pour optimiser des applications de  $\mathbb{Z}$  dans  $\mathbb{Z}^k$ .

En stéréovision (et en traitement d'image de manière plus générale) où l'espace de départ est de dimension 2, on est souvent conduit à balayer l'image suivant les lignes (ou les colonnes, ou n'importe quelle direction) pour utiliser la programmation dynamique. Pour chaque balayage on a alors une application de  $\mathbb{Z}$  dans  $\mathbb{Z}^k$ .

Formellement, on peut décrire ainsi le fonctionnement :

- on dispose de  $N$  positions  $P_k, k \in [1, N]$ ;

- pour chaque position, on dispose de l'ensemble des états possibles que peut prendre cette position  $E_k = e_1^k, \dots, e_{n_k}^k$
- une solution  $S$  est une suite correspondant à la sélection de l'état de chaque position  $\{e_{S(1)}^1 \dots e_{S(N)}^N\}$ ;
- chaque état possède un coût intrinsèque  $C^I(e_i^k)$  et chaque couple d'états successifs possède un coût de transition  $C^T(e_i^k, e_j^{k+1})$

Le coût global d'une solution est défini par :

$$\mathcal{C}(S) = \sum_{k=1}^N C^I(e_{S(k)}^k) + C^T(e_{S(k)}^k, e_{S(k+1)}^{k+1}) \quad (24.3)$$

Une fois effectué l'opération de balayage, la transcription pour chaque ligne au problème d'appariement est immédiate :

- les positions sont les pixels;
- les états d'un pixels sont les paralaxes qu'il peut prendre en fonction du contexte (nappes englobantes ...);
- une solution est un champs de parallaxe;
- les coûts intrinsèques portent l'attache aux données (fonction de la corrélation obtenue pour chaque pixel lorsqu'on le reprojette à la parallaxe  $e_j^k$ );
- les coûts de transitions portent l'a priori; par exemple, si la parallaxe correspondant à l'état  $e_i^k$  s'écrit  $(Px_i^{1k}, Px_i^{2k})$ , on pourrait utiliser l'équation 24.4 comme schéma de discrétisation de 24.2;
- on pourrait aussi modéliser des coûts quadratiques avec l'équation 24.5;
- on peut imposer différentes contraintes, par exemple que les solution retenue respectent un critère de pente maximum  $P_{max}$  en rajoutant une règle du type de l'équation 24.6;

$$C^T(e_i^k, e_j^{k+1}) = \alpha_1 * |Px_i^{1k} - Px_j^{1k+1}| + \alpha_2 * |Px_i^{2k} - Px_j^{2k+1}| \quad (24.4)$$

$$C^T(e_i^k, e_j^{k+1}) = \alpha_1 * (Px_i^{1k} - Px_j^{1k+1})^2 \quad (24.5)$$

$$|Px_i^{1k} - Px_j^{1k+1}| > P_{max} \Rightarrow C^T(e_i^k, e_j^{k+1}) = +\infty \quad (24.6)$$

La programmation dynamique fournit alors un algorithme rapide pour calculer la solution  $\mathcal{S}_{min}$  qui minimise le coût  $\mathcal{C}(S)$ . Rappelons le principe :

- on cherche à calculer la fonction  $\mathcal{C}_{min}^+(e_i^k)$  qui correspondent au coût de la plus petite sous suite de taille  $k$  se terminant par  $k$  (c.a.d. les suites de la forme  $e_{i_0}^0 \dots e_{i_{k-1}}^{k-1} e_i^k$ );
- on effectue le calcul en parcourant les positions suivant les indice croissants;
- en rajoutant un sommet neutre d'indice 0 en début on a initialement  $\mathcal{C}_{min}^+(e_0^0) = 0$
- on utilise ensuite la récurrence donnée par l'équation 24.7, en memorisant pour chaque sommet celui ayant conduit au calcul du minimum ( son "père");

$$\mathcal{C}_{min}^+(e_i^{k+1}) = C^I(e_i^{k+1}) + \text{Min}_{j \in [1, n_k]} (\mathcal{C}_{min}^+(e_j^k) + C^T(e_j^k, e_i^{k+1})) \quad (24.7)$$

Arrivé à la position  $N$ , l'état qui minimise  $\mathcal{C}_{min}^+(e_j^N)$  est celui par lequel se termine la solution  $\mathcal{S}_{min}$ , il suffit alors de parcourir les position vers l'arrière à partir de cet état en remontant aux "pères" pour obtenir l'ensemble de la série  $\mathcal{S}_{min}$ .

## 24.3 Programmation dynamique "multi directionnelle"

Le balayage directionnel, nécessaire pour appliquer la programmation dynamique à des images, conduit à des artefacts qui peuvent être assez gênants. MicMac propose un principe de programmation dynamique "multi-directionnel" qui permet de limiter ces effets. Pour chaque direction de balayage, on calcule :

- d'une part  $\mathcal{C}_{min}^+(e_i^k)$  décrit en 24.2;
- d'autre part  $\mathcal{C}_{min}^-(e_i^k)$  correspondant au coût de la plus petite sous suite de taille  $N - k$  commençant par  $k$  et calculée selon le même principe (mais par un parcours arrière);

On pose :

$$\mathcal{C}_{min}(e_i^k) = \mathcal{C}_{min}^+(e_i^k) + \mathcal{C}_{min}^-(e_i^k) - C^I(e_i^k) \quad (24.8)$$

On montre facilement que  $\mathcal{C}_{min}(e_i^k)$  est le coût de la plus petite solution contrainte à passer par  $e_i^k$ . On pose finalement :

$$\Delta_{min}(e_i^k) = \mathcal{C}_{min}(e_i^k) - \mathcal{C}(\mathcal{S}_{min}) \quad (24.9)$$

La valeur  $\Delta_{min}(e_i^k)$  s'interprète facilement comme le surcoût, par rapport à la solution optimale qu'il y a à passer par l'état  $e_i^k$ . On dispose d'une information sensiblement plus riche que la programmation dynamique habituelle qui ne fournit que la solution optimale : on dispose maintenant, pour chaque état, d'une mesure à valeur réelle quantifiant quelle distance, au sens du critère à minimiser, le sépare de la solution optimum.

L'intérêt est d'avoir une mesure qui va permettre d'aggréger les résultats obtenus par des balayages effectués dans plusieurs directions de l'image afin de limiter les effets directionnels. On pourra par exemple balayer l'image selon  $D$  directions correspondant à des angles  $\frac{d\pi}{D}$  et aggréger suivant une des méthodes ci-dessous chaque  $\Delta_{min}^d(e_i^k)$  obtenu :

1. calculer la moyenne des  $\Delta_{min}^d(e_i^k)$ ;
2. calculer le max des  $\Delta_{min}^d(e_i^k)$ ;
3. utiliser chaque  $\Delta_{min}^d(e_i^k)$  comme coût d'entrée pour le prochain balayage (méthode dite réentrant);

## 24.4 Algorithmes de flots

Lorsque la parallaxe est de dimension 1, les algorithmes de "flots maximum / coupe minimum" permettent de trouver le minimum exact du critère énergétique en  $L_1$ . Une fois le problème discrétilisé et quantifié, la fonctionnelle s'écrit :

$$E(Z) = \sum_{x,y} (A(x,y, Z(x,y)) + \alpha * \sum_{u,v \in V} P_{u,v}^{ds} |Z(x,y) - Z(x+u,y+v)|) \quad (24.10)$$

Avec  $Z$  la fonction inconnue,  $A$  le critère d'attache aux données,  $\alpha$  le coefficient pondérant l'*a priori*,  $V$  le voisinage utilisée (typiquement 4 ou 8 voisinage), et  $P_{u,v}^{ds}$  une éventuelle pondération du voisinage. L'implémentation proposée par MicMac est basée sur le code de Cox & Roy décrite dans [Cox-Roy 98].

## 24.5 Algorithmes de déquantification

## 24.6 Algorithmes variationnels

Pas encore implantés.



# Chapter 25

## Algorithm on orientation

### 25.1 Tomasi-Kanabe

This section gives a brief description of the Tomasi-Kanabe algorithm that is (will be) used in Martini as a supplementary test of orientation for long focal camera. See [Tomasi Kanabe 98], the original paper, for more details.

Let  $C_1, C_2, \dots, C_N$  be  $N$  camera with infinite focal length, classically the projection function are ortho centric projection, and there is a strict redundancy between principal-point/origin and focal/origin. The projection function  $\pi_n$  on camera  $C_n$  is given by :

$$\pi_n(P) = (A_n, B_n) + S_n * (i_n.P, j_n.P) \quad (25.1)$$

Where :

- $i_n$  and  $j_n$  are the axes of the camera they are the two first vector of an orthonormal repair  $\|i_n\| = \|j_n\| = 1$ ,  $i_n \cdot j_n = 0$ , we note  $k_n = i_n \wedge j_n$ .
- the translation  $(A_n, B_n)$  represent the "principal point/origin" on  $i_n, j_n$ ;
- the scaling factor  $S_n$  represent the "focal length/origin" on  $k_n$ .

Having  $M$  points seen in the  $N$  camera (we know the projection of these points but not their 3d coordinates in some common ground space) we note :

- $P_m, m \in [1, M]$  the unknown 3d coordinates of the  $m^{th}$  points;
- $(u_{n,m}, v_{n,m}), m \in [1, M], n \in [1, N]$  the 2d coordinates of projection of  $P_m$  points on  $C_n$  ;
- we have :

$$\pi_n(P_m) = (u_{n,m}, v_{n,m}) \quad (25.2)$$

As the repair in which we want to express  $P_k$  is arbitrary, we decide to set the origin at the center of mass; we then have :

$$\sum_{m=1}^M P_m = 0 \quad (25.3)$$

As equation 25.1 is linear :

$$\forall n : \sum_{m=1}^M \pi_n(P_m) = (A_n, B_n) * M + S_n * (i_n \cdot \sum_{m=1}^M P_m, j_n \cdot \sum_{m=1}^M P_m) = (A_n, B_n) * M = \sum_{m=1}^M (u_{n,m}, v_{n,m}) \quad (25.4)$$

So we see that  $A_n, B_n$  can easily be computed by :

$$A_n = \frac{\sum_{m=1}^M u_{n,m}}{M}; B_n = \frac{\sum_{m=1}^M v_{n,m}}{M}; \quad (25.5)$$

As in [Tomasi Kanabe 98], to ligthen notation, we suppose we begin by normalise  $u_{n,m}$  and  $v_{n,m}$  by substracting the center of mass, and equation 25.5 resume to  $A_n = B_n = 0$  and equation 25.1 can be written :

$$(u_{n,m}, v_{n,m}) = \pi_n(P_m) = S_n * (i_n.P_m, j_n.P_m) \quad (25.6)$$

We note  $M_v^u$  the matrix :

$$M_v^u = \begin{pmatrix} u_{1,1} & u_{2,1} & \dots & u_{N,1} \\ u_{1,2} & u_{2,2} & \dots & u_{N,2} \\ \dots & \dots & \dots & \dots \\ u_{1,M} & u_{2,M} & \dots & u_{N,M} \\ v_{1,1} & v_{2,1} & \dots & v_{N,1} \\ \dots & \dots & \dots & \dots \\ v_{1,M} & v_{2,M} & \dots & v_{N,M} \end{pmatrix} \quad (25.7)$$

Using equation 25.6, equation 25.7 can be written :

$$M_v^u = \begin{pmatrix} S_1 * i_1^x & S_1 * i_1^y & S_1 * i_1^z \\ S_2 * i_2^x & S_2 * i_2^y & S_2 * i_2^z \\ \dots & \dots & \dots \\ S_M * i_M^x & S_M * i_M^y & S_M * i_M^z \\ S_1 * j_1^x & S_1 * j_1^y & S_1 * j_1^z \\ \dots & \dots & \dots \\ S_M * j_M^x & S_M * j_M^y & S_M * j_M^z \end{pmatrix} * \begin{pmatrix} P_1^x & P_2^x & \dots & P_N^x \\ P_1^y & P_2^y & \dots & P_N^y \\ P_1^z & P_2^z & \dots & P_N^z \end{pmatrix} = M_j^i * M^P \quad (25.8)$$

We can still suppose that  $M_j^i$  is a square Matrix by padding is with 0 column or lines. Padding it with 0 column correspond to add the projection of a point of coordinate  $(0, 0, 0)$ , padding with 0 line correspond to add a camera with scaling factor 0. Using a singular value decomposition,  $M_v^u$  can be written :

$$M_v^u = {}^t R_1 \Delta R_2 \quad (25.9)$$

With :

- $R_1^t R_1 = Id$
- $R_2^t R_2 = Id$
- $\Delta$  diagonal matrix.

As  $M_j^i$  (or  $M^P$ ) is of rank 3,  $M_v^u$  is also of rank 3. So theoretically,  $\Delta$  has only 3 non zero value, due to measurement error in  $u_{n,m}, v_{n,m}$  (and also numerical error in computation), the value may be not exactly equal to zero; however,  $\Delta$  can be best approximated by the  $3 * 3$  matrix corresponding to the highest eigen values (in absolute values). Suppressing the corresponding void row of  $R_1$  and void column of  $R_2$  we obtain a decomposition of  $M_v^u$  in the form :

$$M_v^u = r_1 \delta r_2 \quad (25.10)$$

With  $\delta$  a  $3 * 3$  matrix,  $r_1$  a  $N * 3$  matrix and  $r_2$  a  $3 * N$  matrix, seting arbirtarily  $r'_2 = \delta r_2$ , we can write :

$$M_v^u = r_1 r'_2 \quad (25.11)$$

Also this formula is close to 25.8, we cannot identify  $M_j^i = r_1$  and  $M^P = r'_2$  because the decomposition is far from unique; in fact for any  $3 * 3$  invertible matrix  $Q$ , we still have :

$$M_v^u = (r_1 Q)(Q^{-1} r'_2) \quad (25.12)$$

To determine  $Q$  we are now using the fact that the  $(i_m, j_m)$  are orthonormal, remember we have :

$$M_j^i = {}^t ( i_1, i_2 \dots i_M \quad j_1 \quad \dots \quad j_M ) \quad (25.13)$$

We can write the metric constraints :

- ${}^t i_1 Q^t Q i_1 = 1$
- ${}^t j_1 Q^t Q j_1 = 1$
- $\forall n : {}^t i_n Q^t Q j_n = 0$
- $\forall n \neq 1 : {}^t i_n Q^t Q i_n = {}^t j_n Q^t Q j_n$

Note that, as there is a global arbirtary scaling, we make the camera 1 play a particular role setting its scaling to 1, for the other we just impose to have the same scaling in  $x$  and  $y$ .

As  $W = Q^t Q$  is a symetric matrix, we can estimate  $W$  by least mean square using the above linear equation (there is 6 unknow on  $Q$ , we have  $1 + 2 * N$  equations and  $N \geq 3$  ). Knowing  $W$ , it's easy to recover  $Q$ , we do a singular value decomposition of  $W$  :

$$W = {}^t R D R \quad (25.14)$$

When all the eigen value are positive,<sup>1</sup> we can compute a possible value of  $Q$  by

---

1. we see just after when it is not the case

$$Q = {}^t R \sqrt{D} \quad (25.15)$$

This value is not unique, because for any rotation  $r$ ,  $Q' = Qr$  will be also a solution. This non unicity simply reflect the fact that the orientation of the camera can only be computed up to a global rotation. We will use this fact for processing the case with negative values.

When there exist negative value, the computation is still easy, also we have to use hermitian produce and hermitian matrices to prove the validity, we can write :

$$Q = {}^t R \sqrt{|D|} \mathbb{I} = Q' \mathbb{I} \quad (25.16)$$

Where  $\mathbb{I}$  is a diagonal complex matrix containing only 1 or  $i$ . We still have :

$$W = {}^t \bar{Q} Q \quad (25.17)$$

And  $\mathbb{I}$  is a hermitian unitary matrix as :

$${}^t \bar{\mathbb{I}} \mathbb{I} = Id \quad (25.18)$$

As  $Q$  is defined up to a global unitary matrix, we can use  $Q'$  instead of  $Q$ .

## 25.2 Triplet selection algorithm

This section describe the method and code use for selecting triplet, as last time I had to go back to it, it seemed to me quite obscure .... It is implemnted in file `src/uti.phgrm/cNew0_OldGenTriplets.cpp`.

Le  $S_1, S_2$  be images,

The idea is to select a subset of images, with have a maximum of triple point, that these point cover the cloud of

Let :

- $Depth = \text{mHautBase}$  be the average depth of the scene
- $B_H = \frac{B}{H}$  be base to height ratio
- $B_H^{Lim} = 0.15 = \text{TBSurHLim}$  be the a constant seting when  $\frac{B}{H}$  ratio becomes big;
- Also the code is obscured by the fact that all computation are done in integer.
- $Q = 30 = \text{TQuant}$  be the constant of quantification.
- $Q_{Bh} = 100 = \text{TQuantBsH}$  be the constant of quantification for  $\frac{B}{H}$
- $NbP1 = 6 = \text{TNbCaseP1}$ , use for digitalizing the image space
- $Nb_C = NbP1 * NbP1 = \text{TNbCaseP1}$ , for efficiency the 2d distribution are stored in 1 board of size  $Nb_C$
- The gain when adding a summit is controled by the distance to others, thi gains is controled by the function :
- $B_H(S_1, S_2) = d(S_1, S_2) / Depth$
- $GB_H(S1, S2) = Q_{Bh} * \frac{B_H}{B_H + B_H^{Lim}}$



# Chapter 26

## Sensibility Analysis

### 26.1 Theoreticall consideration

#### 26.1.1 some tricks

##### 26.1.1.1 tricks

When  $A$  and  $B$  are vector,  ${}^t AB$  is as scalar so :

$${}^t AB = {}^t ({}^t AB) = {}^t BA \quad (26.1)$$

Then :

$$({}^t AB)^2 = {}^t AB {}^t BA = {}^t A(B {}^t B)A \quad (26.2)$$

So the term can interpred as the application of the quadratic form  $B {}^t B^{-1}$  to vector  $A$ .

##### 26.1.1.2 tricks

If  $A$  is a symetric positive matrix, the minimum of quadratic form  $F(X) = {}^t XAX - 2{}^t BX$  is reached for  $X = A^{-1}B$ . If we write  $X' = X + \delta$  :

$$F(X') - F(X) = {}^t (A^{-1}B + \delta)A(A^{-1}B + \delta) - 2{}^t B(A^{-1}B + \delta) - F(X) = {}^t \delta A \delta \quad (26.3)$$

Which is always positive as  $A$  is positive.

##### 26.1.1.3 tricks

We have also the well known block matrix inverse identity :

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} A' & B' \\ C' & D' \end{pmatrix} = \begin{pmatrix} (A - BD^{-1}C)^{-1} & -(A - BD^{-1}C)^{-1}BD^{-1} \\ -D^{-1}C(A - BD^{-1}C)^{-1} & D^{-1} + D^{-1}C(A - BD^{-1}C)^{-1}BD^{-1} \end{pmatrix} \quad (26.4)$$

### 26.1.2 Least square notation

Suppose we have  $M$  equation of observation with  $N$  unknown,  $M > N$  :

$$\sum_{i=1}^N l_i^m x_i = o_m ; \quad (26.5)$$

Noting :

$$L^m = {}^t (l_1^m \ l_2^m \ \dots \ l_N^m) m \in [1, M]; X = {}^t (x_1 \ x_2 \ \dots \ x_N) \quad (26.6)$$

Equation 26.5 writes :

$${}^t L^m X = o_m , m \in [1, M]; \quad (26.7)$$

---

1. of rank 1

As  $M > N$  it is generally impossible to annulate all the term , instead we minimise the square of residual  $R_2(X)$  :

$$R^2(X) = \sum_{m=1}^M (^t L^m X - o_m)^2 \quad (26.8)$$

Using tricks 26.1 and 26.2 we can write :

$$R^2(X) = \sum_{m=1}^M ((^t L^m X)^2 - 2o_m ^t L^m X + o_m^2) = \sum_{m=1}^M (^t X (L^{m t} L^m) X - (2o_m ^t L^m) X + o_m^2) \quad (26.9)$$

Noting the  $N \times N$  matrix A, B the  $N$  vector and the scalar C :

$$A = \sum_{m=1}^M (L^{m t} L^m); B = \sum_{m=1}^M (o_m L^m); C = \sum_{m=1}^M o_m^2 \quad (26.10)$$

We have :

$$R^2(X) = ^t X A X - 2^t B X + C \quad (26.11)$$

Obviously A is positive as being the some of squares. The minimum is reached for :

$$\hat{X} = A^{-1} B \quad (26.12)$$

### 26.1.3 Variance

The system being not exactly invertible, for each observation  $m$ , the equation 26.5 is only approximatively satisfied by  $\hat{X}$ , so we introduce the residual  $\epsilon$  to modelize this uncertainty:

$$^t L^m X = o_m + \epsilon_m \quad (26.13)$$

To evaluate variance on  $X$  we consider  $\epsilon_m$  as the realization of random variable. Here I consider that each  $\epsilon_m$  is an independant variable, of average 0 and variance  $r_m^2$  where  $r_m^2$  is the empirical residual :

$$r_m = ^t L^m \hat{X} - o_m; Var(\epsilon_m) = r_m^2 \quad (26.14)$$

We can then modelize the probabilistic aspect of evaluation of  $X$  by the random vector  $\tilde{X}$  :

$$\tilde{X} = A^{-1} \sum_{i=1}^M L^m (o_m + \epsilon_m) \quad (26.15)$$

As  $o_m$  is deterministic the variance is :

$$Var(\tilde{X}) = Var(A^{-1} \sum_{i=1}^M L^m \epsilon_m) \quad (26.16)$$

Noting the element of  $A^{-1}$  :

$$A^{-1} = (a'^j_i) \quad (26.17)$$

$$Var(\tilde{x}_i) = Var(\sum_{m=1}^M \sum_{j=1}^N a'^j_i l_j^m \epsilon_m) \quad (26.18)$$

$$Var(\tilde{x}_i) = \sum_{m=1}^M Var(\epsilon_m) (\sum_{j=1}^N a'^j_i l_j^m)^2 \quad (26.19)$$

$$Var(\tilde{x}_i) = \sum_{m=1}^M (^t L^m \hat{X} - o_m)^2 (\sum_{j=1}^N a'^j_i l_j^m)^2 \quad (26.20)$$

---

2. well it's probably an heresy for stastical point of view to try ro extract information from a single realisation ?

### 26.1.4 Covariance

Similarly, we can compute the covariance :

$$Cov(\tilde{x}_i \tilde{x}_j) = \mathbb{E}\left(\left(\sum_{m=1}^M \sum_{k=1}^N a'_i l_k^m \epsilon_m\right)\left(\sum_{n=1}^M \sum_{k=1}^N a'_j l_k^n \epsilon_n\right)\right) \quad (26.21)$$

Under the independance hypothesis, we have

$$\forall m, n, m \neq n : \mathbb{E}(\epsilon_m \epsilon_n) = 0 \quad (26.22)$$

$$Cov(\tilde{x}_i \tilde{x}_j) = \sum_{m=1}^M Var(\epsilon_m) \left( \sum_{k=1}^N a'_i l_k^m \right) \left( \sum_{k=1}^N a'_j l_k^m \right) \quad (26.23)$$

### 26.1.5 Unknown elimination

Computation of variance and co-variance requires some additional precaution when using the schurr complement technique as described in C.2 and C.1. The computation of this paragraph are rather destinated to understand the code modification impacted by unkown elimination.

We separate the unkown in  $X$  and  $Y$ , where  $X$  is the unknown we want to eliminate.

$${}^t K^m X + {}^t L^m Y = o_m \quad (26.24)$$

And the residual writes :

$$R^2(X, Y) = {}^t X \left( \sum_{m=1}^M K^m {}^t K^m \right) X + 2 \left( \sum_{m=1}^M ({}^t L^m Y - o_m) {}^t K^m \right) X + \sum_{m=1}^M ({}^t L^m Y - o_m)^2 \quad (26.25)$$

We split  $R^2$  in  $R_y^2(Y)$  and  $r_Y(X)$ , where  $R_y^2$  is the part that do not depends of  $X$  :

$$\Lambda = \sum_{m=1}^M K^m {}^t K^m ; \Gamma(Y) = \sum_{m=1}^M ({}^t L^m Y - o_m) K^m \quad (26.26)$$

$$r_Y(X) = {}^t X \Lambda X + 2 {}^t \Gamma(Y) X ; R_y^2(Y) = \sum_{m=1}^M ({}^t L^m Y - o_m)^2 \quad (26.27)$$

$$R^2(X, Y) = r_Y(X) + R_y^2(Y) \quad (26.28)$$

To eliminate  $X$  in the minimisation, we set  $X$  to the value that minimize  $r_Y(X)$  for a given  $Y$ , that is :

$$\hat{X}(Y) = -\Lambda^{-1} \Gamma(Y) \quad (26.29)$$

And the minimum value is :

$$r_Y(\hat{X}(Y)) = {}^t \hat{X}(Y) \Lambda \hat{X}(Y) + 2 {}^t \Gamma(Y) \hat{X}(Y) = -{}^t \Gamma(Y) \Lambda^{-1} \Gamma(Y) \quad (26.30)$$

In unkown elimination we suppose that  $X = \hat{X}(Y)$  and compute only :

$$\check{R}^2(Y) = R^2(\hat{X}(Y), Y) = R_y^2(Y) - {}^t \Gamma(Y) \Lambda^{-1} \Gamma(Y) \quad (26.31)$$

We develop :

$$\check{R}^2(Y) = \sum_{m=1}^M ({}^t L^m Y - o_m)^2 - \left( \sum_{m=1}^M ({}^t L^m Y - o_m) {}^t K^m \right) \Lambda^{-1} \left( \sum_{m=1}^M ({}^t L^m Y - o_m) K^m \right) \quad (26.32)$$

Noting :

$$\check{A} = \sum_{m=1}^M L^{mt} L^m - \left( \sum_{m=1}^M L^{mt} K^m \right) \Lambda^{-1} \left( \sum_{m=1}^M K^{mt} L^m \right) \quad (26.33)$$

$$\check{B} = \sum_{m=1}^M (o_m L^m) - \left( \sum_{m=1}^M L^{mt} K^m \right) \Lambda^{-1} \left( \sum_{m=1}^M o_m K^m \right) \quad (26.34)$$

We have :

$$\check{R}^2(Y) = {}^t Y \check{A} Y - 2 {}^t \check{B} Y + Cste \quad (26.35)$$

And the estimation of  $\check{Y}$  of  $Y$  by least square :

$$\check{Y} = \check{A}^{-1} \check{B} \quad (26.36)$$

We write :

$$\Theta = \sum_{m=1}^M L^{mt} K^m \quad (26.37)$$

$$\check{A} = \sum_{m=1}^M L^{mt} L^m - \Theta \Lambda^{-1t} \Theta \quad (26.38)$$

$$\check{B} = \sum_{m=1}^M (o_m L^m) - \Theta \Lambda^{-1} \left( \sum_{m=1}^M o_m K^m \right) = \sum_{m=1}^M o_m (L^m - \Theta \Lambda^{-1} K^m) \quad (26.39)$$

Defining  $\check{L}^m$ , we have :

$$\check{L}^m = (L^m - \Theta \Lambda^{-1} K^m) \quad (26.40)$$

$$\check{B} = \sum_{m=1}^M o_m \check{L}^m \quad (26.41)$$

Using  $\check{A}$ ,  $\check{B}$  and  $\check{L}^m$  we finaly can compute the variance and covariance of  $\check{Y}$  with formula equivalent to 26.1.3 and 26.1.4 . We consider the random vector  $\tilde{Y}$  :

$$\tilde{Y} = \check{A}^{-1} \sum_{m=1}^M (o_m + \epsilon_m) \check{L}^m \quad (26.42)$$

We have :

$$Var(\tilde{y}_i) = \sum_{m=1}^M Var(\epsilon_m) \left( \sum_{j=1}^N \check{a}'_i \check{l}_j^m \right)^2 \quad (26.43)$$

$$Cov(\tilde{y}_i \tilde{y}_j) = \sum_{m=1}^M Var(\epsilon_m) \left( \sum_{k=1}^N \check{a}'_j \check{l}_k^m \right) \left( \sum_{k=1}^N \check{a}'_i \check{l}_k^m \right) \quad (26.44)$$

### 26.1.6 Practicle aspects on unknown elimination in MicMac

Practically, in MicMac, the unkown elimination is essentially used to eliminate, for each tie points, the 3d point that project on each image. This is done "à la volée" (on the flight ?) with the following procedure, for each tie point :

- the unknown of the 3d point are always located to the same place (say they are unknown 1,2,3)
- the observation are used in accumulator matrix  $A, B, C$  using equation 26.10 (ignoring for now the future elimination);
- then  $\Lambda$  and  $\Theta$  are computed and equations 26.38 and 26.39 to modify the accumulator  $A, B, C$
- the part of the accumulator  $A, B, C$  corresponding to unknow [1 - 3] are reseted;

So at the end, the accumulator  $A$  and  $B$  contains the global  $\check{A}$  and  $\check{B}$  and allow to compute the optimal value  $\check{Y}$  . For variance-covariance, this way of proceeding , raise a probleme for computing the residual for estimation of  $Var(\epsilon_m)$  that require the value of unknowns as shown in 26.14, we know the value for  $Y$ , but not for  $X$ . There is two possibility :

- use formula 26.29 , with  $\check{Y}$  as value, this create an iteration offset;
- use the value computed from bundle intersection, which is also an approximation.

For now the second solution is used .

### 26.1.7 Sensibility

$$F(X) = {}^t X M X = F(y, Z) = \begin{pmatrix} y & {}^t Z \end{pmatrix} \begin{pmatrix} a & B \\ {}^t B & D \end{pmatrix} \begin{pmatrix} y \\ Z \end{pmatrix} = a y^2 + 2 y {}^t B Z + {}^t Z D Z \quad (26.45)$$

For a given  $y$ ,  $F(y, Z)$  is minimal for :

$$Z_{min}(y) = -y D^{-1} B \quad (26.46)$$

And the minimal value is :

$$V_{min}(y) = F(y, Z_{min}(y)) = y^2(a - {}^t BD^{-1}B) \quad (26.47)$$

In our case where  $A = a$  is a 1 dimensionnal (scalar) we can then write :

$$V_{min}(y) = y^2(a - {}^t BD^{-1}B) = \frac{x^2}{a'} \quad (26.48)$$

So using equation 26.4,  $V_{min}(x)$  can be easily computed from the inverse matric. If we have a "bad" value of  $y$ , we have two estimation of the impact on  $F$  :

- a "pessimistic"  $ay^2$ ;
- a "optimistic"  $\frac{y^2}{a'}$ .

So know, if we explain the empirical least square error  $R$ , by a bad estimation on  $y$ , we have two estimation of the sensibility/accuracy of  $y$ , optimisitic in 26.49 and pessimistic in ??:

$$\sqrt{\frac{a}{R}} \quad (26.49)$$

$$\sqrt{a'R} \quad (26.50)$$

## 26.2 Use in MicMac

The computation of these different value can be done in **Martini** command, by setting to true the optional parameter **ExportSensib**. Different value are exported at the end of computation; all the file are located in the same folder containing the orientation generated and they have name begining **Sensib**.

