0.1 C++ course under MicMac's library : Elise

0.1.1 Introduction and generalities

A `C++` course was organised 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 informations 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 executables 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 informations

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

0.1.2 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.

0.1.2.1 Hello World !

The first exrcice 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 favourite IDE, for example Geany, start by including this file **StdAfx.h** wich 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 tutoriel:

```
#${TDPPMD_DIR}/ExoMM_CorrelMulImage.cpp
```

In order to call our program we need to add in **culture3d/src/CBinairies/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/CBinairies/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.

0.1.3 Mandatory or Optionnal Argument?

If you are a user of MicMac you know that calling a mandatory argument doesn't recquire to specify the name of the option. For example **Tapas** recquires 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
```

ElInitArgMain 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
```

0.1.4 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>2.41677870000000006 42.595951300000003 496.235299999999995</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)** [1] 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
```

1. PCP means ParamChantierPhotogram.h

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 dictionnary ==> 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
          }
```

0.1.5 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 exercice 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;
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 (
44             std::list<std::string>::iterator itS=mLFile.begin();
               itS != mLFile.end();
46             itS ++)
                    { std::cout << "      F = " << *itS << "\n"; }}}
48
int ExoMCI_main(int argc,char ** argv)
50 {
     cMCI_Appli anAppli(argc,argv);
52   return EXIT_SUCCESS;
}
```

Try to compile and from the **Mini-Cuxha** folder type in the following command:

```
1 mm3d TestLib ExoMCI ".*jpg"
```

Your prompt should display:

```
1 Nb Files 48
     F = Abbey-IMG\_0173.jpg
3    F = Abbey-IMG\_0191.jpg
```

0.1.6 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:

```
1 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:

```
1 #include "StdAfx.h"

3 int  ExoMCI_main(int argc,char ** argv)
{
5    std::string aNameI1,aNameI2;
     int aPxMax= 199;
7    int aSzW = 5;
     ElInitArgMain
9    (
         argc,argv,
11       LArgMain()  << EAMC(aNameI1,"Name Image1")
                     << EAMC(aNameI2,"Name Image2"),
13       LArgMain()  << EAM(aPxMax,"PxMax",true,"Pax Max")
     );
15
     Im2D_U_INT1 aI1 = Im2D_U_INT1::FromFileStd(aNameI1);
17   Im2D_U_INT1 aI2 = Im2D_U_INT1::FromFileStd(aNameI2);

19   Pt2di aSz1 = aI1.sz();
     Im2D_REAL4  aIScoreMin(aSz1.x,aSz1.y,1e10);
21   Im2D_REAL4  aIScore(aSz1.x,aSz1.y);
     Im2D_INT2   aIPaxOpt(aSz1.x,aSz1.y);
23
     Video_Win aW = Video_Win::WStd(Pt2di(1200,800),true);
25
     for (int aPax = -aPxMax ; aPax <=aPxMax ; aPax++)
27   {
         std::cout << "PAX tested " << aPax << "\n";
29       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
```

0.1.7 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 + patterne
        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 *                        mMastIm; //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);
```

