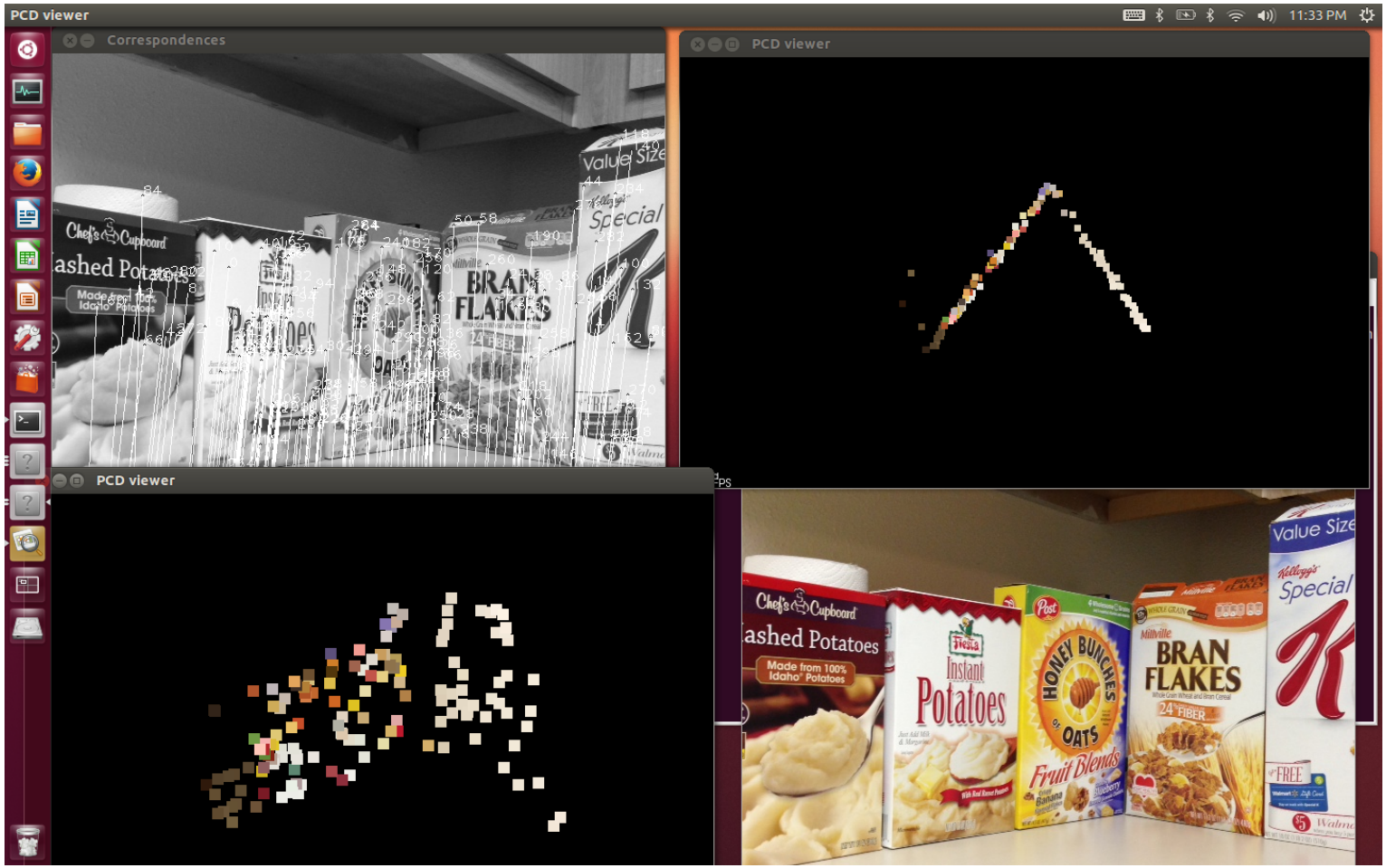
Computer Vision

3D reconstruction from motion

Lioulemes Alexandros

3D reconstruction from motion



Project Pipelines

Implemented

- Captured images of the same scene from the same camera in different position.
- Extraction of Keypoints using the SURF feature detector.
- Estimation of Fundamental and Essential Matrix.
- Find Rotation and Translation matrix from the decomposition of the Essential Matrix.
- Find 3D points cloud from the Linear Triangulation method .
- Display the 3D points in the world using (PCL) Point Cloud Libraty.