There is four matrix file, exported as images in float format. This files are :

- **Sensib-MatriceCov.tif** contain the covariance matrix, this is the matrix resulting from unknown elimination (this of 26.38);
- **Sensib-MatriceCorrelDir.tif** contain the correlation extracted from direct covariance matrice i.e.  $\frac{a_j^i}{\sqrt{a_i^i * a_j^j}}$ ;
- **Sensib-MatriceCorrelInv.tif** contain the correlation extracted from invert covariance matrice  $\check{A}^{-1}$  ;

Probably the **Sensib-MatriceCorrelDir.tif** is what is most currently used and known as correlation matrices.

When exploring the images with the **Vino** tool, the value are printed using short names, for example when grabing the window, one can get the following messages :

- **V=0.452** and **[Ima4:cZ] [Ima3:T12] (P=41,32)**
- this mean than the correlation is 0.452 between  $Z$  coordinate of center image 4 and  $\theta_{12}$  of image 3;
- as short name are used for variables in **Vino** , a file containing conversion between short and long name is generated.

An example of conversion file :

```
#####
# Intrinseque Calibration Correspondance #####
Cal0 => ./Ori-AllRel/AutoCal_Foc-24000_Cam-PENTAX_K5.xml
#####
# Extrinsic Calibration Correspondance #####
Ima0 => IMGP7029.JPG
Ima1 => IMGP7030.JPG
...
```

The file **Sensib-Data.xml** contains information on variance, uncertainty ... regarding individual variable . Three value are given, corresponding to different formula :

- formula 26.49 correspond to **<SensibParamDir>**;
- formula 26.50 correspond to **<SensibParamInv>**;
- formula 26.43 correspond to **<SensibParamVar>**.

Here is an example with an acquisition mixing GPS and photogrammetry. Probably the **<SensibParamVar>** is the more realistic evaluation of uncertainty.

```
<SensibDateOneInc>
  <NameBloc>cBaseGPS</NameBloc>
  <NameInc>x</NameInc>
  <SensibParamDir>0.0985765205323673732</SensibParamDir>
  <SensibParamInv>0.577266441576786526</SensibParamInv>
  <SensibParamVar>0.00228406097850316443</SensibParamVar>
</SensibDateOneInc>
```

```
...
<SensibDateOneInc>
  <NameBloc>Cal0</NameBloc>
  <NameInc>F</NameInc>
... <SensibParamVar>15.0011827674349423</SensibParamVar>
</SensibDateOneInc>
<SensibDateOneInc>
  <NameBloc>Cal0</NameBloc>
  <NameInc>PPx</NameInc>
... <SensibParamVar>1.68730018610615495</SensibParamVar>
</SensibDateOneInc>
...
<SensibDateOneInc>
  <NameBloc>Ima0</NameBloc>
  <NameInc>Cx</NameInc>
  <SensibParamDir>0.10381674122562666</SensibParamDir>
  <SensibParamInv>1.08339374408743527</SensibParamInv>
  <SensibParamVar>0.00384393697511320725</SensibParamVar>
</SensibDateOneInc>
...
...
```

## **Part IV**

# **Documentation utilisateur**



# Chapter 27

## Mécanismes Généraux

Ce chapitre décrit différents mécanismes généraux à MICMAC qu'il est nécessaire de maîtriser avant de pouvoir aborder la spécification d'un certain nombre de tags.

### 27.1 Généralités et notations

Cette section, ou un sous-ensemble, sera sans doute transférée ultérieurement dans la partie algorithmique.

#### 27.1.1 Boîtes englobantes

Une boîte est caractérisée par  $P^- = (x^-, y^-)$  et  $P^+ = (x^+, y^+)$ , elle définit la zone du plan :

$$[x^-, x^+] \otimes [y^-, y^+] \quad (27.1)$$

Si  $E$  est un ensemble de point, on note  $B^x(E)$  la boîte englobante de  $E$ . Si  $P_t$  est un point, on note  $B^x(P_t)$  la boîte contenant le singleton . On note  $\emptyset$  la boîte vide.

La boîte qui englobe deux boîtes  $B_1$  et  $B_2$  est notée  $B_1 + B_2 = B^x(B_1 \cup B_2)$ . On pose  $\emptyset + B = B$ .

L'opération est évidemment associative et commutative et idempotente. Si  $B_k, k \in [1 N]$  est une collection de boîtes, on pose :

$$\sum_{k \in [1 N]} B_k = B_1 + B_2 + \cdots + B_N \quad (27.2)$$

Si  $B_k, k \in [1 N]$  est une collection de boîtes, avec  $N \geq 2$  on définit  $\sqsupseteq B_k$  la boîte obtenue en prenant pour valeurs limites les min et max à l'exclusion des valeurs extrêmes. Par exemple, le  $x^-$  de cette boîte est la deuxième plus petite valeur des  $x^-$  des  $B_k$ .

### 27.2 Géométries

#### 27.2.1 Géométries intrinsèque et de restitution

De manière générale, la spécification de la géométrie  $\pi_k$  d'une image, se fait sous MICMAC par la description de deux transformations :

- une transformation intrinsèque  $\dot{\pi}_k$  de  $\mathcal{T} \otimes \mathcal{E}_{px}$  dans  $\mathcal{I}_k$ ; cette transformation correspond aux caractéristiques propres du "chantier" (par exemple poses et calibrations internes des caméras, système de coordonnées géodésique dans lequel sont exprimées les poses);
- une transformation de restitution  $\pi_T$  de  $\mathcal{T} \otimes \mathcal{E}_{px}$  dans  $\mathcal{T} \otimes \mathcal{E}_{px}$  qui permet d'effectuer les calculs dans un espace différent de celui du chantier;

On a la relation :

$$\pi_k = \dot{\pi}_k \circ \pi_T \quad (27.3)$$

Les motivations pour effectuer le calcul de mise en correspondance dans une géometrie différente de la géometrie intrinsèque peuvent être de deux natures :

- **formatage**; dans le cas d'une prise de vue aérienne, par exemple, il est courant que les fichiers d'orientations soient donnés dans un repère euclidien local alors que l'on souhaite exploiter le MNT résultat dans un référentiel géodésique (lambert ou autre); plutôt que de faire un basculement *a posteriori*, il est plus simple et plus précis de demander à MICMAC d'effectuer directement tous les calculs dans le système dans lequel seront exploitées les données;
- **algorithmiques**; par exemple, lors d'une mise en correspondance avec peu d'images (typiquement 2 ou 3), l'expérience et la théorie montrent que l'on obtient des résultats de mise en correspondance plus fiables lorsqu'il existe une image "maîtresse"<sup>1</sup>; dans ce contexte il peut être intéressant de faire les calculs dans une géométrie dans laquelle les "verticales" sont les faisceaux issus d'une image à spécifier (l'image "maîtresse").

Les termes utilisés par MICMAC pour désigner ces géométries sont un héritage des prises de vue aérienne standard et sont un peu impropre dans le cadre général :

- *géométrie image* pour la géométrie intrinsèque;
- *géométrie terrain* pour la géométrie de restitution;

## 27.2.2 Géométrie intrinsèque (image)

### 27.2.2.1 Description générale

Les géométries intrinsèques actuellement connues par MICMAC sont :

- `<eGeomImageOri>` correspondant à une géométrie sténopée stockée selon le format *historique Ori* du MATIS;
- `<eGeomImageModule>` correspondant à une géométrie spécifiée par l'utilisateur sous forme de librairie dynamique (permettant de représenter de manière générique toutes les géométries dérivées d'un capteur physique par un projection  $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ );
- `<eGeomImage_Epip>` correspondant à une géométrie épipolaire classique;
- `<eGeomImageDHD_Px>`, `<eGeomImage_Hom_Px>` `<eGeomImageDH_Px_HD>` correspondant à différentes variantes de géométries qui ne sont pas directement liées à une position dans l'espace du capteur;
- `<eNoGeomIm>` valeur spéciale, permettant d'utiliser certaines fonctionnalités de MICMAC, non liées à la mise en correspondance, ne nécessitant aucune connaissance de la géométrie;

### 27.2.2.2 Géométrie `<eGeomImageOri>`

Il s'agit de la géométrie sténopée classique, soit :

$$\dot{\pi}_k(P) = \text{Dist}_k^{-1}(Pp_k + F_k * \pi_0(\mathcal{R}_k(\mathcal{C}_k - P))) \quad (27.4)$$

Avec pour l'image  $k$ :

- paramètres extrinsèques de la caméra; centre optique  $\mathcal{C}_k$  ; matrice de rotation  $\mathcal{R}_k$  ;
- $\pi_0$  projection canonique  $\pi_0(x, y, z) = \frac{(x, y)}{z}$
- paramètres intrinsèques de la caméra (souvent indépendant de  $k$ ); distance focale  $F_k$  ; point principal  $Pp_k$ ; distortion  $\text{Dist}_k$

### 27.2.2.3 Géométrie `<eGeomImageModule>`

Cette valeur permet d'ajouter dans MICMAC n'importe quelle géométrie décrivant un système imageur caractérisable par une fonction de projection  $\mathbb{R}^3 \rightarrow \mathbb{R}^2$  et sa fonction "réciproque"  $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ .

Dans ce mode l'utilisateur indique à MICMAC le chemin d'accès à une librairie partagée qui sera chargée dynamiquement. C'est par cette librairie que des objets dérivant d'une classe abstraite `ModuleOrientation` seront créés. La classe `ModuleOrientation` définit dans son interface deux méthodes virtuelles permettant de définir  $\pi$  et  $\pi^{-1}$ .

Il existe déjà un module adapté au format grille mis en place par IGN-espace.

### 27.2.2.4 Géométries `<eGeomImage_Epip>`, `<eGeomImageDHD_Px>` ...

Ces géométries permettent de faire de la mise en correspondance "image à image", c'est à dire en ignorant la caractérisation géométrique (localisation en 3D) rigoureuse des capteurs imageurs.

Dans ce cas l'espace "terrain" est l'espace de la première image. La parallaxe est toujours bi-dimensionnelle et exprimée en pixel. Les variantes correspondent à différents prédicteurs de la fonction de correspondance à priori. La parallaxe étant exprimée en différentiel par rapport à ces prédicteur, cela permet d'un part de réduire

1. image pour laquelle,  $\pi_k$  ne dépendant pas de la parallaxe

les intervalles de recherche et d'autre part de corriger des distorsions trop fortes (pour que les "vignettes" de corrélation soient réellement superposables).

En notant  $D$  des distorsions,  $H$  des homographies, les fonctions de projection sont les suivantes :

- **<eGeomImage\_Epip>**  $\hat{\pi}(x, y, u, v) = (x + u, y + v)$ , cette géométrie est donc adaptée à la mise en correspondance des images mise en épipolaire; la parallaxe "transverse" permet, le cas échéant, de modéliser des imperfections dans la géométrie épipolaire;
- **<eGeomImage\_Hom\_Px>**  $\hat{\pi}(x, y, u, v) = H(x, y) + (u, v)$ , correspondance *a priori* modélisée par une homographie, utile quand on n'a pas mieux (par exemple en initialisation de la "calibration par points de liaisons");
- **<eGeomImageDHD\_Px>**  $\hat{\pi}(x, y, u, v) = D_1(H(D_2^{-1}(x, y)) + (u, v))$ , correspondance *a priori* modélisée par une homographie une fois les distorsions corrigées; c'est la modélisation utilisée pour la superposition des canaux de la caméra numérique;
- **<eGeomImageDH\_Px\_HD>**  $\hat{\pi}(x, y, u, v) = D_1(H_1(H_2^{-1}(D_2^{-1}(x, y)) + (u, v)))$ ; géométrie pas encore supportée; l'intérêt serait de pouvoir modéliser la géométrie épipolaire à la volée uniquement avec les briques de base  $D$  et  $H$ ;

### 27.2.2.5 Lecture des paramètres de géométrie

Pour aller lire les paramètres de géométrie, MicMac fonctionne ainsi :

- **eGeomImageOri** un fichier *ori* par image doit être lu, son nom est calculé à partir du tag **<PatNameGeom>** (voir 28.2.2.2);
- **eGeomImageModule** un fichier par image doit être lu, son nom est calculé à partir du tag **<PatNameGeom>** (voir 28.2.2.2), ce nom de fichier sera passé en argument à la fonction contenu dans la librairie dynamique;
- quand une distorsion est nécessaire (**eGeomImageDHD\_Px**), celle-ci doit être disponible sous forme d'un fichier **xml** au format décrit en A.2, le nom est calculé à partir du tag **<PatNameGeom>**; la valeur **GridDistId** est conventionnelle pour signifier une grille identité;
- quand une homographie est nécessaire (**eGeomImageDHD\_Px** et **eGeomImage\_Hom\_Px**) celle-ci est calculée à partir de points homologues fournis par l'utilisateur; ces points homologues doivent être contenus dans un fichier dont le nom est calculé à partir du tag **<NomHomologues>** et qui doit être au format décrit dans A.3; pour passer de  $N$  points homologues à une homographie, on utilise les conventions suivantes :
  - $N = 0$  : identité;
  - $N = 1$  : translation;
  - $N = 2$  : similitude;
  - $N = 3$  : affinité;
  - $N = 4$  : homographie exacte;
  - $N > 4$  : homographie ajustée par moindres carrés;

## 27.2.3 Géométries de restitution (terrain)

### 27.2.3.1 Description générale

Les géométries de restitution actuellement connues par MICMAC sont :

- **<eGeomMNTCarto>** **<eGeomMNTEuclid>** : géométries terrain classique : le relief est restitués en coordonnées cartographiques ou dans le repère euclidien local;
- **<eGeomMNTfaisceauIm1ZTerrain\_Px1D>** **<eGeomMNTfaisceauIm1ZTerrain\_Px2D>** : géométries où la première image est maîtresse; (les "verticales" sont les rayons perspectifs de la première image);
- **<eGeomMNTfaisceauIm1PrCh\_Px1D>** **<eGeomMNTfaisceauIm1PrCh\_Px2D>** : géométries où la première image est maîtresse avec une dynamique en  $\frac{1}{z}$ ;
- **<eGeomPxBiDim>** valeur qui doit être associée aux géométries images (décrisées en 27.2.2.4);
- **<eNoGeomMNT>** valeur spéciale, homologue de **<eNoGeomIm>**, décrite en 27.2.2.1

### 27.2.3.2 Géométries terrain

La distinction entre **<eGeomMNTCarto>** et **<eGeomMNTEuclid>** n'est utile aujourd'hui que lorsque la géométrie intrinsèque est conique fournie au format *ori*.

### 27.2.3.3 Géométries **<eGeomMNTfaisceauIm1ZTerrain\_Px1D>**

Ces géométries de restitution sont accessibles lorsque la géométrie intrinsèque correspond à une modélisation 3d du capteur (**<eGeomImageOri>** et **<eGeomImageModule>**).

Soit  $\hat{\pi}_1$ , la fonction de projection intrinsèque de la première image, on pose alors pour **<eGeomMNTfaisceauIm1ZTerrain\_Px1D>**:

$$\pi_k(x, y, z) = \dot{\pi}_k(\dot{\pi}_1^{-1}(x, y, z), z) \quad (27.5)$$

Une conséquence immédiate est que :

$$\pi_1(x, y, z) = (x, y) \quad (27.6)$$

Ces formules ont une interprétation intuitive simple :

- l'espace de restitution est l'espace des coordonnées de la première image;
- $(\dot{\pi}_1^{-1}(x, y, z), z)$  est le point d'altitude  $z$  situé sur le rayon perspectif issu de la première image au point  $x, y$ ;
- ce point est reprojeté sur le  $k^{ième}$  image par  $\dot{\pi}_k$ ;

#### 27.2.3.4 Géométries <eGeomMNTFaisceauIm1ZTerrain\_Px2D>

Le mode <eGeomMNTFaisceauIm1ZTerrain\_Px2D> est identique au précédent, avec en plus une parallaxe transverse permettant de corriger d'éventuelles imprécisions dans la modélisation du capteur. Soit  $\pi_k^{1D}$  la projection associée à <eGeomMNTFaisceauIm1ZTerrain\_Px1D>, on a :

$$\pi_k(x, y, z, t) = \pi_k^{1D}(x, y, z) + t * (\bar{u}_k, \bar{v}_k) \quad (27.7)$$

Où  $(\bar{u}_k, \bar{v}_k)$  est une direction de parallaxe transverse définie comme la direction orthogonale à la projection dans l'image  $k$  du rayon perspectif issu du centre du "chantier":

$$(u_k, v_k) = \frac{\partial \pi_k^{1D}(x, y, z)}{\partial z} (x_0, y_0, z_0) \quad (27.8)$$

$$(\bar{u}_k, \bar{v}_k) = \frac{(v_k, -u_k)}{\sqrt{u_k^2 + v_k^2}} \quad (27.9)$$

Pour  $k = 1$  on conserve la définition précédente:

$$\pi_1(x, y, z) = (x, y) \quad (27.10)$$

#### 27.2.3.5 Géométries <eGeomMNTFaisceauIm1PrCh\_Px?D>

Il s'agit de variantes des deux géométries précédentes spécialisées pour le cas particulier où la géométrie intrinsèque est conique et où il y a éventuellement de grandes variations relatives de la profondeur de champs (scènes terrestres par exemple). Les différences sont les suivantes :

- tout se passe comme si avant d'utiliser l'équation 27.5 on se mettait dans un repère, d'origine le centre optique et où l'axe optique et la verticales sont confondus (si bien que  $z$  est une profondeur de champ);
- la dynamique est en  $\frac{1}{z}$ , afin que les variations de  $z$  soient proportionnelles à des parallaxes;

Soit  $(x, y, z)$  un point de l'espace,  $(x, y)$  est un point de l'image 1, soit  $(\tilde{x}, \tilde{y}, 1)$  la direction du rayon perspectif dans le repère image; avec les notations de l'équation 27.4, on pose :

$$\pi_k(x, y, z) = \dot{\pi}_k(C_1 + R_1^t \frac{(\tilde{x}, \tilde{y}, 1)}{z}) \quad (27.11)$$

Ces géométries sont surtout utilisées pour le calcul de points homologues denses.

### 27.2.4 Caractéristiques liées aux géométries

#### 27.2.4.1 Combinaisons de géométries, dimension de parallaxe

Comme indiqué en 27.2.1, MicMac prévoit une approche "matricielle" pour la description de la géométrie où la géométrie finale est obtenue par combinaison d'une géométrie intrinsèque et d'une géométrie de restitution. En pratique, toutes les combinaisons "intrinsèque/restitution" ne conduisent pas forcément à une géométrie cohérente et ne sont donc pas autorisées. Le tableau 27.1 synthétise les combinaisons de géométries autorisées et les dimensions de parallaxes associées.

On voit que pour les géométries purement images (celle décrites en 27.2.2.4) la séparation en deux géométries n'a pas de sens, c'est pour cela que ces géométries sont conventionnellement associées à eGeomPxBiDim.

#### 27.2.4.2 Unités de parallaxes

Une conséquence des géométries variées gérées par MicMac est que les parallaxes ont des significations très différentes suivant les cas. L'unité en laquelle est exprimée la parallaxe varie donc en fonction de la géométrie de restitution. Le tableau 27.2 synthétise ces variations.

	eGeomMNTCarto	eGeomMNTEuclid	eGeomMNTFaisceauIm1ZTerrain_Px1D	eGeomMNTFaisceauIm1ZTerrain_Px2D	eGeomMNTFaisceauIm1PrCh_Px1D	eGeomMNTFaisceauIm1PrCh_Px2D	eGeomPxBiDim	eNoGeomMNT
eGeomImageOri	1	1	1	2	1	2	X	X
eGeomImageModule	X	1	1	2	X	X	X	X
eGeomImageDHD_Px	X	X	X	X	X	X	2	X
eGeomImage_Hom_Px	X	X	X	X	X	X	2	X
eGeomImageDH_Px_HD	X	X	X	X	X	X	2	X
eGeomImage_Epip	X	X	X	X	X	X	2	X
eNoGeomIm	X	X	X	X	X	X	X	0

Figure 27.1: Combinaisons autorisées entre géométrie intrinsèque et géométries de restitution. Dimensions de parallaxe associées. X : combinaison interdite.

Géométrie	Unité Px1	Unité Px2	Unité terrain
eGeomMNTCarto	Mètre	x	Mètre
eGeomMNTEuclid	Mètre	x	Mètre
eGeomMNTFaisceauIm1ZTerrain_Px1D	Mètre	x	Pixel
eGeomMNTFaisceauIm1ZTerrain_Px2D	Mètre	Pixel	Pixel
eGeomMNTFaisceauIm1PrCh_Px1D	Mètre <sup>-1</sup>	x	Pixel
eGeomMNTFaisceauIm1PrCh_Px2D	Mètre <sup>-1</sup>	Pixel	Pixel
eGeomPxBiDim	Pixel	Pixel	Pixel
eNoGeomMNT	x	x	x

Figure 27.2: Unités des parallaxe et des coordonnées terrain en fonction des géométries de restitution

## 27.3 Patrons de sélection et de transformations de chaînes

### 27.3.1 Utilisation avec un nom

A plusieurs endroits du fichier <ParamMICMAC.xml>, MicMac utilise un mécanisme de patrons de chaînes de caractères. Cette fonctionnalité peut être utilisée soit pour spécifier un ensemble de chaînes, soit pour spécifier un mécanisme d'association automatique entre deux familles de chaînes; ces deux utilisations peuvent être conjointes.

Un exemple extrait d'un fichier MicMac :

```
<Images >
  <Im1> ess238.10_229.ng_16b.tif </Im1>
  <Im2> ess238.10_228.ng_16b.tif </Im2>
  <ImPat> .*\.tif </ImPat>
</Images>
<NomsGeometrieImage>
  <PatternSel >(.*)\.tif</PatternSel>
  <PatNameGeom > $1.ori </PatNameGeom>
</NomsGeometrieImage>
```

Le premier patron, <ImPat> est une utilisation en sélection, il permet de rajouter dans la liste des images du chantier tous les fichiers dont le nom porte l'extension ".tif".

Le deuxième patron, <PatternSel > est une utilisation en association qui, couplée avec <PatNameGeom>, permet d'indiquer à MicMac que le nom du fichier de géométrie associé au fichier image s'obtient en remplaçant l'extension ".tif" par ".ori" ; cela fonctionne notamment parce que \$k, dans le pattern de substitution, signifie "doit être remplacé par la k<sup>ième</sup> expression parenthésée du pattern de sélection".

Ce mécanisme est tout à fait standard. Les patrons sont utilisés selon ce que l'on appelle les expressions régulières modernes dont la description détaillée peut être, par exemple, trouvée en recherchant `regex` dans les différentes pages du manuel `unix`.

### 27.3.2 Utilisation avec deux noms

*Les fonctionnalités décrites ici sont surtout utiles lorsque MicMac est rappelé par un programme maître sur un grand nombre de couple d'images.*

Il existe des utilisations MicMac où l'on souhaite que des noms calculés dépendent non pas seulement du nom d'une seule image mais de plusieurs (généralement les images 1 et 2). La solution que propose alors MicMac pour répondre à ces besoins est de fournir une expression régulière qui sera appariée sur la concaténation des noms <Im1> et <Im2>. Pour être sûr que l'appariement soit non ambigu quelque soient <Im1> et <Im2> on rajoute un séparateur entre les noms, ce séparateur peut être fourni par l'utilisateur, par défaut il vaut "@".

```
<NomsGeometrieImage>
  ...
  <PatternNameIm1Im2>
    DSC_Gray_(.*)\.tif@DSC_Gray_(.*)\.tif
  </PatternNameIm1Im2>
  <PatNameGeom> OriRelStep1_$1_For_$2.ori </PatNameGeom>
  ...
</NomsGeometrieImage>

<CalcNomChantier>
  <PatternSelChantier >
    DSC_Gray_(.*)\.tif###DSC_Gray_(.*)\.tif
  </PatternSelChantier>
  <PatNameChantier >
    ChantierHelico2_$1-$2_
  </PatNameChantier>
  <SeparateurChantier > ### </SeparateurChantier>
</CalcNomChantier>
```

L'extrait de fichier xml ci-dessous illustre ce mécanisme. Appelé dans un contexte où, par exemple, Im1=DSC\_Gray\_5255.tif et Im2=DSC\_Gray\_5256.tif, on aura :

- pour <PatternNameIm1Im2>, c'est DSC\_Gray\_5255.tif@DSC\_Gray\_5256.tif qui va être apparié sur DSC\_Gray\_(.\*)\\*.tif@DSC\_Gray\_(\*)\\*.tif; PatNameGeom générera alors le nom OriRelStep1\_5255\_For\_5256.ori; pour ce tag, le séparateur est d'office @;
- pour le calcul du nom de chantier, le séparateur est fourni par l'utilisateur; ici <PatNameChantier> générera ChantierHelico2\_5255-5256\_;

## 27.4 Librairies Dynamiques

### 27.4.1 Fonctionnement Général

Afin de pouvoir spécialiser MicMac, de manière "fine", en rajoutant des morceaux de codes, sans avoir à les recompiler dans l'environnement MicMac, il est offert un mécanisme de chargement de librairies dynamiques pour certaines caractéristiques de MicMac.

La manière d'écrire une librairie dynamique est décrite dans la partie *Documentation programmeur*, ici on se limite à la description du mécanisme d'insertion d'un librairie dynamique, supposée écrite correctement, via le fichier de paramètres effectifs.

Pour spécifier un module chargé dynamiquement, il suffit de de spécifier deux noms :

- un nom de fichier, qui est le nom absolu de la librairie partagée qui doit être chargée;
- un nom de symbole qui servira de point d'entrée à MicMac pour aller créer les objets correspondants une fois la librairie chargée (étant entendu que la même librairie partagée peut contenir plusieurs services);

Actuellement deux fonctionnalités sont accessibles via des librairies dynamiques :

- la géométrie des images ;
- la gestion de la pyramide d'image;

### 27.4.2 Utilisation pour la géométrie

A titre d'exemple pour la géométrie :

- le tag <ModuleGeomImage>, à l'intérieur de la section <Section\_PriseDeVue>, permet de spécifier une librairie dynamique décrivant la géométrie des capteurs;
- à l'intérieur, le tag <NomModule> permet de spécifier le nom du fichier contenant la librairie et le tag <NomGeometrie> permet de spécifier un point d'entrée dans la librairie;
- ces valeurs seront recherchées lorsque la géométrie image vaut <eGeomImageModule>.

Grégoire Maillet a écrit une librairie dynamique permettant de prendre en compte le format grille d'IGN-espace; les lignes ci-dessous sont extraites d'un fichier utilisant ce module pour faire fonctionner MicMac avec des images au format grille.

```
<Section_PriseDeVue >
  <GeomImages> eGeomImageModule </GeomImages>
  <ModuleGeomImage>
    <NomModule>./applis/MICMAC/ModuleGrille/.libs/libmodulegrille.so</NomModule>
    <NomGeometrie> Grille </NomGeometrie>
  </ModuleGeomImage>
  <NomsGeometrieImage>
    <PatternSel >(.*)\.tif</PatternSel>
    <PatNameGeom > $1.GRI </PatNameGeom>
  </NomsGeometrieImage>
  ...
</Section_PriseDeVue>
```

### 27.4.3 Utilisation pour les pyramides

Un module pour utiliser le format JPEG-2000 comme format de pyramide d'images a été écrit. Voir Grégoire Maillet et/ou Gilles Martinoty pour ce module.

## 27.5 Types réutilisés

Cette section décrit des types d'arbres qui, après avoir été définis, sont utilisés plusieurs fois selon le mécanisme décrit en 10.4.

### 27.5.1 Le type FileOriMnt

```
<FileOriMnt Nb="1" Class="true" ToReference="true">
  <NameFileMnt Nb="1" Type="std::string">           </NameFileMnt>
  <NameFileMasque Nb="?" Type="std::string">         </NameFileMasque>
  <NombrePixels Nb="1" Type="Pt2di">                </NombrePixels>
  <OriginePlani Nb="1" Type="Pt2dr">               </OriginePlani>
  <ResolutionPlani Nb="1" Type="Pt2dr">             </ResolutionPlani>
  <OrigineAlti Nb="1" Type="double">                </OrigineAlti>
  <ResolutionAlti Nb="1" Type="double">              </ResolutionAlti>
  <NumZoneLambert Nb="?" Type="int">                </NumZoneLambert>
  <Geometrie Nb="1" Type="eModeGeomMNT">            </Geometrie>
  <OrigineTgtLoc Nb="?" Type="Pt2dr">              </OrigineTgtLoc>
</FileOriMnt>
```

Le type `FileOriMnt` permet de décrire un modèle numérique de terrain sous la forme d'un fichier image et des métadonnées associées. Il s'agit essentiellement d'un *xml-isation* de l'ancien format `ori` pour les fichiers MNT. Il contient:

- un nom de fichier image `NameFileMnt` ainsi qu'un nom optionnel, `NameFileMasque`, de fichier masque (généralement sur 1 bit);
- soit  $I, J$  un pixel du fichier `NameFileMnt`, il est valide si  $NameFileMasque(I, J) = true$  (toujours valide si `NameFileMasque` n'est pas spécifié);
- soit  $K = NameFileMnt(I, J)$ ;
- soit  $P = OriginePlani + ResolutionPlani * (I, J)$ ;
- soit  $Z = OrigineAlti + ResolutionAlti * K$ ;
- alors le point  $(P, Z)$  est un point du MNT dans le système de coordonnées spécifié par `Geometrie` ainsi que les paramètres optionnels `NumZoneLambert` et `OrigineTgtLoc`.

### 27.5.2 Le type SpecFitrageImage

## 27.6 Gestion des erreurs

### 27.6.1 Bugs

### 27.6.2 Erreurs mal signalées

### 27.6.3 Erreurs cataloguées

# Chapter 28

## Sections hors mise en correspondance

### 28.1 Section Terrain

```
<Section_Terrain Nb="1">
    <IntervAltimetrie Nb="?"> ...
    <IntervParalaxe Nb="?"> ...
    <Planimetrie Nb="?"> ...
    <RugositeMNT Nb="?"> ...
</Section_Terrain>
```

Cette section contient les informations liées au terrain lui même, indépendamment de la façon dont il est imaginé.

#### 28.1.1 <IntervParalaxe> et <IntervAltimetrie>

```
...
    <IntervAltimetrie Nb="?">
        <ZMoyen Nb="?" Type="double"> ...
        <ZIncCalc Nb="1" Type="double"> ...
        <ZIncZonage Nb="?" Type="double"> ...
        <MNT_Init Nb="?"> ...
    </IntervAltimetrie>

    <IntervParalaxe Nb="?">
        <Px1Moy Nb="?" Type="double"> ...
        <Px2Moy Nb="?" Type="double"> ...
        <Px1IncCalc Nb="1" Type="double"> ...
        <Px2IncCalc Nb="1" Type="double"> ...
        <Px1IncZonage Nb="?" Type="double"> ...
        <Px2IncZonage Nb="?" Type="double"> ...
    </IntervParalaxe>
...

