Colloque:

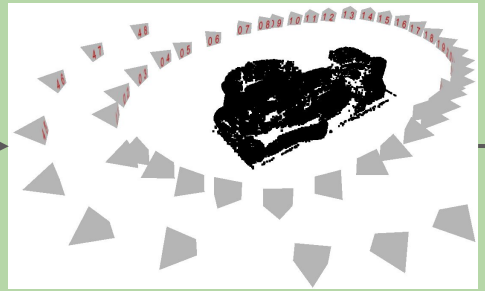
Photogrammétrie Numérique et Perception 3D : les nouvelles conquêtes.

OpenMVG Tutorial

Pierre Moulon
Pascal Monasse (Imagine/LIGM/ENPC/Université Paris Est)

Structure from Motion (SfM)

Photogrammetry: from pictures to 3D scenes

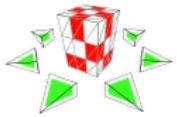


Covering these topics today

Pictures → Image matching → Structure from Motion → Surface estimation → Color restitution

MVG: multi-view geometry

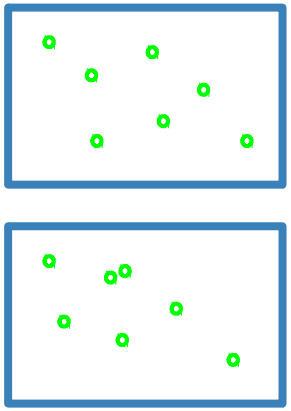
MVS: multi-view stereovision

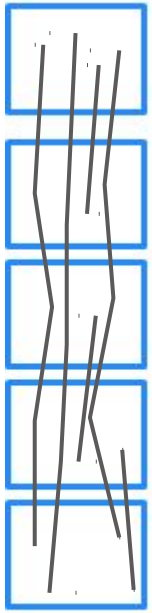


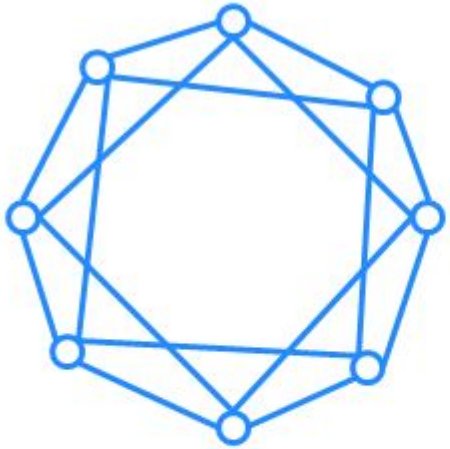
Structure from motion

Algorithmic formulation

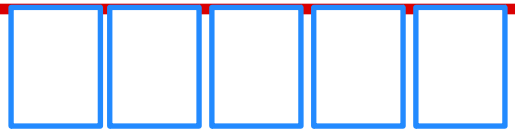
Image Processing	Nearest Neighbors	Multiple-View Geometry	
Detection	Matching	Filtering	Optimization



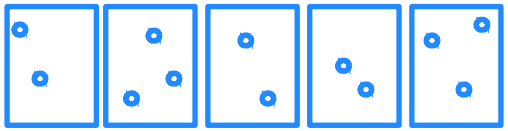




Vocabulary & notations

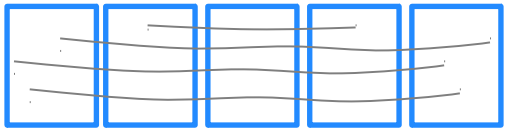


Views/Frames/Images: {I_i}



Features: {x_j + attributes (description)}

- salient regions of an image (corners, blob,...)

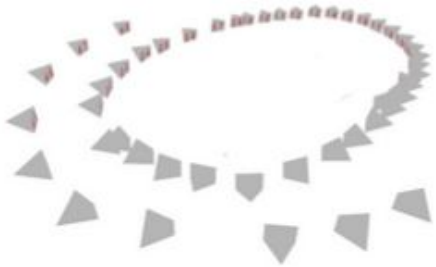


Tracks: {x_j,x_k,...,x_l}

- 2D observation x_j of a putative 3D points X_i

Structure/Landmarks: {X_i, {x_j,x_k,...,x_l}}

- A 3D point + visibility information



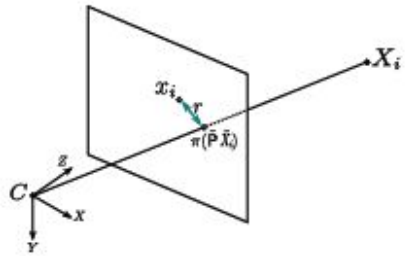
Cameras: {intrinsics;extrinsics}

Intrinsic camera parameters:

- Define projection from 3D to the image/camera plane

Extrinsic camera parameters:

- 3D **Pose**: From the scene world to the local camera coordinates system. A rotation + translation : {R_i,C_i}



Residual error: r = || x - P(X_i) ||

- Measure fitting between the 3D point X_i; the camera location and the 2d x_i fixed observation.

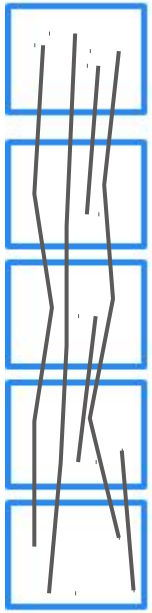
An optimization problem

Unknown parameters	**Known** parameters	**The** “math” **problem**
Poses: {P_i} = {R_i|C_i} **3D Structure**: {X_i;{{x_{i,j}}	**Camera intrinsics**: {K_i} **2D correspondences**: {x_{i,j}}	Minimization of a residual error: sum_(i,j)(ϵ)

Multiple-View Geometry

Input data

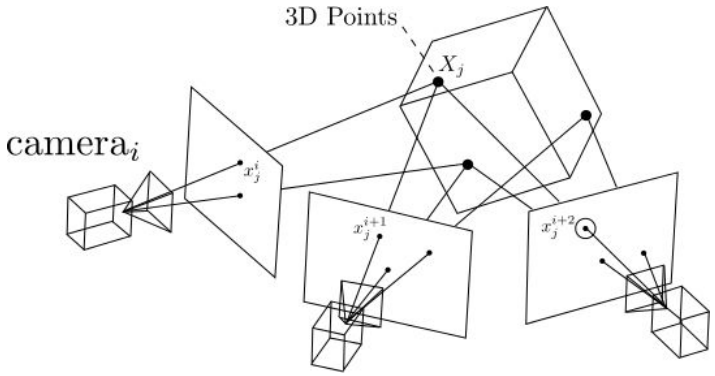
2D points correspondences



Model estimation

parameters initialization

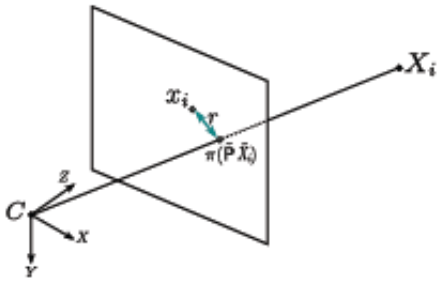
$\{X_j\}, \{P_i\}$



Parameters optimization

Minimization of cost functions

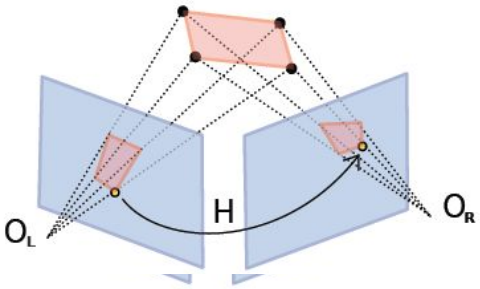
$$\min\left(\sum_i \sum_j ||x_{i,j} - P_i \quad X_j||\right)$$

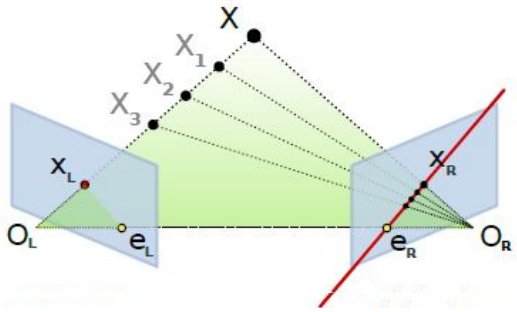


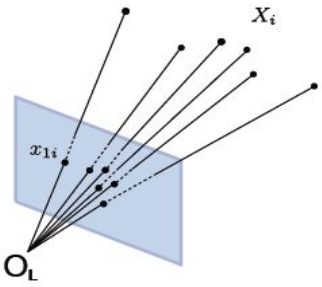
OpenMVG
Open Multiple View Geometry

What is OpenMVG?

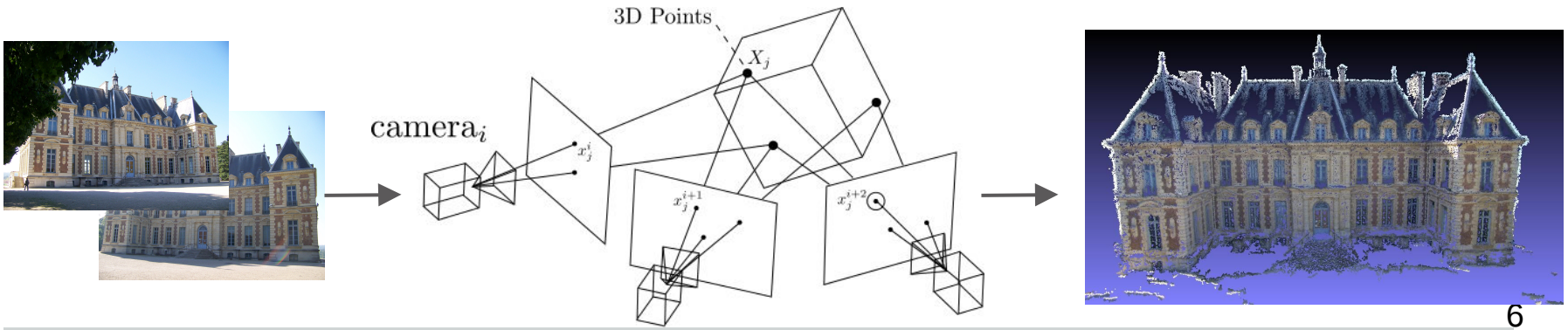
OpenMVG is a list of libraries to solve **M**ulti**V**iew **G**eometry problems,







and **SfM** (Structure from Motion):



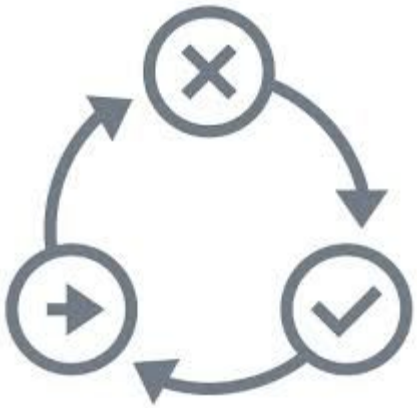
OpenMVG
Open Multiple View Geometry

Design credos:

1. **"Keep it simple, keep it maintainable":**
   Easy to read,
   Easy to maintain,
   Easy to re-use (MPL2 licensed – permissive license).

2. **Test driven development**
   Assert that code is working as we want,
   Regression testing.

   Provide simple usage examples,
   Help people to implement new things.



OpenMVG
Open Multiple View Geometry

Accessibility & Continuous integration

Github

Easy access and collaboration

Issue tracking, Milestones, fork + pull request

V0.1 (8 feb 2013) => V0.9 (5 oct 2015) => V1.0 (2016)

25 external commiters (from 1700 lines to 1)

→ show that people can really make deep change

OpenMVG
Open Multiple View Geometry

Accessibility & Continuous integration

Github

Easy access and collaboration

Issue tracking, Milestones, fork + pull request

Travis/AppVeyor 

Continuous integration (Linux/Windows):

- Compilation
- Unit test -> regression checking & tutorials
- Code coverage

Multiplatform

Linux, MacOS, Windows, ARM

Documentation

http://openmvg.readthedocs.org/

OpenMVG
Open Multiple View Geometry

Supported by the research & users & the industry:

IMAGINE

Research Laboratory

ATOR (Arc-Team Open Research)

Archeologist (users)

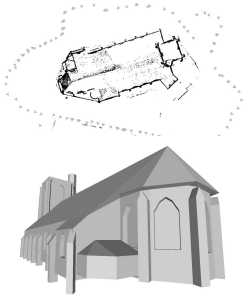
OpenMVG
Open Multiple View Geometry

Supported by the research & users & the industry:

MIKROS Image

VFX / Post-production company

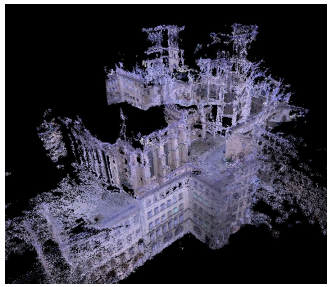
- photogrammetry,
- image based-modeling.



FOXEL

Mobile-mapping company

- Photogrammetry.



....