Not Implemented

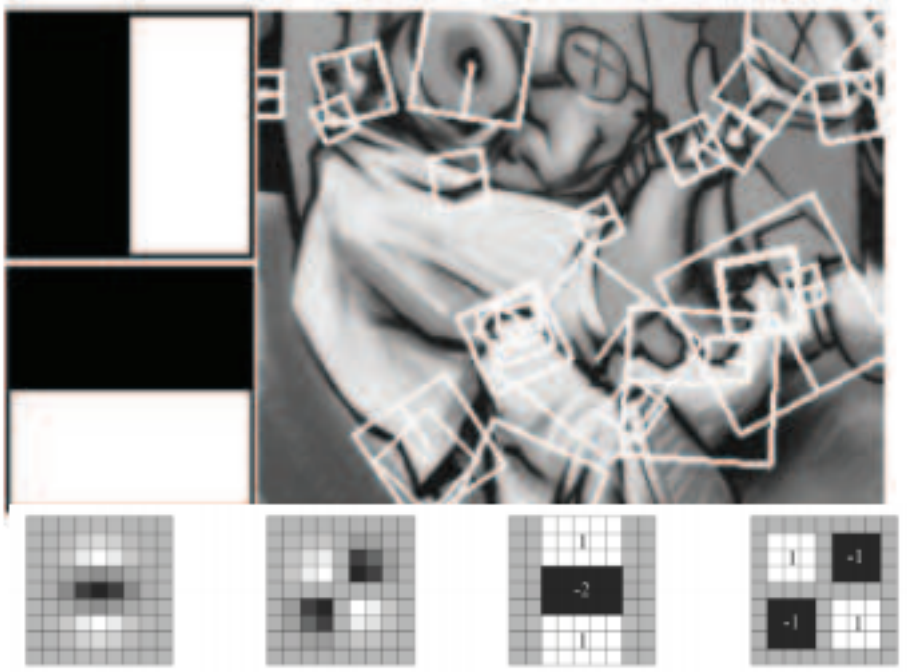
- Combine the 3D point cloud data from the different camera's positions.

SURF features

- It uses an integer approximation to the determinant of Hessian blob detector, which can be computed extremely quickly with an integral image.
- SURF is based on sums of 2D Haar wavelet responses and makes an efficient use of integral images.
- In order to extract four descriptor values for each subregion:

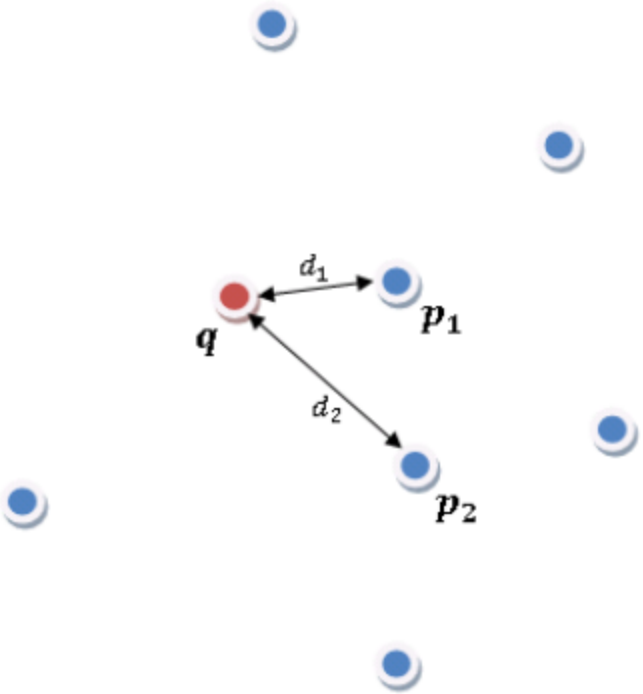
$$\left[\sum dx \quad \sum dy \quad \sum |dx| \quad \sum |dy|\right]$$

- Since there are 4 × 4 = 16 sub-regions, we have a total of 64 descriptor values.