```

Ces deux sections jouent le même rôle, <IntervParalaxe> est utilisé lorsque la parallaxe est de dimension 2 et <IntervAltimetrie> lorsqu'elle est de dimension 1. Elle ne diffère que par <MNT\_Init>, spécifique à <IntervAltimetrie> et décrit en 28.1.1.4.

L'unité des valeurs exprimées dans cette section est celle donnée par le tableau 27.2 sauf lorsque cette unité est du Mètre<sup>-1</sup> auquel cas les valeurs sont exprimées en mètre.

##### 28.1.1.1 Valeurs moyennes de parallaxe

Les tags sont <ZMoyen> ( $d = 1$ ) ou <Px1Moy> et <Px2Moy> ( $d = 2$ ).

	Z	Px1	Px2
Moyenne	ZMoyen	Px1Moy	Px2Moy
Incert-Calc	ZIncCalc	Px1IncCalc	Px2IncCalc
Incert-Zone	ZIncZonage	Px1IncZonage	Px2IncZonage

Figure 28.1: Equivalence de noms sur les intervalles de parallaxe

Ces tags fixent la valeur moyenne de la parallaxe; elles ont donc une influence directe sur la zone de recherche explorée lors du premier niveau de la pyramide de résolution. Accessoirement, elles ont une influence sur le formatage du résultat (le contenu des fichiers est exprimé en relatif par rapport à cette valeur moyenne).

Dans le fichier de spécifications, on peut voir que l’arité de ces valeurs est ”?”; ces valeurs sont optionnelle lorsque la géométrie intrinsèque fournit une valeur moyenne par défaut. Plus précisément :

- optionnelle pour les géométries purement image, la valeur par défaut est 0 les paramètres annexes (par ex : distorsion et homographie) étant supposés fournir une bonne prédiction;
- optionnelle pour `eGeomImageOri`, le format `Ori` contenant une information d’altitude moyenne;
- pour `<eGeomImageModule>` ca dépend de ce qui est implémenté dans la librairie dynamique . . . ; obligatoire pour l’implémentation actuelle du format grille d’IGN-espace;

On notera  $\tilde{P}_x$  cette valeur.

### 28.1.1.2 Incertitude de calcul

Les tags sont `<ZIncCalc>` ( $d = 1$ ) ou `<Px1IncCalc>` et `<Px2IncCalc>` ( $d = 2$ ) et ils sont obligatoires. Ces valeur permettent de définir les deux nappes encadrantes utilisées pour définir la zone de recherche au premier niveau de la pyramide.

Par exemple, pour  $d = 1$ , la nappe initiale est définie par l’intervalle `[ZMoyen-ZIncCalc , ZMoyen+ZIncCalc]`.

### 28.1.1.3 Incertitude pour le calcul d’emprise

Les tags sont `<ZIncZonage>` ( $d = 1$ ) ou `<Px1IncZonage>` et `<Px2IncZonage>` ( $d = 2$ ). Il sont facultatifs et lorsqu’ils sont omis, ils prennent la même valeur que `<ZIncCalc>` . . . .

Ces valeurs servent à contrôler l’emprise du chantier lorsqu’elle celle ci est calculée automatiquement par MicMac voir 28.1.2.2

On notera  $P_x^Z$  cette valeur.

### 28.1.1.4 MNT initial

```
<MNT_Init Nb="?">
  <MNT_Init_Image Nb="1" Type="std::string"> </MNT_Init_Image>
  <MNT_Init_Xml Nb="1" Type="std::string">   </MNT_Init_Xml>
  <MNT_Offset Nb="?" Type="double" Def="0.0"> </MNT_Offset>
</MNT_Init>
```

Cette structure permet d’utiliser un MNT pour initialiser le calcul. La structure est fille de `<IntervAltimetrie>` et n’est accessible qu’en dimension 1. Les trois champs qui la composent sont :

- `<MNT_Init_Image>` le nom du fichier image contenant le MNT;
- `<MNT_Init_Xml>` le nom du fichier `xml` contenant le géoréférencement du MNT au format `<FileOriMnt>` (décrit en A.4); les contraintes sont les mêmes qu’en 28.1.2.4;
- `<MNT_Offset>` un éventuel offset rajouté au MNT avant de l’utiliser comme valeur initiale; cette valeur est utile lorsque , un fois le MNT connu, les intervalle d’incertitude sur le relief sont asymétriques; typiquement, les altitudes attendues au dessus du MNT peuvent être élevée car elle sont dues à tous le sursol, alors que celle en dessous restent faible car elles ne refléteront que l’erreur sur le MNT lui même ; si par exemple, l’intervalle d’incertitude est  $[-20, 80]$ , on pourra fixer `<MNT_Offset>` à 30 et `<ZIncCalc>` à 50;

## 28.1.2 Planimétrie

```
<Planimetrie Nb="?">
  <BoxTerrain Nb="?" Type="Box2dr"> </BoxTerrain>
  <ListePointsInclus Nb="*">
    <Pt Nb="+" Type="Pt2dr"> </Pt>
    <Im Nb="1" Type="std::string"> </Im>
```

```

</ListePointsInclus>
<ResolutionTerrain Nb="?" Type="double"> </ResolutionTerrain>
<MasqueTerrain Nb="?"> ... </MasqueTerrain>
<RecouvrementMinimal Nb="?" Type="double"> </RecouvrementMinimal>
</Planimetrie>

```

La section `<Planimetrie>` ainsi que tous ses sous-sections sont optionnelles.

### 28.1.2.1 Calcul de l'emprise spécifiée

L'emprise est spécifiée par l'utilisateur lorsque le champs `<BoxTerrain>` a une valeur ou que la liste `<ListePointsInclus>` est non vide.

Chaque élément  $l$  de la liste  $L$  de `<ListePointsInclus>`, permet de spécifier une image  $I_l$ , et un point  $P_l$  de cette image dont on souhaite que l'homologue terrain soit présent dans l'emprise terrain.

On note  $B^T$  la boîte englobante qui vaut `<BoxTerrain>` quand elle est définie et  $\emptyset$  sinon.

Lorsque l'emprise est spécifiée elle vaut:

$$B^T + \sum_{l \in L} B^x(\pi_l^{-1}(P_l, \tilde{P}_x)) \quad (28.1)$$

Conventionnellement, si `Im` vaut `Terrain` il s'agit d'un point terrain (au sens de la géométrie intrinsèque).

### 28.1.2.2 Calcul de l'emprise par défaut

Lorsqu'aucune emprise terrain n'est donnée MicMac en calcule une selon les spécifications suivantes: la boîte englobante de l'ensemble des points terrains qui, compte-tenu des incertitudes sur la parallaxes, soient susceptibles d'être vus dans au moins 2 images.

Formellement on calcule cette boîte par la formule suivante :

$$\bigcup_{k \in [1, N]} (B^x(\pi_k^{-1}(\mathcal{I}_k, P_x^Z)) + B^x(\pi_k^{-1}(\mathcal{I}_k, -P_x^Z))) \quad (28.2)$$

### 28.1.2.3 Résolution terrain

Le tag `<ResolutionTerrain>` permet de spécifier la résolution à laquelle l'espace terrain est discrétré. Cette valeur est exprimée dans l'unité terrain associée à la géométrie de restitution telle qu'exprimée en 27.2.

Lorsque cette valeur est omise, une valeur est calculée en faisant la moyenne des résolutions qui sont indiquées par les prises de vues. Ces valeurs par défaut sont les suivantes:

- pour les géométries purement images, la valeur par défaut est 1.0;
- pour la géométrie `eGeomImageOri` une valeur est calculée en tenant compte de la focale, de la hauteur de vol et de l'altitude sol;
- pour `<eGeomImageModule>` ça dépend de ce qui est implémenté dans la librairie dynamique ... ; dans l'implémentation actuelle des géométries grilles, une valeur est calculée en utilisant la grille au centre de l'image;

### 28.1.2.4 Masque terrain

```

<MasqueTerrain Nb="?">
  <MT_Image Nb="1" Type="std::string"> </MT_Image>
  <MT_Xml Nb="1" Type="std::string"> </MT_Xml>
</MasqueTerrain>

```

La structure `<ImMasqImported>` permet de spécifier un masque utilisateur qui va se superposer au masque calculé automatiquement par MicMac (voir 29.2.2). Ce masque doit utiliser le même système de coordonnées que la géométrie utilisée sur le chantier (par exemple même zone lambert ...), par contre il n'a pas nécessairement la même origine ou la même échelle que le chantier.

Le tag `<MT_Image>` spécifie un nom de fichier image et le tag `<MT_Xml>` un nom de fichier `xml` permettant de géo-référencer le fichier image. Le fichier `xml` doit être au format `<FileOriMnt>` (voir A.4, ceci implique qu'un résolution et un origine alti doivent être spécifiées même si elles ne seront pas utilisées).

### 28.1.2.5 Recouvrement minimal

Ce tag permet de spécifier une valeur minimale de l'emprise terrain. Cette valeur est spécifiée en proportion de la taille de la première image (par exemple, 0.01 correspond à  $160Ko$  avec des images  $4000 * 4000$ ).

Ce tag a été rajouté lors de l'utilisation de MicMac pour le calcul automatique de points homologues, il permet d'arrêter MicMac au plus tôt lorsque la zone de recouvrement est trop petite et risque de créer des dégénescences ultérieure.

Dès que la taille de zone a été calculée, si elle est inférieure à la valeur seuil, MicMac s'arrête en générant l'erreur cataloguée **eErrRecouvrInsuffisant** (voir 27.6.3).

### 28.1.3 Paramètres liés à la "rugosité"

```
<RugositeMNT Nb=?>
  <EnergieExpCorrel Nb=? Type="double" Def="-2.0">      </EnergieExpCorrel>
  <EnergieExpRegulPlani Nb=? Type="double" Def="-1.0">    </EnergieExpRegulPlani>
  <EnergieExpRegulAlti Nb=? Type="double" Def="-1.0">     </EnergieExpRegulAlti>
</RugositeMNT>
```

Trois paramètres liés à la rugosité du terrain qui permettent de spécifier comment les paramètre de régularisation doivent évoluer en fonction du facteur d'échelle.

Non détaillés pour l'instant, car je ne suis pas convaincu qu'il y ait le moindre intérêt à utiliser autre chose que les valeurs par défaut.

## 28.2 Section Prise de Vue

```
<Section_PriseDeVue Nb="1">
  <Images Nb="1"> ... </Images>

  <ValSpecNotImage/ Nb=? Type="int">
  <PrefixMasqImRes Nb=? Type="std::string" Def="MasqIm">
  <DirMasqueImages Nb=? Type="std::string" Def="">
  <MasqImageIn Nb="*">
    <OneMasqueImage Nb="*">> ... </OneMasqueImage>
  </MasqImageIn>

  <GeomImages/ Nb="1">
  <NomsGeometrieImage Nb="+">... </NomsGeometrieImage>
  <ModuleGeomImage Nb=?>... </ModuleGeomImage>
  <NomsHomomologues Nb=?> ... </NomsHomomologues>
</Section_PriseDeVue>
```

Cette section décrit l'ensemble des caractéristiques du chantier liées à la prise de vue, c'est à dire à l'ensembles des images et de leur géométrie.

### 28.2.1 Images

#### 28.2.1.1 Ensemble des images

```
<Images Nb="1">
  <Im1 Nb=? Type="std::string">   </Im1>
  <Im2 Nb=? Type="std::string">   </Im2>
  <ImPat Nb="*" Type="std::string"> </ImPat>
</Images>
```

Cette section permet de spécifier l'ensemble des images qui constituent le chantier. Les noms et patron de noms sont toujours indiqués en relatif par rapport à la directory d'instalation des images (voir 28.5.1).

**<Im1>** et **<Im2>** ( facultatifs ) sont des noms d'images tandis que **<ImPat>** correspond à une liste de pattern de sélection. C'est l'ensemble des images spécifiées qui constituera la sélection retenu pour le chantier. Les noms spécifiés peuvent se retrouver en plusieurs endroit, MicMac supprimera les duplicitas. La seule contrainte réelle est qu'il y ait en tout au moins 2 noms d'images différents.

**<Im1>** et **<Im2>** peuvent paraître redondants compte-tenu du mécanisme assez souple offert par les patrons **<ImPat>**. Les raison qui justifient la présence de ces tags sont les suivantes :

- Si la géométrie sélectionnée contient la notion d'images maîtresse la présence de <Im1> permet de spécifier laquelle des images sera maîtresse;
- lorsque MicMac est rappelé par un programme *maitre* pour générer des mises en correspondances sur un grand nombre de couples (aérotriangulation, superposition multi-spectrale), la présence de <Im1> et <Im2> permet de spécifier chaque couple d'image par ligne de commande (ce qui ne serait pas possible aujourd'hui avec les <ImPat> à cause de son arité, voir 10.6.1);
- en géométrie image, il est plus naturel de spécifier explicitement quelles sont <Im1> et <Im2> plutôt que de reposer sur l'ordre des noms ...

Supposons par exemple que la directory images contienne des fichiers Im000.tif Im001.tif ...Im999.tif, le code ci dessous sélectionnera les images de numéros compris entre 200 et 299 ou 400 et 499, et l'image maîtresse sera 293 (si cela a un sens pour la géométrie).

```
<Images>
  <Im1> Im293.tif  </Im1>
  <ImPat> Im2..\*.tif  </ImPat>
  <ImPat> Im4..\*.tif  </ImPat>
</Images>
```

### 28.2.1.2 Masque images

```
<PrefixMasqImRes Nb=?> Type="std::string" Def="MasqIm"      </PrefixMasqImRes>
<ValSpecNotImage Nb=?> Type="int"      </ValSpecNotImage>

<DirMasqueImages Nb=?> Type="std::string" Def=""> </DirMasqueImages>
<MasqImageIn Nb="*>
  <!-- En cas de match multiple, c'est le dernier qui compte -->
  <OneMasqueImage Nb="*>
    <PatternSel Nb="1" Type="cElRegex_Ptr">      </PatternSel>
    <!-- Si NomMasq=PasDeMasqImage -> Valeur Tag
         pour ne pas aller chercher de masque -->
    <NomMasq Nb="1" Type="std::string">      </NomMasq>
  </OneMasqueImage>
</MasqImageIn>
```

MicMac maintient pour chaque image un masque binaire indiquant la zone valide de l'image. Lors du calcul d'un score de corrélation, dans la phase de mise en correspondance, pour chaque point et chaque parallaxe seules seront conservées les images telles que tous les points de la vignette se projettent en des points valides. Le masque est représenté et mémorisé explicitement par une pyramide d'image. Les pyramides créées sont stockées dans le répertoire <TmpPyr> et les noms des fichiers de masque sont construits en rajoutant au nom du fichier image le préfixe <PrefixMasqImRes> (qui par défaut vaut **MasqIm**).

On peut spécifier plusieurs masques qui se superposent. Soit par exemple *Im* une image, l'ensemble des masques se superposant est construit de la manière suivante :

- si on ne spécifie rien, toute la zone de l'image est valide.
- si l'on spécifie une valeur *Val* dans <ValSpecNotImage>, un premier masque est construit correspondant aux seuls points ayant une radiométrie différente de *Val* dans l'image *Im* seront valides (méthode un peu archaïque, maintenue surtout par compatibilité);
- à cet éventuel masque, on peut superposer autant de masque que l'on veut via la liste <MasqImageIn>; le fonctionnement de <MasqImageIn> est le suivant :
  - chaque élément de la liste <MasqImageIn> doit permettre de calculer le masque image associé à *Im*;
  - on parcourt la liste <OneMasqueImage>; soit le dernier élément pour lequel l'expression régulière <PatternSel> est appariable sur le nom *Im*, on utilise alors <NomMasq> pour calculer le masque associé selon le mécanisme habituel (voir 27.3.1); ces images sont recherchées dans <DirMasqueImages>;
  - si aucune expression régulière n'est appariable, une erreur est générée; il est cependant possible de spécifier qu'aucun masque n'est associé en faisant que <NomMasq> renvoie la valeur clé "PasDeMasqImage"

L'extrait ci-dessous est un exemple d'utilisation réel des masques:

- pour toutes les images, les points valant 0 sont hors masques;
- pour l'image 299.tif, c'est le seul masque utilisé;
- pour les autres images *Im.tif*, on va superposer en plus l'image de nom *Im.masque.tif* dans le répertoire *masques/*;

```
<ValSpecNotImage> 0 </ValSpecNotImage>
<DirMasqueImages> masques/ </DirMasqueImages>
```

```

<MasqImageIn>
    <OneMasqueImage>
        <PatternSel> (.*)\.tif</PatternSel>
        <NomMasq> $1.masque.tif </NomMasq>
    </OneMasqueImage>
    <OneMasqueImage>
        <PatternSel> 299.tif </PatternSel>
        <NomMasq> PasDeMasqImage </NomMasq>
    </OneMasqueImage>
</MasqImageIn>

```

### 28.2.1.3 Gestionnaire de pyramide d'image

```

<NomsGeometrieImage Nb="+">
    ...
    <ModuleImageLoader Nb="?">
        <NomModule Nb="1" Type="std::string"> </NomModule>
        <NomLoader Nb="1" Type="std::string"> </NomLoader>
    </ModuleImageLoader>
    ...
</NomsGeometrieImage>

```

Le tag `<ModuleImageLoader>` permet de spécifier un gestionnaire de pyramide d'image. Ce tag a été mis comme fils du tag `<NomsGeometrieImage>` pour factoriser le mécanisme de sélection par le nom de l'image (voir 28.2.2.2) et pouvoir, si nécessaire, associer différents gestionnaires en fonction du nom de l'image.

Si rien n'est spécifié, c'est le gestionnaire par défaut de MicMac qui est utilisé; ce gestionnaire s'attend à ce que le nom de l'image soit un fichier `tif` (ou `thom`) contenant l'image à résolution 1, il générera sa propre pyramide au fur et à mesure des besoins.

S'il est spécifié, alors `<NomModule>` est le nom d'une librairie dynamique et `<NomLoader>` est une entrée dans cette librairie (selon le mécanisme décrit en 27.4). Grégoire Maillet a commencé à écrire un module utilisant une librairie JPEG-2000 pour en faire un gestionnaire de pyramide utilisable dans MicMac. Ce gestionnaire s'attend à ce que le nom de l'image soit un fichier JPEG-2000 et utilise ce seul fichier pour retrouver tous les niveaux de la pyramide.

Il serait possible d'écrire une gestionnaire pour utiliser le format `DMR` utilisé par IGN-espace.

## 28.2.2 Géométrie (intrinsèque)

### 28.2.2.1 Type de Géométrie

```

<GeomImages Nb="1" Type="eModeGeomImage" /> </GeomImages>
<ModuleGeomImage Nb="?">
    <NomModule Nb="1" Type="std::string"> </NomModule>
    <NomGeometrie Nb="1" Type="std::string"> </NomGeometrie>
</ModuleGeomImage>

```

Ce tag obligatoire spécifie le format de géométrie intrinsèque utilisée. Il doit avoir une valeur dans l'énumération `eModeGeomImage` (voir 27.2.2.1).

Si le `GeomImages` a la valeur `eGeomImageModule`, alors `ModuleGeomImage` doit avoir une valeur pour spécifier une librairie dynamique (fichier `<NomModule>`, entrée `NomGeometrie`) permettant de charger le code utilisateur décrivant la géométrie (voir 27.4 et 27.2.2.3).

**MODIF:** Aujourd'hui il n'est pas possible de mélanger plusieurs formats de géométrie dans un même chantier. Ajouter cette possibilité ne devrait pas poser de vrai problème il est prévu de le faire à terme (priorité *a priori* assez basse).

### 28.2.2.2 Association images/géométries, cas standard

```

<NomsGeometrieImage Nb="+">
    ...
    <PatternSel Nb="1" Type="std::string" /> </PatternSel>
    <PatNameGeom Nb="1" Type="std::string" /> </PatNameGeom>
    ...
</NomsGeometrieImage>

```

La liste non vide (arité ="+") `NomsGeometrieImage` contient les informations pour calculer automatiquement le nom du fichier de géométrie à partir du nom de fichier image. Le mécanisme a été mis en place avec l'objectif d'être suffisamment souple pour qu'il ne soit jamais nécessaire de renommer les fichiers en entrée de MicMac.

Soit une image de nom de fichier `UnNomImage`, pour calculer le nom de la géométrie, MicMac procède ainsi :

- la liste `NomsGeometrieImage` est parcourue dans l'ordre;
- MicMac s'arrête au premier élément tel que `UnNomImage` soit appariable sur l'expression régulière `<PatternSel>` ;
- à partir de cet appariement, le pattern `PatNameGeom` est utilisé pour calculer le nom de la géométrie selon (voir 27.3.1);
- si aucun appariement n'est trouvé, une erreur se produit;

Dans la majorité des cas, `<NomsGeometrieImage>` ne contiendra qu'un seul élément parce que l'association "image/géométrie" est complètement systématique.

### 28.2.2.3 Nom calculé sur Im1-Im2

*Ce paragraphe peut facilement être omis en première lecture.*

```
<NomsGeometrieImage Nb="+">
  ...
    <PatternNameIm1Im2 Nb="?" Type="std::string" > </PatternNameIm1Im2>
    <AddNumToNameGeom Type="bool" Nb="?" Def="false" > </AddNumToNameGeom>
  ...
</NomsGeometrieImage>
```

La fonctionnalité décrite ici est apparue en utilisant MicMac lors du calcul automatique de points homologues pour l'aérotriangulation. Dans ce contexte, on veut utiliser MicMac de la manière suivante :

- MicMac est appelé avec un grand nombre de couples; la même image se retrouvant en général dans plusieurs couples;
- pour chaque couple, MicMac est appelé en plusieurs étapes; une première fois en utilisant en entrée des orientations *a priori*, éventuellement assez "grossière";
- ensuite MicMac est rappelé dans différentes étapes de "raffinement" successif où chaque étape génère une nouvelle orientation qui servira d'entrées à l'étape suivante (le principe étant que, sauf bug, la précision des orientations relatives s'améliorant à chaque étape, on peut relancer le calcul avec des contraintes de plus en plus fortes sur les parallaxes transverses);

Pour éviter tout conflit de nom entre les nombreux chantiers qui vont être créés (surtout dans le cas où le calcul est parallélisé par un outil de calcul distribué), il importe que chaque calcul puisse spécifier ses propres noms de fichiers d'orientation. Supposons que les images mises en correspondance soit `Im1=ImAA.tif` et `Im2=ImBB.tif` et qu'après l'étape 3, soit générés deux fichiers d'orientation relative, selon le mécanisme décrit en `?2Def?`, qui s'appellent `Ori_Etap3_Im1_AABB.ori` et `Ori_Etap3_Im2_AABB.ori`; toujours pour éviter les conflits de noms, ceux-ci sont choisis de manière à être différent du cas "symétrique" `Im2=ImAA.tif` et `Im1=ImBB.tif`.

Ensuite à l'étape 4 il faut pouvoir calculer le nom des fichiers d'orientation en entrée d'une part en prenant en compte le nom de `Im1` et `Im2` et d'autre part en pouvant spécifier une construction qui soit différente pour chacune des deux images (la sélection étant basée ici sur leur rang et non sur leur nom). Ces possibilités sont offertes par les tags `<PatternNameIm1Im2>` et `<AddNumToNameGeom>` :

- si le tag `<PatternNameIm1Im2>` est spécifié , alors pour c'est la concaténation `Im1@Im2` (voir 27.3.2) qui sera appariée sur sa valeur considérée comme une expression régulière qui sera utilisée pour étendre `<PatNameGeom>` (mais c'est toujours sur l'expression `PatternSel` et sur le nom d'une seule image que sera faite la sélection);
- si le tag `<AddNumToNameGeom>` est spécifié à vrai, alors `<PatNameGeom>` n'est pas apparié sur le nom tout seul, mais sur une concaténation du nom suivi de `@k` où `k` est le rang de l'image (en commençant à 0);

Dans notre exemple, les bon nom de fichiers pourraient alors être calculés par quelque chose comme :

```
</Section_PriseDeVue>
  ...
<NomsGeometrieImage>
  <AddNumToNameGeom> true </AddNumToNameGeom>
  <PatternSel>   (*.*)\.\.tif@00 </PatternSel>
  <PatternNameIm1Im2>
    Im(*.)\.\.tif@Im(*.)\.\.tif
  </PatternNameIm1Im2>
  <PatNameGeom> Ori_Etap3_Im1_\$1\$2.ori </PatNameGeom>
</NomsGeometrieImage>
```

```

<NomsGeometrieImage>
    <AddNumToNameGeom> true           </AddNumToNameGeom>
    <PatternSel>   (*.*)\.tif@1  </PatternSel>
    <PatternNameIm1Im2>
        Im(..)\.tif@Im(..)\.tif
    </PatternNameIm1Im2>
    <PatNameGeom> Ori_Etap3_Im2_$1$2.ori  </PatNameGeom>
</NomsGeometrieImage>
...
</Section_PriseDeVue>

```

#### 28.2.2.4 Points homologues

```

<NomsHomologues Nb=?>
    <PatternSel Nb="1" Type="std::string">      </PatternSel>
    <PatNameGeom Nb="1" Type="std::string">      </PatNameGeom>
    <SeparateurHom Nb=? Type="std::string" Def="">  </SeparateurHom>
</NomsHomologues>

```

Ce tag permet de calculer le fichier de points homologues utilisé pour calculer, le cas échéant, l'homographie de passage de **Im1** à **Im2** (voir 27.2.2.5). Ce nom de fichier est calculé par appariement sur la concaténation des noms (voir 27.3.2) :

- **<PatternSel>** est l'expression sur laquelle est appariée la concaténation;
- **<PatNameGeom>** est le patron qui génère le nom defichier;
- **<SeparateurHom>** est un éventuel séparateur;

Voici un exemple pris d'un paramétrage pour de la superposition multi-canal:

```

<Images >
    <Im1 > FD0027_r.047_3113 </Im1>
    <Im2 > FD0027_v.047_3113 </Im2>
</Images>
....
<NomsHomologues >
    <PatternSel > FD0027_(.)\.(.*)_@FD0027_(.)\.(.*)_(</PatternSel>
    <PatNameGeom > Liaison_$1$4.xml  </PatNameGeom>
    <SeparateurHom > @  </SeparateurHom>
</NomsHomologues>

```

Dans cet exemple, le nom de fichier de points homologues calculé sera **Liaison\_rv.xml**; l'objectif était bien d'avoir un nom qui dépend de la natures des canaux (ici rouge et vert) mais pas du numéro des images.

### 28.3 Génération de Résultat

Cette section, ainsi que 28.4, décrit des tags permettant de générer des résultat intermédiaires. Ils sont dans la descendance de **EtapeMEC** afin de pouvoir, éventuellement, générer ces résultat à chaque étape dans une phase de mise au point. Ils sont décrit ici car ils ne sont en général pas logiquement actif sur le processus de mise en correspondance.

#### 28.3.1 Image 8 Bits

#### 28.3.2 Image de corrélation

#### 28.3.3 Basculement dans une autre géométrie

```

<BasculeRes Nb="*>
    <Ori Nb="1" RefType="FileOriMnt"> </Ori>
</BasculeRes>

```

Ce tag permet de génér à chaque étape un (ou plusieurs) basculement dans une géométrie différente de l'acquisition. Le tag **Ori** est du type défini en 27.5.1.

**28.3.4 Parallaxe relative ??**

**28.4 Modèles analytiques**

**28.5 Section Espace de travail**

**28.5.1 Directory Image**

**28.6 Section dite "Vrac"**



# Chapter 29

## Mise en correspondance

### 29.1 Généralité

*N.B.* : MEC= Mise En Correspondance.

#### 29.1.1 Organisation

La section `<Section_MEC>` est constituée de quelques tags globaux à la mise en correspondance puis d'une liste de descripteurs d'étapes `<EtapeMEC>`.

Afin de conserver un maximum de souplesse, MicMac étant aussi un outil de R&D , la grande majorité des tags se retrouvent en fait dans les descripteurs d'étapes ce qui permet, au cas où, d'avoir un contrôle étape par étape sur les valeur. La section 29.2 est consacrée aux quelques tags globaux; les sections suivantes sont consacrées aux descendants de `<EtapeMEC>`.

#### 29.1.2 Mode différentiel, valeurs par défaut

La liste `<EtapeMEC Nb="+" DeltaPrec="1">` est en mode différentiel (voir 10.5), ce qui permet de concilier un fonctionnement simple dans le cas général avec un réglage fin pour des utilisations plus pointues. Le premier élément de cette liste d'étapes de mise en correspondance ne sera pas exécuté, il permet de régler les valeur par défaut qui seront commune à la majorité des étapes. Pous s'assurer qu'il n'y a pas d'ambiguité sur ce point, MicMac impose que la valeur de résolution (tag `DeZoom`) de ce premier élément soit `-1` (valeur impossible).

Le mode différentiel complique un peu la gestion des valeur par défaut. Il existe un seul fils de `<EtapeMEC>` obligatoire à toutes le étapes : `DeZoom`. Les autres sont d'arité ?, ce qui peut en fait correspondre à plusieurs réalité différentes :

- les tags qui sont logiquement obligatoires mais qui, du fait du mode différentiel, peuvent n'être spécifiés qu'à la première étape;
- les tags qui sont logiquement facultatifs, pour des raisons internes de programmation il n'est pas possible d'utiliser le mécanisme de spécification par l'attribut `Def` (voir 10.6.2) ; la valeur par défaut est gérée dans le code MicMac; sauf oubli, elle est spécifiée dans cette doc et en commentaire.

#### 29.1.3 Equivalence de noms

Comme en 28.1.1, les mêmes paramètres ont des noms différents suivant la dimension de parallaxe. Le tableau 29.1 résume les équivalences.

### 29.2 Paramètres globaux

```
<Section_MEC Nb="1">
  <ProportionClipMEC/>
  <NbMinImagesVisibles/>
  <DefCorrelation/>
  <EpsilonCorrelation/>
  ...
</Section_MEC>
```

	Z	Px1	Px2	Nom	Math
Pas	ZPas	Px1Pas	Px2Pas	$r_k$	
Régularisation	ZRegul	Px1Regul	Px2Regul	$\alpha_k$	
Dilatation-Px	ZDilatAlti	Px1DilatAlti	Px2DilatAlti		
Dilatation-XY	ZDilatPlani	Px1DilatPlani	Px2DilatPlani		
Redressement	ZRedrPx	Px1RedrPx	Px2RedrPx		
Dequantif Redr	ZDeqRedr	Px1DeqRedr	Px2DeqRedr		

Figure 29.1: Equivalence de noms

### 29.2.1 Clip de la zone de MEC

```
<ProportionClipMEC Type="Box2dr" Nb="?"> </ProportionClipMEC>
```

Cette valeur permet de restreindre la zone sur laquelle est réellement effectuée la mise en correspondance . Cette restriction ne porte que sur le calcul et ne modifie pas l'emprise terrain du chantier; par exemple, si on lance plusieurs fois MicMac avec le même fichier de paramètres, sauf cette valeur `<ProportionClipMEC>`, on ne verra pas de problème de raccord entre les différents morceaux calculés.