```
55          ELISE_COPY(mW->all_pts(),mIm.in(),mW->ogray());
57          // 2- Test with functional approach
59          ELISE_COPY
61          (
61             mIm.all_pts(),
63             mTifIm.in(),
63             // Output is directed both in window & Im
65             mIm.out() | mW->ogray()
65           );
67           ELISE_COPY
69          (
69             disc(Pt2dr(200,200),150),
71             255-mIm.in()[Virgule(FY,FX)],
71             // Output is directed both in window & Im
73              mW->ogray()
73           );
75           int aSzF = 20;
77          ELISE_COPY
77          (
79             rectangle(Pt2di(0,0),Pt2di(400,500)),
79             // rect_som(mIm.in(),20)/ElSquare(1+2*aSzF),
81             rect_som(mIm.in_proj(),aSzF)/ElSquare(1+2*aSzF),
81             // Output is directed both in window & Im
83              mW->ogray()
83           );
85           Fonc_Num aF = mIm.in_proj();
87           aSzF=4;
87           int aNbIter = 5;
89           for (int aK=0 ; aK<aNbIter ; aK++)
89               aF = rect_som(aF,aSzF)/ElSquare(1+2*aSzF);
91          ELISE_COPY(mIm.all_pts(),aF,mW->ogray());
93         //  ELISE_COPY(mIm.all_pts(),mTifIm.in(),mIm.out());
95
97          ELISE_COPY(mIm.all_pts(),mIm.in(),mW->ogray());
99          // 3- Test with Tpl  approach
101         Im2D<U_INT1,INT4> aDup(mSz.x,mSz.y);
101         TIm2D<U_INT1,INT4> aTplDup(aDup);
103         TIm2D<U_INT1,INT4> aTIm(mIm);
105         for (int aK=0 ; aK<aNbIter ; aK++)
105         {
107            for (int anY=0 ; anY<mSz.y ; anY++)
107                for (int anX=0 ; anX<mSz.x ; anX++)
109                {
109                    int aNbVois = ElSquare(1+2*aSzF);
111                    int aSom=0;
111                    for (int aDx=-aSzF; aDx<=aSzF ; aDx++)
113                       for (int aDy=-aSzF; aDy<=aSzF ; aDy++)
113                           aSom += aTIm.getproj(Pt2di(anX+aDx,anY+aDy));
115                    aTplDup.oset(Pt2di(anX,anY),aSom/aNbVois);
115                }
117            Pt2di aP;
119            for (aP.y=0 ; aP.y<mSz.y ; aP.y++)
119                for (aP.x=0 ; aP.x<mSz.x ; aP.x++)
121                aTIm.oset(aP,aTplDup.get(aP));
121         }
123         ELISE_COPY(mIm.all_pts(),mIm.in(),mW->ogray());
    }
```

```
125
127 }
129 void cMCI_Ima::CalculImOrthoOfProf(double aProf,cMCI_Ima * aMaster)
    {
131     TIm2D<U_INT1,INT> aTIm(mIm);
        TIm2D<U_INT1,INT> aTImOrtho(mImOrtho);
133     int aSsEch = 10;
        Pt2di aSzR = aMaster->mSz/ aSsEch;
135     TIm2D<float,double> aImX(aSzR);
        TIm2D<float,double> aImY(aSzR);
137
        Pt2di aP;
139     for (aP.x=0 ; aP.x<aSzR.x; aP.x++)
        {
141         for (aP.y=0 ; aP.y<aSzR.y; aP.y++)
            {
143             Pt3dr aPTer = aMaster->mCam->ImEtProf2Terrain(Pt2dr(aP*aSsEch),aProf);
                Pt2dr aPIm = mCam->R3toF2(aPTer);
145             aImX.oset(aP,aPIm.x);
                aImY.oset(aP,aPIm.y);
147         }
        }
149
        for (aP.x=0 ; aP.x<aMaster->mSz.x; aP.x++)
151     {
            for (aP.y=0 ; aP.y<aMaster->mSz.y; aP.y++)
153         {
                /*
155             Pt3dr aPTer = aMaster->mCam->ImEtProf2Terrain(Pt2dr(aP),aProf);
                Pt2dr aPIm0 = mCam->R3toF2(aPTer);
157             */
                Pt2dr aPInt = Pt2dr(aP) / double(aSsEch);
159             Pt2dr aPIm (aImX.getr(aPInt,0),aImY.getr(aPInt,0));
                float aVal = aTIm.getr(aPIm,0);
161             aTImOrtho.oset(aP,round_ni(aVal)); //returns nearest integer
            }
163     }

165     if ( 0 &&  (mName=="Abbey-IMG_0250.jpg"))
        {
167         static Video_Win * aW = Video_Win::PtrWStd(Pt2di(1200,800));
            ELISE_COPY(mImOrtho.all_pts(),mImOrtho.in(),aW->ogray());
169     }
    }
171

