Réalité augmenté

Projet tuteuré de 1 mars 2016 – 30 mars 2016

Semestre 3, DUT Département Informatique

Bohao LI

Tuteur : Pierre-Fréderic Villard

Année 2015 – 2016

Table des matières :

Compte rendu PT4

Bohao LI

I. Introduction :

1. Qu'est-ce que la réalité augmentée ?

La réalité augmentée désigne les systèmes informatiques qui rendent possible la superposition d'un modèle virtuel 2D ou 3D à la perception que nous avons naturellement de la réalité et ceci en temps réel.

Ex : On prend la photo d'une salle vide, et à partir de cette photo, en utilisant la technologie de la réalité augmentée, on est capable d'ajouter des chaises, un tapis, ou n'importe quel autre objet sur la photo (c'est-à-dire dans la chambre), et ceci nous donne la perception de la salle avec des objets qu'on y est ajoutés, c'est bien un mélange du monde réel et le monde virtuel, autrement dit, on a impression que le monde réel est « renforcé ».

2. Qu'est-ce que la bibliothèque PoLAR ?

PoLAR (portable Library for Augmented Reality == librairie portable pour la réalité augmentée).

C'est un framwork qui existe pour nous aider à créer des applications pour la réalité augmentée, visualisation d'image et des images médicales. Le framwork nous offert les outils nécessaires pour créer des applications graphiques. Ce framwork est écrit en langage c++ et a été publiée sous License GNU GPL.

3. Quel est objectif du projet ?

L'objectif du projet, est d'utiliser la librairie PoLAR pour mettre en œuvre un projet de la réalité augmentée, pour nous, cette fois ci, le projet consiste à mettre des objets virtuels dans une salle (une photo qu'on a prise préalablement). Cela nous donnera la perception de la salle avec des objets qu'on y ajoutera.

II. Installation de PoLAR

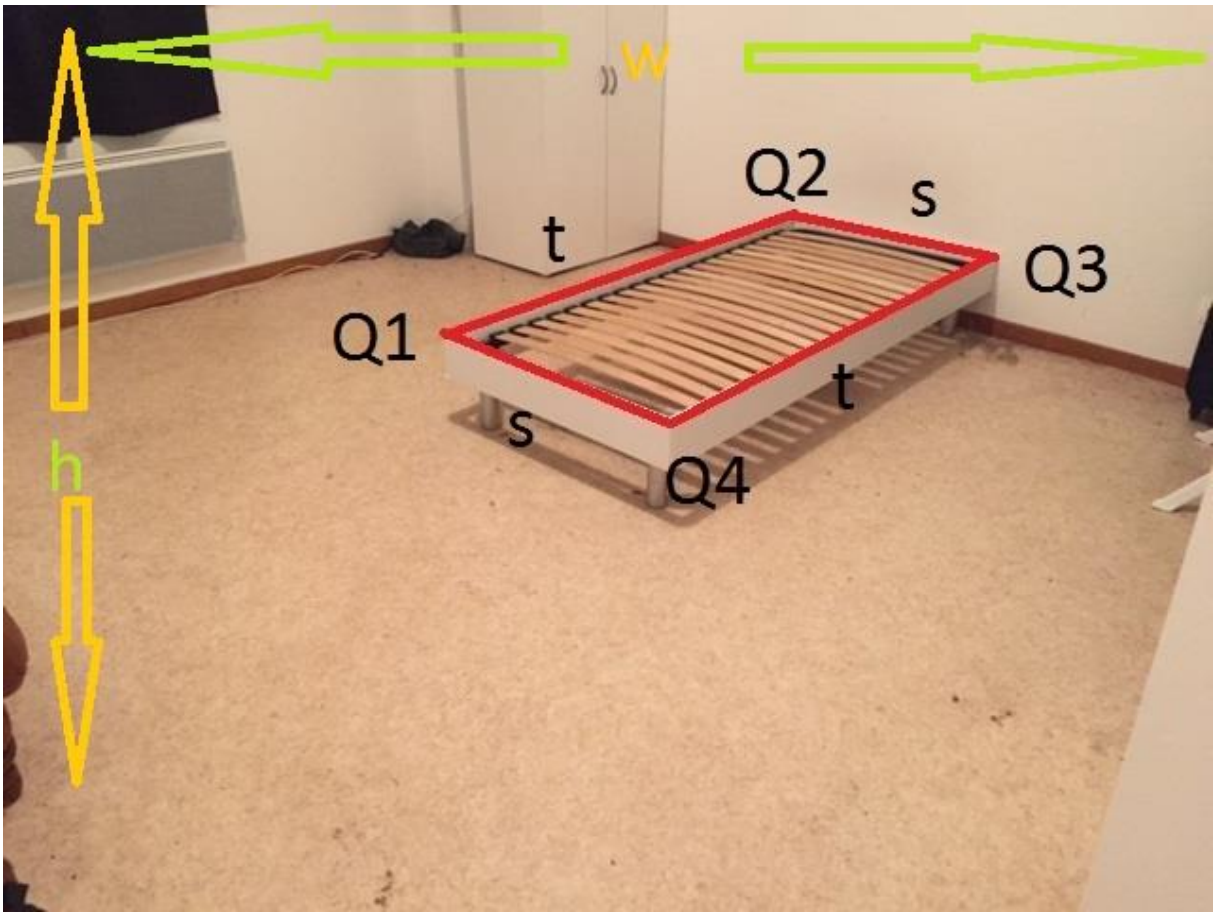
Il faut suivre les indications sur le site du PoLAR pour installer PoLAR sur linux :

