Dimitri Charneux

Rendu tp:

Mise en correspondance stéréoscopique.

Introduction

Durant ce TP, nous allons voir comment estimer la troisième dimension de chaque point d'une image par triangulation à partir de paires de points homologues dans deux images différentes. Nous allons pour cela devoir calculer la matrice fondamentale du stéréoscope, détecter les pixels d'intérêts des deux images à l'aide de l'opérateur de Shi et Tomasi, calculer les distances entre les points de chaque paire puis enfin mettre en correspondance les paires de points homologues.

1. Calcul de la matrice fondamentale

La matrice fondamentale F est une matrice permettant de calculer la droite épipolaire d'un point de l'espace π1 dans l'espace π2.

La droite épipolaire d'un point est la projection de ce point dans un autre plan projectif comme le montre l'image ci-dessous. Cette droite va nous permettre par la suite de calculer la distance entre une paire de points.

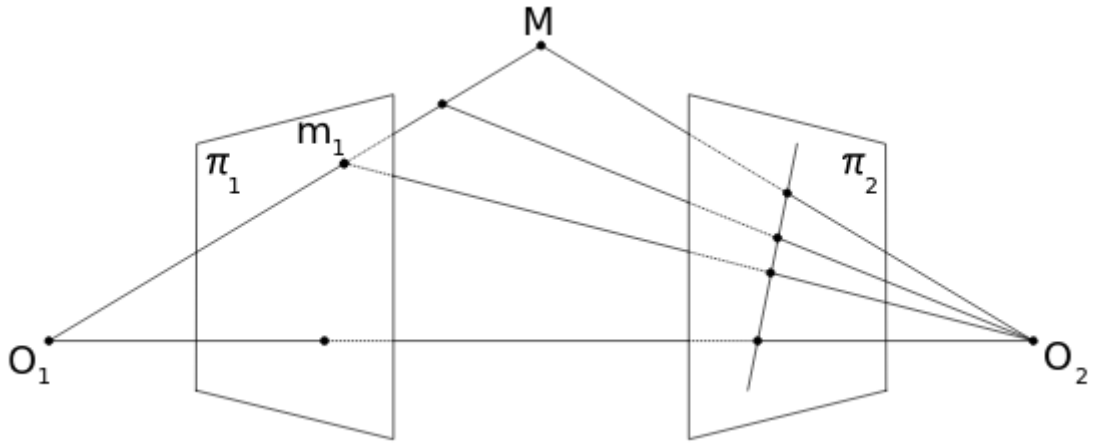

Illustration 1: Droite épipolaire d'un point m1 dans le plan π2.

Pour calculer cette matrice, nous avons d'abord eu besoin d'écrire une méthode pour déterminer une matrice de transformation linéaire équivalente à un produit vectoriel qui est considéré comme la matrice d'homographie suivante :

$$\mathbf{p}^{\times} = \begin{pmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{pmatrix}$$

Illustration 2: Matrice de transformation linéaire équivalente à un produit vectoriel.

Le code de la fonction ***iviVectorProductMatrix*** réalisant cette transformation est fourni en annexe.

Nous avons ensuite calculé la matrice fondamentale en utilisant la formule $\mathbf{F} = (\mathbf{P}_2\mathbf{O}_1)^{\times}\mathbf{P}_2\mathbf{P}_1^{+}$ où P2O1 est la projection de O1 sur le plan π2. Ce calcule a été effectué par la fonction suivantes :

```
Mat iviFundamentalMatrix(const Mat& mLeftIntrinsic,
                         const Mat& mLeftExtrinsic,
                         const Mat& mRightIntrinsic,
                         const Mat& mRightExtrinsic) {
     Mat mFundamental = Mat::eye(3, 3, CV_64F);
     Mat r = (Mat_<double>(3,4) <<
          1.0, 0.0, 0.0, 0.0,
          0.0, 1.0, 0.0, 0.0,
          0.0, 0.0, 1.0, 0.0
          );
     Mat p1 = mLeftIntrinsic * r * mLeftExtrinsic;
     Mat p2 = mRightIntrinsic * r * mRightExtrinsic;
     Mat iE1 = mLeftExtrinsic.inv();
     Mat o1 = iE1.col(3);
     mFundamental = iviVectorProductMatrix(p2 * o1) * p2 * p1.inv(DECOMP_SVD);
     return mFundamental;
}
```

Par exemple, l'équation de la droite épipolaire de l'image droite associé au centre de l'image gauche est : d2 = Fm1 tandis que celle de la droite épipolaire de l'image gauche associé au centre de l'image droite est : d1 = F^tm2.

Cette matrice va nous servir plus tard pour calculer les droites épipolaires de chaque point ainsi que la distance entre une paire de points.

Maintenant que nous avons vu comment calculer la matrice fondamentale et à quoi elle sert, nous allons voir comment extraire les points d'intérêts, appelés ***coins***, dans les deux images.

2. Extraction des coins

Pour pouvoir donner de la profondeur à notre image, il faut d'abord trouver des points d'intérêt dans les deux images afin de pouvoir les mettre en correspondance.