173 Fonc_Num  cMCI_Ima::FCorrel(cMCI_Ima *aMaster)
    {
175     int aSzW = 2;
        double aNbW = ElSquare(1+2*aSzW);
177     Fonc_Num aF1 = mImOrtho.in_proj();
        Fonc_Num aF2 = aMaster->mImOrtho.in_proj();
179


181     Fonc_Num aS1 = rect_som(aF1,aSzW) / aNbW;
        Fonc_Num aS2 = rect_som(aF2,aSzW) / aNbW;
183
        Fonc_Num aS12 = rect_som(aF1*aF2,aSzW) / aNbW - aS1*aS2;
185     Fonc_Num aS11 = rect_som(Square(aF1),aSzW) / aNbW - Square(aS1);
        Fonc_Num aS22 = rect_som(Square(aF2),aSzW) / aNbW - Square(aS2);
187
        Fonc_Num aRes = aS12 / sqrt(Max(1e-5,aS11*aS22));
189
    //static Video_Win * aW = Video_Win::PtrWStd(Pt2di(1200,800));
191 //ELISE_COPY(aW->all_pts(),128*(1+aRes),aW->ogray());

193     return aRes;
    }
```

```
195
197 void cMCI_Ima::InitMemImOrtho(cMCI_Ima * aMas)
{
199     mImOrtho.Resize(aMas->mIm.sz());
}
201
Pt2dr cMCI_Ima::ClikIn()
203 {
    return mW->clik_in()._pt; //returns 2D point when click on image
205 }

207 void  cMCI_Ima::DrawFaisceaucReproj(cMCI_Ima & aMas,const Pt2dr & aP)
{
209     if (!mW) return ;
    double aProfMoy =  aMas.mCam->GetProfondeur();
211     double aCoef = 1.2;

213     std::vector<Pt2dr> aVProj;
    for (double aMul = 0.2; aMul < 5; aMul *=aCoef) //steps of depth
215     {
          Pt3dr aP3d =  aMas.mCam->ImEtProf2Terrain(aP,aProfMoy*aMul);
217           Pt2dr aPIm = this->mCam->R3toF2(aP3d);

219           aVProj.push_back(aPIm); //creation of polyline
    }
221     for (int aK=0 ; aK<((int) aVProj.size()-1) ; aK++)
        mW->draw_seg(aVProj[aK],aVProj[aK+1],mW->pdisc()(P8COL::red));
223 }

225 double cMCI_Ima::EstimateStep(cMCI_Ima * aMas)
{
227
   std::string aKey = "NKS-Assoc-CplIm2Hom@@dat";
229
   std::string aNameH =    mAppli.Dir()
231                         + mAppli.ICNM()->Assoc1To2
                          (
233                             aKey,
                              this->mName,
235                           aMas->mName,
                              true
237                       );
   ElPackHomologue aPack = ElPackHomologue::FromFile(aNameH);
239
   Pt3dr aDirK = aMas->mCam->DirK();
241    for
   (
243        ElPackHomologue::iterator iTH = aPack.begin();
       iTH != aPack.end();
245        iTH++
   )
247    {
       Pt2dr aPInit1 = iTH->P1();
249        Pt2dr aPInit2 = iTH->P2();

251        double aDist;
       Pt3dr aTer = mCam->PseudoInter(aPInit1,*(aMas->mCam),aPInit2,&aDist);
253
       double aProf2 = aMas->mCam->ProfInDir(aTer,aDirK);
255

257        Pt2dr aProj1 = mCam->R3toF2(aTer);
       Pt2dr aProj2 = aMas->mCam->R3toF2(aTer);
259
      // std::cout << aMas->mCam->ImEtProf2Terrain(aProj2,aProf2) -aTer << "\n";
261

263        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;
     }

     // Ckeck 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 an 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**.