- *sudo apt-get install freeglut3 freeglut3-dev*
- *sudo apt-get install qt5-default qttools5-dev-tools libqt5opengl5-dev qtmultimedia5-dev libqt5multimedia5-plugins qtdeclarative5-dev libqt5svg5-dev*
- *sudo apt-get install openscenegraph libopenscenegraph-dev*
- *sudo apt-get install libgstreamer0.10-0 libgstreamer0.10-dev gstreamer-tools*
- *sudo apt-get install doxygen graphviz*
- *ccmake*
- *sudo apt-get install cmake-curses-gui*
- *git clone https://scm.gforge.inria.fr/anonscm/git/polar/polar.gitgit clone https://scm.gforge.inria.fr/anonscm/git/polar/polar.git*

- *git clone https://scm.gforge.inria.fr/anonscm/git/polar/polar.git*
- *sudo apt-get install git*
- *git clone https://scm.gforge.inria.fr/anonscm/git/polar/polar.git*
- *cd polar*
- *sudo apt-get build-dep openscenegraph*
- *./polar_config*
- *ccmake .*
- *make*
- *./bin/svisu Examples/data/example1.ppm*
- *./bin/testpick Examples/data/example1.ppm Examples/data/example1.proj Examples/data/armchair.obj*

III. Calcul de la matrice de projection

D'abord, on prend une photo d'une scène qu'on connaît :



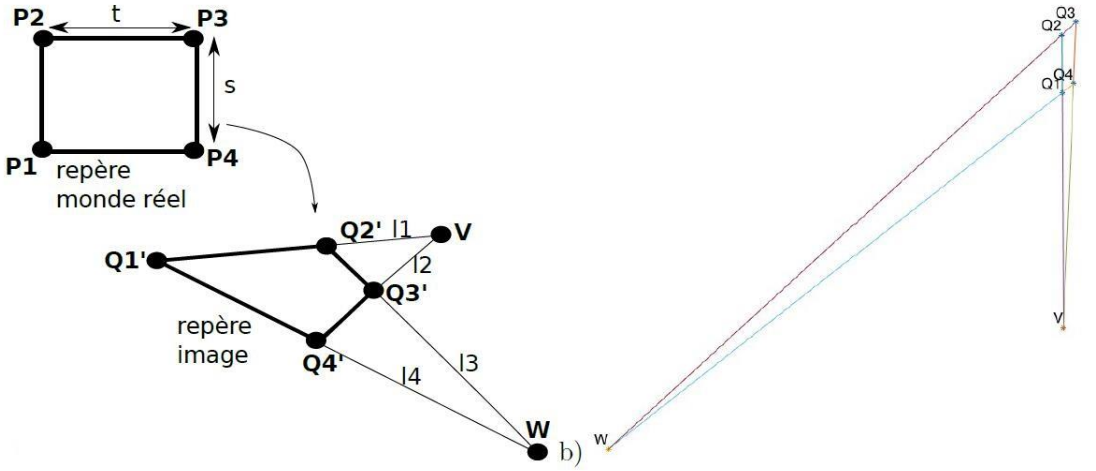
Par exemple, sur la photo, le lit est choisi pour construire un repère. Q1, Q2, Q3, Q4 sont les quatre points d'un rectangle (du lit). t et s sont la longueur et la largeur du lit. De plus, on précise la hauteur (représentée par h, en anglais height), et la largeur (représentée par w, en anglais width).