OpenMVG
Open Multiple View Geometry

Goals

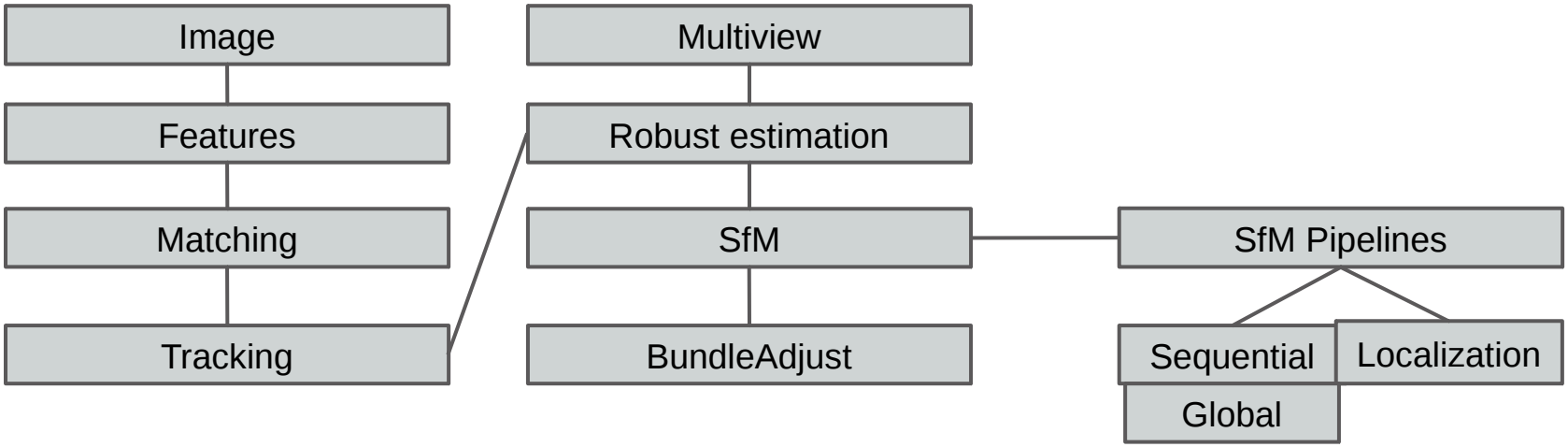
- Ease reproducible research in Multiple View Geometry,
- Create tools to build Accurate **S**tructure **f**rom **M**otion frameworks,
- Easy to use and modify.

- Provide a framework to develop Multiple View Geometry application.
  - C++, CMake.

- Making photogrammetry more accessible to people.

OpenMVG library structure

A-Z needed libraries to solve MVG problems and **S**tructure **f**rom **M**otion:



Each module is a tiny library within a specific directory and namespace.

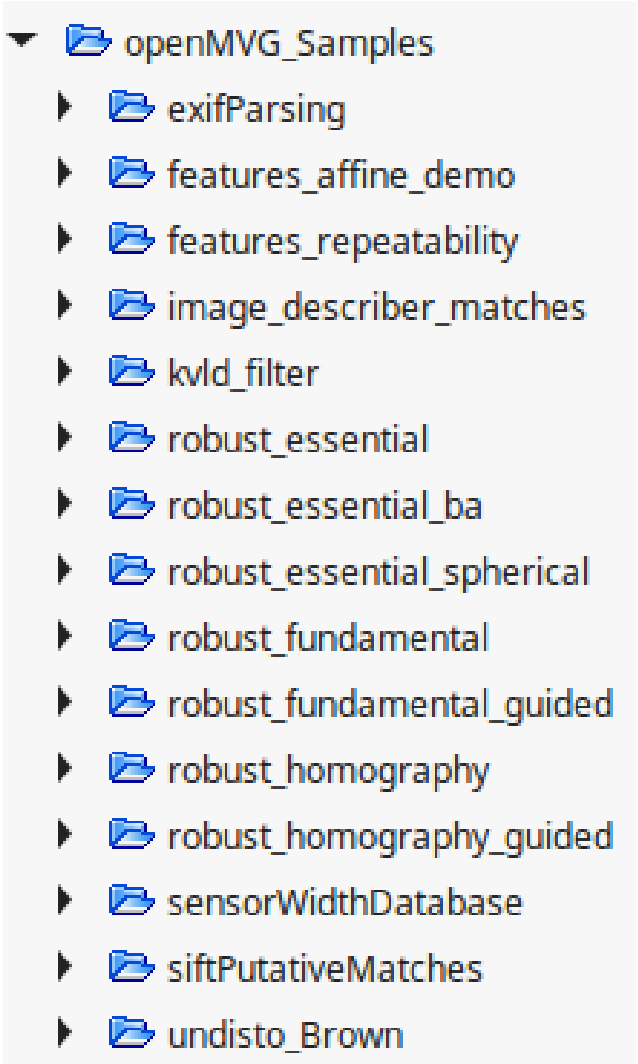
OpenMVG library structure

A-Z needed libraries to solve MVG problems and **S**tructure **f**rom **M**otion:

- Image: Pixel container + Image processing
- Features: Feature&Descriptor detection/storage => Regions concept
- Matching: Nearest Neighbor search & filtering
- Tracking: Unordered feature tracking (Link 2D features in tracks)
- Cameras: Generic camera projection
- Multiview: Multi-view geometry solvers
- Robust estimation: Robust estimation of models on datum
- ….
- SfM: Link algorithm into Structure from Motion pipelines
  - BundleAdjust: Bundle Adjustment
  - SfM Pipelines: Sequential | Global | Localization

OpenMVG code samples

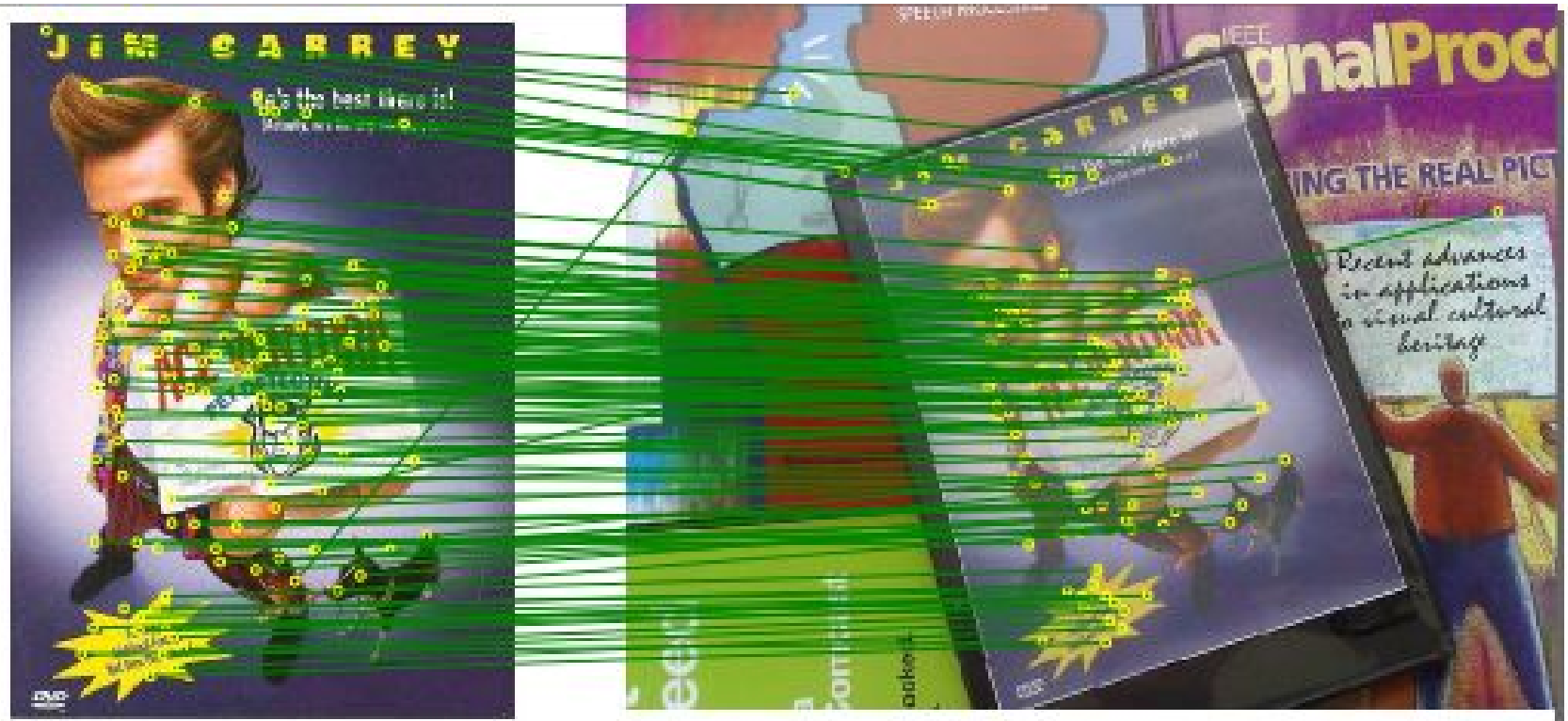
A collection of samples to show how is working the library:



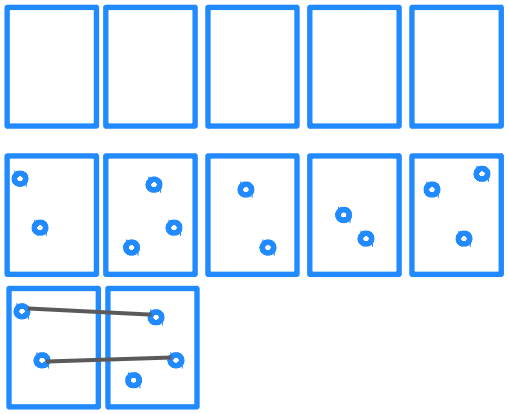
OpenMVG code samples

=> Compute photometric corresponding points between two images:
/openMVG/src/openMVG_Samples/image_describer_matches/describe_and_match.cpp
Using three openMVG modules:

- Image,
- Features,
- Matching



Vocabulary & notations



Views/Frames/Images: {I_i}

Features: {x_j + attributes (description)}
- salient regions of an image (corners, blob,...)

Pairwise matches: {i,j}
- collection of pair (feature indexes)

openMVG concepts

Image<X>

Regions
Feature/Descriptor

PairwiseMatches
IndMatch

OpenMVG code samples

=> Compute photometric corresponding points between two images:
/openMVG/src/openMVG_Samples/image_describer_matches/describe_and_match.cpp

Includes:

```
#include "openMVG/image/image.hpp"
#include "openMVG/features/features.hpp"
#include "openMVG/matching/matching_filters.hpp"
#include "openMVG/matching/regions_matcher.hpp"

#include "nonFree/sift/SIFT_describer.hpp" // Feature detection/description

using namespace openMVG;
using namespace openMVG::image;
using namespace openMVG::features;
using namespace openMVG::matching;
```

OpenMVG code samples

=> Compute photometric corresponding points between two images:

Open the two images:

```
const string jpg_filenameL = "...";
const string jpg_filenameR = “...";

Image<unsigned char> imageL, imageR;
If (!ReadImage(jpg_filenameL.c_str(), &imageL))
  return EXIT_FAILURE;
If (!ReadImage(jpg_filenameR.c_str(), &imageR))
  return EXIT_FAILURE;
```

OpenMVG code samples

=> Compute photometric corresponding points between two images:

Extracting SIFT regions:

```
// Instantiation of a Region extractor
std::shared_ptr<Image_describer> image_describer =
   std::make_shared<SIFT_Image_describer>(SiftParams());

//--
// Detect regions thanks to the image_describer
//--
std::map<IndexT, std::unique_ptr<features::Regions> > regions_perImage;
image_describer->Describe(imageL, regions_perImage[0]);
image_describer->Describe(imageR, regions_perImage[1]);
```

OpenMVG code samples

=> Compute photometric corresponding points between two images:

Matching SIFT regions:

```
//--
// Compute corresponding points
//--
//-- Perform matching -> find Nearest neighbor, filtered with Distance ratio
matching::IndMatches vec_PutativeMatches;
matching::DistanceRatioMatch(
  0.8,                                  // Distance ratio parameter
  matching::BRUTE_FORCE_L2,  // Nearest neighbor method
  *regions_perImage[0].get(),        // Query regions
  *regions_perImage[1].get(),        // Database regions
  vec_PutativeMatches);              // Corresponding output feature index
```

OpenMVG code samples

=> Compute photometric corresponding points between two images:

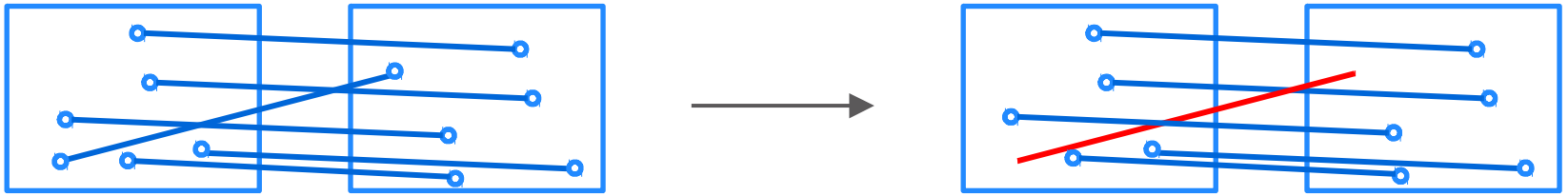
Exporting the result as vector data:

```
const std::vector<PointFeature>
    featsL = regions_perImage.at(0)->GetRegionsPositions(),   featsR = regions_perImage.at(1)-
>GetRegionsPositions()

  // Draw correspondences after Nearest Neighbor ratio filter
  {
    svgDrawer svgStream( imageL.Width() + imageR.Width(), max(imageL.Height(),
imageR.Height()));
    svgStream.drawImage(jpg_filenameL, imageL.Width(), imageL.Height());
    svgStream.drawImage(jpg_filenameR, imageR.Width(), imageR.Height(), imageL.Width());
    for (size_t i = 0; i < vec_PutativeMatches.size(); ++i)
    {
      // Link corresponding features (draw circles for features and a line between them)
      const PointFeature & L = featsL[vec_PutativeMatches[i].i_],
      const PointFeature & R = featsR[vec_PutativeMatches[i].j_];
      svgStream.drawLine(L.x(), L.y(), R.x()+imageL.Width(), R.y(), svgStyle().stroke("green", 2.0));
      svgStream.drawCircle(L.x(), L.y(), 3.0f, svgStyle().stroke("yellow", 2.0));
      svgStream.drawCircle(R.x()+imageL.Width(), R.y(), 3.0f,svgStyle().stroke("yellow", 2.0));
    }
    std::ofstream svgFile( "02_Matches.svg" );
    svgFile << svgStream.closeSvgFile().str();
    svgFile.close();
  }
```

OpenMVG code samples

Detection of false positive matches is crucial in computer vision

This task can be performed thanks to robust model estimation:
- Find the model that fit most of the datum.
  → Homography/Essential/Fundamental matrix...