Comme son nom l'indique, il s'agit d'une proportion; la boîte [0.0 0.0 1.0 1.0] correspond à l'ensemble du chantier; lorsque `<ProportionClipMEC>` est omis (arité=?), la valeur par défaut est naturellement [0.0 0.0 1.0 1.0].

`<ProportionClipMEC>` est utile en phase mise au point lorsque l'on veut tester rapidement un jeu de paramètres sans créer un chantier *ad hoc*.

### 29.2.2 Calcul du masque de MEC

La section planimétrie(28.1.2) permet de spécifier l'emprise (boîte englobante) terrain et un éventuel masque utilisateur ( 28.1.2.4). Les tags de cette section permettent de paramétriser le masque calculé automatiquement par MicMac.

#### 29.2.2.1 Nombre minimal d'images

```
<NbMinImagesVisibles Nb="?" Type="int" Def="2"> </NbMinImagesVisibles>
```

MicMac calcul un masque qui se superpose à l'éventuel masque terrain spécifié par l'utilisateur. Soit  $Nb^{im}$  la valeur de ce paramètre; ce masque est défini comme l'ensemble des points terrain qui sont vus d'au moins  $Nb^{im}$  avec l'hypothèse de parallaxe moyenne. La formule utilisée par MicMac est donc :

$$\{p \in \mathcal{T} / \text{card}\{k \in [1 N] / \pi_k(p, \tilde{P}_x) \in \mathcal{I}_k\} \geq Nb^{im}\} \quad (29.1)$$

### 29.2.3 Divers

#### 29.2.3.1 Valeur par défaut de l'attache aux données

```
<DefCorrelation Nb="?" Type="double" Def="0.0"> </DefCorrelation>
```

Lors du calcul d'un coefficient de ressemblance inter-image, différentes raisons peuvent empêcher MicMac de calculer un coefficient de corrélation (par exemple parce que, compte-tenu des masques image et terrain, il n'y a pas au moins deux images visibles). Dans ces cas MicMac renvoie la valeur de `<DefCorrelation>`.

#### 29.2.3.2 Corrélation dégénérées

```
<EpsilonCorrelation Nb="?" Type="double" Def="1e-10"> </EpsilonCorrelation>
```

L'estimation du coefficient de corrélation est d'autant plus bruitée que l'écart-type est faible (et indéfini quand il est nul). Pour éviter tout problème d'exception arithmétique, MicMac utilise `Max(Variance, <EpsilonCorrelation>)` dans le calcul du coefficient de corrélation.

## 29.3 Géométrie et nappes englobantes

### 29.3.1 Gestion des résolutions

#### 29.3.1.1 Résolution terrain

```
<DeZoom Nb="1" Type="int"> </DeZoom>
```

Cette valeur fixe le ratio de résolution terrain de l'étape; il s'agit du seul paramètre obligatoire à chaque étape. Le pas de résolution terrain de l'étape, le  $\Delta^{xy}$  défini en 20.2, sera égal à `DeZoom * ResolutionTerrain` (où `ResolutionTerrain` est la valeur définie en 28.1.2.3). Les contraintes sont les suivantes :

- la valeur de la première étape, qui n'est pas exécutée, doit valoir conventionnellement  $-1$  (voir 29.1.1);
- les valeurs suivantes doivent être des puissances de  $2$  et  $\geq 1$  ;
- ces valeurs doivent être décroissantes (de manière non stricte) avec un ratio au maximum de  $2$  entre valeurs consécutives; autrement dit, si à une étape la valeur est  $2^n, n \in \mathbb{N}$ , à l'étape suivante cette valeur est soit  $2^n$ , soit  $2^{n-1}$ .

#### 29.3.1.2 Résolution image

*Valeur par défaut 1.0*

```
<RatioDeZoomImage Nb="?" Type="double"> </RatioDeZoomImage>
```

En général, pour des questions de cohérence d'échantillonage, on souhaite avoir des résolutions comparables pour le terrain et l'image. C'est ainsi que pour les géométries terrain, la résolution terrain proposée par défaut par MicMac est égale à la résolution estimée de la prise de vue (voir 28.1.2.3). En conséquence, pour maintenir cette cohérence à chaque étape, le ratio de résolution image de l'étape (le  $\Delta^i$  de 20.2) est lui aussi, en général, égal au ratio Terrain.

Cependant cette égalité des ratio, n'est pas imposée et MicMac permet de le contrôler avec le paramètre `RatioDeZoomImage` de la manière suivante :

- le  $\Delta^i$  vaut `RatioDeZoomImage * DeZoom`;
- par défaut `RatioDeZoomImage` vaut  $1$  (donc assure l'égalité des ratios);

Le seul cas cohérent répertorié d'utilisation de `RatioDeZoomImage` correspond au contexte suivant :

- on dispose d'un prise de vue à une résolution donnée, disons  $20\text{cm}$  pour fixer les choses;
- on espère (peut être à cause du multi-vues) pouvoir corrélérer les images avec un pas terrain de  $10\text{cm}$ ;
- si on laissait les options par défaut de MicMac on aurait :
  - à `DeZoom=1`, des images à  $20\text{cm}$  pour  $10\text{cm}$  terrain (ça c'est inévitable);
  - à `DeZoom=2`, des images à  $40\text{cm}$  pour  $20\text{cm}$  terrain ;
  - à `DeZoom=4`, des images à  $80\text{cm}$  pour  $40\text{cm}$  terrain ;
  - ...

Or, tant que  $\text{DeZoom} \geq 2$ , on dispose d'images ayant une résolution adaptée au terrain, et on n'a pas intérêt à utiliser des images moins bien résolues. Pour utiliser les images à la bonne résolution, on pourra fixer `<RatioDeZoomImage>` à  $0.5$  dans la première étape. Cette valeur est ensuite transmise aux suivantes via le mode différentiel, on aura simplement soin de rétablir `<RatioDeZoomImage>` à  $1$  lorsque `DeZoom=1`.

### 29.3.2 Pas de quantification

*Pas de valeur par défaut*

```
<ZPas Nb="?" Type="double"> </ZPas>
<Px1Pas Nb="?" Type="double"> </Px1Pas>
<Px2Pas Nb="?" Type="double"> </Px2Pas>
```

Ces paramètres permettent de fixer les  $\Delta_k^{px}$  décrits en 20.2. Ils expriment un ratio; soit  $r_k$  leur valeur, la formule utilisée pour calculer  $\Delta_k^{px}$  est la suivante :

$$\Delta_k^{px} = r_k * \Delta^{xy} * \rho_k^G \quad (29.2)$$

Commentons :

- $\rho_k^G$  est un coefficient, calculé par MicMac, qui ne dépend que de la géométrie et sur lequel nous allons revenir (il vaut souvent  $1$ );
- couramment, les valeurs fixées à la première étape peuvent être conservées tout au long du calcul (puisque  $\Delta_k^{px}$  est proportionnel à  $\Delta^{xy}$  et donc à `DeZoom`);

- le coefficient  $\rho_k^G$  assure que, en général  $0.5 \leq r_k \leq 1.0$ , est une plage de valeur "raisonnable" pour débouter un essai par tatonnement.

Le coefficient  $\rho_k^G$  est calculé en fonction de la géométrie de restituation de la manière suivante :

- $\rho_k^G = 1$  pour `eGeomMNTCarto`, `eGeomMNTEuclid`, `eGeomPxBiDim`;
- pour `eGeomMNTFaisceauIm1ZTerrain_Px?D`,  $\rho_1^G$  est la résolution terrain de la géométrie intrinsèque ;  $\rho_2^G = 1$  (si utile);
- pour `eGeomMNTFaisceauIm1PrCh_Px?D`, s'il y a deux images  $\rho_1^G$  est calculé à partir du  $\frac{B}{H}$  pour qu'un écart de 1 en parallaxe corresponde à peu près à un écart de 1 pixel sur les images (la correspondance étant exacte, en cas de vues parallèles, grâce à la dynamique en  $\frac{1}{Z}$ ) ; avec  $N$  image, c'est une moyenne de cette formule qui est utilisée;  $\rho_2^G = 1$  (si utile);

### 29.3.3 Calcul des nappes englobantes

*Pas de valeur par défaut*

```
<ZDilatAlti  Nb="?" Type="int">  </ZDilatAlti>
<ZDilatPlani Nb="?" Type="int">  </ZDilatPlani>

<Px1DilatAlti Nb="?" Type="int">  </Px1DilatAlti>
<Px1DilatPlani Nb="?" Type="int">  </Px1DilatPlani>

<Px2DilatAlti Nb="?" Type="int">  </Px2DilatAlti>
<Px2DilatPlani Nb="?" Type="int">  </Px2DilatPlani>
```

Ces valeurs fixent le  $\delta_P^k$  et le  $\delta_A^k$  définis en 21.2. Ce sont des valeurs entières qui seront appliquées à la fonction de parallaxe quantifiée. *Si l'on change les pas de quantification définis en 29.3.2 et que l'on veut conserver la même signification physique à la dilatation, il faut adapter le  $\delta_A^k$ .*

### 29.3.4 Redressement des images

#### 29.3.5 Divers

##### 29.3.5.1 Différentiabilité de la géométrie

*Valeur par défaut 2*

```
<SzGeomDerivable Nb="?" Type="int"> </SzGeomDerivable>
```

Pour limiter le volume de calcul, fait l'hypothèse que les fonctions de projections sont dérivables et localement approximable par leur différentielle (dérivées secondes "faible" à l'échelle des variations considérées). Typiquement, étant donnée une parallaxe quantifiée  $u_0, v_0$  et un point terrain quantifié  $i_0, j_0$  pour estimer les projection sur un voisinage de taille  $N^b$  de  $i_0, j_0$ , posons :

$$P_0(i, j) = p_k(i_0 + i, j_0 + j, u_0, v_0) \quad (i, j) \in [-N^b, +N^b]^2 \quad (29.3)$$

MicMac utiliser un schéma classique de différences finies, pour faire les calculs à partir des estimations de  $\pi_k$  sur les "4-voisins" de  $P_0(0, 0)$  soit :

$$V_0 = \frac{P_0(1, 0) + P_0(-1, 0) + P_0(0, 1) + P_0(0, -1)}{4} \quad (29.4)$$

$$\Delta_x = \frac{P_0(1, 0) - P_0(-1, 0)}{2} \quad \Delta_y = \frac{P_0(0, 1) - P_0(0, -1)}{2} \quad (29.5)$$

$$P_0(i, j) \approx V_0 + i * \Delta_x + j * \Delta_y \quad (29.6)$$

En toute rigueur, il est possible que ce mode de calcul conduise à des approximations inacceptables. `SzGeomDerivable` permet d'indiquer à MicMac la taille du voisinage  $N^b$  sur lequel l'approximation est licite.

## 29.4 Autres paramètres d'entrées

### 29.4.1 Sélection des images

*Valeur par défaut : pas de sélection*

```
<ImageSelecteur Nb="?">
  <ModeExclusion Nb="1" Type="bool"> </ModeExclusion>
  <PatternSel Nb="*" Type="std::string"> </PatternSel>
</ImageSelecteur>
```

Parfois, on ne souhaite pas que toutes les images du chantier soient utilisées à toutes les étapes. Lorsque **ImageSelecteur** est spécifié, il permet de filtrer les images qui seront utilisées à l'étape courante. Si **ModeExclusion** vaut **true** (resp **false**) on exclut (resp inclut) les images dont le nom est appariable sur au moins une des expressions régulières (voir 27.3.1) de la liste **PatternSel**.

## 29.4.2 Interpolation

*Valeur par défaut eInterpolBiLin*

```
<ModeInterpolation Nb="?" Type="eModeInterpolation"> </ModeInterpolation>
<CoefInterpolationBicubique Nb="?" Type="double" Def="-0.5"> </CoefInterpolationBicubique>
<TailleFenetreSinusCardinal Nb="?" Type="int" Def="3"> </TailleFenetreSinusCardinal>
```

Les  $\pi_k$  des points quantifiés sont à coordonnées réelles; pour accéder aux radiométries des images en ces points, avec un minimum de perte d'information, il convient de se donner un schéma d'interpolation. Le paramètre **ModeInterpolation** permet de spécifier l'interpolateur à utiliser ; il est à valeur dans l'énumération **eModeInterpolation**:

- **eInterpolPPV** : plus proche voisin;
- **eInterpolBiLin** : bi-linéaire;
- **eInterpolBiCub** : bi-cubique , il existe une famille d'interpolateur bicubique définie par un paramètre (dérivée en 1 ?), ce paramètre est contrôlé par le tag **CoefInterpolationBicubique** (celui utilisé par défaut est celui qui est restore correctement les signaux affines);
- **eInterpolSinCard** : sinus cardinal, l'interpolateur étant en théorie à support infini , le tag **TailleFenetreSinusCardinal** permet de choisir sur quelle taille de fenêtre on "l'appodise"; par **TailleFenetreSinusCardinal** (par défaut 3);

## 29.5 Approche énergétique

### 29.5.1 Attache aux données

#### 29.5.1.1 fenêtres de corrélation

*Par défaut TypeWCorr=eWInCorrelFixe*

*Pas de valeur par défaut pour SzW.*

*Par défaut SzWInt=SzW.*

*Par défaut SzWy=SzW.*

```
<TypeWCorr Nb="?" Type="eTypeWinCorrel"> </TypeWCorr>
<SzW  Nb="?"  Type="double">          </SzW>
<SzWy Nb="?"  Type="double">          </SzWy>
<SzWInt Nb="?" Type="int">           </SzWInt>
```

On a vu (22.4.1) que les fenêtre de corrélation ne sont pas forcément rectangulaire avec un poids constant. Le paramètre **TypeWCorr** permet de régler le choix du type de fenêtre, il peut prendre les valeurs suivantes :

- **eWInCorrelFixe** fenêtre "classique";
- **eWInCorrelExp** fenêtre exponentielle;

Le paramètre **SzW** fixe la taille de la vignette de corrélation telle que définie en 22.2. Lorsque la fenêtre choisie n'est pas un fenêtre rectangulaire mais, par exemple, une fenêtre exponentielle à support infini, ce paramètre sert en fait à fixer par analogie le paramètre d'échelle : MicMac choisit le paramètre qui donne à la largeur moyenne du filtre la même valeur que la fenêtre classique de taille **SzW**. Pour les fenêtre exponentielles, il est tout à fait cohérent de lui assigner des valeurs réelles et est donc, pour rester général de type **double**.

Les tailles en x et en y de la fenêtre n'ont pas de raison d'être égales: le tag **SzWy** permet d'assigner une valeur différente à la taille en y (attention, pas encore accessible avec les fenêtres classiques).

Comme défini en 22.3, pour les fenêtres classiques, le paramètre **SzWInt**, permet éventuellement d'avoir une taille différente pour la mesure d'écart (fenêtre de taille **SzWInt**) et la normalisation en homothétie-translation des radiométries (fenêtre de taille **SzW**).

### 29.5.1.2 Multi-corrélation

```
<AggregCorr Nb="?" Type="eModeAggregCorr"> </AggregCorr>
```

Ce tag permet de spécifier comment agrégger les coefficients de corrélation lorsqu'il y a plus de deux images (voir 22.5). Les valeurs possibles sont :

- **eAggregSymetrique** coefficient où toutes les images jouent le même rôle (équation 22.25);
- **eAggregIm1Maitre** coefficient où la première image est considérée comme maîtresse (équation 22.24).

### 29.5.1.3 Dynamique de corrélation

```
<DynamiqueCorrel Nb="?" Type="eModeDynamiqueCorrel"> </DynamiqueCorrel>
```

Comme vu avec l'équation 22.10, le coefficient de corrélation est directement fonction de la distance au carré entre des vignettes normalisées. Ce tag permet de spécifier comment convertir le coefficient de corrélation en un coût utilisé pour l'attache aux données. Soit *Cost* le coût et *Corr* le coefficient de corrélation, les valeurs possibles sont :

- **eCoeffCorrelStd** le coût est donné par  $Cost = 1 - Corr$ , ce coût est donc un écart quadratique entre les vignettes normalisées;
- **eCoeffAngle** le coût est donné par  $Cost = \text{acos}(Corr)$ ; lorsque l'on a deux images, en revenant à l'interprétation du coefficient de corrélation comme le cosinus de l'angle entre  $\bar{u}$  et  $\bar{v}$  (dérivé de la formule 22.9), ce coût peut être interprété comme l'angle entre  $\bar{u}$  et  $\bar{v}$ ; de manière plus générale, ce coût s'interprète comme un pénalisation  $L^1$  alors que **eCoeffCorrelStd** est plutôt une pénalisation  $L^2$ .

**MODIF:** Pour que l'utilisateur puisse spécifier n'importe quelle fonction rajouter la possibilité d'avoir une dynamique "tabulée".

### 29.5.2 A priori

#### 29.5.2.1 Régularisation

*Pas de valeur par défaut pour les 3 premiers*

*Nuls pas défauts pour les couts quadratiques*

```
<ZRegul Nb="?" Type="double">      </ZRegul>
<Px1Regul Nb="?" Type="double">     </Px1Regul>
<Px2Regul Nb="?" Type="double">     </Px2Regul>

<ZRegul_Quad Nb="?" Type="double">    </ZRegul_Quad>
<Px1Regul_Quad Nb="?" Type="double">  </Px1Regul_Quad>
<Px2Regul_Quad Nb="?" Type="double">  </Px2Regul_Quad>
```

Ces valeurs correspondent aux termes de régularisation  $\alpha_k$  définis en 24.1; il permettent de pondérer l'attache aux données par rapport à l'*a priori* de régularité (plus ils sont fort, plus on impose un résultat régulier). Ces paramètres posent parfois problème aux utilisateurs MicMac car il est difficile de donner des consignes rationnelles sur les moyen de régler leur valeur. Une remarque qui peut être faite, est qu'il n'est en général pas nécessaire de modifier leur valeur en fonction du niveau de **DeZoom**; d'un part, en première approximation il peuvent être considérés comme invariant à l'échelle et, d'autre part, leur réglage pour les basses résolution n'est pas critique. A défaut de règles rationnelles, on peut donner quelques plages de valeurs empiriques sur les principaux cas d'utilisation de MicMac :

- pour les MNE urbain, en multi-vues à haute résolution,  $\alpha_1$  est typiquement dans l'intervalle [0.01 0.05]; la valeur est faible car, d'une part le multi-vues rend plus fiable l'attache aux données et d'autre part l'**a priori** de régularité est faible sur le relief urbain et ses fortes discontinuités;
- pour les MNT spot-5, en stéréo classique,  $\alpha_1$  est typiquement dans l'intervalle [0.1 0.2];
- pour la superposition des canaux de la caméra numérique,  $\alpha_1$  et  $\alpha_2$  sont typiquement dans l'intervalle [0.5 2.0]; ici le champ recherché est très régulier (fonction basse fréquence, faible amplitude);
- pour du calcul de points de liaisons denses en aéro-triangulation, on peut partir de  $\alpha_1$  dans [0.05 0.2] et  $\alpha_2$  dans [0.25 2.0];  $\alpha_1$  et  $\alpha_2$  sont a priori très différents puisque  $\alpha_1$  est déterminé par l'incertitude sur le relief alors que  $\alpha_2$  est déterminé par l'incertitude sur la mise en place initiale; le ratio de variation possible de  $\alpha_2$  est très grand car l'**a priori** sur les orientation peut être très variable (typiquement quelques pixels à quelques centaines suivant les contextes);

Ces règles peuvent servir de valeur initiale dans une approche par tatonnement, il ne faut surtout pas hésiter à les remettre en cause à la première occasion.

Les trois derniers paramètres permettent de rajouter un terme quadratique dans le critère de coût (comme par exemple 24.5). Ces paramètres ne sont accessibles qu'avec un optimisation de type programmation dynamique. Ils peuvent être utilisés par exemple pour de la production de modèles numériques de terrain.

### 29.5.2.2 Post-filtrage

### 29.5.3 Minimisation

#### 29.5.3.1 Choix d'un algorithmes

```
<AlgoRegul Nb="?" Type="eAlgoRegul"> </AlgoRegul>
```

#### 29.5.3.2 Paramètre spécifiques à Cox-Roy

*Par défaut CoxRoy8Cnx vaut false, CoxRoyUChar vaut true*

```
<CoxRoy8Cnx Nb="?" Type="bool"> </CoxRoy8Cnx>
<CoxRoyUChar Nb="?" Type="bool"> </CoxRoyUChar>
```

Si **CoxRoy8Cnx** vaut **true** le voisinage défini dans 24.10 est le 8-voisinage, sinon c'est le 4-voisinage. L'utilisation du 8-voisinage permet de diminuer sensiblement les artefacts directionnels qui apparaissent avec le 4-voisinage au prix d'un doublement, en moyenne, du temps de calcul.

Les algorithme de flots sont assez consommateurs en mémoire; pour chaque pixel et chaque parallaxe et chaque, il faut mémoriser autant de valeur numérique qu'il y a de voisins (4 ou 8). Le tag **CoxRoyUChar** permet de contrôler le compromis "encombrement mémoire / dynamique" :

- si **CoxRoyUChar** vaut **true**, les valeurs numérique sont stockées sur un octet; cela a pour conséquence que les coefficient de corrélation et les terme de régularisation sont arrondies au  $\frac{1}{100}$ ; cette option est donc déconseillée avec des régularisations très faibles;
- si **CoxRoyUChar** vaut **false**, les valeurs numérique sont stockées sur deux octets; les valeurs sont arrondies au  $\frac{1}{10000}$ ;

#### 29.5.3.3 Paramètre spécifiques à la programmation dynamique

*ModulationProgDyn est obligatoire si le mode algorithmique choisi est eAlgo2PrgDyn*

```
<ModulationProgDyn Nb="?">
  <EtapeProgDyn Nb="+">
    <Px1MultRegul Nb="?" Type="std::vector<double>"> </Px1MultRegul>
    <Px2MultRegul Nb="?" Type="std::vector<double>"> </Px2MultRegul>
    <NbDir Nb="?" Type="int" Def="2"> </NbDir>
    <ModeAgreg Type="eModeAggregProgDyn" Nb="1"> </ModeAgreg>
    <Teta0 Nb="?" Type="double" Def="0.0"> </Teta0>
  </EtapeProgDyn>
  <Px1PenteMax Nb="?" Type="double" Def="10.0"> </Px1PenteMax>
  <Px2PenteMax Nb="?" Type="double" Def="10.0"> </Px2PenteMax>
</ModulationProgDyn>
```

MicMac utilise le mécanisme de "programmation dynamique multi-directionnelle" décrit en 24.3 auquel on pourra se référer. L'image est balayée selon les angles  $Teta0 + k \frac{\pi}{NbDir}$ . Les mesures obtenues pour chaque direction sont agrégés en fonction de **ModeAgreg** qui peut prendre une des valeurs énumérées :

- **ePrgDAgrSomme**, on agrège en calculant la moyenne;
- **ePrgDAgrMax**, on agrège en retenant le maximum
- **ePrgDAgrReinject**, on utilise un mode réentrant (le résultat de chaque balayage est utilisé comme entrée pour le prochain);

**Px1PenteMax** et **Px2PenteMax** permettent d'imposer une contrainte de pente maximale. On a tout intérêt à imposer cette contrainte lorsque cela est pertinent pour le problème posé. En plus de l'avantage évident d'ajouter une connaissance *a priori*, cela permet de limiter la combinatoire (en évitant d'explorer tous les couples de parallaxe de pixels voisins).

Lorsqu'il y a plusieurs étape **EtapeProgDyn** elle sont enchaînées selon un mode réentrant.

**Px1MultRegul** et **Px2MultRegul** sont des paramètres ésotériques.

**29.5.4 Sous résolution des algorithmes****29.5.5 Option non implantées****29.6 Gestion mémoire**

# **Chapter 30**

## **Cas d'utilisation**

Cette section présentera et commenterá des paramétrages types adaptés aux configurations les plus courantes d'utilisation de MICMAC.

**30.1 MNT Spots**

**30.2 MNE Urbains**

**30.3 Superposition d'images colorées**

**30.4 Points homologues pour l'aéro-triangulation**



# Chapter 31

# Programmes Utilitaires

Cette section décrit différents *petits utilitaires* de traitement d'images qui, bien que ne faisant pas partie de la mise en correspondance à proprement parler peuvent rendre services dans le contexte d'utilisation de MicMac, en tant que pré ou post traitements.

## 31.1 Généralités

### 31.1.1 Génération du binaire

S'il n'existe pas, chaque binaire appelé `UnUtil`, peut être généré :

- se mettre sous la directory d'installation d'`Elise`;
- taper `make bin/UnUtil`;

Il suffit ensuite de lancer le binaire, avec ses argument; il est prudent de se placer sous la directory d'installation d'`Elise`(certains binaires pouvant aller y chercher des ressources à partir d'un chemin relatif):

- `bin/UnUtil arg0 arg1 ...`

### 31.1.2 Liste des arguments

Chaque commande possède des arguments obligatoires et des arguments optionnels. Les arguments obligatoires sont passés en premiers et ils sont identifiés par leur ordre de passage. Les arguments optionnels sont passé par noms, sous une suite de la forme `Tag=Val`. Ci dessous trois exemples d'utilisation de la commande `ScaleIm` :

- `bin/ScaleIm Lena.tif 1.2`
- `bin/ScaleIm Lena.tif 1.2 YScale=1.3 PO=[100,300]`
- `bin/ScaleIm Lena.tif 1.2 Y PO=[100,300] YScale=1.3`

Les deux dernières lignes sont équivalentes car l'ordre des arguments optionnels n'a pas d'importance. Les types d'argument connus et la syntaxe associée sont :

- **entier, réel, chaînes de caractères**, syntaxe "naturelle";
- **points 2D**, syntaxe `[x,y]`;

Il est important de n'avoir aucun blanc à l'intérieur d'un argument.

### 31.1.3 Aide en ligne

Pour chaque commande, un aide sommaire peut être obtenue en tapant :

- `bin/UnUtil -help`

L'aide est purement syntaxique, elle donne la liste des types des paramètres , ainsi que leur tag associés pour les paramètres optionnels. Par exemple, avec la commande `ScaleIm`, on obtiendra :

```
bin/ScaleIm -help
*****
* Help for Elise Arg main *
*****
Unnamed args :
 * string
 * REAL
Named args :
 * [Name=Out] string
```

```
* [Name=YScale] REAL
* [Name=Sz] Pt2dr
* [Name=P0] Pt2dr
```

## 31.2 L'utilitaire de changement d'échelle ScaleIm

### 31.2.1 Fonctionnalités

Cet utilitaire permet de calculer, à partir d'un fichier image, une même image à une échelle différente. Elle fonctionne aussi bien en agrandissement qu'en réduction. En agrandissement, c'est un interpolateur bicubique classique qui est utilisé (avec le paramètre  $-0.5$  qui interpole rigoureusement les fonctions linéaires). En réduction, c'est un bicubique sans valeur négative (paramètre  $0.0$ , avec une transition continue pour les échelles entre  $1$  et  $1.5$ ) qui est utilisé, après avoir été dilaté du facteur de réduction

Par défaut, le calcul est fait sur toute l'image, mais un cliping est possible.

Soit  $S^x$  et  $S^y$  les facteur de changement d'échelle et  $P_0$  une origine, la radiométrie de l'image  $I^{out}$  est définie à partir de celle de l'image d'entrée  $I^{in}$  par :

$$I^{out}(x, y) = I^{in}(P_0^x + S^x * x, P_0^y + S^y * y) \quad (31.1)$$

### 31.2.2 Paramètres

La liste des arguments a été donnée comme exemple de la section 31.1.3:

- le premier argument est le nom du fichier image en entrée;
- le deuxième argument est le facteur de changement d'échelle, comme l'indique la formule 31.1 un facteur  $S$  avec  $S < 1$  correspond à un agrandissement;
- le paramètre optionnel **Out** indique le nom du fichier de sortie, s'il est omis un nom est calculé à partir du fichier d'entrée selon la transformation régulière `.*\tif` donne `$1_Scaled\tif`;
- le paramètre optionnel **YScale** permet de donner le changement d'échelle en  $Y$  (par défaut égal à  $S$ );
- le paramètre optionnel **Sz** permet de spécifier la taille de l'image sur laquelle il faut effectuer la transformation (par défaut le maximum, donc toute l'image quand  $P^0 = (0, 0)$ );
- le paramètre optionnel **P0**, correspond au  $P_0$  de la formule 31.1.3, il définit donc l'origine "haut-gauche" de la portion d'image à laquelle est appliquée la transformation; par défaut  $P_0 = [0, 0]$ .

### 31.2.3 Evolutions possibles

Il serait possible assez facilement de pouvoir paramétriser totalement le noyau de convolution à condition qu'il reste séparable.

## 31.3 L'utilitaire d'ombrage GrShade

### 31.3.1 Fonctionnalités

Cette utilitaire part d'un image assimilée à un relief  $z = f(x, y)$  (par exemple un résultat de MicMac) et calcule son *ombrage* défini comme la portion de ciel visible. Cet ombrage a la caractéristique de mettre en relief les défaut de la corrélation. Ce n'est donc pas toujours l'outil adéquat pour comprendre le relief réel, mais c'est souvent un outil assez pratique pour juger de la qualité d'un résultat de corrélation.

Pour effectuer le calcul de la portion de ciel visible avec une complexité acceptable, le programme discrétise les directions du plan sur  $N$  valeurs et effectue un balayage sur ces  $N$  directions; pour chaque direction, on a un signal  $1D$  et le problème revient au calcul en chaque point de la pente du rayon tangent au point d'ombrage; pour ce problème, un algorithme récursif rapide permet de faire le calcul en temps linéaire.

Globalement, le temps de calcul est donc en  $N^{pix} * N^{dir}$  où  $N^{pix}$  est le nombre de pixels de l'image et  $N^{dir}$  est le nombre sur lequel on choisit de discrétiser les directions du plan.

### 31.3.2 Paramètres

Il y un seul paramètre obligatoire qui indique le nom du fichier image en entrée.