(ps : l'unité de w et h est pixel, nous pouvons utiliser un logiciel (par exemple paint sur Windows) pour le savoir)

Ensuite, il s'agit de faire des calculs suivants :

$$Q1' = \begin{bmatrix} Q1_x - w/2 \\ Q1_y - h2 \\ 1 \end{bmatrix}; Q2' = \begin{bmatrix} Q2_x - w/2 \\ Q2_y - h2 \\ 1 \end{bmatrix}; Q3' = \begin{bmatrix} Q3_x - w/2 \\ Q3_y - h2 \\ 1 \end{bmatrix}; Q4' = \begin{bmatrix} Q4_x - w/2 \\ Q4_y - h2 \\ 1 \end{bmatrix}$$

$$l1 = \begin{bmatrix} Q1'_x \\ Q1'_y \\ 1 \end{bmatrix} \otimes \begin{bmatrix} Q2'_x \\ Q2'_y \\ 1 \end{bmatrix}; l2 = \begin{bmatrix} Q4'_x \\ Q4'_y \\ 1 \end{bmatrix} \otimes \begin{bmatrix} Q3'_x \\ Q3'_y \\ 1 \end{bmatrix}; l3 = \begin{bmatrix} Q2'_x \\ Q2'_y \\ 1 \end{bmatrix} \otimes \begin{bmatrix} Q3'_x \\ Q3'_y \\ 1 \end{bmatrix}; l4 = \begin{bmatrix} Q1'_x \\ Q1'_y \\ 1 \end{bmatrix} \otimes \begin{bmatrix} Q4'_x \\ Q4'_y \\ 1 \end{bmatrix}$$



$$V^H = l1 \otimes l2; W^H = l3 \otimes l4; V = \begin{bmatrix} V_x^H / V_z^H \\ V_y^H / V_z^H \\ 1 \end{bmatrix}; W = \begin{bmatrix} W_x^H / W_z^H \\ W_y^H / W_z^H \\ 1 \end{bmatrix}$$

$$f = \sqrt{-(V_x.W_x + V_y.W_y)}$$

$$K = \begin{bmatrix} f & 0 & w/2 \\ 0 & f & h/2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & -Q2'_x & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & -Q2'_y & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & -Q3'_x & -Q3'_x \\ 0 & 0 & 0 & 1 & 1 & 1 & -Q3'_y & -Q3'_y \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & -Q4'_x \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & -Q4'_y \end{bmatrix}; Qi' = \begin{bmatrix} Q1'_x \\ Q1'_y \\ Q2'_x \\ Q2'_y \\ Q3'_x \\ Q3'_y \\ Q4'_x \\ Q4'_y \end{bmatrix}$$

$$\begin{bmatrix} h11 \\ h12 \\ h13 \\ h21 \\ h22 \\ h23 \\ h31 \\ h32 \end{bmatrix} => \begin{bmatrix} h11 & h12 & h13 \\ h21 & h22 & h23 \\ h31 & h32 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 1/f & 0 & 0 \\ 0 & 1/f & 0 \\ 0 & 0 & 1 \end{bmatrix} .h = \begin{bmatrix} \vdots & \vdots & \vdots \\ B1 & B2 & B3 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

La matrice de projection est **M=K.E** avec :