OpenMVG implements 3 robust estimation algorithms:
- Max-Consensus (RANSAC)
- Lmeds,
- AC-RANSAC (A Contrario Ransac)
  - Automatic parameter estimation.

OpenMVG code samples

=> Compute geometrically meaningful matches:
/openMVG/src/openMVG_Samples/robust_homography/robust_homography.cpp
Using four openMVG modules:

- Image, features, matching, **robust_estimation**

OpenMVG code samples

=> Compute geometrically meaningful matches:
/openMVG/src/openMVG_Samples/robust_homography/robust_homography.cpp

Includes:

```
// Model solver
#include "openMVG/multiview/solver_homography_kernel.hpp"
#include "openMVG/multiview/conditioning.hpp"
// Robust estimator
#include "openMVG/robust_estimation/robust_estimator_ACRansac.hpp"
#include "openMVG/robust_estimation/robust_estimator_ACRansacKernelAdaptator.hpp"


// Namespace
using namespace openMVG::robust;
```

OpenMVG code samples

=> Compute geometrically meaningful matches:
/openMVG/src/openMVG_Samples/robust_homography/robust_homography.cpp

Robust estimation of a Homography:

```
std::vector<IndMatch> vec_PutativeMatches; // Initialized with some photometric matches
const PointFeatures
  featsL = regions_perImage.at(0)->GetRegionsPositions(), featsR = regions_perImage.at(1)->GetRegionsPositions();

// Homography geometry filtering of putative matches

  //-- Build array with the corresponding photometric interest points
  Mat xL(2, vec_PutativeMatches.size()), xR(2, vec_PutativeMatches.size());
  for (size_t k = 0; k < vec_PutativeMatches.size(); ++k)  {
    const PointFeature imaL = featsL[vec_PutativeMatches[k].i_], imaR = featsR[vec_PutativeMatches[k].j_];
    xL.col(k) = imaL.coords().cast<double>();  xR.col(k) = imaR.coords().cast<double>();
  }

  //-- Definition of the HOMOGRAPHY Kernel (Embed minimal solver & fitting error computation)
  typedef ACKernelAdaptor<   openMVG::homography::kernel::FourPointSolver,
    openMVG::homography::kernel::AsymmetricError,  UnnormalizerI, Mat3> KernelType;
  KernelType kernel (xL, imageL.Width(), imageL.Height(),   xR, imageR.Width(), imageR.Height(), false);

  //-- Robust estimation (return the most meaningful model found)
  Mat3 H; // MODEL
  std::vector<size_t> vec_inliers; // INLIER list
  const std::pair<double,double> ACRansacOut = ACRANSAC(kernel, vec_inliers, 1024, &H,
    std::numeric_limits<double>::infinity(),
    True);
```

OpenMVG – Brief summary

A collection of:

- librairies
- samples
- softwares

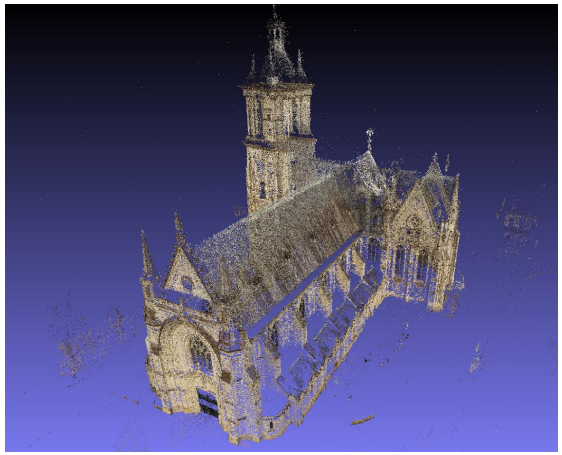
What can the users can do with OpenMVG?

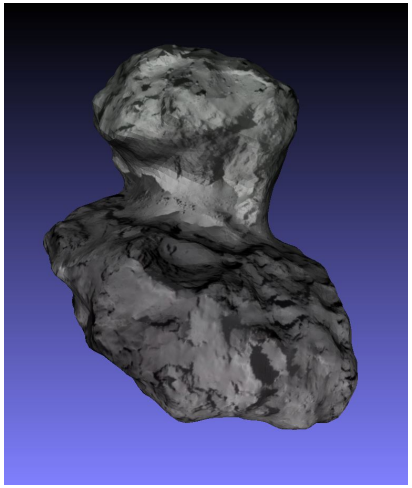
- use it for fun,
- use it for practical applications,
- to develop their own tools,

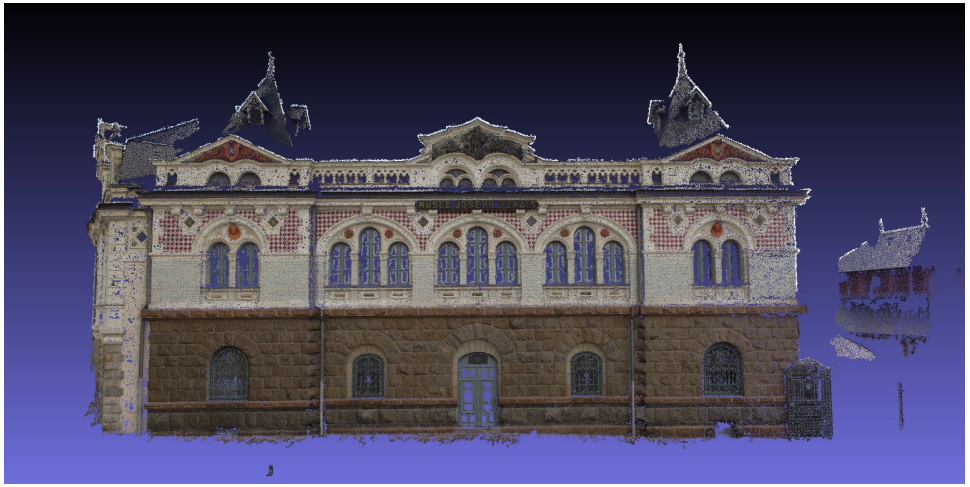
...

User results

Cultural heritage:







User results

OpenMVG in a GUI

=> Projet: Regard3D

=> Projet: SfM-Worflow

User results

OpenMVG + Interoperability + Densification(3rd party)

- PMVS/CMVS → Point clouds

Credits: Romuald Perrot

User results

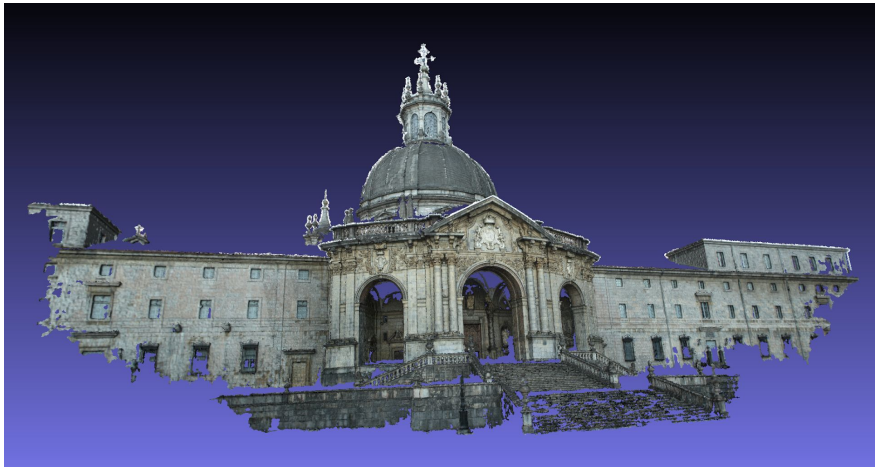
OpenMVG + Interoperability + Densification(3rd party)

- CMPMVS

- MVE

CMPMVS: Mesh

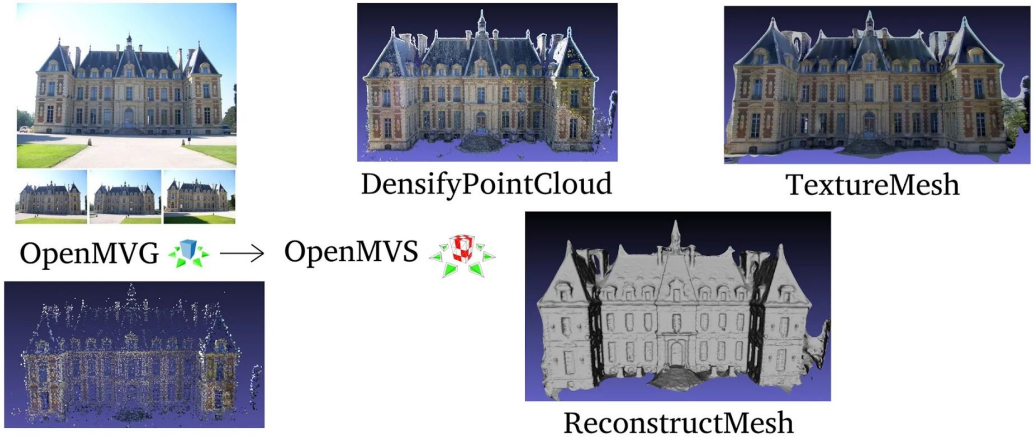
MVE: Mesh



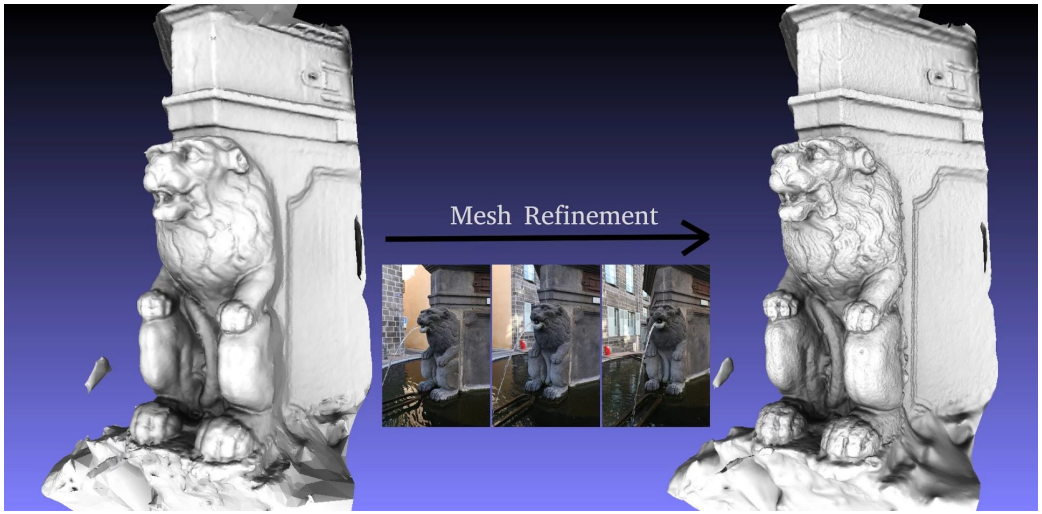
Credits: Pierre-Yves Paranthoën / Romuald Perrot

User results

OpenMVG + Interoperability + Densification(3rd party)
- OpenMVS



OpenMVS: Mesh



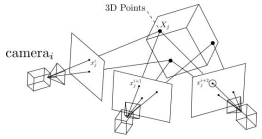
User results

OpenMVG + Interoperability + Densification(3rd party)

- Export to MicMac

→ WIP

Photogrammetry



What **is required**:

- “nice” overlapping pictures,
- initial correspondences set is crucial for:
  - completeness of the structure & camera poses.
- robustness.

What is **hard**:

- accuracy & speed,
- deal with false positive (robustness),
- deal with image distortions:
  - many methods use hard threshold & distortion free models.