Feature Matching

- Extract all the descriptors in the search image.
- Compute the distance between $\mathbf{d}_q$ and each descriptor in the search image.
- Extract the first closest descriptor in the search image $\mathbf{d}_1$
- Extract the second closest descriptor in the search image $\mathbf{d}_2$



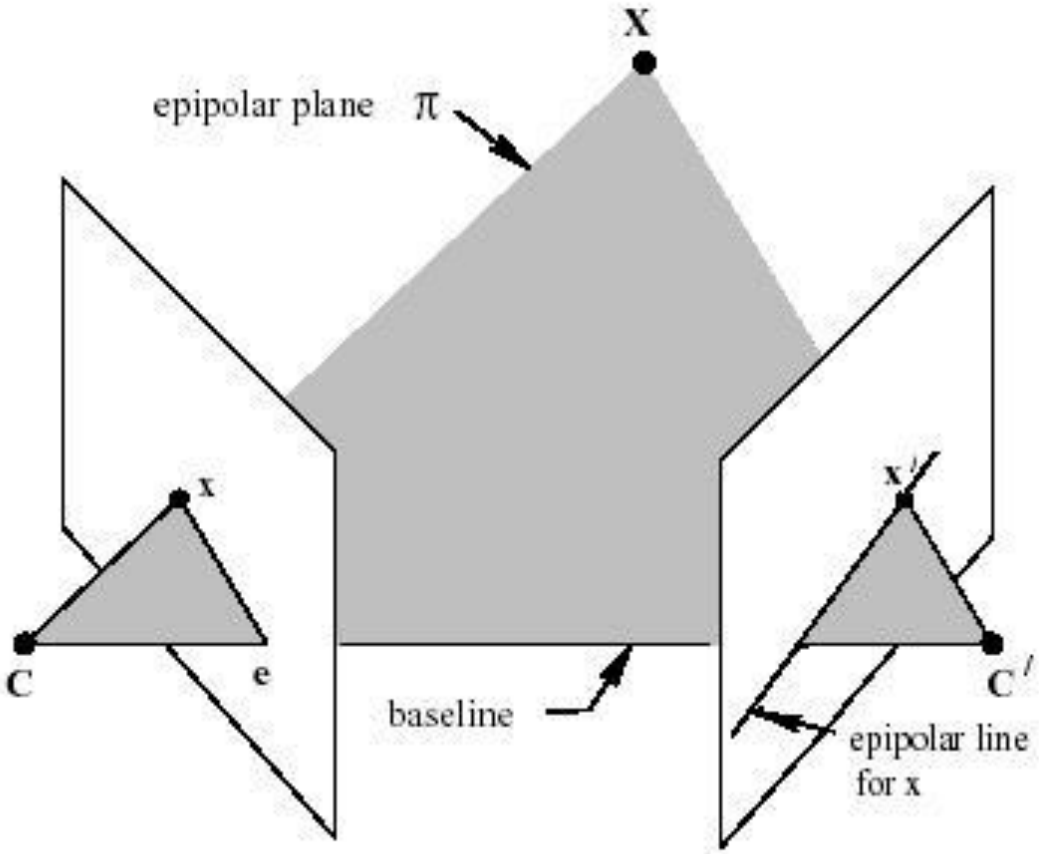
Brute force matching

```
static void findPairs(const CvSeq* objectKeypoints, const CvSeq* objectDescriptors, const CvSeq* imageKeypoints, const CvSeq* imageDescriptors, vector<int>& ptpairs ){

CvSeqReader reader, kreader;
cvStartReadSeq( objectKeypoints, &kreader );
cvStartReadSeq( objectDescriptors, &reader );
ptpairs.clear();
for(int  i = 0; i < objectDescriptors->total; i++ )
{
    const CvSURFPoint* kp = (const CvSURFPoint*)kreader.ptr;
    const float* descriptor = (const float*)reader.ptr;
    CV_NEXT_SEQ_ELEM( kreader.seq->elem_size, kreader );
    CV_NEXT_SEQ_ELEM( reader.seq->elem_size, reader );
    int nearest_neighbor = naiveNearestNeighbor( descriptor, kp->laplacian, imageKeypoints, imageDescriptors );
     if( nearest_neighbor >= 0 )
     {
        ptpairs.push_back(i);
        ptpairs.push_back(nearest_neighbor);
     }
}
}
```