Les principaux paramètres optionnels sont les suivants :

- **FZ** : (*real*) définit le pas en  $z$  par lequel est multiplié le relief avant d'être ombré, vaut  $1.0$  par défaut; peut être négatif;

- **Out** : (*string*) nom du fichier de sortie, s'il n'est pas spécifié il vaut **InputShade.tif** (en supposant que l'entrée s'appelle **Input.tif**);
- **Anisotropie** : (*real*) si l'on vaut 0, on a un ombrage isotrope (toutes les directions sont équivalentes); plus il est proche de 1, plus on fait jouer un rôle privilégié aux directions proche du "nord"; il doit obligatoirement être compris entre 0 et 1; sa valeur par défaut est 0.95;
- **Dequant** : (*int*), les MNT sont souvent quantifiés, c'est le cas de ceux produits par MicMac; une fois ombré le MNT quantifié a un aspect relief en plateau (type rizière) qui peut être considéré comme un désagrément; si ce paramètre est  $\neq 0$ , un algorithme de "déquantification" est appliqué en amont de l'ombrage (algorithme de type interpolation linéaire), sa valeur par défaut est 0;
- **HypsoDyn** : (*real*), **HypsoSat** :

Les paramètres optionnels suivants sont d'usage moins courant:

- **Visu** : (*int*) indique de manière booléenne s'il y a visualisation en temps réel du calcul, vaut 0 par défaut;
- **P0** : (*Pt2di*), **Sz** : (*Pt2di*) définissent la boîte englobante sur laquelle est effectué le calcul;
- **NbDir** : (*int*) nombre de valeurs sur lequel sont discrétisées les directions du cercle, correspond à un compromis "qualité/temps de calcul" (valeur par défaut 20);
- **Brd** : (*int*) souvent les valeurs en bord d'images sont bruitées (voir sans significations), si ce sont des valeurs élevées elles ont une forte influence sur l'ombrage; ce paramètre permet de mettre sur un bord de taille **Brd** la valeur minimale du fichier afin que le bord n'influence pas l'ombrage, vaut 0 par défaut;
- **TypeMnt** : (*int*), **TypeShade** : (*int*), type des images temporaires utilisées pour le calcul temporaire; peut valoir **u\_int1,int1,u\_int2,int2,real4**, par défaut vaut **real4**; permet éventuellement d'économiser de la mémoire, usage déconseillé;

### 31.3.3 Evolutions possibles

L'algorithme permet, sans changer de complexité de prendre n'importe quelle fonction d'illumination (densité de lumière/stéradian en chaque point de la sphère). Il serait possible de laisser l'utilisateur spécifier sa propre fonction, ce qui pourrait avoir un intérêt dans l'utilisation pour des simulations "physique".

## 31.4 L'utilitaire de déquantification Dequant

## 31.5 L'utilitaire d'information sur un fichier tiff tiff\_info

## 31.6 L'utilitaire de test des expression régulière test\_regex

## 31.7 SupMntIm to superpose image and DTM

- ```
bin/SupMntIm A1 A2
— A1 = Absolute Name of Image
— A2 = Absolute Name of Mnt
— DynCoul , dynamique of colour, real, Def = 1.0
— CDN , generate level curve ? Boolean, Def = 0
```



# Part V

## Documentation programmeur



## **Chapter 32**

# **Ateliers**

## 32.1 C++ course under MicMac's library : Elise

### 32.1.1 Introduction and generalities

A C++ course was organized at ENSG (The National School of Geographic Sciences). The purpose of this course is to be able to create your own programs using the Elise library used in MicMac. In this section we start by giving some information about the location of each files that will be needed, then some examples that have been implemented during this session will be detailed and explained.

Here is a list of locations of files that will be used along the course:

- **/culture3d/src** : contains source files (extension .cpp)
- **/culture3d/include** : contains header files (extension .h)
- **/culture3d/include/XML\_MicMac** : contains **xml** files describing parameters for simplified tools as **Tapioca**, **Tapas**, ... etc
- **/culture3d/include/XML\_GEN** : contains **xml** files, and an associated header file (generated automatically from the **xml** file)
- **culture3d/src/CBinaires/** : contains binaries that can be called without using **mm3d**. This is maintained, but using **mm3d** is recommended
- **mm3d.cpp** specifies each command, with some commentary and log information

Some convention are used while developing MicMac tools. Here we give some of them in order to understand the approach adopted:

- a class called **toto** in an **xml** file will become **cToto**
- a member called **toto** in an **xml** file will become **mToto**

### 32.1.2 How to create a new .cpp file and compile it using the library Elise?

Start by creating a new file under the folder **/culture3d/src/TpMMPD** and call it **cExoMM\_CorrelMulImage.cpp**. Note that the file **ExoMM\_CorrelMulImage.cpp** under the same folder contains the solution of this course.

#### 32.1.2.1 Hello World !

The first exercise is displaying the famous “hello world” under the prompt. The most important thing here is to succeed to compile your directory **culture3d** with your new file **cExoMM\_CorrelMulImage.cpp**.

Under your favorite IDE, for example Geany, start by including this file **StdAfx.h** which contains all the headers of the library Elise. You are invited to check what is contained under **/culture3d/include/StdAfx.h**

```
#include "StdAfx.h"

int ExoMCI_main(int argc, char ** argv)
{
    std :: cout << "hello world" << "\n";
    return EXIT_SUCCESS;
}
```

**Achtung !** We need to tell the compiler that there is a new source file.  
Edit **Source.cmake** under the same folder and add this line :

```
set(Src_TD_PPMD ${TDPPMD_DIR}/cExoMM_CorrelMulImage.cpp
```

You also need to comment this line while your are doing this tutorial:

```
#${TDPPMD_DIR}/ExoMM_CorrelMulImage.cpp
```

In order to call our program we need to add in **culture3d/src/CBinaries/mm3d.cpp** the following line:

```
aRes.push_back(cMMCom("ExoMCI",ExoMCI_main,"Exo: Multi Correlation Image"));
```

This line should be added under :

```
const std::vector<cMMCom> & TestLibAvailableCommands() {...}
```

Check that the compilation works properly by typing as usual **make install** under “**/culture3d/build**”. Then if you type in:

```
mm3d TestLib ExoMCI
```

Your prompt should display “**hello world**”.

**Achtung !** In **/culture3d/src/CBinaries/mm3d.cpp**, *getAvailableCommands()* contains a list of commands accessible through the syntax: **mm3d MyCommand**. Its declaration in the file is:

```
const std::vector<cMMCom> & getAvailableCommands() {...}
```

*TestlibAvailableCommands()* vector contains the commands accessible through the syntax: **mm3d TestLib MyCommand**. It's our case above.

### 32.1.3 Mandatory or Optional Argument?

If you are a user of MicMac you know that calling a mandatory argument doesn't require to specify the name of the option. For example **Tapas** requires at least two arguments : model of distortion and a pattern, while optional arguments are specified by a name. For instance the option **InCal=** or **InOri=**.

Edit **cExoMM\_CorrelMulImage.cpp** and add this loop at the beginning :

```
for (int aK=0; aK<argc ; aK++)
    std::cout << "Argv[" << aK << "]=" << argv[aK] << endl;
```

Now, compile as before then type in the following command:

```
mm3d TestLib ExoMCI MyArg1 MyArg2
```

The prompt should display:

```
Argv[0] = ExoMCI
Argv[1] = MyArg1
Argv[2] = MyArg2
```

**EIInitArgMain** function is used to specify which arguments are optional and which are mandatory. This function is also used to display the help.

Here is a second example dealing with manipulation of arguments.

Modify your file **cExoMM\_CorrelMulImage.cpp** in order to contain the following code :

```
#include "StdAfx.h"

int ExoMCI_main(int argc,char ** argv)
{
    int I,J; //declaration of two arguments
    double D=1.0; //default value (for optional args)
```

```

ElInitArgMain
(
    argc, argv, //list of args
    LArgMain() << EAMC(I,"Left Operand")
//EAMC means mandatory argument
                << EAMC(J,"Right Operand"),
    LArgMain() << EAM(D,"D",true,"divisor of I+J")
//EAM means optional argument
);

    std::cout << "(I+J)/D = " <<(I+J)/D << std::endl;

    return EXIT_SUCCESS;
}

```

Compile again and type in the following command:

```
mm3d TestLib ExoMCI 1 4
```

The prompt should display:

```
(I+J)/D = 5
```

If you type:

```
mm3d TestLib ExoMCI 1 4 2
```

Then your prompt should display:

```
(I+J)/D = 2.5
```

### 32.1.4 How to load an xml file and read its informations?

Right now we will keep working with the Mini-Cuxha dataset (see **micmac.data**).

First, take a look at “**ParamChantierPhotogram.xml**”.

This file is under the folder “**culture3d/include/XML\_GEN/**”. It describes for each object, its type, options, ... etc under an xml formalism. In the Mini-Cuxha folder, the file **120601.xml** contains ground control points. Here are the first seven lines of this file:

```

<?xml version="1.0" ?>
<DicoAppuisFlottant>
    <OneAppuisDAF>
        <Pt>2.4167787000000006 42.59595130000003 496.23529999999995</Pt>
        <NamePt>12060100_172</NamePt>
        <Incertitude>1 1 1</Incertitude>
    </OneAppuisDAF>

```

You can check if the file “**120601.xml**” respects the formalism described in **ParamChantierPhotogram.xml**:

```

<DicoAppuisFlottant Nb="1" Class="true">
    <OneAppuisDAF Nb="*>
        <Pt Nb="1" Type="Pt3dr"> </Pt>
        <NamePt Nb="1" Type="std::string"> </NamePt>
        <Incertitude Nb="1" Type="Pt3dr"> </Incertitude>
    </OneAppuisDAF>
</DicoAppuisFlottant>

```

**Nb** parameter can be set to :

- 1 : this tag should appear once
- ? : this tag can appear once, but it's not mandatory
- \* : there can be given as many of these tags

Type parameter is a classical C++ type class, indeed some of MicMac's classes, as for instance: Pt3dr means a real 3D point, Pt2di means a 2D integer point, ... etc.

The function **StdGetFromPCP(aStr,aObj)**<sup>1</sup> describes how to link the xml file to its description. You can check /culture3d/include/private/files.h for more details.

Now, edit **cExoMM\_CorrelMulImage.cpp** in order to contain the following code:

```
#include "StdAfx.h"

int ExoMCI_main(int argc,char ** argv)
{
    std::string aNameFile; //will store your xml filename
    double D=1.0; //default value for optional argument
    ElInitArgMain //displays the help, and affect your command line to members
    (
        argc, argv, /arguments list
        LArgMain() << EAMC(aNameFile,"Left Operand"), //EAMC = mandatory argument
        LArgMain() << EAM(D,"D",true,"Unused") //EAM = optional argument
    );
    //the DicoAppuisFlottant (xml file) is converted to cDicoAppuisFlottant (c++)
    cDicoAppuisFlottant aDico= StdGetFromPCP(aNameFile,DicoAppuisFlottant);

    std::cout << "NbPts = " << aDico.OneAppuisDAF().size() << std::endl;

    return EXIT_SUCCESS;
}
```

Try to compile again an typing the following command :

```
mm3d TestLib ExoMCI 120601.xml
```

Your prompt should display : NbPts = 11

If you want to access to each individual element and display for instance the name and coordinates for each point, you should add a loop and browser each **OneAppuisDAF** like following:

```
std::list<cOneAppuisDAF> & aLGCP = aDico.OneAppuisDAF();
//OneAppuisDAF (xml) becomes cOneAppuisDAF (C++)
for ( //as long as we are in the same dictionary ==> browse each point
    std::list<cOneAppuisDAF>::iterator iT = aLGCP.begin();
    iT != aLGCP.end();
    iT++ )
{
    std::cout << iT->NamePt() << " " << iT->Pt() << "\n";
    //NamePt and Pt are the classes names in the xml
}
```

### 32.1.5 How to get list of files in a folder?

Here we start by declaring and defining two classes that we will use later in our global exercise which contains the algorithm of Multi Correlation Images. Our first class **cMCI\_Appli** concerns the application, and the second one **cMCI\_Ima** deals with image manipulations and will be defined in next section.

Now, edit again your file **cExoMM\_CorrelMulImage.cpp** in order to contain the following code:

```
#include "StdAfx.h"

//list of class
class cMCI_Appli;
```

---

1. PCP means ParamChantierPhotogram.h

```

class cMCI_Ima;

//classes declaration
class cMCI_Appli
{
    public :
        cMCI_Appli(int argc,char ** argv);
    private :
        std::list<std::string> mLFile;
        std::string mFullName;
        std::string mDir; //directory in which we are working
        std::string mPat; //pattern of images
        cInterfChantierNameManipulateur * mICNM;
};

cMCI_Appli::cMCI_Appli(int argc,char ** argv)
{
    bool aShowArgs=true;
    ElInitArgMain
    (
        argc, argv, //list of args
        //EAMC = mandatory argument
        LArgMain() << EAMC(mFullName,"Full Name (Dir+Pat)") ,
        //EAM = optional argument
        LArgMain() << EAM(aShowArgs,"Show",true,"Gives details on arguments")
    );

    SplitDirAndFile(mDir, mPat, mFullName);
    mICNM = cInterfChantierNameManipulateur::BasicAlloc(mDir);
    mLFile = mICNM->StdGetListOfFile(mPat);

    if (aShowArgs) ShowArgs();
}

void cMCI_Appli::ShowArgs()
{
    std::cout << "DIR = " << mDir << "Pat = " << mPat << "\n";
    std::cout << "Nb Files " << mLFile.size() << "\n";
    for (
        std::list<std::string>::iterator itS=mLFile.begin();
        itS != mLFile.end();
        itS++)
        { std::cout << "      F = " << *itS << "\n"; }

}
int ExoMCI_main(int argc,char ** argv)
{
    cMCI_Appli anAppli(argc,argv);
    return EXIT_SUCCESS;
}

```

Try to compile and from the **Mini-Cuxha** folder type in the following command:

```
mm3d TestLib ExoMCI ".*jpg"
```

Your prompt should display:

```
Nb Files 48
F = Abbey-IMG\_0173.jpg
F = Abbey-IMG\_0191.jpg
```

### 32.1.6 Epipolar geometry

Here, first we need to create a couple of images with an epipolar rectification. In the directory **Mini-Cuxha**, we can use the tool **mm3d CreateEpip** for completing this. If the name of our orientations computed is **RTL-Init**, type in the following command gives our couple of images needed:

```
mm3d CreateEpip Abbey-IMG_0173.jpg Abbey-IMG_0191.jpg RTL-Init
```

Then, edit your file **cExoMM\_CorrelMulImage.cpp** in order to contain the following code:

```
#include "StdAfx.h"

int ExoMCI_main(int argc,char ** argv)
{
    std::string aNameI1,aNameI2;
    int aPxMax= 199;
    int aSzW = 5;
    ElInitArgMain
    (
        argc,argv,
        LArgMain() << EAMC(aNameI1,"Name Image1")
        << EAMC(aNameI2,"Name Image2"),
        LArgMain() << EAM(aPxMax,"PxMax",true,"Pax Max")
    );

    Im2D_U_INT1 aI1 = Im2D_U_INT1::FromFileStd(aNameI1);
    Im2D_U_INT1 aI2 = Im2D_U_INT1::FromFileStd(aNameI2);

    Pt2di aSz1 = aI1.sz();
    Im2D_REAL4 aIScoreMin(aSz1.x,aSz1.y,1e10);
    Im2D_REAL4 aIScore(aSz1.x,aSz1.y);
    Im2D_INT2 aIPaxOpt(aSz1.x,aSz1.y);

    Video_Win aW = Video_Win::WStd(Pt2di(1200,800),true);

    for (int aPax = -aPxMax ; aPax <=aPxMax ; aPax++)
    {
        std::cout << "PAX tested " << aPax << "\n";
        Fonc_Num aI2Tr = trans(aI2.in_proj(),Pt2di(aPax,0));
        ELISE_COPY
        (
            aI1.all_pts(),
            rect_som(Abs(aI1.in_proj()-aI2Tr),aSzW),
            aIScore.out()
        );
        ELISE_COPY
        (
            select(aI1.all_pts(),aIScore.in()<aIScoreMin.in()),
            Virgule(aPax,aIScore.in()),
            Virgule(aIPaxOpt.out(),aIScoreMin.out())
        );
    }

    ELISE_COPY
    (
        aW.all_pts(),
        aIPaxOpt.in()[Virgule(FY,FX)]*3,
        aW.ocirc()
    );
    aW.clik_in();
```

```
    return EXIT_SUCCESS;  
}
```

Try to compile and from the **Mini-Cuxha** folder type in the following command:

```
mm3d TestLib ExoMCI Epi_Im1_Left_Abbey-IMG_0173_Abbey-IMG_0191.tif Epi_Im2_Right_Abbey-IMG_0173_Abbey-IMG_0191.tif
```

### 32.1.7 Multi Image Correlation

As seen above, we start by defining our two main classes. The class **cMCI\_Ima** contains for each image the information to store, geometry and radiometry:

```
class cMCI_Ima
{
public:
    cMCI_Ima(cMCI_Appli & anAppli,const std::string & aName);

    Pt2dr ClikIn();
    // Renvoie le saut de prof pour avoir un pixel
    double EstimateStep(cMCI_Ima *);

    void DrawFaisceaucReproj(cMCI_Ima & aMas,const Pt2dr & aP);
    Video_Win * W() {return mW;};
    void InitMemImOrtho(cMCI_Ima *); //initialization to right size

    void CalculImOrthoOfProf(double aProf,cMCI_Ima * aMaster);

    Fonc_Num FCorrel(cMCI_Ima *);
    Pt2di Sz(){return mSz;}

private :
    cMCI_Appli & mAppli;
    std::string mName;
    Tiff_Im mTifIm;
    Pt2di mSz;
    Im2D_U_INT1 mIm;
    Im2D_U_INT1 mImOrtho;

    Video_Win * mW;
    std::string mNameOri;
    CamStenope * mCam;

};
```

The class **cMCI\_Appli** contains the information of our application:

```
class cMCI_Appli
{
public :

    cMCI_Appli(int argc, char** argv);
    const std::string & Dir() const {return mDir;}
    bool ShowArgs() const {return mShowArgs;}
    std::string NameIm2NameOri(const std::string &) const;
    cInterfChantierNameManipulateur * ICNM() const {return mICNM;}

    Pt2dr ClikInMaster();

    void TestProj();
    void InitGeom();
    void AddEchInv(double aInvProf,double aStep)
    {
        mNbEchInv++;
        mMoyInvProf += aInvProf;
        mStep1Pix += aStep;
    }
private :
    cMCI_Appli(const cMCI_Appli &); //to avoid unwanted copies
```

```

void DoShowArgs(); //display args

std::string mFullName; //directory + pattern
std::string mDir; //directory of my dataset
std::string mPat; //pattern containing images
std::string mOri;
std::string mNameMast;
std::list<std::string> mLFile;
cInterfChantierNameManipulateur * mICNM;
std::vector<cMCI_Ima *> mIms; //vector of images
cMCI_Ima * mMasterIm; //master image because GeomImage
bool mShowArgs;
int mNbEchInv;
double mMoyInvProf;
double mStep1Pix;
};

}

```

Then for each class we define non inline functions members. For **cMCI\_Ima** :

```

//**
/*
*          cMCI_Ima
*/
//**StdCorrecNameOrient*****



cMCI_Ima::cMCI_Ima(cMCI_Appli & anAppli,const std::string & aName) :
    mAppli (anAppli),
    mName (aName),
    mTifIm (Tiff_Im::StdConvGen(mAppli.Dir() + mName,1,true)),
    mSz (mTifIm.sz()),
    mIm (mSz.x,mSz.y),
    mImOrtho (1,1),
    mW (0),
    mNameOri (mAppli.NameIm2NameOri(mName)),
    mCam (CamOrientGenFromFile(mNameOri,mAppli.ICNM()))
{
    ELISE_COPY(mIm.all_pts(),mTifIm.in(),mIm.out());

    if (0) // (mAppli.ShowArgs())
    {
        std::cout << mName << mSz << "\n";
        mW = Video_Win::PtrWStd(Pt2di(1200,800));
        mW->set_title(mName.c_str());
        ELISE_COPY(mW->all_pts(),mTifIm.in(),mW->ogray());
        //mW->clik_in();

        ELISE_COPY(mW->all_pts(),255-mIm.in(),mW->ogray());
        //mW->clik_in();
        std::cout << mNameOri
            << " F=" << mCam->Focale()
            << " P=" << mCam->GetProfondeur()
            << " A=" << mCam->GetAltiSol()
            << "\n";
        // 1- Test at very low level
        Im2D<U_INT1,INT> aImAlias = mIm;
        ELISE_ASSERT(aImAlias.data()==mIm.data(),"Data");
        U_INT1 ** aData = aImAlias.data();
        for (int anY=0 ; anY<mSz.y/3 ; anY++)
            for (int anX=0 ; anX<anY ; anX++)

```

```

{
    ElSwap(aData[anY][anX],aData[anX][anY]);
}
U_INT1 * aDataL = aImAlias.data_lin();

for (int anY=0 ; anY<mSz.y ; anY++)
{
    ELISE_ASSERT
    (
        aData[anY]==(aDataL+anY*mSz.x),
        "data"
    );
}
memset(aDataL+mSz.x*50,128,mSz.x*100);
ELISE_COPY(mW->all_pts(),mIm.in(),mW->ogray());

// 2- Test with functional approach

ELISE_COPY
(
    mIm.all_pts(),
    mTifIm.in(),
    // Output is directed both in window & Im
    mIm.out() | mW->ogray()
);

ELISE_COPY
(
    disc(Pt2dr(200,200),150),
    255-mIm.in()[Virgule(FY,FX)],
    // Output is directed both in window & Im
    mW->ogray()
);

int aSzF = 20;
ELISE_COPY
(
    rectangle(Pt2di(0,0),Pt2di(400,500)),
    // rect_som(mIm.in(),20)/ElSquare(1+2*aSzF),
    rect_som(mIm.in_proj(),aSzF)/ElSquare(1+2*aSzF),
    // Output is directed both in window & Im
    mW->ogray()
);

Fonc_Num aF = mIm.in_proj();
aSzF=4;
int aNbIter = 5;
for (int aK=0 ; aK<aNbIter ; aK++)
    aF = rect_som(aF,aSzF)/ElSquare(1+2*aSzF);

ELISE_COPY(mIm.all_pts(),aF,mW->ogray());

// ELISE_COPY(mIm.all_pts(),mTifIm.in(),mIm.out());

ELISE_COPY(mIm.all_pts(),mIm.in(),mW->ogray());

// 3- Test with Tpl approach

Im2D<U_INT1,INT4> aDup(mSz.x,mSz.y);
TIm2D<U_INT1,INT4> aTplDup(aDup);

```

```

TIm2D<U_INT1, INT4> aTIm(mIm);

for (int aK=0 ; aK<aNbIter ; aK++)
{
    for (int anY=0 ; anY<mSz.y ; anY++)
        for (int anX=0 ; anX<mSz.x ; anX++)
    {
        int aNbVois = ElSquare(1+2*aSzF);
        int aSom=0;
        for (int aDx=-aSzF; aDx<=aSzF ; aDx++)
            for (int aDy=-aSzF; aDy<=aSzF ; aDy++)
                aSom += aTIm.getproj(Pt2di(anX+aDx,anY+aDy));
        aTplDup.oset(Pt2di(anX,anY),aSom/aNbVois);
    }

    Pt2di aP;
    for (aP.y=0 ; aP.y<mSz.y ; aP.y++)
        for (aP.x=0 ; aP.x<mSz.x ; aP.x++)
            aTIm.oset(aP,aTplDup.get(aP));
    }
    ELISE_COPY(mIm.all_pts(),mIm.in(),mW->ogray());
}

void cMCI_Ima::CalculImOrthoOfProf(double aProf,cMCI_Ima * aMaster)
{
    TIm2D<U_INT1, INT> aTIm(mIm);
    TIm2D<U_INT1, INT> aTImOrtho(mImOrtho);
    int aSsEch = 10;
    Pt2di aSzR = aMaster->mSz/ aSsEch;
    TIm2D<float,double> aImX(aSzR);
    TIm2D<float,double> aImY(aSzR);

    Pt2di aP;
    for (aP.x=0 ; aP.x<aSzR.x; aP.x++)
    {
        for (aP.y=0 ; aP.y<aSzR.y; aP.y++)
        {
            Pt3dr aPTer = aMaster->mCam->ImEtProf2Terrain(Pt2dr(aP*aSsEch),aProf);
            Pt2dr aPIm = mCam->R3toF2(aPTer);
            aImX.oset(aP,aPIm.x);
            aImY.oset(aP,aPIm.y);
        }
    }

    for (aP.x=0 ; aP.x<aMaster->mSz.x; aP.x++)
    {
        for (aP.y=0 ; aP.y<aMaster->mSz.y; aP.y++)
        {
            /*
            Pt3dr aPTer = aMaster->mCam->ImEtProf2Terrain(Pt2dr(aP),aProf);
            Pt2dr aPIm0 = mCam->R3toF2(aPTer);
            */
            Pt2dr aPInt = Pt2dr(aP) / double(aSsEch);
            Pt2dr aPIm (aImX.getr(aPInt,0),aImY.getr(aPInt,0));
            float aVal = aTIm.getr(aPIm,0);
            aTImOrtho.oset(aP,round_ni(aVal)); //returns nearest integer
        }
    }
}

```

```

    }

    if ( 0 && (mName=="Abbey-IMG_0250.jpg"))
    {
        static Video_Win * aW = Video_Win::PtrWStd(Pt2di(1200,800));
        ELISE_COPY(mImOrtho.all_pts(),mImOrtho.in(),aW->ogray());
    }
}

Fonc_Num cMCI_Ima::FCorrel(cMCI_Ima *aMaster)
{
    int aSzW = 2;
    double aNbW = ElSquare(1+2*aSzW);
    Fonc_Num aF1 = mImOrtho.in_proj();
    Fonc_Num aF2 = aMaster->mImOrtho.in_proj();

    Fonc_Num aS1 = rect_som(aF1,aSzW) / aNbW;
    Fonc_Num aS2 = rect_som(aF2,aSzW) / aNbW;

    Fonc_Num aS12 = rect_som(aF1*aF2,aSzW) / aNbW - aS1*aS2;
    Fonc_Num aS11 = rect_som(Square(aF1),aSzW) / aNbW - Square(aS1);
    Fonc_Num aS22 = rect_som(Square(aF2),aSzW) / aNbW - Square(aS2);

    Fonc_Num aRes = aS12 / sqrt(Max(1e-5,aS11*aS22));

//static Video_Win * aW = Video_Win::PtrWStd(Pt2di(1200,800));
//ELISE_COPY(aW->all_pts(),128*(1+aRes),aW->ogray());

    return aRes;
}

void cMCI_Ima::InitMemImOrtho(cMCI_Ima * aMas)
{
    mImOrtho.Resize(aMas->mIm.sz());
}

Pt2dr cMCI_Ima::ClikIn()
{
    return mW->clik_in()_.pt; //returns 2D point when click on image
}

void cMCI_Ima::DrawFaisceauReproj(cMCI_Ima & aMas,const Pt2dr & aP)
{
    if (! mW) return ;
    double aProfMoy = aMas.mCam->GetProfondeur();
    double aCoef = 1.2;

    std::vector<Pt2dr> aVProj;
    for (double aMul = 0.2; aMul < 5; aMul *=aCoef) //steps of depth
    {
        Pt3dr aP3d = aMas.mCam->ImEtProf2Terrain(aP,aProfMoy*aMul);
        Pt2dr aPIm = this->mCam->R3toF2(aP3d);

        aVProj.push_back(aPIm); //creation of polyline
    }
    for (int aK=0 ; aK<((int) aVProj.size()-1) ; aK++)
        mW->draw_seg(aVProj[aK],aVProj[aK+1],mW->pdisc()(P8COL::red));
}
}

```

```

double cMCI_Ima::EstimateStep(cMCI_Ima * aMas)
{
    std::string aKey = "NKS-Assoc-CplIm2Hom@@dat";

    std::string aNameH = mAppli.Dir()
        + mAppli.ICNM()->Assoc1To2
        (
            aKey,
            this->mName,
            aMas->mName,
            true
        );
    ElPackHomologue aPack = ElPackHomologue::FromFile(aNameH);

    Pt3dr aDirK = aMas->mCam->DirK();
    for
    (
        ElPackHomologue::iterator iTH = aPack.begin();
        iTH != aPack.end();
        iTH++
    )
    {
        Pt2dr aPInit1 = iTH->P1();
        Pt2dr aPInit2 = iTH->P2();

        double aDist;
        Pt3dr aTer = mCam->PseudoInter(aPInit1,*(aMas->mCam),aPInit2,&aDist);

        double aProf2 = aMas->mCam->ProfInDir(aTer,aDirK);

        Pt2dr aProj1 = mCam->R3toF2(aTer);
        Pt2dr aProj2 = aMas->mCam->R3toF2(aTer);

        // std::cout << aMas->mCam->ImEtProf2Terrain(aProj2,aProf2) -aTer << "\n";

        if (0)
            std::cout << "Ter " << aDist << " " << aProf2
                << " Pix " << euclid(aPInit1,aProj1)
                << " Pix " << euclid(aPInit2,aProj2) << "\n";
        double aDeltaProf = aProf2 * 0.0002343;
        Pt3dr aTerPert = aMas->mCam->ImEtProf2Terrain
            (aProj2,aProf2+aDeltaProf);

        Pt2dr aProjPert1 = mCam->R3toF2(aTerPert);

        double aDelta1Pix = aDeltaProf / euclid(aProj1,aProjPert1);
        double aDeltaInv = aDelta1Pix / ElSquare(aProf2);
        // std::cout << "Firts Ecart " << aDelta1Pix << " " << aDeltaInv << "\n";
        mAppli.AddEchInv(1/aProf2,aDeltaInv);
    }
    return aPack.size();
}

```

Also for cMCI\_Appli:

```
/***/
```

```

/*
 *          cMCI_Appli
 */
/*
 ****
 */

cMCI_Appli::cMCI_Appli(int argc, char** argv):
    mNbEchInv (0),
    mMoyInvProf (0),
    mStep1Pix   (0)
{
    // Reading parameter : check and convert strings to low level objects
    mShowArgs=false;
    ElInitArgMain
    (
        argc,argv,
        LArgMain()  << EAMC(mFullName,"Full Name (Dir+Pat)")
                    << EAMC(mNameMast,"Name of Master Image")
                    << EAMC(mOri,"Used orientation"),
        LArgMain()  << EAM(mShowArgs,"Show",true,"Give details on args")
    );
}

// Initialize name manipulator & files
SplitDirAndFile(mDir,mPat,mFullName); //get our directory
mICNM = cInterfChantierNameManipulateur::BasicAlloc(mDir);
mLFile = mICNM->StdGetListOfFile(mPat); //get all files in the pattern

StdCorrecNameOrient(mOri,mDir); //correct given name

if (mShowArgs) DoShowArgs();

// Initialize all the images structure
mMastIm = 0;
for (
    std::list<std::string>::iterator itS=mLFile.begin();
    itS!=mLFile.end();
    itS++)
{
    cMCI_Ima * aNewIm = new cMCI_Ima(*this,*itS);
    mIms.push_back(aNewIm);
    if (*itS==mNameMast)
        mMastIm = aNewIm;
}

// Check the master is included in the pattern
ELISE_ASSERT
(
    mMastIm!=0,
    "Master image not found in pattern"
);

if (mShowArgs)
    TestProj();

InitGeom();
Pt2di aSz = mMastIm->Sz();
Im2D_REAL4 aImCorrel(aSz.x,aSz.y);
Im2D_REAL4 aImCorrelMax(aSz.x,aSz.y,-10);
Im2D_INT2 aImPax(aSz.x,aSz.y);

```

```

double aStep = 0.5; //unit in pixel
for (int aKPax = -60 ; aKPax <=60 ; aKPax++)
{
    std::cout << "ORTHO at " << aKPax << "\n";
    double aInvProf = mMoyInvProf + aKPax * mStep1Pix * aStep;
    double aProf = 1/aInvProf;

    for (int aKIm=0 ; aKIm<int(mIms.size()) ; aKIm++)
        mIms[aKIm]->CalculImOrthoOfProf(aProf,mMastIm);

    Fonc_Num aFCorrel = 0;
    for (int aKIm=0 ; aKIm<int(mIms.size()) ; aKIm++)
    {
        cMCI_Ima * anIm = mIms[aKIm];
        if (anIm != mMastIm)
            aFCorrel = aFCorrel+anIm->FCorrel(mMastIm);
    }
    ELISE_COPY(aImCorrel.all_pts(),aFCorrel,aImCorrel.out());
    ELISE_COPY
    (
        select(aImCorrel.all_pts(),aImCorrel.in()>aImCorrelMax.in()),
        Virgule(aImCorrel.in(),aKPax),
        Virgule(aImCorrelMax.out(),aImpax.out())
    );
}
Video_Win aW = Video_Win::WStd(Pt2di(1200,800),true);
ELISE_COPY(aW.all_pts(),aImpax.in()*6,aW.ocirc());
aW.clik_in();

}

void cMCI_Appli::InitGeom()
{
    for (int aKIm=0 ; aKIm<int(mIms.size()) ; aKIm++)
    {
        cMCI_Ima * anIm = mIms[aKIm];
        if (anIm != mMastIm)
        {
            anIm->EstimateStep(mMastIm) ;
        }
        anIm->InitMemImOrtho(mMastIm) ;
    }
    mMoyInvProf /= mNbEchInv;
    mStep1Pix /= mNbEchInv;
}

void cMCI_Appli::TestProj()
{
    if (! mMastIm->W()) return;
    while (1)
    {
        Pt2dr aP = ClikInMaster();
        for (int aKIm=0 ; aKIm<int(mIms.size()) ; aKIm++)
        {
            mIms[aKIm]->DrawFaisceaucReproj(*mMastIm,aP);
        }
    }
}

```

```
Pt2dr cMCI_Appli::ClikInMaster()
{
    return mMastIm->ClikIn();
}

std::string cMCI_Appli::NameIm2NameOri(const std::string & aNameIm) const
{
    return mICNM->Assoc1To1
    (
        "NKS-Assoc-Im2Orient@" + mOri + "@",
        aNameIm,
        true
    );
}

void cMCI_Appli::DoShowArgs()
{
    std::cout << "DIR=" << mDir << " Pat=" << mPat << " Orient=" << mOri << "\n";
    std::cout << "Nb Files " << mLFile.size() << "\n";
    for (
        std::list<std::string>::iterator itS=mLFile.begin();
        itS!=mLFile.end();
        itS++)
    {
        std::cout << "      F=" << *itS << "\n";
    }
}
```

Finally :

```
int ExoMCI_main(int argc, char** argv)
{
    cMCI_Appli anAppli(argc,argv);

    return EXIT_SUCCESS;
}
```

Try to compile again and type in the following command:

```
mm3d TestLib ExoMCI ".*jpg" Abbey-IMG_0279.jpg RTL-Init Show=1
```

**Achtung !** Note that you need to compute an orientation before. Here it's name is **RTL-Init** and the master image is **Abbey-IMG\_0279.jpg**.