What is **trendy**:

- deal with large scale image collection,
- deal with external sensors (GPS, IMU)
  - accuracy, noise, drift, synchronization

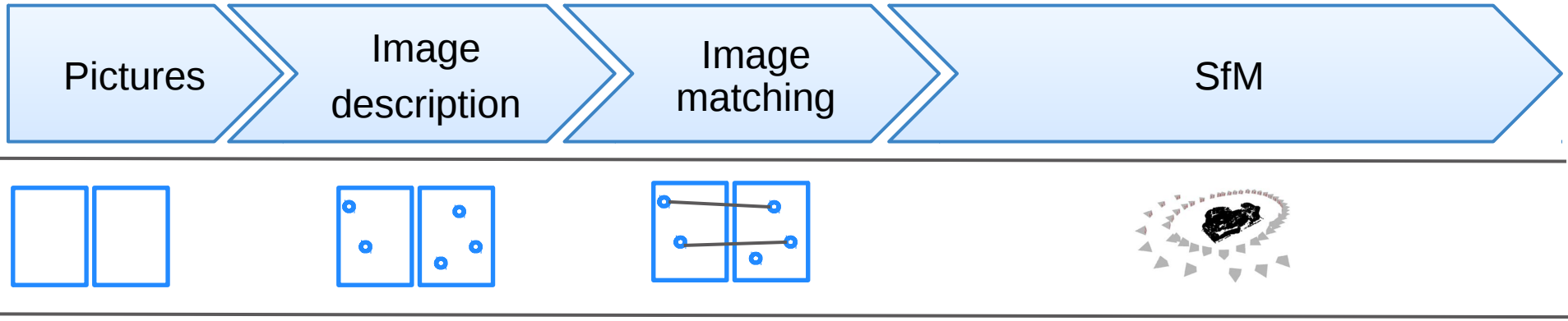
Understanding OpenMVG pipelines

- from data to pipelines

OpenMVG – SfM Pipelines

From concept to pipelines & process.

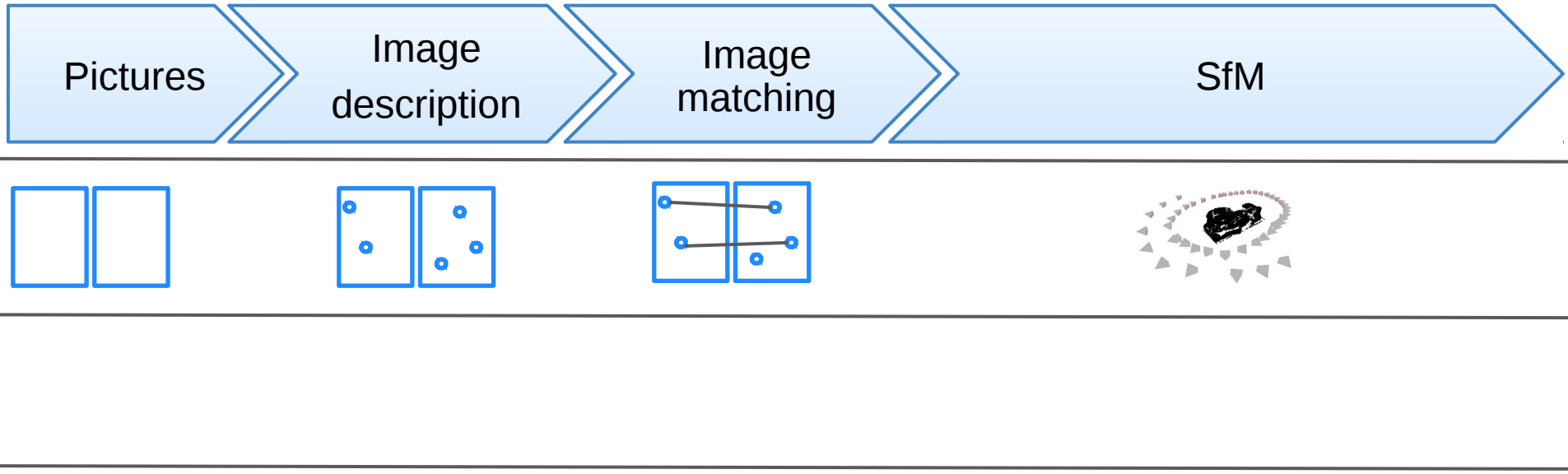
- OpenMVG splits the toolchain in small concepts that are easy to use and manipulate.



OpenMVG – SfM Pipelines

From concept to pipelines & process.

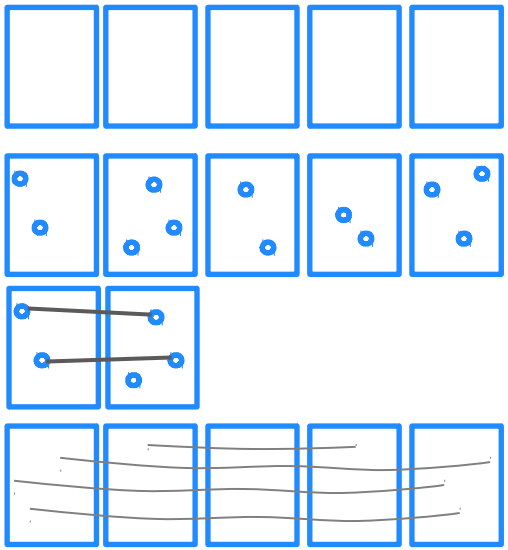
- OpenMVG splits the toolchain in small concepts that are easy to use and manipulate.

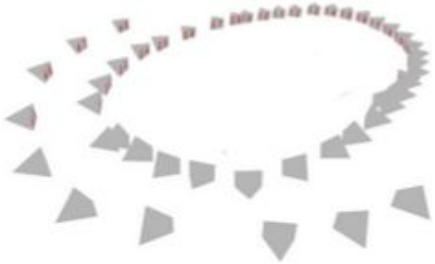


Using Abstract elements to support user customizations:

- view description (image metadata),
- camera model (camera intrinsics),
- features/descriptors (image regions).

Vocabulary & notations





Views/Frames/Images: {I_i}

Features: {x_j + attributes (description)}
- salient regions of an image (corners, blob,...)

Pairwise matches: {i,j}
- collection of pair (feature indexes)

Tracks: {x_j,x_k,...,x_l}
- 2D observation x_j of a putative 3D points X_i

Structure/Landmarks: {X_i, {x_j,x_k,...,x_l}}
- A 3D point + visibility information

Cameras: {intrinsics;extrinsics}
Intrinsic camera parameters:
- Define projection from 3D to the image/camera plane

Extrinsic camera parameters:
- 3D **Pose**: From the scene world to the local camera coordinates system. A rotation + translation : {R_i,C_i}

openMVG concepts

View
Image name, metadata

Regions
Feature/Descriptor

PairwiseMatches
IndMatch

Tracks

Landmark
[{Observations, Vec3}]

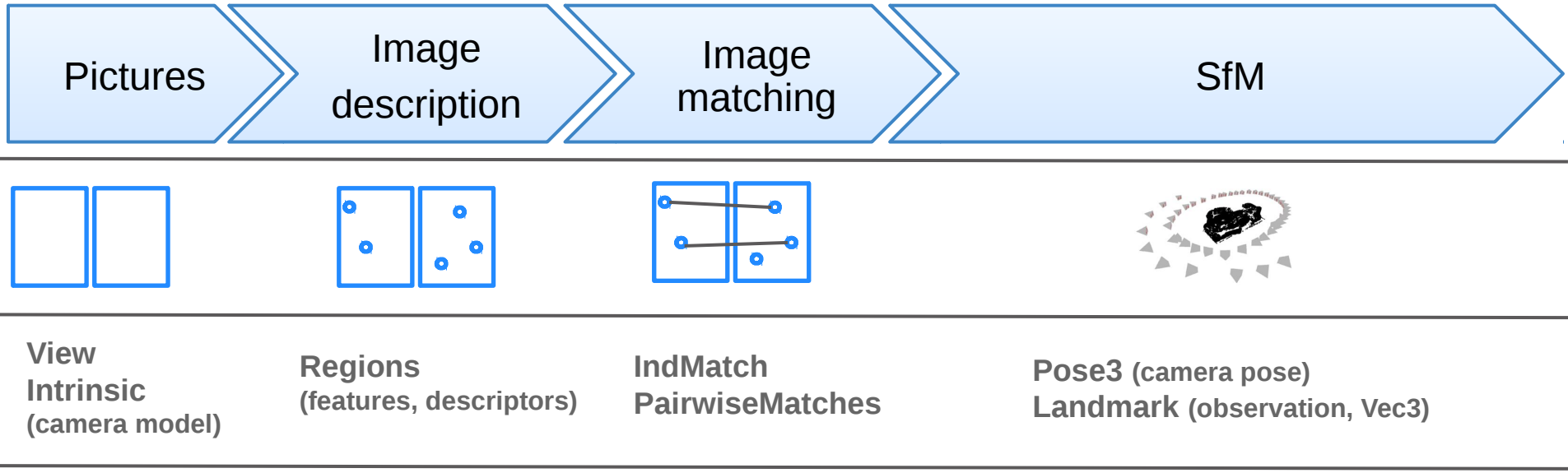
IntrinsicBase
Pinhole_Intrinsic...

Pose3

OpenMVG – SfM Pipelines

From concept to pipelines & process.

- OpenMVG splits the toolchain in small concepts that are easy to use and manipulate.

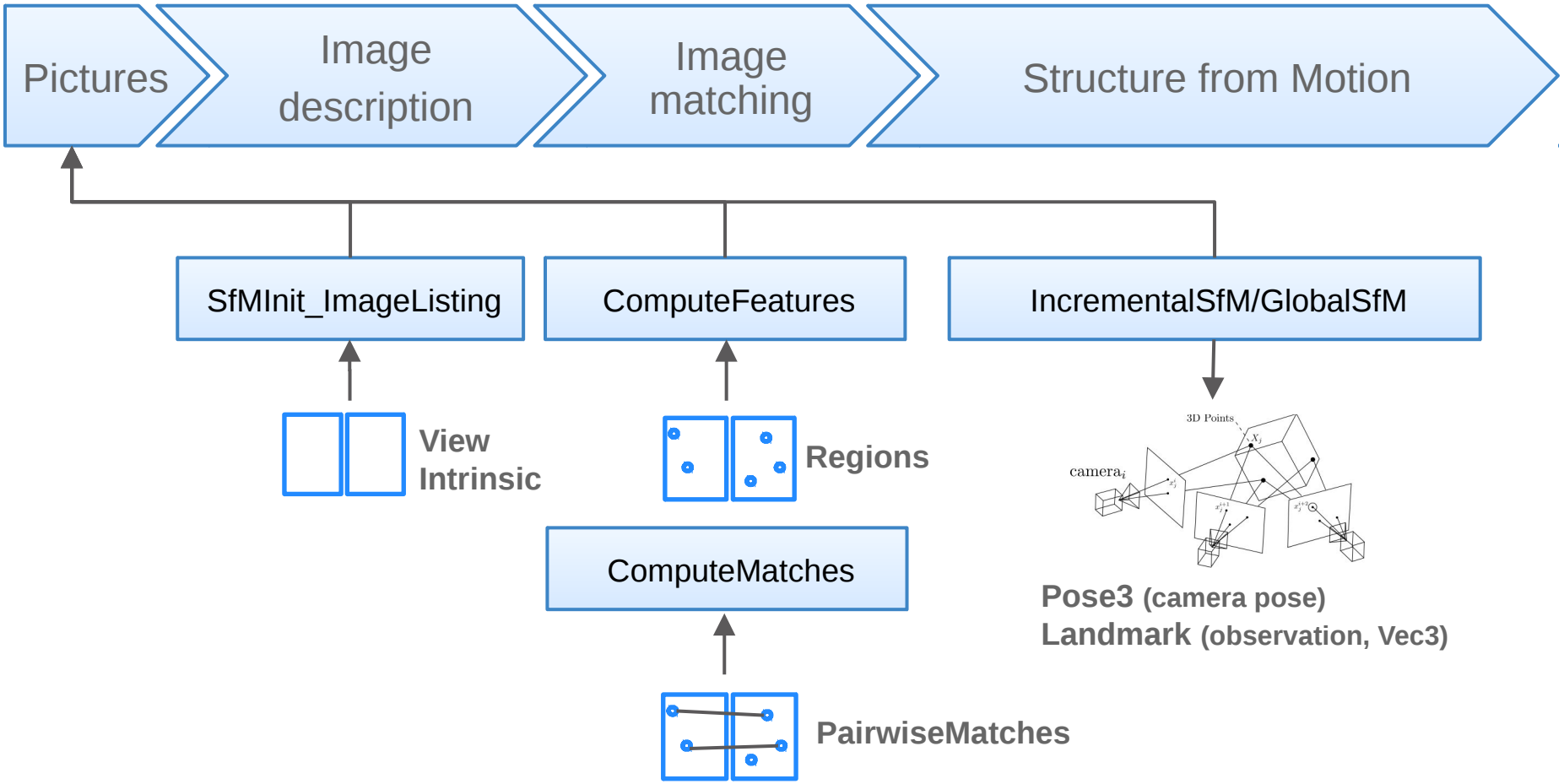


Using Abstract elements to support user customizations:

- view description (image metadata),
- camera model (camera intrinsics),
- features/descriptors (image regions).