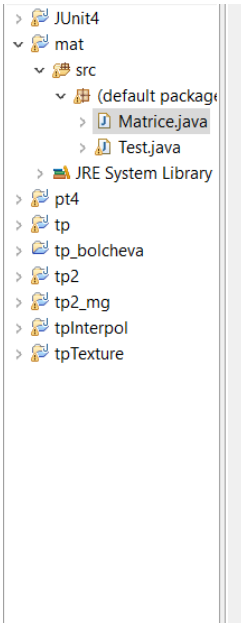
$$E = \begin{bmatrix} \vdots & \vdots & \vdots & \\ R1 & R2 & R3 & T \\ \vdots & \vdots & \vdots & \end{bmatrix}$$

avec les valeurs suivantes :

$$\begin{array}{lcl} R1 & = & B1/\|B1\| \\ R3 & = & R1 \otimes B2/(\|R1 \otimes B2\|) \\ R2 & = & R3 \otimes R1 \\ \lambda & = & \|B1\|/s \\ t_{calcule} & = & \|B2\|/\lambda \\ T & = & B3/\lambda \end{array}$$

Pour calculer la matrice de projection, nous pouvons utiliser n'importe lequel langage de programmation, par exemple, ici java est utilisée. Il faut d'abord trouver une classe matrice sur internet qui permet de faire tous les calculs concernant les matrices (calculer son inverse, sa transposé, la multiplication entre deux matrice……) après, il s'agit de réaliser les calculs expliqué dans le sujet.

Voici en bas quelques captures d'écran :

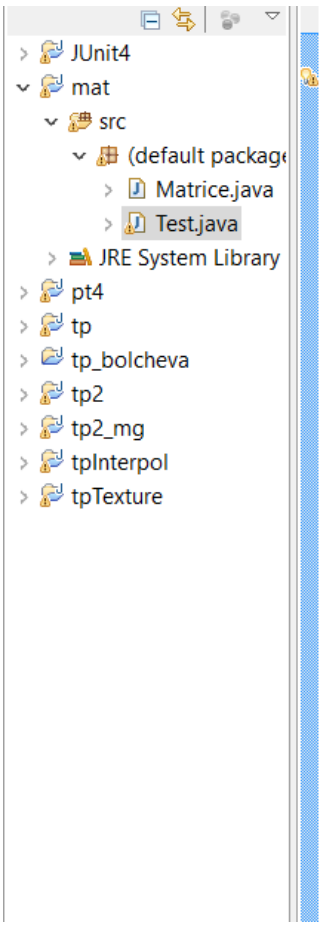



```java
public class Matrice
{
    private double[][] coeff = null;
    private int diametre = 0;
    private int distance = 0;

    //-------------------------------------------------//
    //                    CONSTRUCTOR                  //
    //-------------------------------------------------//
    /** Constructeur Matrice
      * @param
      * int i - ligne
      * int j - colonne
      */
    public Matrice(int i, int j)
    {
        this.setLength(i,j);
    }

    public Matrice()
    {
        this(0,0);
    }

    public Matrice(double[][] mat)
    {
        this.coeff = mat;
```




```java
public class Test {
    public static void main(String[] args) {
    double t = 2.05; //en metres
    double s = 0.95; //en metres
    double w = 640; //en pixel
    double h = 480; //en pixel
    double q1x = 236;
    double q1y = 174;
    double q2x = 419;
    double q2y = 112;
    double q3x = 527;
    double q3y = 129;
    double q4x = 352;
    double q4y = 219;
    double[][] tab1 = {{q1x-w/2},{q1y-h/2},{1}};
    double[][] tab2 = {{q2x-w/2},{q2y-h/2},{1}};
    double[][] tab3 = {{q3x-w/2},{q3y-h/2},{1}};
    double[][] tab4 = {{q4x-w/2},{q4y-h/2},{1}};
    Matrice q1prime = new Matrice(tab1);
    Matrice q2prime = new Matrice(tab2);
    Matrice q3prime = new Matrice(tab3);
    Matrice q4prime = new Matrice(tab4);

    double q1primex = q1prime.getCoeff()[0][0];
    double q1primey = q1prime.getCoeff()[1][0];
    double q2primex = q2prime.getCoeff()[0][0];
    double q2primey = q2prime.getCoeff()[1][0];
    double q3primex = q3prime.getCoeff()[0][0];
    double q3primey = q3prime.getCoeff()[1][0];
    double q4primex = q4prime.getCoeff()[0][0];
    double q4primey = q4prime.getCoeff()[1][0];
```