## 32.2 C++ course under MicMac's library : Elise

This section contains the note taken by Ewelina Rupnik that I integrated in this chapter.

### 32.2.1 Introduction and generalities

A C++ course was organized at ENSG (The National School of Geographic Sciences). The purpose of this course was to learn how to add own equations in the bundle adjustment (Apero), as well as how to perform dense matching in epipolar geometry (MicMac). Additionally, a brief introduction to creation of own image filters within the Elise library was given.

This section provides the documentation of the 'matching' part of the course and is structured into three subsections. The subsections are as follows

- \* overview of the new classes and interfacing with *mm3d* in subsection 32.2.2,
- \* preprocessing (epipolar images; 3D mask; multiscale) in subsection 32.2.3,
- \* the matching (with and without regularization) in subsection 32.2.4,

The dataset used to test the code was a pair of terrestrial images. The images and its corresponding orientation files can be found in mercurial repository in the directory *YYYYY-EXO-XXXXXX*.

### 32.2.2 Overview of the new classes and interfacing with *mm3d*

Classes created to manipulate the images during matching

- \* **cOneImTDEpip** to store an image, the camera data, plus generate names and commands for downscaled images,
- \* **cLoadedImTDEpip** to store the image pair, parallax image and to do matching (in short)
- \* **cAppliTDEpip** to manage the entire pipeline of the matching as well as to store the user's input

Below are the steps to follow in order to interface the code with the *mm3d* tool

- \* create *cTD\_Epip.cpp* file to store the code and put it in *../culture3d/src/TpMMPD/*
- \* update *../culture3d/src/TpMMPD/Sources.cmake* to include the declaration of the file with your code  
 $\$\{TDPPMD_DIR\}/cTD_Epip.cpp$
- \* declare the main function in *../culture3d/include/general/arg\_main.h*  
 $\text{int TDEpip\_main(int argc, char \*\*argv);}$
- \* link the main function with the auxiliary list in *mm3d* (under *TestLibAvailableCommands()*)  
 $\text{aRes.push\_back(cMMCom("TDEpi", TDEpip\_main, "Test epipolar matcher"))};$

*TDEpi* is invoked from the terminal with

```
mm3d TestLib TDEpi
```

### 32.2.3 Preprocessing

#### 32.2.3.1 Epipolar images

The images are transformed to epipolar geometry by altering their orientations. In epipolar geometry homologous points of the left image lie along lines in the right image. The lines are coincident with the row of the left image and run parallel to the axes of the pixel coordinate system. This way the matching problem is reduced to a 1D search along image rows. Generating epipolar images is done with

```
mm3d CreateEpip IMGP7032.JPG IMGP7032.JPG RTL-Init Ori-CalPerIm
```

#### 32.2.3.2 3D mask

It is possible to restrict the volume that will be considered in matching directly in 3D space with the command below

```
mm3d SaisieMasqQT AperiCloud_CalPerIm.ply
```

**SaisieMasqQT** tool allows one to select the region of interest with polylines. With the F9 key one switches between selection and manipulation mode. F1 causes the help menu to appear. Two files are created at the output of the above operation. One of the files contains the list of selected 3D points, and the other one holds some transformation parameters.

### 32.2.3.3 Multiscale approach

Dense image matching is generally performed in a multiscale approach. Starting with low resolution and very rough approximation of the 3D scene, the result is propagated and improved at higher resolutions. Inside the **TDEpi** tool, the downscaling of the image is handled inside the constructor of the **cAppliTDEpip** class. The function executing the task (*GenerateDownScale*) has two input arguments that are the starting and final zoom (i.e. resolution). As a result, the images at given resolutions are saved on the hard drive of your PC.

```
void cAppliTDEpip :: GenerateDownScale( int aZoomBegin , int aZoomEnd )
{
    std :: list <std :: string> aLCom;
    for ( int aZoom = aZoomBegin ; aZoom >= aZoomEnd ; aZoom /=2 )
    {
        std :: string aCom1 = mIm1->ComCreateImDownScale(aZoom);
        std :: string aCom2 = mIm2->ComCreateImDownScale(aZoom);

        if (aCom1!="") aLCom.push_back(aCom1);
        if (aCom2!="") aLCom.push_back(aCom2);
    }

    cEl_GPAO :: DoComInParal(aLCom);
}
```

### 32.2.4 The matching

The general workflow of the dense image matching in MicMac (normalized cross correlation (NCC) as the similarity measure) is

- create the object Zinf, Zsup,
- fill the correlation object Corr(x,y,z),
- run optimization (if with regularization)

#### 32.2.4.1 The similarity measure

There exist a number of local similarity measures that are commonly used for dense matching. They can be either pixel-based (e.g. Census, Mutual Information) or window-based (e.g. normalized cross correlation (NCC)). The pixel-based features must always be accompanied by a regularization scheme in order to remove the matching ambiguities. Window-based features can be used with and without the regularization. The NCC measure is adopted in the following code.

In NCC there are five terms that need computation

- (a) the average intensity of each image,
- (b) the average of the squared value of the image intensities,
- (c) the variances of the two windows being examined, and
- (d) their covariance.

The values of (a) and (b) are computed in the constructor of the **cLoaedImTDEpip** class. They are computed once and their values are stored inside the class.

```
cLoaedImTDEpip :: cLoaedImTDEpip(cOneImTDEpip & aOIE, double aScale , int aSzW) :
    mOIE (aOIE),
    mAppli (aOIE.mAppli),
    mNameIm (aOIE.NameFileDownScale( aScale )),
    mTifIm (mNameIm. c_str ()),
    mSz (mTifIm. sz ()),
    mTIm (mSz),
    mIm (mTIm. _the_im),
    mTImS1 (mSz),
    mImS1 (mTImS1. _the_im),
    mTImS2 (mSz),
    mImS2 (mTImS2. _the_im),
    mTPx (mSz),
```

```

mIPx      (mTPx. _the_im ) ,
mTSc      (mSz) ,
mISc      (mTSc. _the_im ) ,
mMasqIn   (mSz .x ,mSz .y ,1 ) ,
mTMIn    (mMasqIn) ,
mMasqOut (mSz .x ,mSz .y ,0 ) ,
mTMOut   (mMasqOut) ,
mMaxJumpPax (2) ,
mRegul     (0.05)
{
    ELISE_COPY(mIm. all_pts () , mTifIm. in () ,mIm. out ());
    ELISE_COPY(mMasqIn. border (aSzW) ,0 ,mMasqIn. out ());
    ELISE_COPY
    (
        mIm. all_pts () ,
        rect_som(mIm. in_proj () ,aSzW) / ElSquare(1+2*aSzW) ,
        mImS1. out ()
    );
    ELISE_COPY
    (
        mIm. all_pts () ,
        rect_som(Square(mIm. in_proj ()) ,aSzW) / ElSquare(1+2*aSzW) ,
        mImS2. out ()
    );
}

```

The values of (c), (d) and the final NCC measure are computed in the *CrossCorrelation* method:

```

double cLoaedImTDEpip :: CrossCorrelation
(
    const Pt2di & aPIm1 ,
    int aPx ,
    const cLoaedImTDEpip & aIm2 ,
    int aSzW
)
{
    if (! InsideW(aPIm1 ,aSzW)) return TheDefCorrel;

    Pt2di aPIm2 = aPIm1 + Pt2di(aPx ,0 );
    if (! aIm2. InsideW(aPIm2 ,aSzW)) return TheDefCorrel;

    double aS1 = mTImS1. get (aPIm1 );
    double aS2 = aIm2 .mTImS1 .get (aPIm2 );

    double aCov = Covariance(aPIm1 ,aPx ,aIm2 ,aSzW) -aS1*aS2 ;

    double aVar11 = mTImS2 .get (aPIm1 ) - ElSquare(aS1 );
    double aVar22 = aIm2 .mTImS2 .get (aPIm2 ) - ElSquare(aS2 );

    return aCov / sqrt(ElMax(1e-5,aVar11*aVar22));
}

```

### 32.2.4.2 Matching without regularization

Matching without the regularization comes down to finding for each pixel in the left image its parallax in the right image for which the similarity of that pair of pixels is highest.

The matching procedures at each zoom (i.e. resolution) level commence inside the **cAppliTDEpip** class. For every image of the pair the **cLoaedImTDEpip** object is instantiated, and the *ComputePx* initiates the parallax calculation.

```
void cAppliTDEpip :: DoMatchOneScale( int aZoom , int aSzW )
{
    mCurZoom = aZoom ;
    cLoaedImTDEpip aLIm1(*mIm1,aZoom,aSzW) ;
    cLoaedImTDEpip aLIm2(*mIm2,aZoom,aSzW) ;

    aLIm1 . ComputePx( aLIm2 , round_up( mIntPx/aZoom ) ,aSzW ) ;
}
```

The *ComputePx* creates a *matching envelope* constrained by *aTEnvInf* and *aTEnvSup* i.e. a region in the parallax space that will be exploited during matching. Initially the *envelope* is a rectangle constrained by  $< -aPxMax, +aPxMax >$ , and as the matching proceeds at higher resolution levels, the search space adjusts to the true 3D object scene.

```
void cLoaedImTDEpip :: ComputePx
(
    cLoaedImTDEpip & aIm2 ,
    INT aPxMax ,
    int aSzW
)
{
    TIm2D<INT2, INT> aTEnvInf(mSz) ;
    TIm2D<INT2, INT> aTEnvSup(mSz) ;

    ELISE_COPY( aTEnvInf . _the_im . all_pts () , -aPxMax , aTEnvInf . _the_im . out () );
    ELISE_COPY( aTEnvSup . _the_im . all_pts () , 1+aPxMax , aTEnvSup . _the_im . out () );

    ComputePx( aIm2 , aTEnvInf , aTEnvSup , aSzW ) ;
}
```

The overloaded *ComputePx* does the effective job of computing the NCC on a pair of images. The method iterates over all pixels in the left image, and computes the NCC for every parallax from the range embedded inside *aTEnvInf* and *aTEnvSup*. The best parallax (i.e. of highest similarity) is saved to *mTPx*, while the NCC is stored in *mTSc*. At each pixel it is verified whether the matching is within the defined region of interest (line 13).

```
for ( aP . x = 0 ; aP . x < mSz . x ; aP . x++)
{
    for ( aP . y = 0 ; aP . y < mSz . y ; aP . y++)
    {
        int aPxMin = aTEnvInf . get ( aP ) ;
        int aPxMax = aTEnvSup . get ( aP ) ;
        int aBestPax = 0 ;
        double aBestCor = TheDefCorrel ;

        for ( int aPax = aPxMin ; aPax < aPxMax ; aPax++)
        {
            double aCor = TheDefCorrel ;
            if ( In3DMasq( aP , aPax , aIm2 ) )
            {
                aCor = CrossCorrelation( aP , aPax , aIm2 , aSzW ) ;
                if ( aCor > aBestCor )
                {
                    aBestCor = aCor ;
                    aBestPax = aPax ;
                }
            }
        }
        /// ONLY WHEN REGULARIZATION ON
        /// PRGD 2 : fill the cost
        aSparsPtr [ aP . y ] [ aP . x ] [ aPax ] . SetOwnCost( ToICost( 1-aCor ) ) ;
    }
}
```

```

        /// == End PGRD2
    }
    mTPx. oset (aP , aBestPax );
    mTSc. oset (aP , ElMax (0 , ElMin (255 , round_ni (( aBestCor +1 ) * 128 ))));
}
}

Tiff_Im :: CreateFromIm (mTPx. _the_im , " TestPx . tif " );
Tiff_Im :: CreateFromIm (mTSc. _the_im , " TestSc . tif " );

std :: cout << "DONE PX \n ";

```

### 32.2.4.3 Matching with regularization

Matching with regularization differs from the example shown above in that the computed parallaxes apart from being conditioned on the NCC values, are conditioned on the parallaxes of neighbouring pixels. The images are parsed into lines, and for every line a *tCelOpt* structure is created. Solving for the optimal parallaxes along that line is done with dynamic programming. In order to avoid the 'streaking' effects that are pertinent to 1D matching, the images are parsed into lines along a number of different directions. The end result is a combination of all directions (so it is a version of Semi-Global Matching).

The optimization is handled by the **cProg2DOptimiser** class defined in `../culture3d/include/im_tpl/ProgDyn2D.h` (see an example of using the optimizer in `../culture3d/src/uti_phgrm/MICMAC/FusionCarteProf.cpp`).

First up, an object of the **cProg2DOptimiser** class is created and is templated with the **cLoaedImTDEpip** class. The *aSparseVol* is of type `cDynTplNappe3D` and it stores the matching costs (i.e. the *cube* being the volume between two surfaces that constrain the matching search space). The *cube* is composed of cells – *tCelNap*, that can be accessed via the *aSparsPtr* pointer variable. The code from line 36 – 66 is almost identical to matching without regularization. The only novelty is the update of the *cube* in line 58. Note that (i) the cube elements are accessed in a reversed order of the coordinates i.e. Y, X and Z, and (ii) it is the cost rather than the correlation value that is inserted inside the cube.

Having computed the costs for all pixels and parallaxes, the optimization is run in line 67 where the input argument corresponds to the numbers of directions that will be exploited during the matching. Line 69 recovers the final and hopefully most optimal parallaxes from the optimizer, line 70 saves it to an image file.

```

void cLoaedImTDEpip :: ComputePx
(
    cLoaedImTDEpip & aIm2 ,
    TIm2D<INT2 , INT>      aTEnvInf ,
    TIm2D<INT2 , INT>      aTEnvSup ,
    int aSzW
)
{
    if (0)
    {
        Video_Win aW = Video_Win :: WStd (mSz , 2 );

        Fonc_Num aF = mIm. in (0);
        for (int aK=0 ; aK<4 ; aK++)
            aF = MySom (aF , 2 ) / 25;

        ELISE_COPY
        (
            mIm. all_pts () ,
            rect_min (rect_max (255-aF , 3 ) , 3 ) ,
            aW. ogray ()
        );
        aW. clik_in ();
    }
}

```

```

Pt2di aP;
/// PRGD 1 : create the object

cProg2DOptimiser<cLoaedImTDEpip> aPrgD(*this ,aTEnvInf._the_im ,aTEnvSup._the_im ,0 ,1);
cDynTplNappe3D<tCelNap> & aSparseVol = aPrgD.Nappe();
tCelNap *** aSparsPtr = aSparseVol.Data() ;
/// -- end PRGD 1

for (aP.x =0 ; aP.x < mSz.x ; aP.x++)
{
    for (aP.y =0 ; aP.y < mSz.y ; aP.y++)
    {
        int aPxMin = aTEnvInf.get(aP);
        int aPxMax = aTEnvSup.get(aP);
        int aBestPax = 0;
        double aBestCor = TheDefCorrel;

        for (int aPax = aPxMin ; aPax < aPxMax ; aPax++)
        {
            double aCor = TheDefCorrel;
            if (In3DMasq(aP,aPax,aIm2))
            {
                aCor = CrossCorrelation(aP,aPax,aIm2,aSzW);
                if (aCor > aBestCor)
                {
                    aBestCor = aCor;
                    aBestPax = aPax;
                }
            }
        }
        // PRGD 2 : fill the cost
        aSparsPtr[aP.y][aP.x][aPax].SetOwnCost(ToICost(1-aCor));
        // == End PGRD2
    }
    mTPx.oset(aP,aBestPax);
    mTSc.oset(aP,ElMax(0,ElMin(255,round_ni((aBestCor+1)*128))));
}
}

/// PRGD3 : run the optim and use the result
aPrgD.DoOptim(7);
Im2D_INT2 aSolPrgd(mSz.x,mSz.y);
aPrgD.TranfereSol(aSolPrgd.data());
Tiff_Im::CreateFromIm(aSolPrgd,"TestPrgPx.tif");
/// end PRGD3

if (1)
{
    Video_Win aW = Video_Win::WStd(mSz,3);

    ELISE_COPY
    (
        mTPx._the_im.all_pts(),
        Min(2,Abs(mTPx._the_im.in()-aSolPrgd.in())),
        aW.odisc()
    );
    aW.clik_in();
}

```

```

Tiff_Im :: CreateFromIm(mTPx._the_im ,” TestPx.tif ”);
Tiff_Im :: CreateFromIm(mTSc._the_im ,” TestSc.tif ”);

std :: cout << ”DONE PX\n”;
}

```

Within the regularization phase, the optimizer runs the *DoConnexion* method for every pixel in every direction. The input arguments are

- \* aPIn, aPOut : pixel coordinates of two neighbouring points on the line
- \* aSens ? aRab aMul
- \* Input : a column of cells corresponding to the costs of aPIn computed at different parallaxes
- \* aInZMin, aInZMax : the min and max parallax at aPIn (i.e. the start and the end of the column)
- \* Output : a column of cells corresponding to the costs of aPOut computed at different parallaxes
- \* aOutZMin, aOutZMax : the min and max parallax at aPOut (i.e. the start and the end of the column)

First, the connectivity of the current pixel is retrieved (line 13, see definition in ..//culture3d/src/util/num.cpp), and further used in assigning the its cost (line 18). Each time the *UpdateCostOneArc* method is called, it adds a connection between the aPOut and the aPIn at the current parallax.

```

void cLoaedImTDEpip :: DoConnexion
(
    const Pt2di & aPIn, const Pt2di & aPOut,
    ePrgSens aSens, int aRab, int aMul,
    tCelOpt*Input, int aInZMin, int aInZMax,
    tCelOpt*Output, int aOutZMin, int aOutZMax
)
{
    for (int aZ = aOutZMin ; aZ < aOutZMax ; aZ++)
    {
        int aDZMin,aDZMax;
        ComputeIntervaleDelta
        (
            aDZMin,aDZMax,aZ,mMaxJumpPax,
            aOutZMin,aOutZMax,
            aInZMin,aInZMax
        );
        for (int aDZ = aDZMin; aDZ<= aDZMax ; aDZ++)
        {
            double aCost = mRegul * ElAbs(aDZ);
            Output[aZ].UpdateCostOneArc(Input[aZ+aDZ],aSens,ToICost(aCost));
        }
    }
}

```

## 32.3 Visual interfaces "vCommands"

### 32.3.1 Introduction

Each command of **MicMac** can be called in a command line prompt with the general syntax:

```
mm3d Command arg1 arg2 ... argn NameOpt1=Argot1 ...
```

For example, a possible call to the **Tapas** tool is:

```
mm3d Tapas RadialStd ".*.PEF" Out=All
```

To help filling arguments for **MicMac** commands, visual interfaces based on **Qt** can be launched by adding the letter "v" in front of the command name. For example, a possible call to the **Tapas** visual interface is:

```
mm3d vTapas
```

An other possible call to the **Tapas** visual interface is:

```
mm3d vTapas RadialStd ".*.PEF" Out=All
```

This will fill visual interface with corresponding arguments.

NB: this is also true for some commands in **TestLib** such as:

```
mm3d TestLib vOriMatis2MM
```

### 32.3.2 Compilation and code

Visual interfaces are available with option **WITH\_QT4** or **WITH\_QT5** activated.

```
cmake -DWITH_QT4=ON ..
```

or

```
cmake -DWITH_QT5=ON ..
```

If necessary, see **CMakeLists.txt**.

At revision 5520, code is located in:

```
include/general/visual_mainwindow.h
include/general/visual_buttons.h
src/util/visual_main_window.cpp
src/util/visual_buttons.cpp
src/util/visual_arg_main.cpp
src/util/arg_main.cpp
src/CBinaires/mm3d.cpp
```

### 32.3.3 How it works?

Each command has a set of mandatory arguments, and may have a set of optional arguments. Basically, a visual interface is shown by parsing the two lists of mandatory and optional arguments, getting their type, and depending on their type, displaying in a widget the corresponding selection object (ComboBox, buttons, text edition field, button, or **SaisieBoxQT**, etc.).

This is based on **EIInitArgMain** method from **arg\_main.cpp**, which is usually called in the main method for each command in **MicMac**. This method fills the two lists of mandatory and optional arguments, following this syntax:

```
std::vector <char *> ElInitArgMain
(
    int argc,char ** argv,
    const LArgMain & LGlob,
    const LArgMain & L1,
    const std::string & aFirstArg = "",
    bool VerifInit=EIAM_VerifInit,
    bool AccUnK=EIAM_AccUnK,
    int aNbArgGlobGlob = EIAM_NbArgGlobGlob
);
```

LGlob contains the list of mandatory arguments, while L1 has optional arguments.  
Adding an argument is usually done like this:

```
int Function_main(int argc,char ** argv)
{
    string aFullName, aOri, aPly, aOut;

    ElInitArgMain
    (
        argc,argv,
        LArgMain() << EAMC(aFullName,"Full Name (Dir+Pat)")
                    << EAMC(aOri,"Orientation path")
                    << EAMC(aPly,"Ply file"),
        LArgMain() << EAM(aOut,"Out",true,"Output filename")
                    etc.
    );

    etc.
}
```

To transform an existing MicMac function into a function which can be launched in visual mode, one has to specify the type of the arguments. For example, here:

```
int Function_main(int argc,char ** argv)
{
    string aFullName, aOri, aPly, aOut;

    ElInitArgMain
    (
        argc,argv,
        LArgMain() << EAMC(aFullName,"Full Name (Dir+Pat)",eSAM_IsPatFile)
                    << EAMC(aOri,"Orientation path",eSAM_IsExistDirOri)
                    << EAMC(aPly,"Ply file", eSAM_IsExistFile),
        LArgMain() << EAM(aOut,"Out",true,"Output filename")
                    etc.
    );

    etc.
}
```

The arguments types can be:

- eSAM\_IsPatFile, for a pattern string,
- eSAM\_IsBool, for a bool,
- eSAM\_IsPowerOf2, for an integer power of 2
- eSAM\_IsDir, for a directory string
- eSAM\_IsExistDirOri, for an existing Orientation directory string,
- eSAM\_IsOutputDirOri, for an Output Orientation directory string,
- eSAM\_IsExistFile, for an existing file string,
- eSAM\_IsExistFileRP, for an existing file to be given with a relative path
- eSAM\_IsOutputFile, for an output file string,

- `eSAM_Normalize`, for a 2d box that has to be normalized (`Box2dr`),
- `eSAM_NoInit`, for an argument that has not been initialized,
- `eSAM_InternalUse`, for an argument that we don't want to display in the visual interface,
- `eSAM_None`, for a list of strings.

Type has not to be specified for an integer, a float, a point (`Pt2di`, `Pt2dr`), a box terrain (`Box2dr`).

To check if a visual interface has to be launched a global variable `MMVisualMode` is set to `true` in `GenMain` in `src/CBinaires/mm3d.cpp`. When calling `mm3d` for a visual interface, we first run the `Function_main` with its `EIInitArgMain` to fill the visual interface, then we run a second time `mm3d` with `MMVisualMode` set to `false`, to take into account modifications done in the visual interface, and to run the actual process.

At the first call to `Function_main`, we just want to go through `EIInitArgMain`, to show the visual interface, so we need to exit this function without doing the main process. This is why we have to add after `EIInitArgMain`:

```
if (MMVisualMode) return EXIT_SUCCESS;
```

Another small trick is done to enable user to set some arguments directly in the command line (which will be automatically filled in the visual interface). If command line contains more than `mm3d vCommand`, we initialize arguments with these values by a call to `EIInitArgMain` in `arg_main.cpp`.

```
if(argc > 1)
{
    MMVisualMode = false;
    EIInitArgMain(argc, argv, LGlob, L1, aFirstArg, VerifInit, AccUnK, aNbArgGlobGlob);
    MMVisualMode = true;
}
```

NB: there is a bug in this part, since we check if an argument has been modified in the visual interface, and this state is set to unchanged when we call `EIInitArgMain` twice.

In `EIInitArgMain`, `aFirstArg` is used to set the widget title.

### 32.3.4 visual\_MainWindow class

In `include/general/visual_mainwindow.h`, we define a class derived from `QWidget`, `visual_MainWindow`. A `visual_MainWindow` is mainly composed of 2 `QGridLayout` where mandatory and optional arguments are displayed in rows. Optional arguments are sorted with regard to their name (`cMMSpecArg::NameArg()`). At the widget's bottom, a button "Run command" runs the command with the selected arguments (*slot onRunCommandPressed*) ; a checkbox "Show dialog when job is done" allows user to continue working, and force a dialog to pop up when process is finished.

Depending on the argument's type, specific objects are created in the corresponding row (see `buildUI` method). A label is added systematically at the left (see `add_label` method), using argument comment (for mandatory argument) or name (for optional argument). A tool tip is added for optional arguments with comment, and is shown (when available) by putting cursor over the argument name.

Dealing with files: many commands need several files, and sometimes several directories, which may be located in the same directory. To help choosing these files or directory, we store the first directory (`mLastDir`). We also store this directory in application settings, so that, at next call, the first open dialog is set to the last directory.

Pushing "Run command" button, we parse vector `vInputs` that stores the argument (as `cMMSpecArg`), and build command line `aCom` for `mm3d`, adding only arguments that has been changed (see `cMMSpecArg::IsDefaultValue()`).

### 32.3.5 Specific functions: vTapioca, vMalt, vC3DC, vSake

Some functions have a slightly different workflow and behavior than the majority. These functions need to choose between several modes before calling `EIInitArgMain`. This mode is recovered with a `QInputDialog`. See for example `CPP_Tapioca.cpp`.

`vMalt` has also some specific behavior depending on mode (Ortho, UrbanMNE, GeomImage). This is dealt in function `visual_MainWindow::moveArgs` with boolean `_bMaltGeomImg`. `vMalt` has also two small special

behaviors, dealt with function **isFirstArgMalt**: disabling mandatory argument edition after choosing mode in **add\_select**, and recovering mode in the final command line in **onRunCommandPressed**.

### 32.3.6 BoxClip and BoxTerrain

Some functions need a 2d rectangular selection information (mainly to perform computations on reduced area). Most of the time, argument name are **BoxClip** and **BoxTerrain**. Previously, user had to measure two corners coordinates in image, and sometimes normalize these coordinates, then type argument for example, **BoxClip=[0.13,0.11,0.89,0.87]**. Here, we use a new tool based on **SaisieQT** (see next chapter), called **SaisieBoxQT**, which is launched with "Selection editor" button (see **visual\_MainWindow::onSaisieButtonPressed**). User first choose which image to open, then draw a rectangle by click-n-drag in the image. User can also edit the rectangle selection afterward, by clicking close to a corner and drag it. We must deal here with 3 cases: true image coordinates, normalized image coordinates, and box terrain coordinates. Normalization is specified using **eSAM\_Normalize**. Difference between box image and box terrain is done with the argument type (**Box2di** or **Box2dr**). For a box terrain, a **FileOriMnt xml** has to be read to convert image coordinates to terrain coordinates through function **visual\_MainWindow::transfoTerrain**.

## 32.4 SaisieQT

### 32.4.1 Introduction

**SaisieQT** is a Qt application gathering a set of commands designed to mimic and extend the **SaisiePts** X11 tool originally used for measuring data in images for **MicMac**. It is cross-platform. At revision 5520, there is 6 Qt tools: **SaisieMasqQT**, **SaisieAppuisInitQT**, **SaisieAppuisPredicQT**, **SaisieCylQT**, **SaisieBascQT** and **SaisieBoxQT**.

**SaisieMasqQT**, **SaisieAppuisInitQT**, **SaisieAppuisPredicQT**, **SaisieCylQT** and **SaisieBascQT** are designed as independent applications, while **SaisieBoxQT** is, for now, only called through the visual interfaces (it does not output measures in a **xml** file, it only sends data through *slot/signal* connections).

### 32.4.2 Compilation and code

**SaisieQT** tools are available with option **WITH\_QT4** or **WITH\_QT5** activated.

```
cmake -DWITH_QT4=ON ..
```

or

```
cmake -DWITH_QT5=ON ..
```

If necessary, see **CMakeLists.txt** and **src/SaisieQT/CMakeLists.txt**.

At revision 5520, code is located in:

```
src/uti_phgrm/CPP_SaisieQT.cpp
src/SaisieQT/
include/qt/
```

When building binaries, one has to copy translation files **.qm** and style sheet **.qss** from **include/qt/** in the same directory, next to **bin/** directory. Scripts that build binaries and packages already do this, but this is to be remembered.