Pour ce faire, nous avons utilisé un opérateur proposé par Jianbo Shi et Carlo Tomasi qui permet de trouver des points situés à l'intersection de deux frontières rectilignes.

Cet opérateur étant déjà implémenté dans openCV par la fonction nommé ***goodFeaturesToTrack***, nous avons juste du écrire la fonction suivante qui extrait les **coins** et les insère dans une matrice :

```
Mat iviDetectCorners(const Mat& mImage,int iMaxCorners) {
	vector<Point> coins;
	//0.01 et 10 sont des valeurs utilisés dans la méthode de calcul.
	goodFeaturesToTrack(mImage, coins, iMaxCorners, 0.01, 10);
	Mat mCorners(3, coins.size(), CV_64F);
	for (int i = 0; i < coins.size(); i++){
		mCorners.at<double>(0,i) = (double)coins[i].x;
		mCorners.at<double>(1,i) = (double)coins[i].y;
		mCorners.at<double>(2,i) = 1.;
	}
	return mCorners;
}
```

Nous pouvons voir dans l'image ci-dessous que les **coins**, représentés par les cercles rouges, ont bien été calculés et sont présents à chaque intersection.

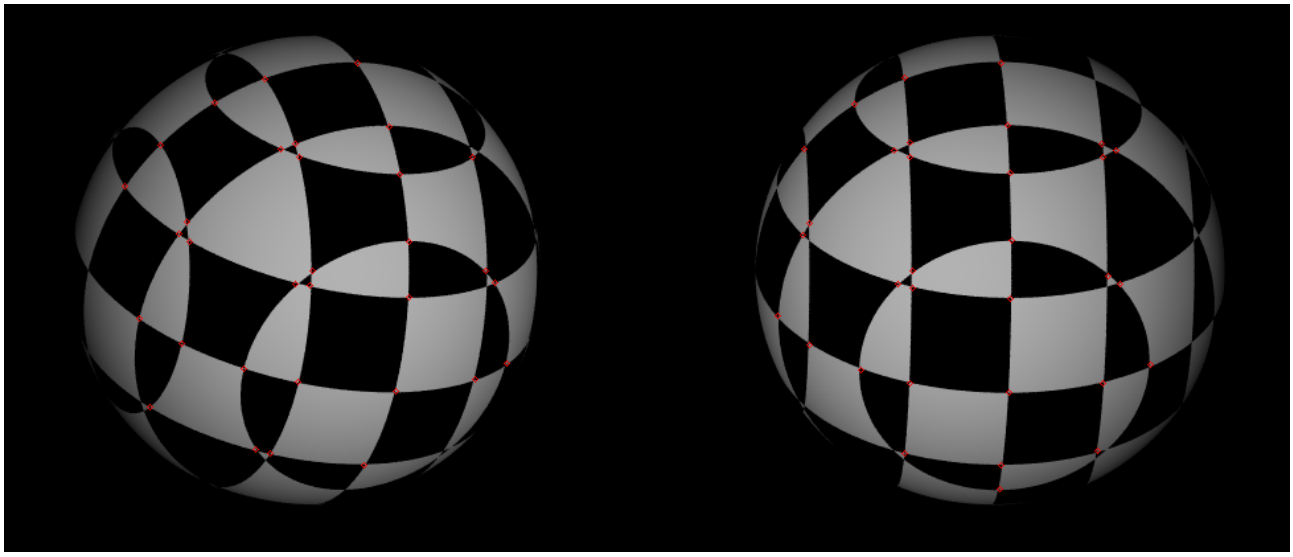

Illustration 3: Coins calculés grâce à l'opérateur de Shi et Tomasi.

Nous allons ensuite calculer les droites épipolaires de chaque **coin** en multipliant leurs coordonnées par la matrice fondamentale précédemment trouvée. Nous pouvons voir les droites épipolaires tracées dans l'image suivantes :