La matrice de projection :

(Résultat obtenu en utilisant la photo prise avant)

```
248.80441851750598      611.9576992246102       -152.88088986608145     18625.245645768882
37.027610975049555      -49.739038857288904     -641.1890632562217      13908.981726883198
-0.6020481214873873     0.6528402821783852      -0.45971472173376665    79.93667659128275
```

IV.a – installation d'un projet PoLAR

1. Trouver le fichier CMakeLists.txt sous répertoire polar/Applications/runPolar/

2. Créer un nouveau dossier « monProjet » ailleurs que le répertoire du PoLAR

3. Faire une copie du fichier CMakLists.txt mentionné avant, le mettre sous le répertoire « monProjet »

4. Créer un fichier runProjet.cpp sous le même répertoire, qui permet de faire toute les opérations incluses dans le projet (par exemple, le fichier .cpp permet de charger une photo, de charger un fichier .txt pour prendre en compte la matrice de projection, de charger un objet sur la scène (photo), etc).

5. Modifier le fichier CMakeLists.txt pour correspondre au projet

6. Utiliser cmake compiler le projet (en linux, taper « mkdir build && cd build && ccmake .. » au terminal sous le répertoire « monProjet »)

7. Pour lancer le projet, il suffit de lancer l'exécutable qui se trouve dans le répertoire monProjet/build/bin.

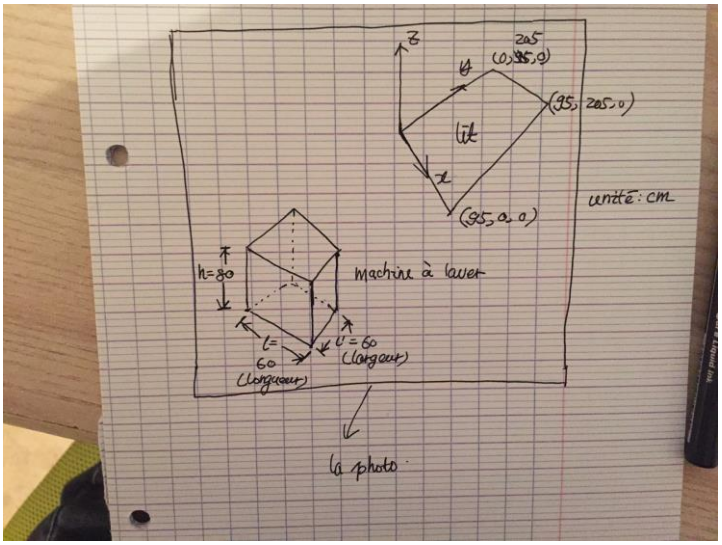
IV.b – Résultat du projet

En utilisant l'exécutable situant sous le répertoire monProjet/build/bin, il n'est possible que de charger une image.

Un exemple de PoLAR (svisu3) est utilisé pour montre la réussite de calcul de la matrice de projection, voici le résultat de l'exemple :

La photo avant l'ajout des objets :

Un croquis de mon résultat :



Le résultat en vrai :

Le projet ne sert qu'à charger une image, du coup, le fichier source runProjet.cpp a pris exactement le même code que l'exemple dans polar (c'est le fichier svisu.cpp sous répertoire polar/Exemples/)

V. Conclusion

PoLAR nous permet d'appliquer la technique de la réalité augmenté sur une photo. Il a beaucoup de fonctionnalités, tel que charger une photo, charger une matrice de projection, mettre un objet sur la scène selon le repère de la photo, etc… En fait, c'est une technologie qui existe aussi partout dans notre monde réel, par exemple :

1) Nous utilisons souvent l'application 'Google Maps' sur nos téléphones portables, il existe des villes en 3D, c'est déjà une réalisation de la réalité augmenté, nous mettons des objets en 3D sur une carte (en 2D), en appliquant la matrice de projection correcte, nous obtenons ce que nous voyons comme un monde 3D.
2) Quand on a une pièce vide, et on veut voir l'effet avec des meubles, nous pouvons bien utiliser la réalité augmenté, en mettant des objets sur la scène pour voir ce que cela nous donne en ajoutant des meubles dans le logement.
3) Et beaucoup plus souvent, quand on regarde les publicités sur les télévisions, cette technique est très largement utilisée quand on applique des effets spéciaux au monde réel.

