Trifocal tensor

Tifaine CAUMONT and Elisa PRANA

January 6, 2013

Contents

Introduction

The aim of this project was to create an application that enables to establish relations between three pictures. The first step consists in clicking on seven points on each picture representing a 3D scene from different points of view. After seven matches, the program is able to find the position of an element on the third picture if you click on the position of the same element on the two other pictures.

1 List of the elements

Display the help in english with -h : element requested, encoded and working.

Launch of the program into the console with the name of the three pictures to load : element requested, encoded and working.

Launch of the program into the console with the name of the three lists to load : element requested, encoded and working.

Back-up of a list of clicked points : element requested, encoded and working.

Calculation of a tensor : element requested, encoded and working.

Transfert pf the points with the tensor : element requested, encoded and working.

Launch of the program into the console with the path of the repository in which the lists are saved : element not requested, encoded and working.

Undo/Redo : element not requested, encoded and but not working.

2 Description of the elements encoded

2.1 Tensor

A trifocal tensor is a 3x3x3 matrix. It contains 27 elements that we need to calculate. The elements T_k^{il} of the tensor have to validate the following expression :

$$\sum_{k=1}^{3} x_p^k (x_p^{'i} x_p^{''3} T_k^{3l} - x_p^{'3} x_p^{''3} T_k^{il} - x_p^{'i} x_p^{''l} T_k^{33} + x_p^{'3} x_p^{''l} T_k^{i3})$$

For each point, we have four equations. So in order to have enough equations, we need seven points. In order to find the tensor, we have to resolve the following system : $At = 0$, where A is a 28x27 matrix and $t = (T_1^{il}, T_2^{il}, T_3^{il})$ a vector with 27 components.

Build A

The first step to resolve the system, is to build the A matrix. Let's call Ap_k^{il} the coefficient of the T_k^{il} elements in all these equations for the point p. We noticed that in each equation, only four tensor coefficients appear. The other are then equal to zero.

The rows of A correspond to $(Ap_1^{il} Ap_2^{il} Ap_3^{il})$. For each point, there are four rows in the matrix. The first correponds to $i = 1$ and $l = 1$, the second to $i = 1$ and $l = 2$, the third to $i = 2$ and $l = 1$ and the forth to $i = 2$ and $l = 2$.

Let's take an example with the p point. We obtain the four following equations :

$$\sum_{k=1}^{3} x_p^k (x_p^{'1} x_p^{''3} T_k^{31} - x_p^{'3} x_p^{''3} T_k^{11} - x_p^{'1} x_p^{''1} T_k^{33} + x_p^{'3} x_p^{''1} T_k^{13}) \text{ with } i=1 \text{ and } l=1$$

$$\sum_{k=1}^{3} x_p^k (x_p^{'1} x_p^{''3} T_k^{32} - x_p^{'3} x_p^{''3} T_k^{12} - x_p^{'1} x_p^{''2} T_k^{33} + x_p^{'3} x_p^{''2} T_k^{13}) \text{ with } i=1 \text{ and } l=2$$

$$\sum_{k=1}^{3} x_p^k (x_p^{'2} x_p^{''3} T_k^{31} - x_p^{'3} x_p^{''3} T_k^{21} - x_p^{'2} x_p^{''1} T_k^{33} + x_p^{'3} x_p^{''1} T_k^{23}) \text{ with } i=2 \text{ and } l=1$$

$$\sum_{k=1}^{3} x_p^k (x_p^{'2} x_p^{''3} T_k^{32} - x_p^{'3} x_p^{''3} T_k^{22} - x_p^{'2} x_p^{''2} T_k^{33} + x_p^{'3} x_p^{''2} T_k^{23}) \text{ with } i=2 \text{ and } l=2$$

So in the A matrix, we have the following values for the first equation:

$$Ap_k^{31} = x_p^k x_p^{'1} x_p^{''3}$$

$$Ap_k^{11} = -x_p^k x_p^{'3} x_p^{''3}$$

$$Ap_k^{33} = -x_p^k x_p^{'1} x_p^{''1}$$

$$Ap_k^{13} = x_p^k x_p^{'3} x_p^{''1}$$

If we adapt this to a general case, we then have :

$$Ap_k^{3l} = x_p^k x_p^{'i} x_p^{''3}$$

$$Ap_k^{il} = -x_p^k x_p^{'3} x_p^{''3}$$

$$Ap_k^{33} = -x_p^k x_p^{'i} x_p^{''l}$$

$$Ap_k^{i3} = x_p^k x_p^{'3} x_p^{''l}$$

The issue is now to fill the A matrix in the program. The rows are depending on p, i and l and the columns on k, i and l. Here is the pseudocode to fill the matrix A.