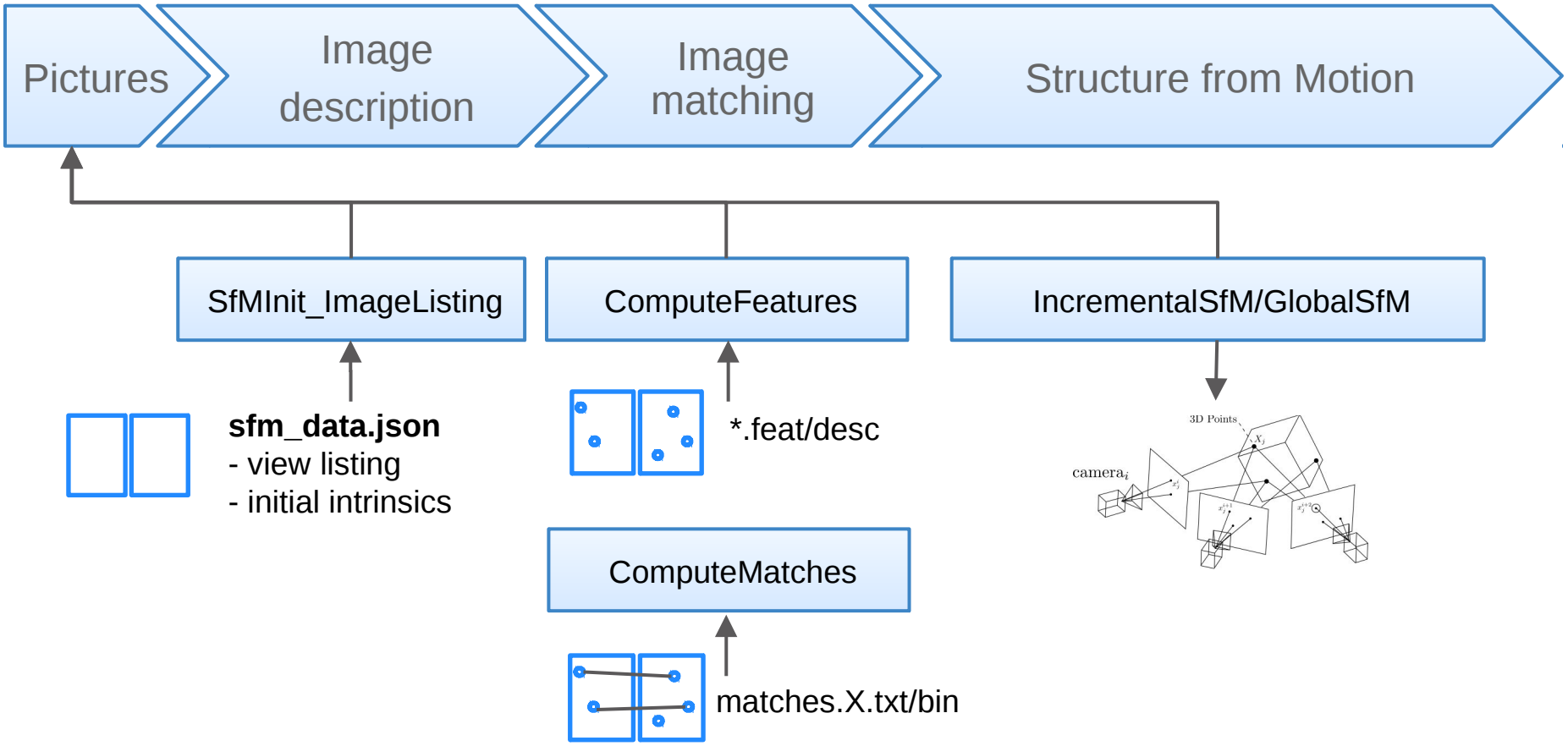
OpenMVG – SfM Pipelines

From concept to pipelines & process.



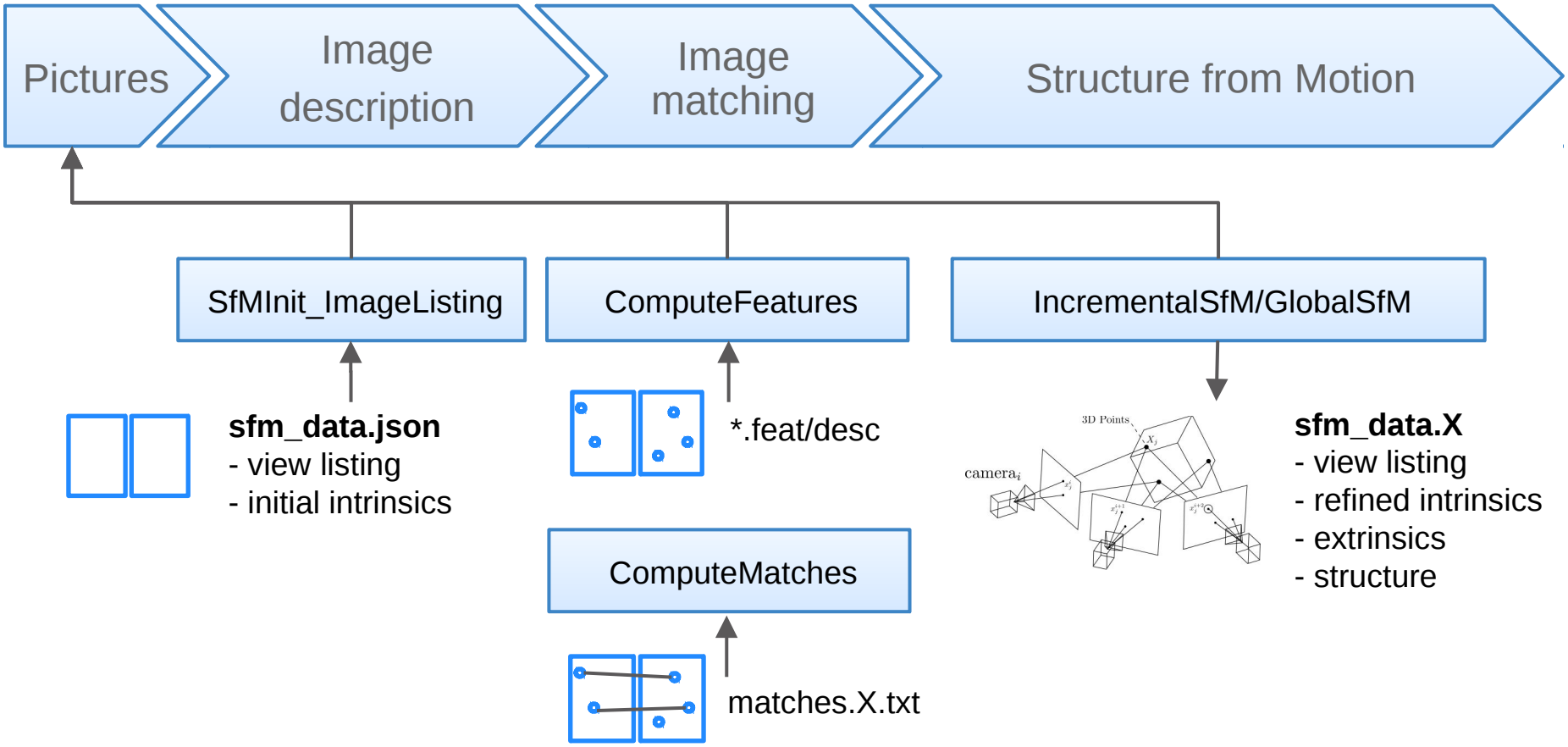
OpenMVG – SfM Pipelines

From concept to pipelines & process.



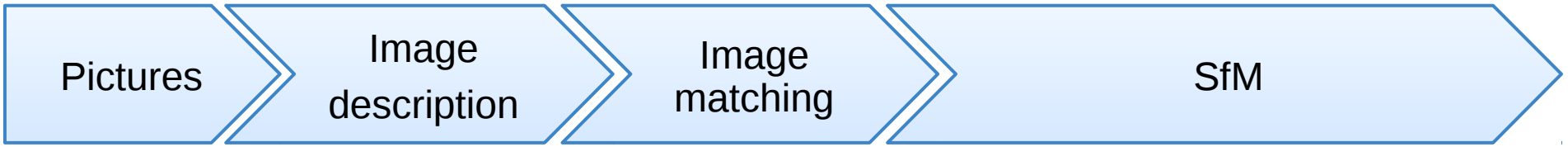
OpenMVG – SfM Pipelines

From concept to pipelines & process.



OpenMVG – SfM Pipelines

Pipeline: process, concepts and data container



To ease the link between the data, an SfM Data container is used.

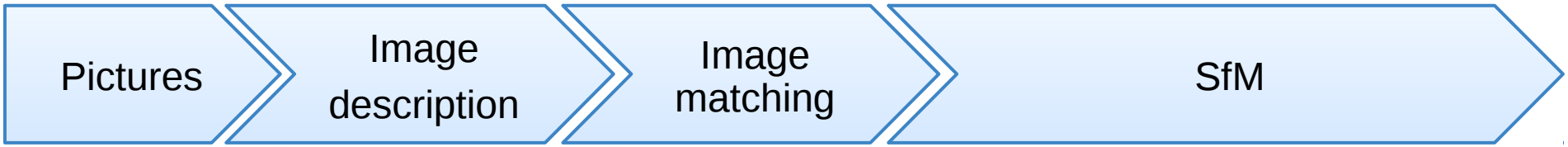
- **Establish the relationship between the parameters and the data.**

SfM_Data

```
{
    "views": [ Abstract views],
    "intrinsics": [ Abstract camera models ],
    "extrinsics": [ Camera poses ],
    "structure": [ Landmarks ]
}
```

OpenMVG – SfM Pipelines

Pipeline: process, concepts and data container



To ease the link between the data, an SfM Data container is used.

- **Establish the relationship between the parameters and the data.**

SfM_Data

{

"**views**": [Abstract views],

"**intrinsics**": [Abstract camera models],

"**extrinsics**": [Camera poses],

"**structure**": [Landmarks]

}

A scene is represented as:

A collection a views

- **image name & optional metadata attributes**

A collection of intrinsics

- **define the camera model that is used**
- **can be shared or not between views**

A collection of extrinsics

- **define the view poses**

A structure

- **define the landmarks**

(3D points with View and Feature Id observations)

Abstract data provider are used to feed the process:

- Features, Regions, Matches ...

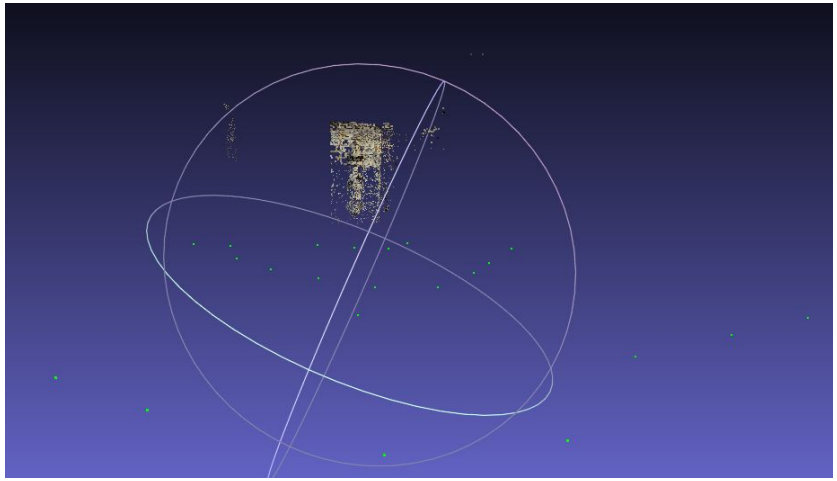
OpenMVG – SfM Pipelines

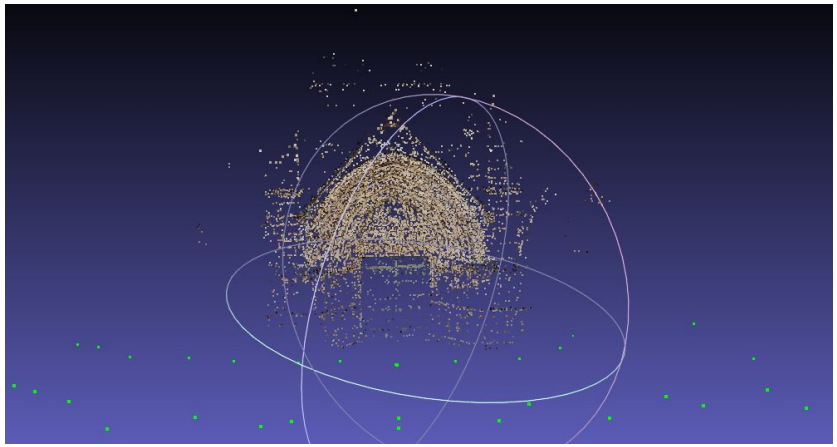
OpenMVG camera models:

- Pinhole
- Pinhole radial distortion K1
- Pinhole radial distortion K3
- Pinhole Brown (radial distortion K3 + T2)
- FishEye

OpenMVG – SfM Pipelines

Run a single python script on an image directory:



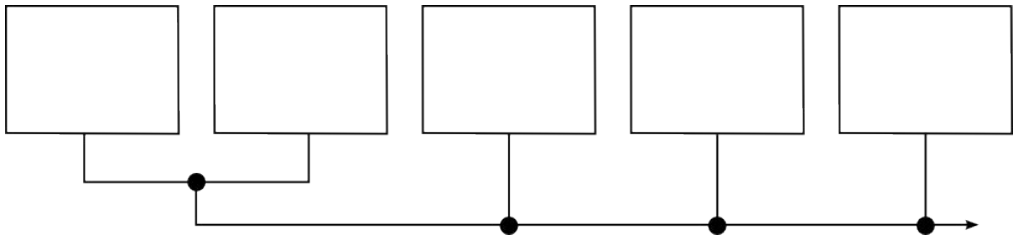


OpenMVG Hands-on

Understand the OpenMVG Structure from Motion toolchains:

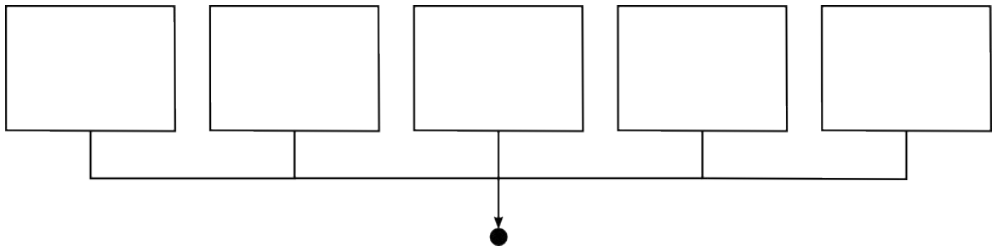
Sequential/Incremental [ACCV12]

best for unknown camera distortions and poor camera overlap



Global [ICCV13]

best for scenes with known image distortions and large camera overlap

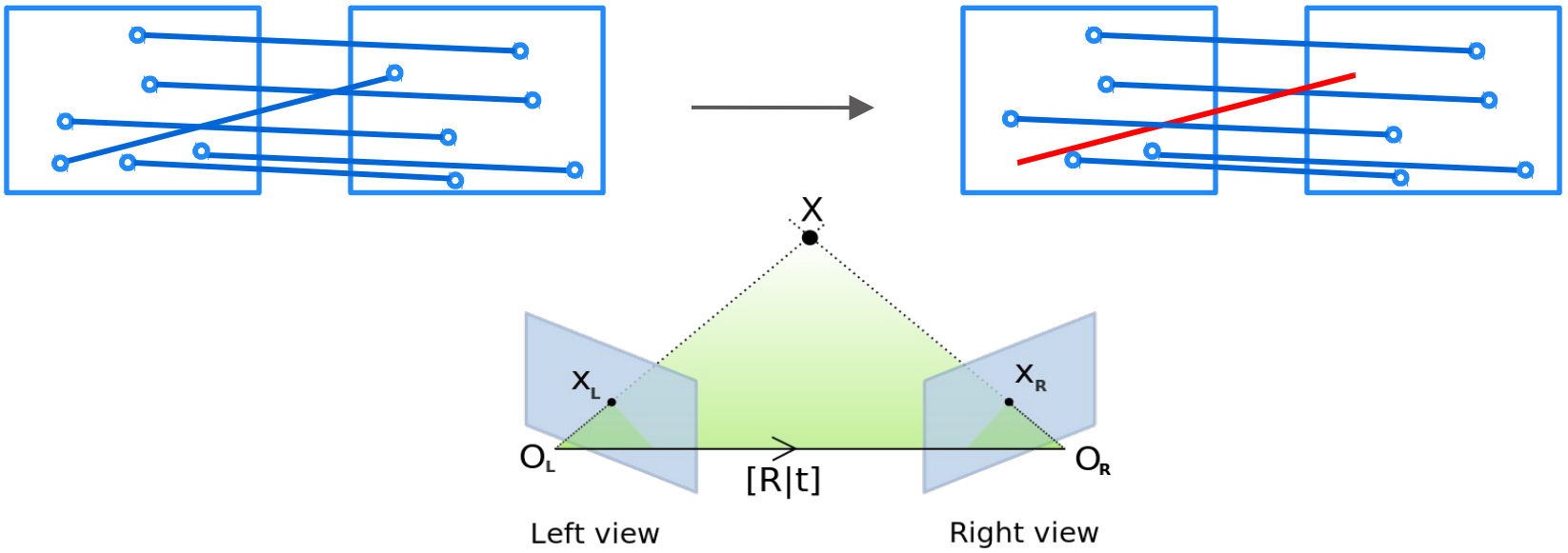


SfM - Reconstruction

The 2-view relative orientation estimation:

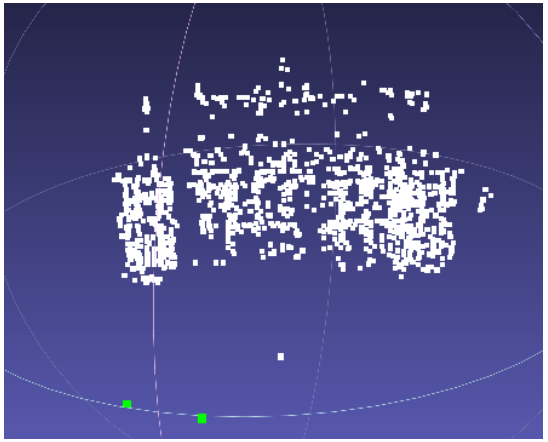
Robust estimation of a geometric model (Ransac & co.):
Essential matrix (5Pt + known intrinsics)

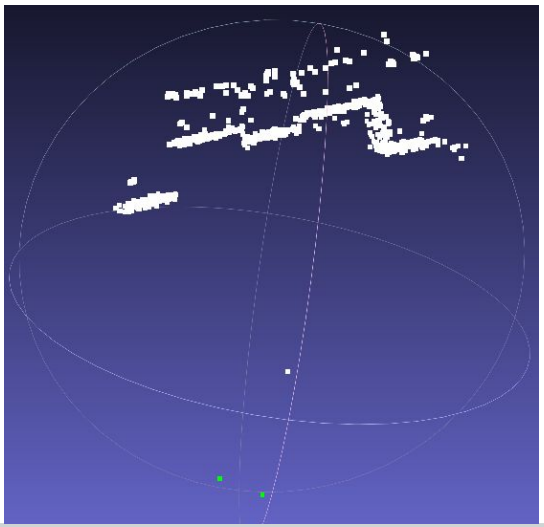
- {R,t} decomposition
  - Translation -> but scale ambiguity remaining



OpenMVG code samples

=> Compute geometrically meaningful matches:
/openMVG/src/openMVG_Samples/robust_essential/robust_essential.cpp





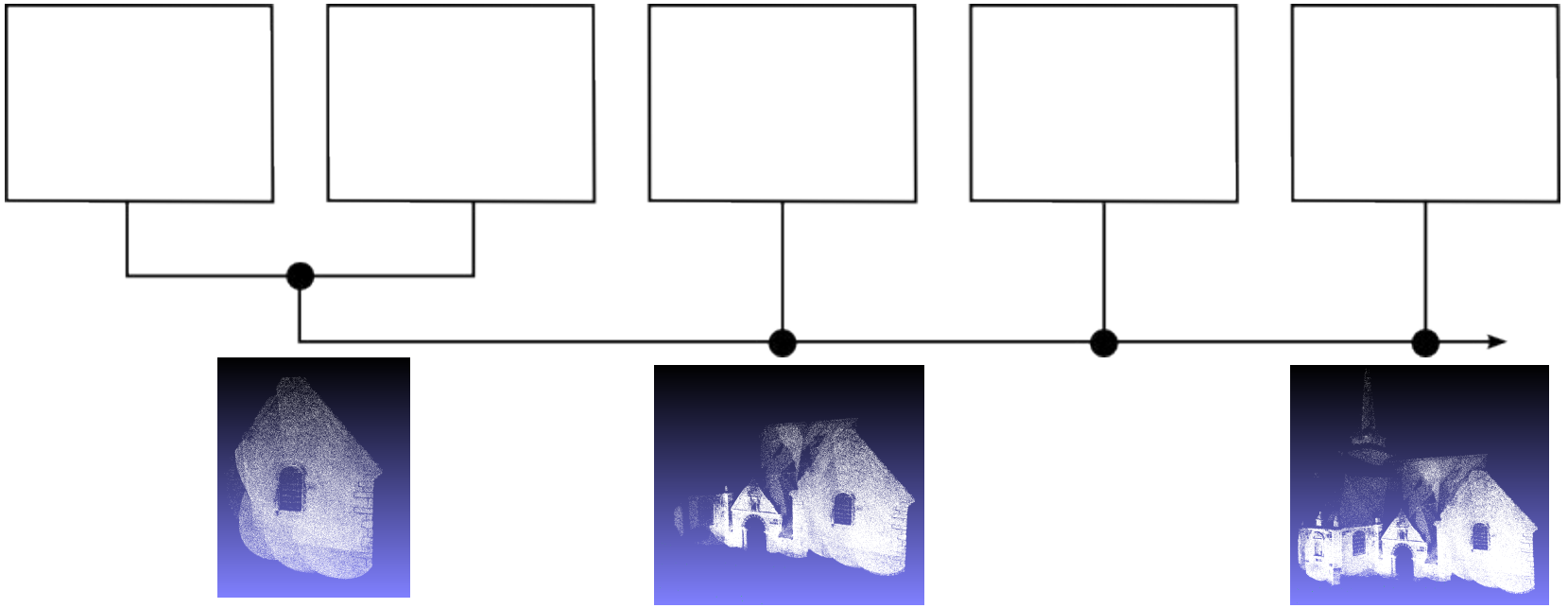
OpenMVG code samples

=> Compute geometrically meaningful matches:
/openMVG/src/openMVG_Samples/robust_essential/robust_essential.cpp

Robust estimation of an Essential matrix:

```
std::vector<IndMatch> vec_PutativeMatches; // Initialized with some photometric matches
const PointFeatures
  featsL = regions_perImage.at(0)->GetRegionsPositions(), featsR = regions_perImage.at(1)->GetRegionsPositions();

// Homography geometry filtering of putative matches

  //-- Build array with the corresponding photometric interest points
  Mat xL(2, vec_PutativeMatches.size()), xR(2, vec_PutativeMatches.size());
  for (size_t k = 0; k < vec_PutativeMatches.size(); ++k)  {
    const PointFeature imaL = featsL[vec_PutativeMatches[k].i_], imaR = featsR[vec_PutativeMatches[k].j_];
    xL.col(k) = imaL.coords().cast<double>();  xR.col(k) = imaR.coords().cast<double>();
  }

  //-- Compute the relative pose thanks to an Essential matrix
  std::pair<size_t, size_t> size_imaL(imageL.Width(), imageL.Height());
  std::pair<size_t, size_t> size_imaR(imageR.Width(), imageR.Height());
  sfm::RelativePose_Info relativePose_info;
  if (!sfm::robustRelativePose(K1, K2, xL, xR, relativePose_info, size_imaL, size_imaR))
  {
    std::cerr << " /!\\ Robust relative pose estimation failure." << std::endl;
    return EXIT_FAILURE;
  }
```

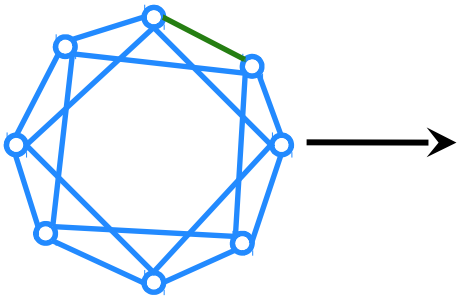
Sequential pipeline

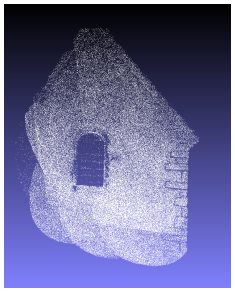


Process “one image” at a time

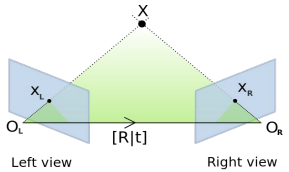
SfM – Incremental Reconstruction

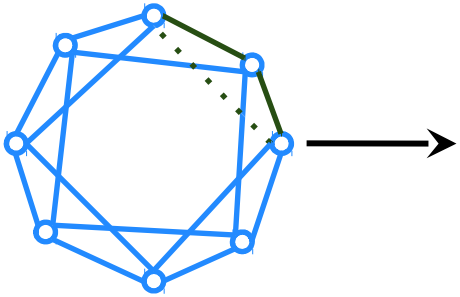
Add one image at a time by model estimation

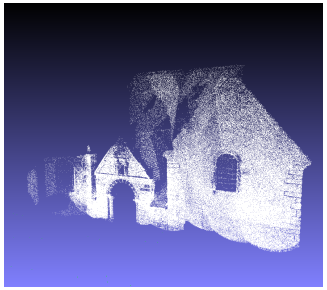




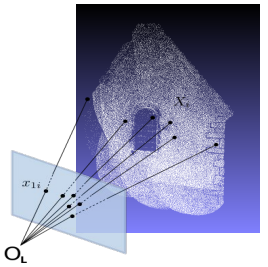
Essential matrix (2d-2d)
+
Triangulation