Correspondence points

Epipolar Geometry



Fundamental Matrix

$$\mathbf{x}'^{\top}\mathbf{F}\mathbf{x} = 0.$$

rank(F)=2

$$\mathbf{K} = \begin{bmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$(\mathbf{y}')^{\top}\,\mathbf{E}\,\mathbf{y} = 0$$

$$\mathbf{E} = \mathbf{K}'^{\top}\,\mathbf{F}\,\mathbf{K}$$

$$\mathbf{E} = \mathbf{R}\,[\mathbf{t}]_{\times}$$

F =
[4.0418792e-07, 2.6479636e-06, -0.0013159743;
2.4267231e-06, 3.855399e-06, -0.0076914625;
-0.00033768022, 0.0035015468, 1]

y0'*F*x0 = +0.0004918
y1'*F*x1 = +0.0024464
y2'*F*x2 = +0.0006018
y3'*F*x3 = +0.0033642
y4'*F*x4 = +0.0009363
y5'*F*x5 = +0.0002915
y6'*F*x6 = +0.0019591
y7'*F*x7 = +0.0000496
y8'*F*x8 = +0.0006706
y9'*F*x9 = +0.0000027

Decompose Essential

$$\mathbf{E} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^T \qquad \det \mathbf{E} = 0 \qquad \mathbf{\Sigma} = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

```
if (determinant(E) > 1e-07)
{
    cout << "Essential Matrix constraint does not apply" << endl;
    return 0;
}
```

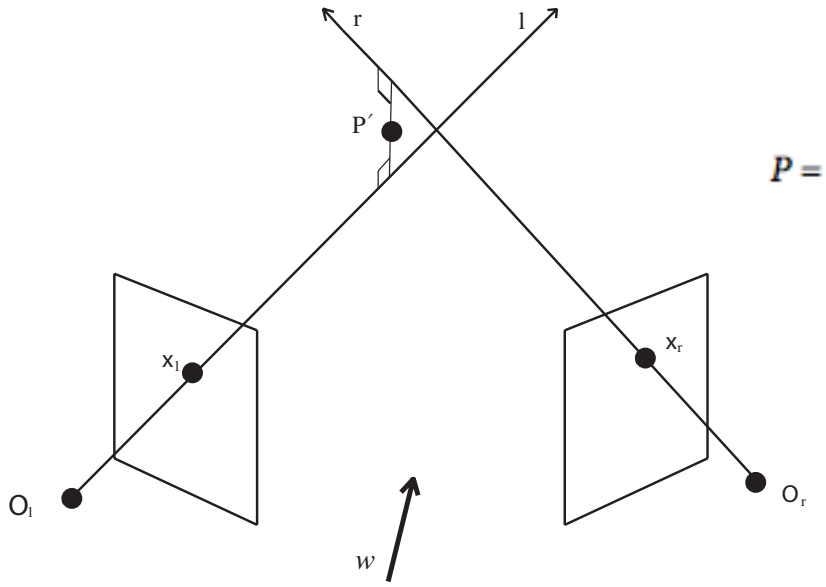
$$\mathbf{W} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{W}^{-1} = \mathbf{W}^T = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R} = \mathbf{U}\,\mathbf{W}^{-1}\,\mathbf{V}^T \qquad \det(\mathbf{R}) = \pm 1$$

$$[\mathbf{t}]_\times = \mathbf{V}\,\mathbf{W}\,\mathbf{\Sigma}\,\mathbf{V}^T$$

Triangulation

$$P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



$$P = [R|t] = \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix}$$

$x_l = P_0 X$

$x_r = PX$

$AX=B$

Least Square Solution: $X = (\mathbf{A}^T\mathbf{A})^{-1}\,\mathbf{A}^T\mathbf{B}$