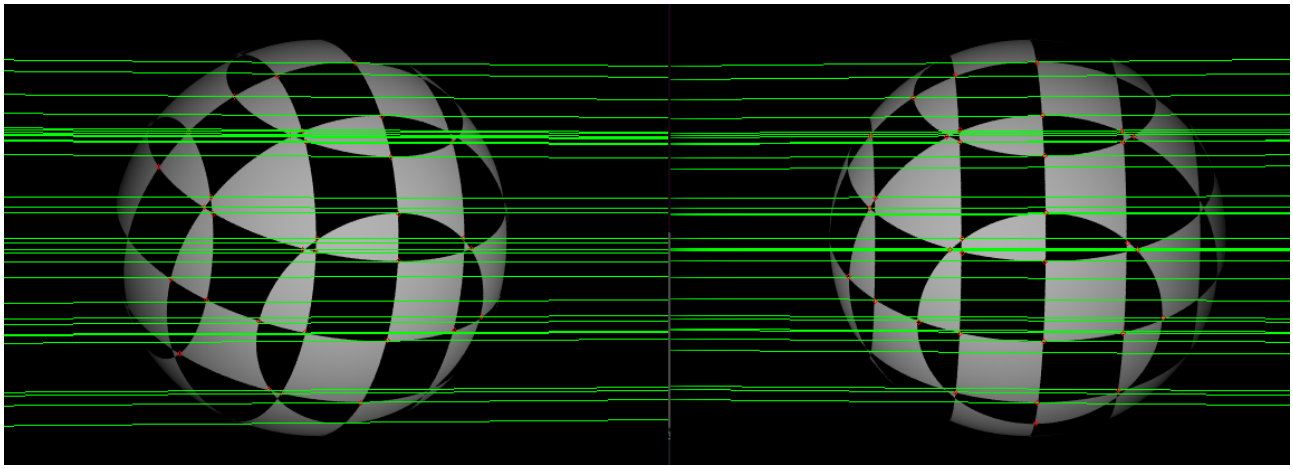

Illustration 4: Droites épipolaires tracées sur les deux images.

Maintenant que nous avons extrait les coins de nos deux images et calculé leur droite épipolaire, nous allons calculer les distances entre chaque paire de ces points.

3. Calcul des distances et mise en correspondance

Pour trouver la meilleur correspondance entre les points des images gauche et droite, nous avons calculé les distances entre les points de chaque paire en incluant la transformation homographique définie par la matrice fondamentale.

Pour ce faire, nous avons écrit une fonction ***iviDistancesMatrix*** qui calcules ces distances à partir de deux distances euclidienne :

- la distance entre le point de l'image de gauche et la droite épipolaire de l'image gauche associé au point de l'image droite.
- la distance entre le point de l'image droite et la droite épipolaire associé au point de l'image gauche.

Le code de cette fonction est fourni en annexe.

Ensuite, il faut mettre en correspondance les points en décidant si deux points sont suffisamment similaires pour être associés. Pour décider ceci, nous utiliserons un seuil qui défini la distance maximum entre deux points associés.

Par manque de temps, je n'ai pas pu réaliser cette mise en correspondance.

Conclusion

Durant ce TP, nous avons appris à calculer la profondeur d'un points grâce à deux images de ce point. Cette technique peut être utilisé dans de nombreux domaine, par exemple dans le cinéma pour donner une impression de profondeur à un film sans avoir besoin de matériel très sophistiqué.

Annexe

Code de la fonction ***iviVectorProductMatrix*** :

```
Mat iviVectorProductMatrix(const Mat& v) {
	Mat mVectorProduct = Mat::eye(3, 3, CV_64F);
	mVectorProduct.at<double>(Point(0,0)) = 0;
	mVectorProduct.at<double>(Point(0,1)) = -v.at<double>(Point(0,2));
	mVectorProduct.at<double>(Point(0,2)) = v.at<double>(Point(0,1));

	mVectorProduct.at<double>(Point(1,0)) = v.at<double>(Point(0,2));
	mVectorProduct.at<double>(Point(1,1)) = 0;
	mVectorProduct.at<double>(Point(1,2)) = -v.at<double>(Point(0,0));

	mVectorProduct.at<double>(Point(2,0)) = -v.at<double>(Point(0,1));
	mVectorProduct.at<double>(Point(2,1)) = v.at<double>(Point(0,0));
	mVectorProduct.at<double>(Point(2,2)) = 0;
	return mVectorProduct;
}
```

Code de la fonction ***iviDistancesMatrix*** :

```
Mat iviDistancesMatrix( const Mat& m2DLeftCorners,
                        const Mat& m2DRightCorners,
                        const Mat& mFundamental) {

      Mat mDistances = Mat(m2DLeftCorners.cols, m2DRightCorners.cols, CV_64F);
      for(int i=0; i<m2DLeftCorners.cols; i++){
            for(int j=0; j<m2DRightCorners.cols; j++){
                  Mat m1 = m2DLeftCorners.col(i);
                  Mat m2 = m2DRightCorners.col(j);
                  Mat d2 = mFundamental * m1;
                  Mat d1 = mFundamental.t() * m2;

                  double dividende = fabs(d2.at<double>(0,0)*m1.at<double>(0,0)
+ d2.at<double>(1,0)*m1.at<double>(1,0) + d2.at<double>(2,0));

                  double diviseur = sqrt(d2.at<double>(0,0)*d2.at<double>(0,0) +
d2.at<double>(1,0)*d2.at<double>(1,0));

                  double dist1 = dividende / diviseur;

                  dividende = fabs(d1.at<double>(0,0)*m2.at<double>(0,0) +
d1.at<double>(1,0)*m2.at<double>(1,0) + d1.at<double>(2,0));

                  diviseur = sqrt(d1.at<double>(0,0)*d1.at<double>(0,0) +
d1.at<double>(1,0)*d1.at<double>(1,0));

                  double dist2 = dividende / diviseur;
                  mDistances.at<double>(i,j) = dist1 + dist2;
            }
      }
      return mDistances;
}
```