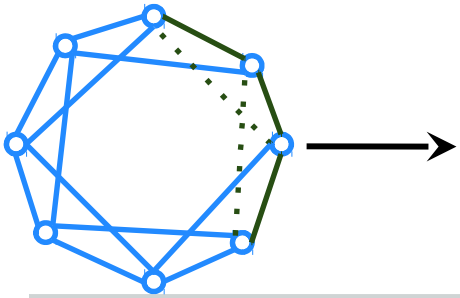


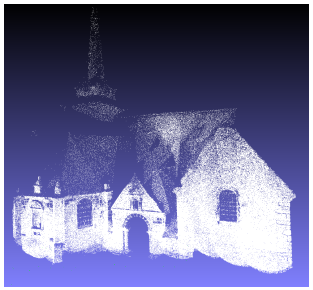




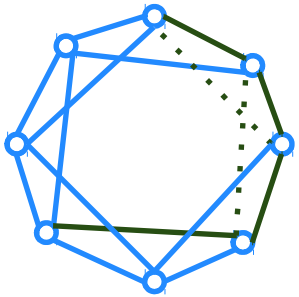
Absolute pose estimation (3d-2d)
+
Triangulation



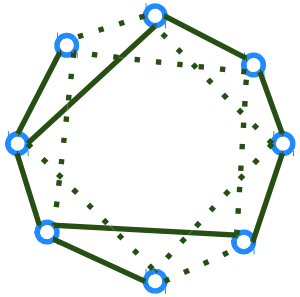




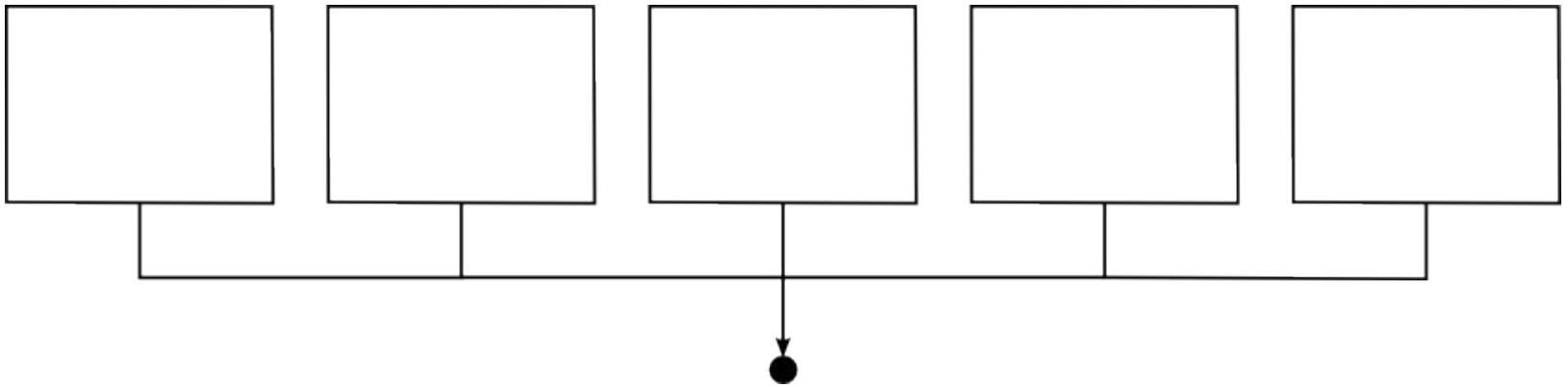
...



...



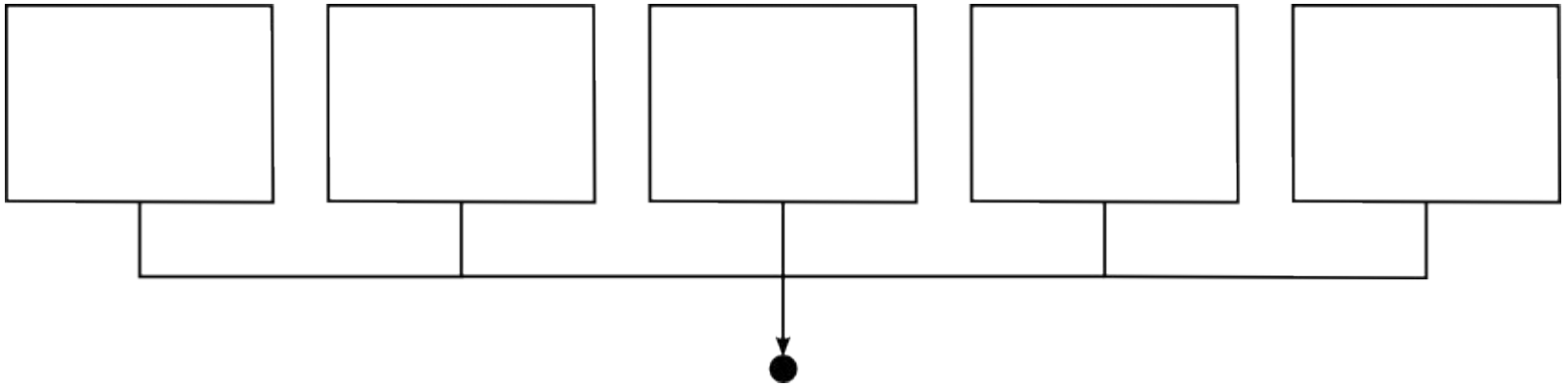
Global pipeline



All-in-one estimation:

- A motion averaging problem

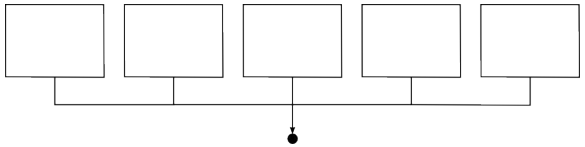
Global pipeline



A motion averaging problem:

- How to fuse relative motions:
  - Rotations => chaining or global estimation,
  - Translations => deal with the scale ambiguity.

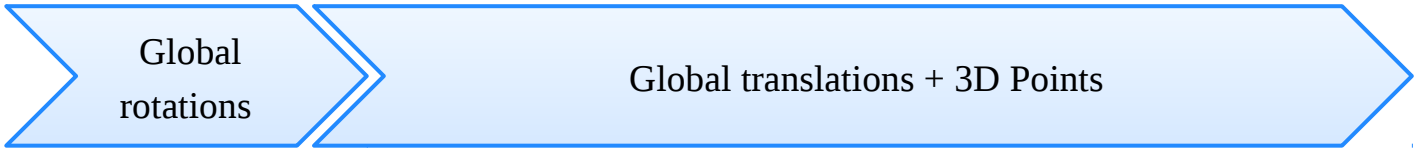
Global pipelines



Motion averaging *"relative motions to global motions"*

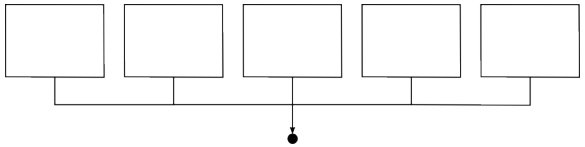
$\{R_{ij}\}$ $\{R_i\}$ $\{t_{ij}\}$ $\{\mathrm{T}_i\}$ $\{\mathrm{X}_j\}$

Structure and motion *"from known rotations"*



$\{R_i\}$ $\{\mathrm{T}_i;\mathrm{X}_j\}$

Global pipelines



Motion averaging:

Rotation Averaging:

- L2 + bundle adjustment (refinement)
- L1

Translation averaging:

- LInfinity (linear programming)
- SoftL1
- L2 (Chordal distance)

Translation & Structure computation

- LInfinity (linear programming)

Hands-on

Discover openMVG

0. SfM_Data container
1. Features computation & matching

Playing with openMVG SfM pipelines

2. Sequential SfM,
3. Global SfM.

Using a scene with known “geometry”

4. compute structure from known camera poses.

Using other scenes

5. Testing other scenes

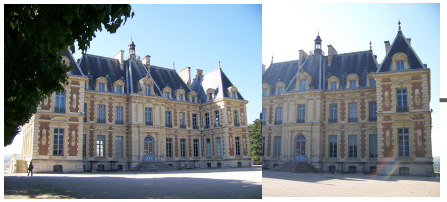
Exercice 0 => Image Listing

Image description

=> List the project images

What's new:

- Initialization of the SfM_Data container



SfMInit_ImageListing

sfm_data.json

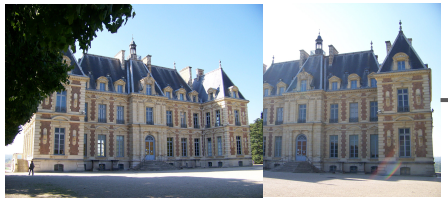
```
{
    "sfm_data_version": "0.1",
    "root_path": "...",
    "views": [],
    "intrinsics": [],
    "extrinsics": [],
    "structure": []
}
```

Exercice 0 => Image Listing

Image description

What's new:

- View, Intrinsic parameters concept.
- Parameter sharing



SfMInit_ImageListing

sfm_data.json

```
{
    "sfm_data_version": "0.1",
    "root_path": "...",
    "views": [],
    "intrinsics": [],
    "extrinsics": [],
    "structure": []
}
```

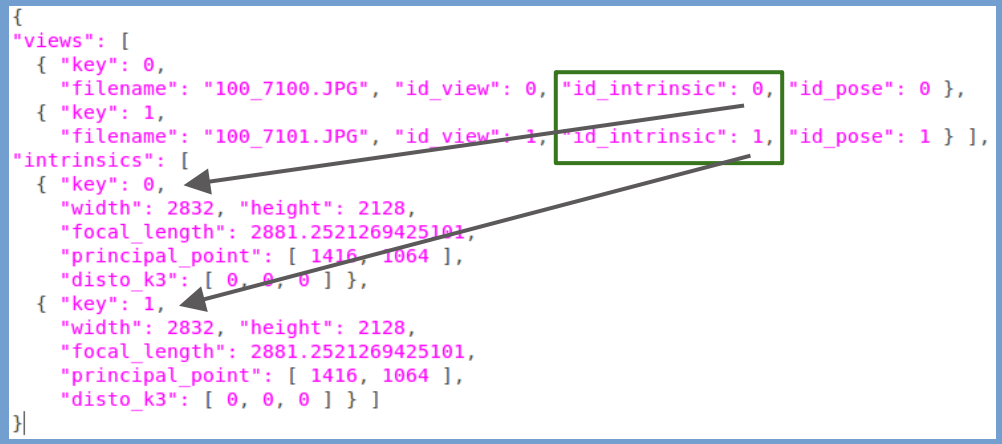
```
{
"views": [
  { "key": 0,
    "filename": "100_7100.JPG", "id_view": 0, "id_intrinsic": 0, "id_pose": 0},
  { "key": 1,
    "filename": "100_7101.JPG", "id_view": 1, "id_intrinsic": 1, "id_pose": 1}
  ]
}

{
"intrinsics": [
  { "key": 0,
    "width": 2832, "height": 2128,
    "focal_length": 2881.2521269425101,
    "principal_point": [ 1416, 1064 ],
    "disto_k3": [ 0, 0, 0 ]
  }
```

Exercice 0 => Image Listing

Image description

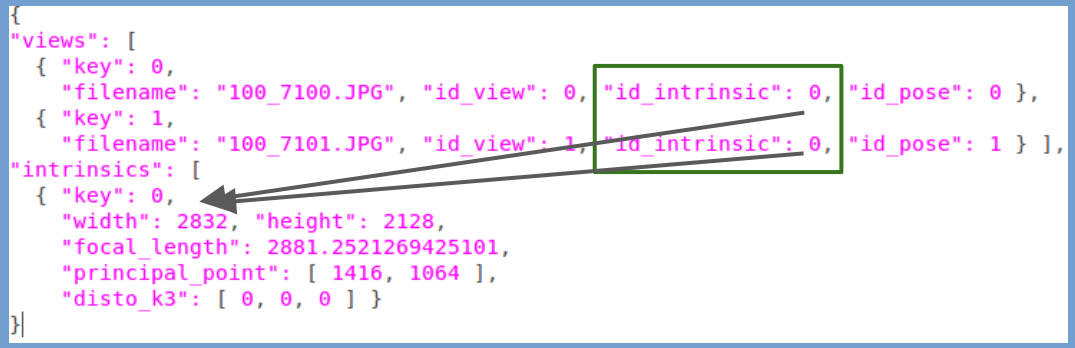
Intrinsics parameters sharing

Per view intrinsic
-g 0



Shared intrinsic
-g 1 (default)

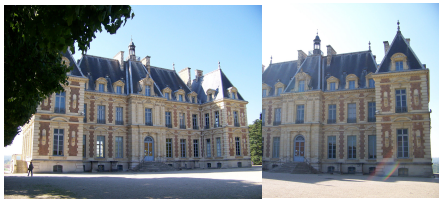
Leads to more stable parameter estimation