La réalité augmentée nous permet de percevoir un mélange de monde réel et le monde virtuel, autrement dit, la réalité augmentée renforce le monde réel, qui nous permet de faire des aventures qu'on ne peut pas faire dans le monde réel. Cette technologie sera de plus en plus utilisée au cours du temps.

V. Annexe : (calculs de la matrice de projection)

```
import Jama.Matrix;
public class Test {
	public static void main(String[] args) throws
CloneNotSupportedException {

	/**
	 *
	 */
	double s = 20.5; //en dm
	double t = 9.5; //en dm

	double w = 640; //en pixel
	double h = 480; //en pixel
	double q1x = 233;
	double q1y = 174;
	double q2x = 351;
	double q2y = 217;
	double q3x = 523;
	double q3y = 131;
	double q4x = 416;
	double q4y = 109;
	/**
	 *
	 */

	double[][] tab1 = {{q1x-w/2},{q1y-h/2},{1}};
	double[][] tab2 = {{q2x-w/2},{q2y-h/2},{1}};
	double[][] tab3 = {{q3x-w/2},{q3y-h/2},{1}};
	double[][] tab4 = {{q4x-w/2},{q4y-h/2},{1}};
	Matrice q1prime = new Matrice(tab1);
	Matrice q2prime = new Matrice(tab2);
	Matrice q3prime = new Matrice(tab3);
	Matrice q4prime = new Matrice(tab4);

	double q1primex = q1prime.getCoeff()[0][0];
	double q1primey = q1prime.getCoeff()[1][0];
	double q2primex = q2prime.getCoeff()[0][0];
	double q2primey = q2prime.getCoeff()[1][0];
	double q3primex = q3prime.getCoeff()[0][0];
	double q3primey = q3prime.getCoeff()[1][0];
	double q4primex = q4prime.getCoeff()[0][0];
	double q4primey = q4prime.getCoeff()[1][0];


	Matrice temp1 = (new Matrice(new
double[][]{{q1primex},{q1primey},{1}}));
	Matrice temp2 = (new Matrice(new
double[][]{{q2primex},{q2primey},{1}}));
	Matrice temp3 = (new Matrice(new
double[][]{{q3primex},{q3primey},{1}}));
	Matrice temp4 = (new Matrice(new
double[][]{{q4primex},{q4primey},{1}}));

	Matrice l1 = temp1.produitVectoriel(temp2);
	Matrice l2 = temp4.produitVectoriel(temp3);
	Matrice l3 = temp2.produitVectoriel(temp3);
	Matrice l4 = temp1.produitVectoriel(temp4);

	Matrice vh = l1.produitVectoriel(l2);
```

```
Matrice wh = l3.produitVectoriel(l4);

double vhx = vh.getCoeff()[0][0];
double vhy = vh.getCoeff()[1][0];
double vhz = vh.getCoeff()[2][0];

double whx = wh.getCoeff()[0][0];
double why = wh.getCoeff()[1][0];
double whz = wh.getCoeff()[2][0];

double[][] tab5 = {{vhx/vhz},{vhy/vhz},{1}};
double[][] tab6 = {{whx/whz},{why/whz},{1}};

Matrice vMajuscule = new Matrice(tab5);
Matrice wMajuscule = new Matrice(tab6);

double vx = vMajuscule.getCoeff()[0][0];
double vy = vMajuscule.getCoeff()[1][0];
double wx = wMajuscule.getCoeff()[0][0];
double wy = wMajuscule.getCoeff()[1][0];

double f = Math.sqrt(-(vx*wx+vy*wy));
System.out.println("f="+f);
System.out.println("*********************************");
double[][] tab7 = {
            {f, 0, w/2},
            {0, f, h/2},
            {0, 0, 1}
};
Matrice kMajuscule = new Matrice(tab7);
double[][] tab12 = {
            {0,0,1,0,0,0,0,0},
            {0,0,0,0,0,1,0,0},
            {1,0,1,0,0,0,-q2primex,0},
            {0,0,0,1,0,1,-q2primey,0},
            {1,1,1,0,0,0,-q3primex,-q3primex},