### 32.4.3 How it works?

**SaisieQT** is a unique binary, based on a core structure deriving from **QMainWindow** (for now) and from **GLWidgetSet: SaisieQtWindow**. To follow and conform to the *universal* command **mm3d** an alias command is defined in **src/uti\_phgrm/CPP\_SaisieQT.cpp** so

```
mm3d SaisieMasqQT IMG_5059.JPG
```

actually runs:

```
SaisieQT SaisieMasqQT IMG_5059.JPG
```

`SaisieQT` then dispatches to each function main in `SaisieQT/main/saisieQT_main.cpp` depending on second argument (`SaisieMasqQT` etc.).

All applications share the same style sheet, loaded in `saisieQT_main.cpp` and stored in `include/qt/style.qss`.

Each application has its own settings (stored depending on the OS). Most of the settings can be edited through a `QDialog`: `cSettingsDialog` defined in `include_QT/Settings.h`.

Switching between each application in code is managed with private member `_appMode` of `SaisieQtWindow` class. Corresponding enum is defined in `include_QT/Settings.h`.

Some of the Saisie Qt tools make use of Elise library, and use the same core as `SaisiePts` to compute 3D points from image measures, epipolar lines, etc. To mimic the way `SaisiePts` works, a class `cVirtualInterface` has been created in `include/SaisiePts/SaisiePts.h`. This class owns all methods that are shared both by `SaisiePts` and `SaisieQT`. Two classes derive from this mostly virtual class: `cX11_Interface` in `SaisiePts`, and `cQT_Interface` in `SaisieQT`. `cQT_Interface` needs `cAppli_SaisiePts` and a `SaisieQtWindow` to be instantiated.

### 32.4.4 SaisieMasqQT

`SaisieMasqQT` has 2 modes: a 2D mask selection mode, like X11 `SaisieMasq`, and a 3D mask selection mode, useful for C3DC command. `SaisieMasqQT` uses the same command line arguments as `SaisieMasq` which are read in `saisieMasq_EIInitArgMain`, function common to both. These arguments are provided to `SaisieQtWindow` afterward.

All the data in `SaisieMasqQT` are rendered in an OpenGL context. These data are stored in `cGLData` class. We use an ortho projection to render them (see `MatrixManager::mgOrtho`). The main container, after `SaisieQtWindow`, is `GLWidget` (derived from `QGLWidget`). Projection matrix and projection functions (from image to window, and back) can be found in `MatrixManager` class.

#### 32.4.4.1 2D mode

In 2D mode, `SaisieMasqQT` loads one image, and displays it in the center of the viewport.

An image, at first glance, is stored as a `cMaskedImageGL` (`3DObject.h`) which contains both image data and mask information.

To deal with some very big images, we show a rescaled image in full size, and draw only visible tiles, at full scale when zooming in image. In this case, while loading image, a scale factor is computed, and a rescaled image is stored, next to the original image in `cMaskedImageGL`, and a vector of full scale image tiles in `cGLData::glMaskedTiles`.

An image is drawn as a `GL_QUAD` (see `cImageGL::drawQuad`). When a mask has been measured, it is blended over the image (see `cMaskedImageGL::draw()`).

Drawing vector data (polygons, points, text) is done in `GLWidget::overlay()`.

Editing a mask is done in `cGLData::editImageMask` (`cGLData.cpp`) by drawing a polygon (class `cPolygon`).

#### 32.4.4.2 3D mode

3D mode allows loading a ply file (only point clouds, for now). 6 ply formats are currently managed (xyz, xyzrgb, xyz nx ny nz, xyzrgba, xyz nx ny nz rgb, xyz nx ny nz rgba) (see `GlCloud::loadPly`).

There are two interaction modes: selection (one can draw a polygon, as in 2D mode), and move (rotate or translate camera). In `GLWidget` switching between the two modes is done with `getInteractionMode()` and `m_interactionMode`. In the *gui*, F9 shortcut switches between the 2 modes.

Editing a mask is done in `cGLData::editCloudMask` (`cGLData.cpp`). Editing a mask consists of two different operations: for each 3d point, decide if it is inside or outside the mask, and also store the mask selection information (to be able to recover the mask from other `mm3d` commands, such as C3DC). A mask consists of the

intersection of several 3D cones, each 3D cone being defined by a 3D polygon (the cone section) and a direction. 3D polygon is built from the 2D polygon drawn in viewport and its camera and matrix information. Cone direction is known from camera orientation. So for each couple (polygonal selection-openGL camera), a virtual 3D cone has to be stored (see 32.5). These information are stored in a vector of `selectInfos`, in `HistoryManager`. This allows to undo/redo actions, and also to edit actions through a `QAbstractTableModel` (`QTableView`-`Objects`).

### 32.4.5 SaisieAppuisInitQT and SaisieAppuisPredicQT

In `SaisieAppuisInitQT` and `SaisieAppuisPredicQT`, we use the same `cPolygon` object to draw a set of points, but we don't draw lines between points. This is done with boolean `_bShowLines` in `cPolygon`, which can be checked with method `cPolygon::isLinear()`.

### 32.4.6 SaisieBascQT

Mode=0 in `src/uti_phgrm/CPP_SaisieQT.cpp`

At this point, `SaisieBascQT` has exactly the same behavior as `SaisieBasc`: lines are drawn as two points, while it may be useful to display the complete line.

### 32.4.7 SaisieCylQT

Mode=1 in `src/uti_phgrm/CPP_SaisieQT.cpp`

### 32.4.8 SaisieBoxQT

`SaisieBoxQT` is a very simple use of `SaisieQtWindow`: it shows an image, allow to draw a `cRectangle`, which is a `cPolygon` with 4 points defined in `cObject.h`. At this point, `SaisieBoxQT` is only meant to communicate with a visual interface (such as `vMalt`) and we only send a *signal* with `void newRectanglePosition(QVector<QPointF> points)` in `GLWidget::mouseMoveEvent`. This *signal* is connected to `onRectanglePositionChanged` in `visual_MainWindow::onSaisieButtonPressed` in `visual_mainWindow.cpp`. `SaisieBoxQT` is instantiated in `visual_mainWindow` constructor (`visual_mainWindow.cpp`).

## 32.5 Conventions for 3D selection tool

**SaisieMasqQT** allows to open ply files and to do some manual segmentation with a polygonal selection tool.  
User can mainly perform 2 actions:

- move camera around point cloud (rotate and/or translate)
- draw a polygon and select/deselect points inside polygon

**SaisieMasqQT** can store an xml file (selectionInfo.xml) with polygonal selection information (camera position, and viewport coordinates of polygon vertex).

For each pair of camera pose and polygonal selection, xml file contains a tag `<Item>` with:

- camera pose matrix
- openGL viewport size (4 parameters)
- a list of polygon vertex viewport coordinates
- selection mode (add, remove, invert, etc.)

Two camera pose matrix are stored using openGL conventions:

- model-view matrix (16 parameters)
- projection matrix (16 parameters)

For more information on these matrix: <http://www.glprogramming.com/red/chapter03.html>

How to transform polygon viewport coordinates (as stored) into world coordinates:

- In : point in viewport coordinates  $P(x,y)$
- Out : point in world coordinates  $P'(x',y',z')$

First, we map x and y from viewport coordinates to [-1,1]

$$\begin{aligned}x &= 2 * (x - \text{viewport}[0]) / \text{viewport}[2] - 1 \\y &= 2 * (y - \text{viewport}[1]) / \text{viewport}[3] - 1\end{aligned}$$

We compute global projection matrix from camera to world coordinates:

$$M = \text{ModelViewMatrix} * \text{ProjectionMatrix}$$

Let's define a point  $P_h$  in homogeneous coordinates in the image plane:

$$\begin{aligned}P_h[0] &= x \\P_h[1] &= y \\P_h[2] &= 0 \\P_h[3] &= 1\end{aligned}$$

Projection point  $P_r$  will be:

$$P_r = P_h * M$$

And final point in world coordinates is:

$$\begin{aligned}x' &= P_r[0] / P_r[3] \\y' &= P_r[1] / P_r[3] \\z' &= P_r[2] / P_r[3]\end{aligned}$$

These operations are performed in function `getInverseProjection` from `saisieQT/MatrixManager.cpp`  
a function to convert a polygon stored in `selectInfo.xml` (filename) into point cloud local frame looks like:

```
#include MatrixManager.h

HistoryManager *HM = new HistoryManager();
MatrixManager *MM = new MatrixManager();
```

```
HM->load(filename);
QVector <selectInfos> vInfos = HM->getSelectInfos();

for (int aK=0; aK< vInfos.size();++aK)
{
    selectInfos &Infos = vInfos[aK];
    MM->importMatrices(Infos);

    for (int bK=0;bK < Infos.poly.size();++bK)
    {
        QPointF pt = Infos.poly[bK];
        Pt3dr pt3d;
        MM->getInverseProjection(pt3d, pt, 0.f);

        std::cout << pt3d.x << " " << pt3d.y << " " << pt3d.z << std::endl;
    }
}
```



## Chapter 33

# Génération automatique de code



# **Part VI**

# **Annexes**



# Appendix A

## Formats

### A.1 Calibration formats

### A.2 Grilles de calibration

Cette section décrit le format grille utilisé par MicMac pour coder la calibration interne des caméras de manière générique (c.a.d. indépendamment du modèle paramétrique choisi).

#### A.2.1 Format de codage des déformations du plan

##### A.2.1.1 Format

A bas niveau, le format de grille est essentiellement un format **Xml** simple adapté au codage des déformations régulières du plan; c'est à dire des objets qui correspondent au concept mathématique de bijection  $\mathcal{C}^\infty$  d'une partie de  $\mathbb{R}^2$  dans une autre partie. Soit  $\psi$  une telle bijection, le format est spécifié de la manière suivante :

- le tag père porte le nom de `<doublegrid>`;
- ce tag comporte deux fils qui ont rigoureusement la même structure, le fils `<grid_directe>` correspond à une tabulation de valeurs de  $\psi$  et le fils `<grid_inverse>` correspond à une tabulation de de valeurs de  $\psi^{-1}$  ;

Chaque valeur `<grid_directe>` et `<grid_inverse>` correspond donc à la tabulation des valeurs d'une déformation d'un domaine de  $\mathbb{R}^2$ . Considérons par exemple la fonction directe  $\psi$  et notons  $\psi(P) = (\psi_x(P), \psi_y(P))$  ses deux composantes; il faut définir le domaine sur lequel  $\psi$  est tabulé, la résolution choisie, et les valeur  $\psi_x$  et  $\psi_y$ ; la tabulation est alors définie par :

- un origine  $\mathcal{O}$  (tags `<origine><x>` et `<origine><y>`);
- un pas  $\Delta$  ( tag `<step>`);
- deux tableaux de valeurs  $Data_x$  et  $Data_y$  définis par deux noms de fichier (tags `<filename><x>` et `<filename><y>`) et une taille ( $T_x, T_y$ ) (tags `<size><x>` et `<size><y>`); chaque fichier est une suite de  $T_x * T_y$  nombres réels codés sur des doubles (format "raw");

Les valeurs des tableaux  $Data_x$  et  $Data_y$ , contenus dans les fichiers `<filename>`, sont décrites par les équations A.1 et A.2.

$$Data_x[I + T_x J] = \psi_x(\mathcal{O}_x + \Delta I, \mathcal{O}_y + \Delta J) \quad (\text{A.1})$$

$$Data_y[I + T_x J] = \psi_y(\mathcal{O}_x + \Delta I, \mathcal{O}_y + \Delta J) \quad (\text{A.2})$$

##### A.2.1.2 Utilisation

En soi, le format ne spécifie rien de plus que des conventions pour tabuler des valeurs de  $\psi$  sur une portion rectangulaire d'une grille régulière. Une utilisation courante possible, pour estimer  $\psi(P)$  en un point quelconque, pourrait consister à effectuer les opérations suivantes :

- calculer les "indices réels"  $i = \frac{P_x - \mathcal{O}_x}{\Delta}$  et  $j = \frac{P_y - \mathcal{O}_y}{\Delta}$ ;
- considérer  $Data_x$  et  $Data_y$  comme des images et utiliser un schéma d'interpolation (par exemple bilinéaire) pour calculer leurs valeurs au point réel  $(i, j)$ .

## A.2.2 Application à la calibration interne

### A.2.2.1 Rappels et notation

Cette section donne un rappel des principaux modèles de distorsion paramétrique couramment utilisés lors du calcul de la calibration interne. La lecture de cette section n'est pas strictement nécessaire à la compréhension du format.

Soit une caméra  $C$ , on considère un point  $P = {}^t(x, y, z)$  du monde objet (espace  $\mathbb{R}^3$ ) et son homologue  $Q_c = {}^t(u, v)$  dans l'image (espace  $\mathbb{R}^2$ ); on note  $\pi_c$  la fonction qui associe  $Q$  à  $P$  :

$$Q = \pi_c(P) \quad (\text{A.3})$$

On se limite aux caméras sténopé; on note

- $\pi_0({}^t(x, y, z)) = \frac{{}^t(x, y)}{z}$ , projection "canonique";
  - la position d'une caméra dans l'espace objet est définie par ses paramètres "extrinsèques", c'est à dire centre optique  $C$  et matrice de rotation  $\mathcal{R}$
  - la caméra elle-même est définie par ses paramètres intrinsèques distance focale  $F$  et point principal  $P^p$ ;
- Pour une caméra "idéale", on a la relation :

$$\pi(P) = P^p + F * \pi_0(\mathcal{R}(P - C)) \quad (\text{A.4})$$

En réalité différents phénomènes sont susceptible de modifier la relation A.4. Le phénomène prépondérant est en général une distorsion radiale<sup>1</sup>. Toute fonction radiale est caractérisée par son centre de symétrie  $C_r$  et une fonction de distorsion  $\phi$  (nécessairement paire si la fonction résultante est  $C^\infty$ ) qui ne dépend que du rayon. Généralement  $\phi$  est modélisée par un polynôme :

$$\phi(R) = 1 + a_2 R^2 + a_4 R^4 + \dots \quad (\text{A.5})$$

$$P - C_r = {}^t(x_c, y_c) \quad R = \sqrt{x_c^2 + y_c^2} \quad (\text{A.6})$$

$$D_{\phi, C_r}(P) = C_r + (P - C_r) * \phi(R) \quad (\text{A.7})$$

Dans ce modèle on a alors :

$$\pi(P) = D_{\phi, C_r}(P^p + F * \pi_0(\mathcal{R}(P - C))) \quad (\text{A.8})$$

Lorsque l'on suppose que les axes optique des différentes lentilles ne sont pas rigoureusement confondus, un développement limité conduit à introduire une distorsion de décentrement définie par deux paramètres  $\alpha$  et  $\beta$  :

$$D_{\phi, C_r}^{\alpha, \beta}(P) = D_{\phi, C_r}(P) + {}^t(\alpha(2x_c^2 + R^2) + 2\beta x_c y_c, \beta(2y_c^2 + R^2) + 2\alpha x_c y_c) \quad (\text{A.9})$$

Si l'on suppose que le plan du capteur n'est pas rigoureusement orthogonal à l'axe optique, on rajoute un terme affine (voir A.2.3.1 pourquoi il n'y a que deux termes supplémentaires) :

$$D_{\phi, C_r, A, B}^{\alpha, \beta}(P) = D_{\phi, C_r}^{\alpha, \beta}(P) + {}^t(Ax_c + By_c, 0) \quad (\text{A.10})$$

L'équation A.10 correspond au modèle dit *photogramétrique standard*. Ce modèle couvre la plupart des cas courants; cependant d'autres phénomènes non pris en compte par le modèle photogramétrique standard sont susceptibles d'intervenir : non planéité du capteur, déformations planimétriques, défauts haute fréquence dans les filtres placés en amont du système optique ...

Pour les caméras développées à l'IGN, le modèle radial (A.7) pourrait couvrir un peu plus de la majorité des cas testés et le modèle photogramétrique standard (A.10) donne une description satisfaisante pour plus de 80% des caméras. Les autres cas (généralement pour les courtes focales) nécessitent une modélisation plus complète dans laquelle la fonction de distorsion est *a priori* une fonction quelconque (mais "très" régulière) de  $\mathbb{R}^2$  dans  $\mathbb{R}^2$ . On aboutit dans ce cas à la modélisation suivante :

$$\mathcal{D} : \mathbb{R}^2 \rightarrow \mathbb{R}^2 \quad C^\infty \quad (\text{A.11})$$

$$\pi(P) = \mathcal{D}(\pi_0(\mathcal{R}(P - C))) \quad (\text{A.12})$$

Dans l'équation A.12 les paramètres  $F$  et  $P^p$  ont disparu car il seraient redondants avec une application  $D$  qui peut *a priori* être quelconque.

---

1. parce que le système optique est en première approximation à symétrie de révolution

### A.2.2.2 Codage des distorsion par des grilles

Compte tenu de la relative complexité du modèle A.10, et du fait que ce modèle n'est pas toujours suffisant, le format d'export retenu pour les calibrations est un format grille. On part de l'équation A.12, qui correspond au cas le plus général (les équations A.7, A.9 et A.10 en sont évidemment un cas particulier), et on utilise le format grille pour coder  $\psi = \mathcal{D}^{-1}$  considéré comme une application du plan dans lui même.

Plus précisément cela signifie que, une fois la grille lue, on peut par exemple utiliser le mécanisme décrit en A.2.1.2 pour :

- à partir d'un point image  $Q$  calculer la direction du rayon incident dans le repère caméra par  ${}^t(\psi_x(Q), \psi_y(Q), 1)$
- à partir d'un point terrain  $P$ , et des paramètres extrinsèques  $R$  et  $C$ , calculer la projection image par  $Q = \psi^{-1}(\pi_0(\mathcal{R}(P - C)))$

En pratique ce format a été utilisé à l'IGN pour exporter des calibrations faites selon les méthodes suivantes :

- modèle radiale sur polygone de calibration;
- modèle photogramétrique standard sur polygone de calibration;
- modèle grille calculé par auto-calibration ; dans cette méthode on n'utilise que des points homologues calculé de manière dense et c'est directement une grille qui est calculée;
- combinaison de méthodes précédentes.

Il s'agit donc d'un format d'export qui a été choisi pour sa capacité à représenter toutes les distorsions correspondant à des caméras sténopés sans préjuger de la méthode de calcul ou de l'éventuel modèle paramétrique sous-jacent.

### A.2.2.3 Justification des modèles paramétriques

Cette section a essentiellement pour but de mettre au propre mes notes sur la justification du modèle décentrique.

On voit facilement qu'une déformation  $D \in \mathcal{C}^\infty$  à symétrie radiale de centre  ${}^t(a, b)$  s'écrit nécessairement sous la forme :

$$D_x(X, Y) = a + (X - C_x)f(\sqrt{(X - C_x)^2 + (Y - C_y)^2}) \quad (\text{A.13})$$

$$D_y(X, Y) = b + (Y - C_y)f(\sqrt{(X - C_x)^2 + (Y - C_y)^2}) \quad (\text{A.14})$$

Où  $f$  est une fonction paire et  $\mathcal{C}^\infty$ . Classiquement, on suppose alors que  $f$  est approximable<sup>2</sup> par un polynôme en  $\rho^2$  :

$$f(\rho) = \sum_{k=0}^N \alpha_k \rho^{2k} \quad (\text{A.15})$$

Le terme  $\alpha_0$  serait redondant avec les focales, on pose donc  $\alpha_0 = 1$ , ce qui donne :

$$X_C = X - C_x \quad Y_C = Y - C_y \quad (\text{A.16})$$

$$D_x(X, Y) = X + X_C \sum_{k=1}^N \alpha_k (X_C^2 + Y_C^2)^k \quad (\text{A.17})$$

$$D_y(X, Y) = Y + Y_C \sum_{k=1}^N \alpha_k (X_C^2 + Y_C^2)^k \quad (\text{A.18})$$

Le choix d'un modèle radiale de distorsion, vient du fait que, si les lentilles sont des objets de révolution, et que leur axes optiques sont confondus, alors le système optique est lui-même à symétrie de révolution. Lorsque l'on remet en cause l'hypothèse selon laquelle les axes optiques sont confondus, on fait l'hypothèse que la distorsion résultante est une somme de distorsions radiales de centre "très légèrement" différent.

Soient  $C$  et  $C'$  les deux centres optiques quasi confondus, on écrit  $C' = C - {}^t(a, b)$  ou  $X_{C'} = X_C + a, Y_{C'} = Y_C + b$  avec  $a \approx 0$  et  $b \approx 0$ .

On peut écrire :

$$D'_x(X, Y) = X + X_{C'} \sum_{k=1}^N \alpha'_k (X_{C'}^2 + Y_{C'}^2)^k \quad (\text{A.19})$$

$$D'_x(X, Y) \approx X + X_C \sum_{k=1}^N \alpha'_k (X_C^2 + Y_C^2)^k + a \frac{\partial(X_{C'} \sum_{k=1}^N \alpha'_k \rho_{C'}^{2k})}{\partial a} + b \frac{\partial(X_{C'} \sum_{k=1}^N \alpha'_k \rho_{C'}^{2k})}{\partial b} \quad (\text{A.20})$$

---

2. cf théorème de Stone-Weierstrass

On voit que le premier terme est un terme radial de centre  $C$  qui va, additionné à la distorsion de centre  $C$ , va donner des termes  $\alpha_k + \alpha'_k$ . Il nous reste à évaluer le terme proprement "décentrique" :

$$\frac{\partial(X_{C'} \sum_{k=1}^N \alpha'_k \rho_{C'}^{2k})}{\partial a} = \sum_{k=1}^N \alpha'_k \rho_C^{2k-2} (\rho_C^2 + 2kX_C^2) \quad (\text{A.21})$$

$$\frac{\partial(X_{C'} \sum_{k=1}^N \alpha'_k \rho_{C'}^{2k})}{\partial b} = \sum_{k=1}^N \alpha'_k 2k \rho_C^{2k-2} X_C Y_C \quad (\text{A.22})$$

Soit une distorsion de décentrement  $D^{C/C'}$  :

$$D_x^{C/C'} = \sum_{k=1}^N \rho_C^{2k-2} \alpha'_k (a\rho_C^2 + 2kaX_C^2 + 2kbX_C Y_C) \quad (\text{A.23})$$

$$D_y^{C/C'} = \sum_{k=1}^N \rho_C^{2k-2} \alpha'_k (b\rho_C^2 + 2kbX_C^2 + 2kaX_C Y_C) \quad (\text{A.24})$$

Si on ne s'intéresse qu'au premier terme, on tombe sur la formule habituelle :

$$D_{x_1}^{C/C'} = \alpha'_k (a\rho_C^2 + 2aX_C^2 + 2bX_C Y_C) = A_1(\rho_C^2 + 2X_C^2) + 2B_1 X_C Y_C \quad (\text{A.25})$$

$$D_{y_1}^{C/C'} = \alpha'_k (b\rho_C^2 + 2bY_C^2 + 2aX_C Y_C) = B_1(\rho_C^2 + 2Y_C^2) + 2A_1 X_C Y_C \quad (\text{A.26})$$

$$A_1 = \alpha'_k a, \quad B_1 = \alpha'_k b \quad (\text{A.27})$$

La plupart des auteurs, s'arrêtent au premier terme. Rien n'empêche *a priori* de rajouter, par exemple, le terme suivant :

$$D_{x_2}^{C/C'} = A_2(\rho_C^4 + 4X_C^2 \rho_C^2) + 4B_2 X_C Y_C \rho_C^2 \quad (\text{A.28})$$

$$D_{y_2}^{C/C'} = B_2(\rho_C^4 + 4Y_C^2 \rho_C^2) + 4A_2 X_C Y_C \rho_C^2 \quad (\text{A.29})$$

$$A_2 = \alpha'_k a, \quad B_2 = \alpha'_k b \quad (\text{A.30})$$

### A.2.3 Application à une rotation près

Cette section contient des développements qui sont plutôt de l'ordre de la photogrammétrie générale. Son contenu n'est donc pas directement lié au format de grille; mais ces remarques me semblent utiles, et il se trouve qu'elles sont d'autant plus importantes que le modèle choisi pour *calculer* la calibration est peu contraint.

#### A.2.3.1 Formalisation

Soit  $\mathcal{R}$  la rotation associé à une caméra, on note ses coefficients ainsi :

$$\mathcal{R} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \quad (\text{A.31})$$

On remarque que pour  $P = {}^t(xyz)$ :

$$\pi_0(\mathcal{R}P) = \begin{pmatrix} \frac{ax+by+cz}{gx+hy+iz} \\ \frac{dx+ey+fz}{gx+hy+iz} \\ \frac{gz+hz+iz}{gx+hy+iz} \end{pmatrix} \quad (\text{A.32})$$

$$\pi_0(\mathcal{R}P) = \begin{pmatrix} \frac{a\frac{x}{z}+b\frac{y}{z}+cz}{g\frac{x}{z}+h\frac{y}{z}+i} \\ \frac{d\frac{x}{z}+e\frac{y}{z}+fz}{g\frac{x}{z}+h\frac{y}{z}+i} \\ \frac{gz+hz+iz}{gx+hy+iz} \end{pmatrix} \quad (\text{A.33})$$

Notons  $\mathcal{H}_{\mathcal{R}}$  l'homographie  $\mathbb{R}^2 \rightarrow \mathbb{R}^2$  définie par :

$$\mathcal{H}_{\mathcal{R}} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{au+bv+c}{gu+hv+i} \\ \frac{du+ev+f}{gu+hv+i} \end{pmatrix} \quad (\text{A.34})$$

L'équation A.33 peut alors se réécrire :

$$\pi_0 \mathcal{R} = \mathcal{H}_{\mathcal{R}} \pi_0 \quad (\text{A.35})$$

Considérons maintenant  $N$  caméras  $C_k$ ,  $k \in [1 N]$ , et leur équation de projection :

$$\pi_k(P) = \mathcal{D}(\pi_0(\mathcal{R}_k(P - \mathcal{C}_k))) \quad (\text{A.36})$$

On voit que pour toute rotation  $\mathcal{S}$  on peut écrire :

$$\pi_k(P) = \mathcal{D}(\pi_0(\mathcal{S}\mathcal{S}^{-1}\mathcal{R}_k(P - \mathcal{C}_k))) \quad (\text{A.37})$$

Soit encore :

$$\pi_k(P) = (\mathcal{D}\mathcal{H}_{\mathcal{S}})\pi_0((\mathcal{S}^{-1}\mathcal{R}_k)(P - \mathcal{C}_k)) \quad (\text{A.38})$$

On remarque alors que les configuration correspondant aux équations A.36 et A.38 sont complètement indiscernables. Concrètement cela signifie que la calibration d'une caméra n'est jamais définie qu'à une rotation près puisque une rotation quelconque  $\mathcal{S}^{-1}$  de l'ensemble des positions de la caméra peut être exactement absorbée par une modification de la distorsion en remplaçant  $\mathcal{D}$  par  $\mathcal{D}\mathcal{H}_{\mathcal{S}}$

### A.2.3.2 Conséquence pour la comparaison de calibration

Il est assez clair que pour mesurer la proximité entre deux calibrations différentes d'une même caméra, la mesure d'écart entre les valeur du modèle paramétrique est une très mauvaise méthode. Les raisons les plus évidentes sont :

- les différent paramètres ne sont pas tous homogènes;
- dans certaines configurations, un paramètre peuvent varier librement sans que cela ait d'influence sur la calibration; un cas classique est celui du modèle radiale lorsque la distorsion est quasi nulle : le centre de distorsion peut se "ballader" sans que cela ait d'influence sur la qualité du résultat final;
- un phénomène similaire, mais plus complexe à détecter est celui des paramètres fortement corrélés pour lesquels les variations de l'un peuvent être compensés par des variation de l'autre sans affecter le résultat de la calibration (c'est le cas du modèle photogramétrique standard pour le point principal et le centre de distorsion);
- enfin , de manière évidente, cette méthode n'est pas utilisable pour comparer deux calibration issus de modèle paramétriques différents.

Une moins mauvaise méthode pourrait être de comparer directement les les direction des rayons perspectifs dans le repère caméra. Repartant de l'équation A.12, cela revient à se donner une distance sur les déformation de  $\mathbb{R}^2$ . Par exemple, étant donnée  $\mathcal{D}_1$  et  $\mathcal{D}_2$  deux calibration, on pourrait définir en choisissant une norme  $\mathcal{L}^2$  :

$$d_2(\mathcal{D}_1, \mathcal{D}_2) = \iint_{\mathcal{I}} (\mathcal{D}_1(u, v) - \mathcal{D}_2(u, v))^2 du dv \quad (\text{A.39})$$

Cette distance n'est pas bonne non plus ; en effet soit  $\mathcal{R}$  une rotation, la valeur  $d_2(\mathcal{D}, \mathcal{D}\mathcal{H}_{\mathcal{R}})$  peut être grande alors que nous avons qu'il s'agit en fait exactement de la même calibration.

Sans rentrer dans les détails mathématiques, lorsque l'on manipule des objets à une opération près, la bonne distance est la distance quotient<sup>3</sup>. Concrètement, soit  $\mathcal{S}^3$  le groupe des rotations de  $\mathbb{R}^3$ , on utilisera la distance  $d_{/\mathcal{S}^3}^2$  définie par :

$$d_{/\mathcal{S}^3}^2(\mathcal{D}_1, \mathcal{D}_2) = \min_{(\mathcal{S} \in \mathcal{S}^3)} d_2(\mathcal{D}_1, \mathcal{D}_2 \mathcal{H}_{\mathcal{S}}) \quad (\text{A.40})$$

Dans le cas d'une norme  $\mathcal{L}^2$  le calcul de la distance  $d_{/\mathcal{S}^3}^2$  est relativement aisé ; en effet, en prenant quelque précautions, on assure facilement que le minimum de  $d_2(\mathcal{D}_1, \mathcal{D}_2 \mathcal{H}_{\mathcal{S}})$  est réalisé pour un valeur proche de l'identité. En linéarisant le problème, on trouve alors le minimum en une ou deux itérations.

### A.2.4 Point principal

#### A.2.4.1 Le point principal bouge

La nécessité d'utiliser  $d_{/\mathcal{S}^3}^2$  au lieu de  $d_2$  étant d'autant plus grande que le modèle est peu contraint, on pourrait espérer qu'avec les modèles les plus simples on puisse faire l'économie de cette complexité. En fait ce n'est pas le cas notamment pour les grandes focales; considérons par exemple, une "petite" rotation  $\mathcal{R}_{\theta}^x$  d'angle  $\theta$  autour de l'axe  $Ox$ , un développement limité montre que  $\mathcal{H}_{\mathcal{R}_{\theta}^x}$  est égale à une translation sur les  $v$ , donc à une modification du point principal(avec une erreur en  $\mathcal{O}(\theta uv)$ ). Donc, même avec le modèle le plus simple de la caméra idéale sans distorsion (voir A.4) deux calibrations peuvent être très proches en réalité (ie selon  $d_{/\mathcal{S}^3}^2$ ) et assez éloignée avec une mesure de  $d_2$ .