Triangulation

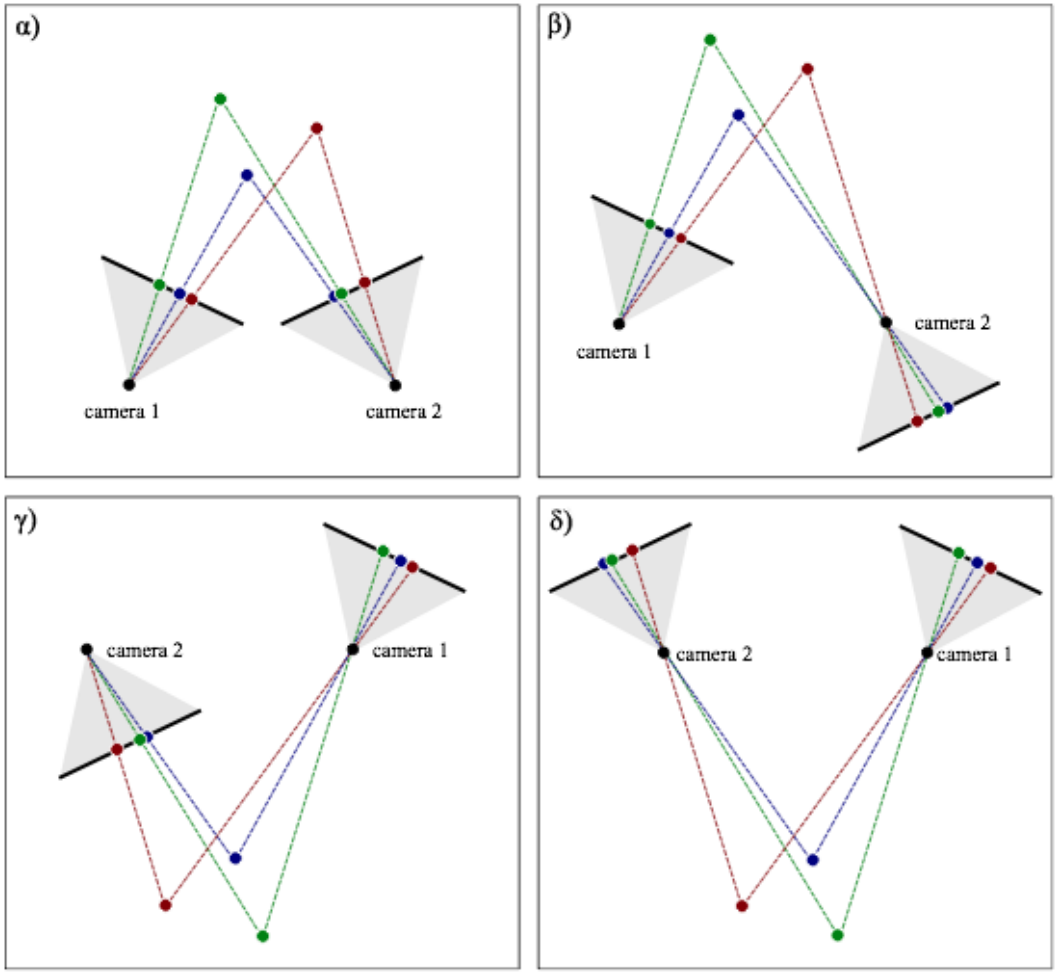
```
Mat_<double> LinearLSTriangulation(Point3d u,Matx34d P,Point3d u1,Matx34d P1)
{
  //build matrix A
  Matx43d A      (u.x*P(2,0)-P(0,0),    u.x*P(2,1)-P(0,1),          u.x*P(2,2)-P(0,2),
                          u.y*P(2,0)-P(1,0),    u.y*P(2,1)-P(1,1),          u.y*P(2,2)-P(1,2),
                u1.x*P1(2,0)-P1(0,0), u1.x*P1(2,1)-P1(0,1), u1.x*P1(2,2)-P1(0,2),
                u1.y*P1(2,0)-P1(1,0), u1.y*P1(2,1)-P1(1,1), u1.y*P1(2,2)-P1(1,2));

   //build B vector
  Matx41d B      (-(u.x*P(2,3)    -P(0,3)),
                -(u.y*P(2,3)    -P(1,3)),
                -(u1.x*P1(2,3)       -P1(0,3)),
                -(u1.y*P1(2,3)       -P1(1,3)));

   //solve for X
  Mat_<double> X;
  solve(A,B,X,DECOMP_SVD);

  return X;
}
```