            {0,0,0,1,1,1,-q3primey,-q3primey},
            {0,1,1,0,0,0,0,-q4primex},
            {0,0,0,0,1,1,0,-q4primey}
};

Matrice aMajuscule = new Matrice(tab12);

double[][] tabbb = aMajuscule.getCoeff();
Matrix coucou = new Matrix(tabbb);
Matrix coucoucou = coucou.inverse();
double[][] coucoucoucou = coucoucou.getArray();

Matrice aMajusculeInverse = new Matrice(coucoucoucou);


double[][] tab13 = {
            {q1primex},
            {q1primey},
            {q2primex},
            {q2primey},
            {q3primex},
            {q3primey},
            {q4primex},
            {q4primey}
};
```

```
		Matrice qiprime = new Matrice(tab13);
		Matrice hMinuscule = aMajusculeInverse.multiply(qiprime);
		System.out.println("***************\nh");
		System.out.println(hMinuscule+"***************\n");

		double[][] tab14 = new double[3][3];
		for(int i = 0; i < 3; i++) {
			for(int j = 0; j < 3; j++)
				tab14[i][j] = 0;
		}

		for(int i = 0; i < 3; i++) {
			for(int j = 0; j < 3; j++) {
				if(i == 2 && j== 2) {
					tab14[i][j] = 1;
					break;
				}
				tab14[i][j] = hMinuscule.getCoeff()[i*3+j][0];
			}
		}

		hMinuscule = new Matrice(tab14);
		System.out.println("h\n"+hMinuscule+"\n");


		double[][] tab15 = {
				{1/f, 0, 0},
				{0, 1/f, 0},
				{0, 0, 1}
		};
		Matrice inter1 = new Matrice(tab15);
		Matrice bMajuscule = inter1.multiply(hMinuscule);
		double[][] termes = bMajuscule.getCoeff();
		double[][] tabB1 = {
				{termes[0][0]},
				{termes[1][0]},
				{termes[2][0]}
		};
		double[][] tabB2 = {
				{termes[0][1]},
				{termes[1][1]},
				{termes[2][1]}
		};
		double[][] tabB3 = {
				{termes[0][2]},
				{termes[1][2]},
				{termes[2][2]}
		};
		Matrice bMajuscule1 = new Matrice(tabB1);
		Matrice bMajuscule2 = new Matrice(tabB2);
		Matrice bMajuscule3 = new Matrice(tabB3);

		Matrice rMajuscule1 = Matrice.diviser(bMajuscule1,
bMajuscule1.norme());
		Matrice inter = rMajuscule1.produitVectoriel(bMajuscule2);
		Matrice rMajuscule3 = Matrice.diviser(inter, inter.norme());
		Matrice rMajuscule2 = rMajuscule3.produitVectoriel(rMajuscule1);
		double lambda = (bMajuscule1.norme())/s;
		double tCalcule = bMajuscule2.norme()/lambda;
		System.out.println("t_calcule="+tCalcule);
		System.out.println("*****************************************");
```

```
		Matrice tMajuscule = Matrice.diviser(bMajuscule3, lambda);

		double[][] tab16 = new double[3][4];
		for(int i = 0; i < 3; i++) {
			for(int j = 0; j < 4; j++) {
				tab16[i][j] = 0;
			}
		}
		for(int i = 0; i < 3; i++) {
			tab16[i][0] = rMajuscule1.getCoeff()[i][0];
		}
		for(int i = 0; i < 3; i++) {
			tab16[i][1] = rMajuscule2.getCoeff()[i][0];
		}
		for(int i = 0; i < 3; i++) {
			tab16[i][2] = rMajuscule3.getCoeff()[i][0];
		}
		for(int i = 0; i < 3; i++) {
			tab16[i][3] = tMajuscule.getCoeff()[i][0];
		}
		Matrice eMajuscule = new Matrice(tab16);
		Matrice projection = kMajuscule.multiply(eMajuscule);
		System.out.println("la matrice de projection\n");
		System.out.println(projection);

		}

}
```