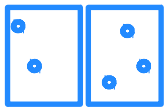


Exercice 1 => Image Matching

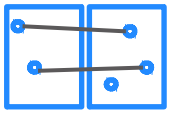
=> Describe & match the listed images

What's new: Regions, Matches

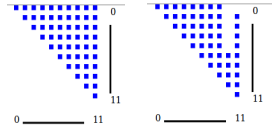




Regions:
*.feat/desc



Matches
matches.X.bin

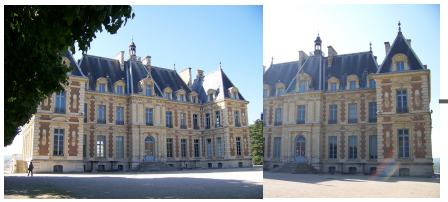


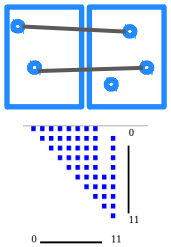
Exercice 2 => Incremental SfM

SfM

=> **Compute the camera motion & the scene**

What's new: extrinsics data and structure



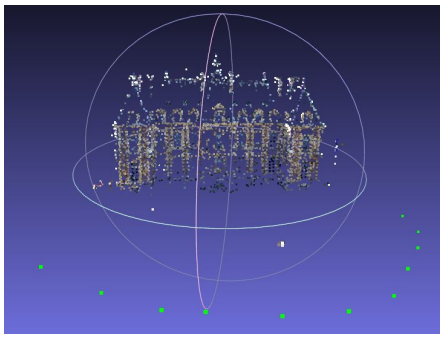


sfm_data.json
matches.f.bin
*.feat

IncrementalSfM

sfm_data.json


```
{
"intrinsics": [
  { "key" : 0
    "focal_length": 2990.8945516342478,
    "principal_point": [ 1468.5793965011931, 1107.2057853699573 ],
    "disto_k3": [ -0.24, 0.28, 0.10 ]
  } ],
"extrinsics": [
  {
    "key": 0,
    "rotation": [...],
    "center": [...]},
  {"key": 1,...},{"key": 2,...}
  ],
"structure": [
  { "key": 0,
    "X": [-0.79, -4.23, 16.62],
    "observations": [
    { "key": 0, "id_feat": 105, "x": [ 1403.43, 397.28 ] },
    { "key": 1, "id_feat": 4,   "x": [ 1280.93, 407.39 ] }
    ]
  }
  { "key": 1, ...
```

Dataset info:
#views: 11
#poses: 11
#intrinsics: 1
#tracks: 5307
#residuals: 17312

IdView	Basename	#Observations	Residuals min	Residuals median	Residuals mean	Residuals max
0	100_7100	1270	5.19211e-06	0.113467	0.214633	2.61144
1	100_7101	2253	1.05774e-05	0.10947	0.190803	2.83349
2	100_7102	2505	1.49996e-07	0.127289	0.204227	3.51394
3	100_7103	2325	3.6981e-06	0.140178	0.214793	3.56388
4	100_7104	2063	6.9085e-06	0.124538	0.20283	2.75675
5	100_7105	1706	1.30537e-06	0.145636	0.22707	1.99196
6	100_7106	1636	1.83511e-05	0.137738	0.220994	2.05416
7	100_7107	1633	2.19582e-05	0.123624	0.206945	2.28715
8	100_7108	1254	1.13327e-05	0.122054	0.208536	2.3185
9	100_7109	357	6.78977e-05	0.116653	0.22921	2.02606
10	100_7110	310	1.60072e-05	0.0964496	0.215712	2.6295

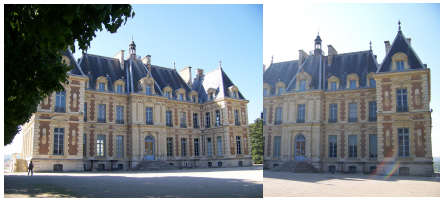
SfM Scene RMSE: 0.330875

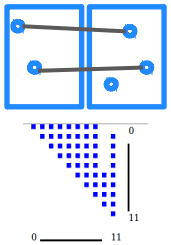
Residuals histogram

Exercice 3 => Global SfM

=> Compute the camera motion & the scene

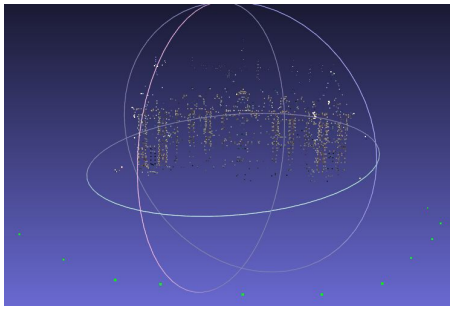
What's new: no initial pair!





sfm_data.json
matches.**e**.bin
*.feat

GlobalSfM



sfm_data.json

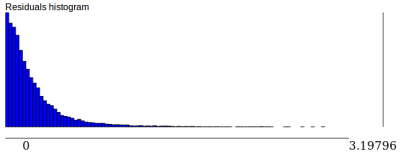

```
{
"intrinsics": [
  { "key" : 0
    "focal_length": 2990.8945516342478,
    "principal_point": [ 1468.5793965011931, 1107.2057853699573 ],
    "disto_k3": [ -0.24, 0.28, 0.10 ]
  } ],
"extrinsics": [
  {
    "key": 0,
    "rotation": [[...]],
    "center": [...]},
  {"key": 1,...},{"key": 2,...}
  ],
"structure": [
  { "key": 0,
    "X": [-0.79, -4.23, 16.62],
    "observations": [
    { "key": 0, "id_feat": 105, "x": [ 1403.43, 397.28 ] },
    { "key": 1, "id_feat": 4,   "x": [ 1280.93, 407.39 ] }
    ]
  }
  { "key": 1, ...
```


Dataset info:
#views: 11
#poses: 11
#intrinsics: 1
#tracks: 1981
#residuals: 8938

IdView	Basename	#Observations	Residuals min	Residuals median	Residuals mean	Residuals max
0	100_7100	589	1.80465e-05	0.170341	0.28987	2.8514
1	100_7101	1088	9.41839e-05	0.142339	0.226296	2.95198
2	100_7102	1350	0.000262609	0.166518	0.237866	2.76662
3	100_7103	1313	0.000112157	0.173754	0.24846	2.29333
4	100_7104	1292	1.0707e-05	0.145177	0.225768	2.47298
5	100_7105	1013	0.000202532	0.184856	0.264398	2.39161
6	100_7106	986	0.000542114	0.168681	0.253159	2.33173
7	100_7107	682	7.53749e-05	0.173086	0.270169	3.19796
8	100_7108	525	0.000335978	0.152994	0.256719	2.45962
9	100_7109	68	0.000709635	0.148987	0.271583	2.36739
10	100_7110	32	0.00198444	0.139737	0.223845	1.32775

SfM Scene RMSE: 0.370092

Residuals histogram



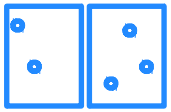
Exercice 4 => Structure [Known poses]

SfM

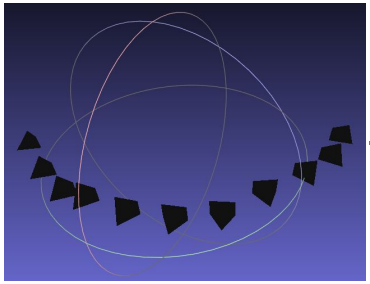
=> Compute the scene structure

What's new: guided matching, triangulation

ComputeFeatures



.feat/.desc



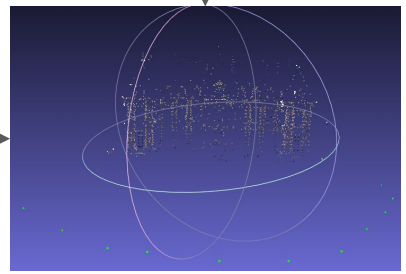
ComputeStructureFromKnownPoses

sfm_data.json

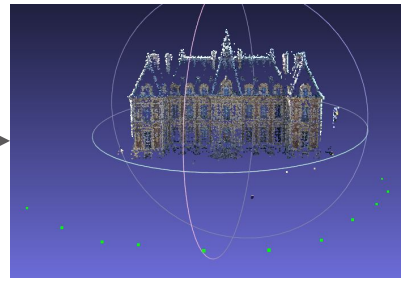
sfm_data.json


```
{
"intrinsics": [
  { "key" : 0
    "focal_length": 2990.8945516342478,
    "principal_point": [ 1468.5793965011931,
    "disto_k3": [ -0.24, 0.28, 0.10 ]
  } ],
"extrinsics": [
  {
    "key": 0,
    "rotation": [...],
    "center": [...]},
  {"key": 1,...},{"key": 2,...}
  ],
```


Sift features + normal preset

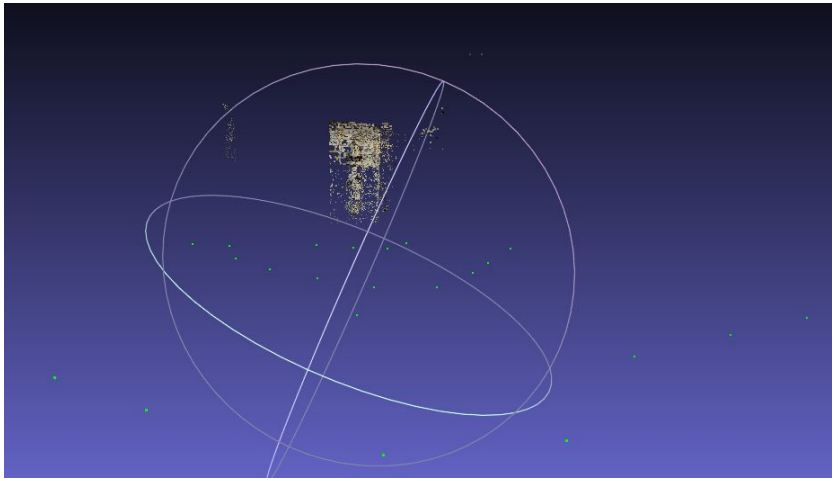


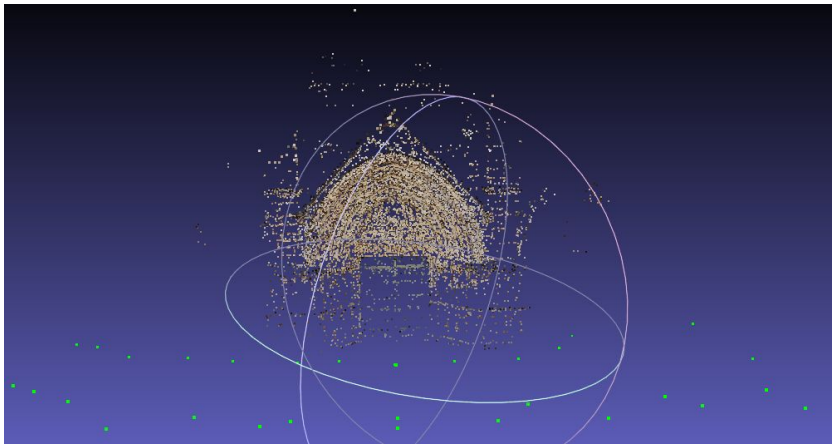
AKAZE features + normal preset



Exercice 5 => Other datasets

=> Global SfM on other datasets





Exercice 6 => OpenMVG as a 3rd party lib.

$ cd 6_codingWithOpenMVG

=> Use an installed OpenMVG as a 3rd party library

```
FIND_PACKAGE(OpenMVG REQUIRED)

INCLUDE_DIRECTORIES(${OPENMVG_INCLUDE_DIRS})

ADD_EXECUTABLE(main main.cpp)

TARGET_LINK_LIBRARIES(main ${OPENMVG_LIBRARIES})
```

Tip:

To use a local install:
cmake ... -DOpenMVG_DIR:STRING="path.../openMVG_Build/install/share/openMVG/cmake/"

Exercice 6 => OpenMVG as a 3rd party lib.

Load an existing scene:

```
  //-----------------------------------
  // Read SfM Scene (image view names)
  //-----------------------------------
  SfM_Data sfm_data;
  if (!Load(sfm_data, sSfM_Data_Filename, ESfM_Data(ALL))) {
    std::cerr << std::endl
      << "The input SfM_Data file \""<< sSfM_Data_Filename << "\" cannot be read." <<
std::endl;
    return EXIT_FAILURE;
  }

  // Display some stats about the loaded data:
  std::cout
    << "Loaded a sfm_data scene with:\n"
    << " #views: " << sfm_data.GetViews().size() << "\n"
    << " #poses: " << sfm_data.GetPoses().size() << "\n"
    << " #intrinsics: " << sfm_data.GetIntrinsics().size() <<  "\n"
    << " #tracks: " << sfm_data.GetLandmarks().size()
    << std::endl;
```

Exercice 6 => OpenMVG as a 3rd party lib.

Computing re-projection errors (residual values):

```
using namespace openMVG;
// For all tracks
for (Landmarks::const_iterator iterTracks = sfm_data.GetLandmarks().begin();
  iterTracks != sfm_data.GetLandmarks().end();  ++iterTracks  )
{
  // For all observations
  const Observations & obs = iterTracks->second.obs;
  const Vec3 X =  iterTracks->second.X; // The landmark 3D position
  for (Observations::const_iterator itObs = obs.begin();  itObs != obs.end(); ++itObs)
  {
    const View * view = sfm_data.GetViews().at(itObs->first).get();
    const geometry::Pose3 pose = sfm_data.GetPoseOrDie(view);
    const cameras::IntrinsicBase * intrinsic = sfm_data.GetIntrinsics().at(view->id_intrinsic).get();
    // Compute the reprojection error
    const Vec2 residual = intrinsic->residual(pose, X , itObs->second.x);
    std::cout << residual.norm() << " ";
  }
  std::cout << std::endl;
}
```

Questions ?

- All in one command line
  - python SfM_SequentialPipeline.py imageDir ResultDir
  - python SfM_GlobalPipeline.py imageDir ResultDir
  - Tip:
    - handle more complex scene => Feature preset NORMAL/HIGH/ULTRA
    - Pre-emptive distortion handling
- Rigid registration
  - GCP Gound Control Point registration
- Future features
  - Geodesy: Spatial Reference System rigid registration (GPS/ECEF)
  - ...

Exercice 7

SfM => Localize an image in an existing scene