4-ambiguous



- Among the 4 possible solutions, keep the one with the **3-D reconstructed points** in front of the two camera

Point Cloud Library

```
# .PCD v.7 - Point Cloud Data file format
VERSION .7
FIELDS x y z rgb
SIZE 4 4 4 1
TYPE F F F I
COUNT 1 1 1 4
WIDTH 145
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 145
DATA ascii
1.24405   -0.856152   1.97062   37    26   122    1
1.19385   -1.1466     1.89827   184  199   254    1
……
……
```

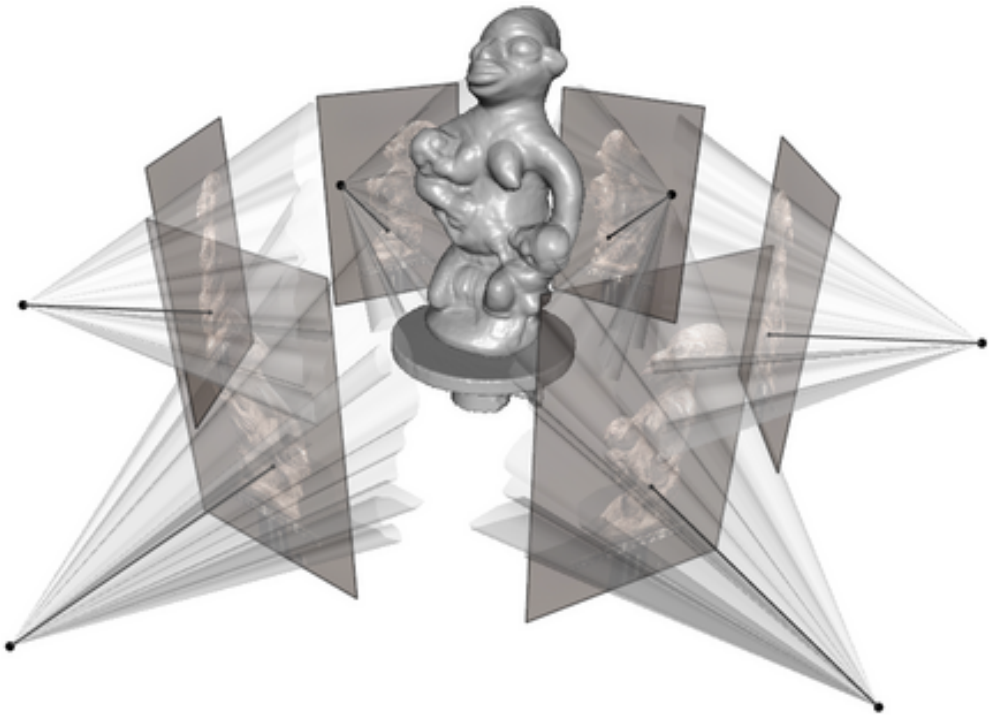
Build-in OpenCV component

- cvSURFParams(...);
- cvExtractSURF(…);
- findFundamentalMat(…)
- projectPoints(…);

Developed components

- findPairs(…);
- naiveNearestNeighbor(…);
- compareSURFDescriptors(…);
- decomposeEssential(…);
- LinearLSTriangulation(…);
- Reprojection_error(…);
- findRGB(…);
- WritePCDfile(…);

Multiple Cameras



Thank you