---

3. on ne rentre pas ici dans les conditions pour que cette distance soit correctement définie

### A.2.4.2 On a perdu le point principal

Avec les formulation habituelles de la calibration utilisée dans les équations A.10, A.9, A.4 ou A.7, le point principal semble être un objet mathématiquement défini puisqu'il est intervenir de manière explicite dans le modèle paramétrique.

Pour définir un équivalent du point principal avec l'équation A.12, on pourrait par analogie avec les autre formule, poser :

$$P^p = \mathcal{D}^{-1}({}^t(0, 0)) \quad (\text{A.41})$$

Cette méthode serait erronée car la calibration n'étant définie qu'à une rotation près, cette définition peut mettre le point principal n'importe où dans l'image suivant la rotation choisie.

On voit donc que la notion de point principal, est :

- une notion peu stable pour les longues focales;
- quelque chose d'indéfini pour les modèle de calibration quelconques.

On peut d'ailleurs pousser le raisonnement un peu plus loin et voir que la focale non plus n'est pas une notion complètement définie. Par analogie avec les modèle simple, tel que A.4, où la focale intervient explicitement, on peut la définir comme un rapport d'homométrie entre les coordonnées image et les coordonnées  $\frac{(x, y)}{z}$ ,

### A.2.4.3 On a retrouvé le point principal ?

La méthodes de calibration mise en place sur les caméras numériques de l'IGN pour modéliser des distorsions quelconques est la suivante :

1. calcul d'un modèle paramétrique approché (par auto-calibration);
2. calcul d'un modèle quelconque considéré comme une petite perturbation corrective du modèle paramétrique;
3. calcul d'un modèle paramétrique sur le polygone, à partir de mesure éventuellement corrigée du modèle précédent;

Lorsque l'on attend des distorsions faibles (typiquement avec les focales  $\geq 50\text{mm}$ ), on passe directement à l'étape de calibration sur polygone. Que l'on commence à l'étape 1, ou directement à l'étape 3,

## A.2.5 Paramètres hors grilles

### A.2.5.1 Paramètres photogramétrique "traditionnel"

### A.2.5.2 Autres paramètres

## A.3 Points Homologues

## A.4 Fichiers MNT

# Appendix B

## Vrac

Ce chapitre comporte un certain nombre de notes, qui n'ont pas forcément grand chose à voir avec MicMac, mais que je met ici en attendant de savoir quoi en faire (bref, c'est du Vrac ...).

### B.1 Notation

On caractérise une pose  $P$  par un couple  $C, R$  où  $C$  est le centre perspectif et  $R$  la rotation. Soit  $p_c$  un point en coordonnées caméra et  $p_m$  sont homologues en coordonnées monde, on a :

$$p_m = C + Rp_c \quad (\text{B.1})$$

Le couple  $C, R$  caractérise une rotation affine; la composition des application donne une structure de groupe naturelle

$$(C_1, R_1) * (C_2, R_2) = (C_1 + R_1 C_2, R_1 R_2) \quad (\text{B.2})$$

Soit  $P_1$  et  $P_2$  deux poses de caméra; on s'intéresse à l'orientation relative; on va calculer les poses  $P_1^1$  et  $P_2^1$  dans le repère lié à la caméra 1. On peut à naturellement  $P_1^1 = Id$ .

$$C_1^1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} R_1^1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (\text{B.3})$$

L'orientation relative étant définie à un facteur d'échelle près on peut poser arbitrairement

$$\|\overrightarrow{C_1^1 C_2^1}\| = \|C_2^1\| = 1 \quad (\text{B.4})$$

Si l'on connaît par ailleurs la pose absolue  $P_1$  et la pose relative  $P_2^1$ , alors la pose absolue de la caméra 2 est :

$$(C_1 + \lambda R_1 C_2^1, R_1 R_2^1) \quad (\text{B.5})$$

Le terme  $\lambda$  représente le facteur d'échelle, pour le reste il suffit de remarquer que  $p_m = P_1 P_1^2 p_2$ .

### B.2 Géométrie épipolaire

On voit facilement qu'il existe une infinité de rotation dont l'image du premier vecteur directeur, ie  ${}^t(1 \ 0 \ 0)$ , est l'axe  $\overrightarrow{C_1 C_2}$ ; elles se déduisent d'ailleurs les unes des autres par une rotation autour de l'axe  $C_1 C_2$ . Soit  $R^e$  une de ces rotations, on appelle poses épipolaires les poses de la forme :  $(C_1 R^e)$  et  $(C_2 R^e)$ .

L'intérêt de ces poses épipolaires vient des remarques suivantes :

L'homologue d'une direction  ${}^t(x_1^e \ y_1^e \ 1)$  est un direction  ${}^t(x_2^e \ y_2^e \ 1)$  avec  $y_1^e = y_2^e$ ; la contrainte d'homologie entre points images y est donc particulièrement facile à exprimer;

$$y_1^e = y_2^e \quad (\text{B.6})$$

Le passage de  $C_1 R_1$  à  $C_1 R^e$  (vs  $C_2 R_2$  à  $C_2 R^e$ ) est purement vectoriel puisque les centres sont confondus

$$p_1^e = (R^{e-1} R_1) p_1 \quad (\text{B.7})$$

On pose :

$$R_1^e = (R^{e-1} R_1) \quad (\text{B.8})$$

On a :

$$(x_1^e \quad y_1^e \quad 1) = R_1^e (x_1 \quad y_1 \quad 1) \quad (\text{B.9})$$

### B.3 Matrice essentielle

Avec des poses epipolaire on a toujours la relation entre directions homologues :

$$(x_1^e \quad y_1^e \quad 1) \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_2^e \\ y_2^e \\ 1 \end{pmatrix} = y_2^e - y_1^e = 0 \quad (\text{B.10})$$

Notons :

$$\mathcal{E}_0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \quad (\text{B.11})$$

En utilisant la relation B.9, on a :

$${}^t p_1 {}^t R_1^e \mathcal{E}_0 R_2^e p_2 = 0 \quad (\text{B.12})$$

On voit donc qu'il existe une matrice,  $\mathcal{E}_{1,2}$ , appelée matrice essentielle telle que pour tout couple de direction homologue  $p_1, p_2$  on ait :

$${}^t p_1 \mathcal{E}_{1,2} p_2 = 0 \quad (\text{B.13})$$

### B.4 Calcul de l'orientation relative par matrice essentielle

### B.5 Cas planaire

### B.6 Grandes focales, projection axo et points triples

Il existe une rotation :

### B.7 Géométrie épipolaire

### B.8 Matrice essentielle

### B.9 Grandes focales, projection axo et points triples

# Appendix C

## Vrac -Bis

Suite du Vrac. Regroupe plutôt des éléments un peu éloigné du traitement d'image et de la photogrammétrie.

### C.1 Introduction-Supression des inconnues auxiliaires

On décrit ici une technique utilisée pour introduire les points terrains dans les équation photogramétriques. L'intérêt de cette technique est d'avoir des formules "naturelles" faisant intervenir la projection des points terrains dans les images sans alourdir le calcul; en effet , une fois connues toutes les équations impliquant le point, on le fait disparaître par un jeu de substitution (on n'augmente donc pas la taille du système quadratique final).

On considère une série de  $K$  observations  $\mathcal{O}_k$  faisant intervenir deux jeux d'inconnues  $X_i, i \in [1, N[$  et  $Y_j, j \in [0, M[$ :

$$\mathcal{O}_k = \sum_{i=0}^N a_i^k X_i + \sum_{j=0}^M b_j^k Y_j + c_k = 0 \quad (\text{C.1})$$

Ces observations font partie d'un plus grand système à résoudre par moindres carrés.

Typiquement dans notre cas :

- les  $X_i$  seront les trois inconnues de coordonnées d'un point terrain,
- les  $Y_j$  seront les inconnues de poses (orientation et centre optiques) et de calibration (ie paramètres extrinsèques et intrinsèques) des images dans lesquelles se projette ce point;
- les  $K$  observations correspondent à la projection du point terrain à un position  $x_k, y_k$  données dans  $\frac{K}{2}$  images (aprés linéarisation);

On note  $X$  le vecteur colonne de coordonnées  $X_k$ : On suppose que les  $K$  observations sont les seules faisant intervenir les  $X_i$ ; la partie de la fonctionnelle impliquant les  $X_i$  est :

$$\mathcal{L}(X) = \sum_{k=0}^K \mathcal{O}_k^2 \quad (\text{C.2})$$

On développe :

$$\mathcal{L}(X) = \sum_{k \in [1, K[} \left( \sum_{i=0}^N a_i^k X_i + \sum_{j=0}^M b_j^k Y_j + c_k \right)^2 \quad (\text{C.3})$$

$$\mathcal{L}(X) = \sum_{k=0}^K \left\{ \left( \sum_{i=0}^N a_i^k X_i \right)^2 + 2 \left( \sum_{i=0}^N a_i^k X_i \right) \left( \sum_{j=0}^M b_j^k Y_j + c_k \right) + \left( \sum_{j=0}^M b_j^k Y_j + c_k \right)^2 \right\} \quad (\text{C.4})$$

Le troisième terme ne fait pas intervenir les  $X_k$ , il est traité comme une contribution "normale" qui va se rajouter à l'accumulateur global; on n'en tient plus compte maintenant (on note  $\mathcal{L}'(X)$  le terme ainsi simplifié)

On note  $\Lambda = (\lambda_{i_1, i_2})$  la matrice  $N * N$  définie par :

$$\lambda_{i_1, i_2} = \sum_{k=0}^K a_{i_1}^k a_{i_2}^k \quad (\text{C.5})$$

On note  $\Gamma = \gamma_i$  le vecteur colonne défini par :

$$\gamma_i = \sum_{k=0}^M a_i^k \left( \sum_{j=N}^M b_j^k Y_j + c_k \right) \quad (\text{C.6})$$

On peut alors écrire :

$$\mathcal{L}'(X) = {}^t X \Lambda X + 2 {}^t \Gamma X \quad (\text{C.7})$$

Raisonnement "avec les mains": pour minimiser la fonctionnelle globale on a tout intérêt à ce que  $X$  prenne la valeur minimum en fonction de  $Y$ .

Classiquement, s'agissant d'une fonctionnelle quadratique, le minimum de  $\mathcal{L}'$  est atteint en  $X = -\Lambda^{-1}\Gamma$  et la valeur de ce minimum est :

$$\mathcal{L}'^0 = -{}^t \Gamma \Lambda^{-1} \Gamma \quad (\text{C.8})$$

Il suffit donc de rajouter le terme  $\mathcal{L}'^0$  qui est un terme quadratique en  $Y$ . Explicitons les valeur de ce terme, on pose :

$$A_i = \sum_{k=0} a_i^k c_k, B_{i,j} = \sum_{k=0} a_i^k b_j^k \quad (\text{C.9})$$

Et on a :

$$\Gamma = \begin{pmatrix} \gamma_0 \\ \dots \\ \gamma_{N-1} \end{pmatrix} = \begin{pmatrix} A_0 \\ \dots \\ A_{N-1} \end{pmatrix} + \begin{pmatrix} B_{0,0} & B_{0,1} & \dots & B_{0,M-1} \\ \dots & \dots & \dots & \dots \\ B_{N-1,0} & B_{N-1,1} & \dots & B_{N-1,M-1} \end{pmatrix} \begin{pmatrix} Y_0 \\ Y_1 \\ \dots \\ Y_{M-1} \end{pmatrix} \quad (\text{C.10})$$

Soit, en notant  $A$  le vecteur colonne de coordonnées  $A_i$  et  $B$  la matrice dont les éléments sont  $B_{i,j}$  :

$$\Gamma = A + BY \quad (\text{C.11})$$

Et :

$$\mathcal{L}'^0 = -{}^t Y ({}^t B \Lambda^{-1} B) Y - 2 ({}^t A \Lambda^{-1} B) Y - {}^t A \Lambda^{-1} A \quad (\text{C.12})$$

Il n'y a plus qu'à rajouter ce terme quadratique à l'accumulateur.

## C.2 Formulation algébrique du traitement des inconnues auxiliaires

On donne une formulation purement algébrique de la technique précédente. Cette formulation bien que moins intuitive est sans doute plus facile d'emploi.

On a un système linéaire à résoudre dans lesquelle on classe les inconnues en trois catégories :

- $X = (X_i)$  les inconnues à éliminer;
- $Y = (X_j)$  les inconnues interférant avec les  $X_i$ ;
- $Z = (Z_k)$  les inconnues n'interférant pas avec les  $X_i$ ;

Avec la notation matrice bloc on écrit :

$$\begin{pmatrix} \Lambda & B & 0 \\ B' & M_{1,1} & M_{2,1} \\ 0 & M_{1,2} & M_{2,2} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} A \\ C_1 \\ C_2 \end{pmatrix} \quad (\text{C.13})$$

Ce que l'on peut écrire :

$$\begin{cases} \Lambda X + BY &= A \\ B'X + M_{1,1}Y + M_{2,1}Z &= C_1 \\ M_{1,2}Y + M_{2,2}Z &= C_2 \end{cases} \quad (\text{C.14})$$

On a :

$$X = \Lambda^{-1}(A - BY) \quad (\text{C.15})$$

On substitue dans la deuxième équation:

$$-B'\Lambda^{-1}BY + M_{1,1}Y + M_{2,1}Z = C_1 - B'\Lambda^{-1}A \quad (\text{C.16})$$

On a donc le système suivant d'où  $X$  est éliminé :

$$\begin{pmatrix} M_{1,1} - B'\Lambda^{-1}B & M_{2,1} \\ M_{1,2} & M_{2,2} \end{pmatrix} \begin{pmatrix} Y \\ Z \end{pmatrix} = \begin{pmatrix} C_1 - B'\Lambda^{-1}A \\ C_2 \end{pmatrix} \quad (\text{C.17})$$

### C.3 Prcision et corrlation sur les paramres

Code voir Prec-Cor-Param.txt



# Appendix D

## Vrac -Bis

Suite du Vrac. Regroupe plutôt des bribes de documentation interne eLiSe .

### D.1 Utilisation des foncteurs compilés

On retrace ici le (long ...) chemin qui conduit de l'utilisation finale d'un foncteur jusqu'à la modification d'un système (L2 ou L1). Il s'agit d'extrait de code avec plein de coupures.

#### D.1.1 Filière non indexee

```
void cEqObsRotVect::AddObservation (...)  
{  
    mN2.SetEtat(aDir2/euclid(aDir2)); ....  
    mSet.AddEqFonctToSys(mFoncEqResidu,aPds,WithD2);  
  
}  
  
REAL cSetEqFormelles::AddEqFonctToSys ( const tContFcteur & aCont ...)  
{  
    VAddEqFonctToSys(aFonct,aPds,WithDerSec);  
}  
  
const std::vector<REAL> & cSetEqFormelles::VAddEqFonctToSys (cElCompiledFonc * aFonct)  
{  
    aFonct->SetCoordCur(mAlloc.ValsVar()); // Met les val courrantes  
    aFonct->SetValDer(); // Fait le calcul  
    aFonct->AddDevLimOrd1ToSysSurRes(*mSys,aPds,true); // Transfert a la matrice  
}  
  
void cElCompiledFonc::AddDevLimOrd1ToSysSurRes  
    (cGenSysSurResol & aSys,REAL aPds,bool EnPtsCur)  
{  
    AddContrainteEqSSR(false,aPds,aSys,EnPtsCur);  
}  
  
// Peut etre appelee par AddDevLimOrd1ToSysSurRe, mais aussi pour mettre  
// des contraintes  
void cElCompiledFonc::AddContrainteEqSSR  
    (bool contr,REAL aPds,cGenSysSurResol & aSys,bool EnPtsCur)  
{  
    // met les valeurs des derivees dans mRealDer qui, apparemment  
    // est un tableau ayant la meme taille que le systeme global !  
    for ( aD ... )  
        aSys.GSSR_AddNewEquation(aPds,&(mRealDer[aD][0]),aB);  
}
```

```

void cGenSysSurResol::GSSR_AddNewEquation(REAL aPds,REAL * aL,REAL aB)
{
    V_GSSR_AddNewEquation(aPds,aL,aB);
}

// Ensuite il y a differente variante, en L2 ca donne
void L2SysSurResol::V_GSSR_AddNewEquation(REAL aPds,REAL * aCoeff,REAL aB)
{
    AddEquation(aPds,aCoeff,aB);
}

void L2SysSurResol::AddEquation(REAL aPds,REAL * aCoeff,REAL aB)
{
    // selectionne (! enfin ...) les coeffs non nuls dans une table d'indexe
    L2SysSurResol::V_GSSR_AddNewEquation_Indexe(VInd,aPds,&VALS[0],aB);
}

// En L1 ca donne simplement

void SystLinSurResolu::V_GSSR_AddNewEquation(REAL aPds,REAL * aCoeff,REAL aB)
{
    PushEquation(aCoeff,aB,aPds);
}

```

Cette filière est très lourde, cependant elle est indispensable en  $L_1$ , puisque chaque équation doit être représentée explicitement et de manière complète.

### D.1.2 Filière indexee

Elle est utilisée par les foncteurs ayant un très grand nombre d'inconnues et (par consequent) des matrices creuses. Par exemple dans `src/photogram/phgr_cGridIncImageMnt.cpp`?

```

void cGridIncImageMnt::OneStepRegulD2 (REAL aPds)
{
    ...
    pSetIncs->AddEqIndexeToSys(pRegD2,aPds/4,mIncs);
}

const std::vector<REAL> & cSetEqFormelles::AddEqIndexeToSys
    ( cElCompiledFonc * aFonct, REAL aPds, const std::vector<INT> & aVInd)
{
    aFonct->SVD_And_AddEqSysSurResol(... ValVar() ...);
}

void cElCompiledFonc::SVD_And_AddEqSysSurResol
    (
        const std::vector<INT> & aVInd,
        REAL aPds,
        REAL *      Pts,
        cGenSysSurResol & aSys,
        bool EnPtsCur
    )
{
    // Si oui le système aSys sait utiliser directement des
    // matrice
    bool UseMat = aSys.GSSR_UseEqMatIndexee();

    ...
    if (UseMat)

```

```

    {
    double ** aDM = aMat.data();
    double * aDF = aFLin.data();

    aSys.GSSR_EqMatIndexee(aVInd,aPds,aDM,aDF,aCste);
    }
    else
    {
    aSys.GSSR_AddNewEquation_Indexe(aVInd,aPds,&(mCompDer[aD][0]),aB);;
    }
}

GSSR_UseEqMatIndexee :
- vrai pour L2SysSurResol (matrice pleine)
- faux pour cGenSysSurResol (donc pour L1 qui en derive sans redef)
- vrai pour cFormQuadCreuse (donc cElMatCreuseMap et cElMatCreuseStrFixe)

```

Par ailleur GSSR\_AddNewEquation\_Indexe , plante en L1.

Optimisation à faire pour les grandes aéros :

- switcher, sans doute à partir de AddEqFonctToSys, vers le mode indexe;
- sauf si le système ne le supporte pas ! Donc par ex L1;

## D.2 Communication avec les systèmes sur-contraints

### D.2.1 Convention d'écritures

Soit un ensemble d'équations d'observations :

$$F^j(X) - O^j = 0; \quad (\text{D.1})$$

Un foncteur correspond à une linéarisation de ces équations. On regroupe plusieurs équations en un seul foncteur, lorsqu'il y a une synergie permettant de mettre en commun les calculs (cas courant, projection image en  $x$  et  $y$ ). À VOIR : PEUT ON GAGNER DU TEMPS EN FACTORISANT TOUTE LES OBSERVATION D'UNE PROJECTION SUR UNE MEME CAMERA.

Si on linéarise l'équation en  $X_0$ , on a :

$$F^j(X_0) + \sum \frac{\partial F^j}{\partial X_k} \delta_k - O^j = 0; \quad (\text{D.2})$$

En notant :

$$D_k^j = \frac{\partial F^j}{\partial X_k} = \text{mCompDer}[j][k] \quad (\text{D.3})$$

$$V^j = F^j(X_0) - O^j = \text{mVal}[j] \quad (\text{D.4})$$

On a l'équation linéaire dont les paramètres sont calculés dans le foncteur:

$$\sum D_k^j \delta_k + V^j = 0; \quad (\text{D.5})$$

Dans l'interface des systèmes linéaires, le méthod `GSSR_AddNewEquation(P,A,B)` correspond à l'observation :

$$(\sum A_k X_k = B) * P; \quad (\text{D.6})$$

C'est pour cela que dans `AddContrainteEqSSR` on a  $B = -V$ .

A priori, dans toutes ces équations, l'inconnue est directement un delta par rapport au point courant (paramètre `EnPtsCur` vaut `true`). Si ce n'était pas le cas, les inconnues seraient  $x_k$  et les équations deviendraient :

$$F^j(X_0) + \sum \frac{\partial F^j}{\partial X_k} (x_k - X_{0k}) - O^j = 0; \quad (\text{D.7})$$

Soit :

$$\sum D_k^j x_k = \sum D_k^j X_{0k} - V; \quad (\text{D.8})$$

D'où le code extrait de `cElCompiledFonc::AddContrainteEqSSR` :

```

REAL aB = -mVal[aD];
...
aB += mCompDer[aD][aIC] * mCompCoord[aIC]

```

### D.2.2 UseEqMatIndexee

Si le système d'équation l'autorise, on va écrire directement une forme quadratique. Chaque équation :

$$P(^t LX = B) \quad (\text{D.9})$$

Apporte une contribution :

$$P(^t LX - B)^2 = P(^t X (^t LL) X - 2B^t LX + B^2) \quad (\text{D.10})$$

## D.3 Mesh computation

Commands are:

- **Malt** with ZoomF=4 (or 8 if you tweak the code...) or use meshlab to downsample point cloud after Nuage2Ply and MergePly (clustered vertex sampling, with cell size = 0,25 %, and average parameter)
- **Nuage2Ply** with Normale=5 option: computes local normal for each point and writes a .ply file with X Y Z NX NY NZ format.
- **MergePly** writes one ply file from multiple ply files
- **PoissonRecon** from Michael Kazhdan and Matthew Bolitho, computes a mesh from the previous ply file (binary can be found in binaire-aux/)

### D.3.1 Command Nuage2Ply :

”Normale” parameters indicates the window size to compute normal: should be 3, 5, 7, etc...

In camera geometry, you can use **Zlimit** command before **Nuage2Ply** to create a mask to remove some parts of the depth image.

Example : **mm3d Zlimit Z\_Num4\_DeZoom8\_GeoI-image1.xml 0.2 1.4 CorrelIm=Correl\_GeoI-image1\_Num\_3.tif**

Will create a mask (Z\_Num4\_DeZoom8\_GeoI-image1.xml\_MasqZminmax.tif) hiding every point with a depth out of [0.2m,1.4m], and points that have a bad correlation value (lower than ValDefCorrel).

This mask can be used in **Nuage2Ply** with the option **Mask**.

### D.3.2 Command MergePly :

MergePly command concats several ply files.

```

mm3d MergePly
*****
* Help for Elise Arg main *
*****
Mandatory unnamed args :
* string :: {Full Name (Dir+Pattern)}
Named args :
* [Name=Out] string
* [Name=Bin] INT :: {Generate Binary or Ascii (Def=true, Binary)}

```

### D.3.3 Example

```

mm3d Nuage2Ply NuageImProf_Geom-Im-5564_Etape_5.xml Normale=5 Out=Fic0.ply
mm3d Nuage2Ply NuageImProf_Geom-Im-5581_Etape_5.xml Normale=5 Out=Fic1.ply
mm3d Nuage2Ply NuageImProf_Geom-Im-5588_Etape_5.xml Normale=5 Out=Fic2.ply

```

```
mm3d MergePly .*ply Out=merged.ply Normale=1
```

```
PoissonRecon --in merged.ply --out result_poisson --depth 10
```

```

PoissonRecon
Usage: PoissonRecon
--in <input points>
[--out <output triangle mesh>]
[--voxel <output voxel grid>]
[--depth <maximum reconstruction depth>=8]
  Running at depth d corresponds to solving on a  $2^d \times 2^d \times 2^d$ 
  voxel grid.
[--fullDepth <full depth>=5]
  This flag specifies the depth up to which the octree should be complete.
[--voxelDepth <depth at which to extract the voxel grid>=<depth>]
[--cgDepth <conjugate-gradients depth>=0]
  The depth up to which a conjugate-gradients solver should be used.
[--scale <scale factor>=1.100000]
  Specifies the factor of the bounding cube that the input
  samples should fit into.
[--samplesPerNode <minimum number of samples per node>=1.000000]
  This parameter specifies the minimum number of points that
  should fall within an octree node.
[--pointWeight <interpolation weight>=4.000000]
  This value specifies the weight that point interpolation constraints are
  given when defining the (screened) Poisson system.
[--iters <iterations>=8]
  This flag specifies the (maximum if CG) number of solver iterations.
  This parameter specifies the number of threads across which
  the solver should be parallelized.
[--confidence]
  If this flag is enabled, the size of a sample's normals is
  used as a confidence value, affecting the sample's
  contribution to the reconstruction process.
[--nWeights]
  If this flag is enabled, the size of a sample's normals is
  used as to modulate the interpolation weight.
[--polygonMesh]
  If this flag is enabled, the isosurface extraction returns polygons
  rather than triangles.
[--density]
  If this flag is enabled, the sampling density is written out with the vertices.
[--double]
  If this flag is enabled, the reconstruction will be performed with double-precision floats.
[--verbose]
  If this flag is enabled, the progress of the reconstructor will be output to STDOUT.

```

For more information on Misha Khazdan's code: <http://www.cs.jhu.edu/~misha/Code/PoissonRecon/>



# Appendix E

## Various formula

I put here different formula that may be useful to understand some part of MicMac. As it's purely utilitarian, I do it the fast way by scanning the paper version.

### E.1 Space resection

As the code for computation of space resection from 3 GCP is a bit tricky, I have summarized the main computation on figure E.1. This notation should be compatible with the variable of class `cNewResProfChamp<Type>` defined in `src/photogram/phgr_low_level.cpp`

### E.2 Stretching (etirement)

I put her the justification of stretching formula in function `cZBuffer::BasculerUnTriangle` :

$$\rho_A = \frac{|\vec{A} - b\vec{B}|^2}{|\vec{A} - c\vec{C}|^2} = \frac{|\vec{BA} - b\vec{B}|^2}{|\vec{CA} - c\vec{C}|^2}$$

$$\rho_A |\vec{CA} - c\vec{C}|^2 = b^2 \vec{B}^2 - 2 \frac{\vec{BA} \cdot \vec{B}}{B^2} b + \vec{BA}^2$$

$$b^2 - 2 \frac{\vec{BA} \cdot \vec{B}}{B^2} b = \frac{1}{B^2} [\rho_A |\vec{CA} - c\vec{C}|^2 - \vec{BA}^2]$$

$$b = \beta \pm \sqrt{\beta^2 + \Delta(c)}$$

$$\boxed{b = \beta + \varepsilon \sqrt{\Delta(c)}}$$

$$\rho_c = \frac{|\vec{B}b - \vec{Cc}|}{|\vec{A} - c\vec{C}|} = \frac{|\vec{CB} + b\vec{B} - c\vec{C}|}{|\vec{CA} - c\vec{C}|^2}$$

$$\rho_c |\vec{CA} - c\vec{C}|^2 = |\vec{CB} + b\vec{B} - c\vec{C} + \varepsilon \sqrt{\Delta(c)} \vec{B}|^2$$

$$\rho_c \omega_1 = (\vec{U} - c\vec{C} + \varepsilon \sqrt{\Delta(c)} \vec{B})^2 = \frac{(\vec{U} - c\vec{C})^2}{\omega_1^2} + B^2 \Delta(c) + 2\varepsilon \sqrt{\Delta(c)} \vec{B} \cdot (\vec{U} - c\vec{C})$$

$$\rho_c \omega_1 - \omega_2 - \Delta(c) B^2 = 2\varepsilon \sqrt{\Delta(c)} \cdot \frac{(\vec{B} \cdot (\vec{U} - c\vec{C}))}{\Delta(c)}$$

$$\boxed{R_{\text{red}}(c) = \Delta(c)^2 - 4 \Delta(c) d^2(c)}$$

Figure E.1: Notation for space resection

$$\begin{aligned}
 & \begin{bmatrix} A_x & B_x \\ A_y & B_y \end{bmatrix} \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} = \omega \cos \theta \vec{A} + \sin \theta \vec{B} = \vec{V}(\theta) \\
 & (\vec{A} \cos \theta + \vec{B} \sin \theta)^2 = A^2 \cos^2 \theta + 2 \vec{A} \cdot \vec{B} \sin \theta + B^2 \sin^2 \theta \\
 & = \frac{A^2 + B^2}{2} \cos^2 \theta + \frac{A^2 - B^2}{2} \cos^2 \theta + \frac{A^2 + B^2}{2} \sin^2 \theta + \frac{B^2 - A^2}{2} \sin^2 \theta + \vec{A} \cdot \vec{B} \sin 2\theta \\
 & = \frac{A^2 + B^2}{2} + \frac{A^2 - B^2}{2} [\cos^2 \theta - \sin^2 \theta] + \vec{A} \cdot \vec{B} \sin 2\theta \\
 & = \frac{A^2 + B^2}{2} + \frac{A^2 - B^2}{2} \cos 2\theta + \vec{A} \cdot \vec{B} \sin 2\theta \\
 & = \frac{A^2 + B^2}{2} + \sqrt{\frac{[A^2 - B^2]^2}{4} + (\vec{A} \cdot \vec{B})^2} [\cos \phi \cos 2\theta + \cos \phi \sin 2\theta] \\
 & = \frac{A^2 + B^2}{2} + \sqrt{[A^2 - B^2]^2 + 4(\vec{A} \cdot \vec{B})^2} \sin [\phi + 2\theta] \\
 & = \frac{A^2 + B^2}{2} + \sqrt{[A^2 - B^2]^2 + 4(\vec{A} \cdot \vec{B})^2} \sin [\phi + 2\theta] \\
 & \|a \times d\| \left\| \vec{V}(\theta) \right\|^2 = \frac{A^2 + B^2 + \sqrt{(A^2 - B^2)^2 + 4(\vec{A} \cdot \vec{B})^2}}{2}
 \end{aligned}$$

Figure E.2: Computation of stretching-étirement formula



## **Appendix F**

## **Référence bibliographique**



# Bibliography

- [Tomasi Kanabe 98] S. Roy, I.J. Cox , 1998, "Shape and Motion from Image Streams under Orthography: a Factorization Method", International Journal of Computer Vision, 9:2, 137-154 (1992)
- [Cox-Roy 98] S. Roy, I.J. Cox , 1998, "A Maximum-Flow formulation of the N-camera Stereo Correspondence Problem", *Proc. IEEE International Conference on Computer Vision*, pp 492–499, Bombay.
- [Fraser C. 97] C. Fraser, 1997, "Digital camera self-calibration", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 52, issue 4, pp. 149-159,
- [Penard L. 2006 ] L. Pnard, N. Paparoditis, M. Pierrot-Deseilligny. "Reconstruction 3D automatique de faades de btiments en multi-vues.", RFIA (Reconnaissance des Formes et Intelligence Artificielle), Tours, France, January 2006.