Initialize A to 0
for *p from 0 to 6* **do**
  for *i from 0 to 1* **do**
    for *l from 0 to 1* **do**
      for *k from 0 to 2* **do**
        $A[4p+2i+l, 9k+3i+l] = -x_p^k x_p^{'3} x_p^{''3}$
        $A[4p+2i+l, 9k+3i+2] = x_p^k x_p^{'3} x_p^{''l}$
        $A[4p+2i+l, 9k+6+l] = x_p^k x_p^{'i} x_p^{''3}$
        $A[4p+2i+l, 9k+8] = -x_p^k x_p^{'i} x_p^{''l}$
        increment k of 1
      end
      increment l of 1
    end
    increment i of 1
  end
  increment p of 1
end

Algorithm 1: Fill the matrix A

Calculate T

Once we have the A matrix, we need to decompose it in $A = UDV^T$ with the SVD of the Eigen Library. The t vector is the last column of the V matrix. We can easily pick it up. We don't forget to put the element of t in the real tensor T with the right loops:

for *i from 0 to tensor.getI()* **do**
  for *j from 0 tensor.getJ()* **do**
    for *k from 0 to tensor.getK()* **do**
      $T(i, j, k) = t(9 * k + 3 * i + j))$
      increment k of 1
    end
    increment j of 1
  end
  increment i of 1
end

Algorithm 2: Calculate T

2.2 Transfert

The transfert was a difficult part of the project. The main difficulty was to transform the equations into matrices and vectors.

The equation used to transfert the points is the same than the one used for the tensor :

$$\sum_{k=1}^{3} x_p^k (x_p^{'i} x_p^{''3} T_k^{3l} - x_p^{'3} x_p^{''3} T_k^{il} - x_p^{'i} x_p^{''l} T_k^{33} + x_p^{'3} x_p^{''l} T_k^{i3})$$

Since we have all the informations about the tensor and two of the three points, the new unknowns are the components of the third point. We decided to put the homogeneous coordinate to 1 in order to simplify the problem. The system to solve is then $Bx = b$ and not $Bx = 0$ as we first thought. B is a 4x2 matrix, x a vector of two components and b a vector of four components.

Let's call B_k^{il} the coefficient of x_p^k depending on what we take for i and l.

We have the four same equations than for the tensor, that we need to rearranged depending of the point transfered.

Transfert on the first image

For the first transfert, the colums of B are corresponding to the two coordinates that we search and the line to the different values of i and l. If we remove from the product in the equations the homogeneous coordinate for all the point, we have the following expression for B:

$$B_k^{il} = x_p^{'i} T_k^{3l} - T_k^{il} - x_p^{'i} x_p^{''l} T_k^{33} + x_p^{''l} T_k^{i3}$$

The method is the same to find b, but we take the coefficient of the homogeneous coordinate and $k = 3$:

$$b^{il} = x_p^{'i} T_3^{3l} - T_3^{il} - x_p^{'i} x_p^{''l} T_3^{33} + x_p^{''l} T_3^{i3}$$

Here is the pseudocode to build B and b for the first transfert :

for *i from 0 to 1* **do**
  for *l from 0 to 1* **do**
    for *k from 0 to 2* **do**
      $B[2*i+j,k] = x^{'i}T_2^{jk} - T_i^{jk} - x^{'i}x^{''j}T_2^{2k} + x^{''j}T_i^{2k}$
      $b[2*i+j] = -x^{'i}T_2^{j2} - T_i^{j2} - x^{'i}x^{'j}T_2^{22} + x^{''j}T_i^{22}$
      increment k of 1
    end
    increment l of 1
  end
  increment i of 1
end

Algorithm 3: Build B and b

This time, we applied the SVD with the method solve(b) on the B matrix in order to have x.

Transfert on the second image

We take time to understand the fact that the columns of the matrix B were not depending on k anymore, but on i for the second picture and l for the third. Once understood, the method is quite the same.

We can also write the equation like this :

$$\sum_{k=1}^{3} x_p^k (x_p^{'i}(T_k^{3l} - x_p^{''l}T_k^{33}) + (x_p^{''l}T_k^{i3} - T_k^{il}))$$

Then we have for B and b :

$$B_i^{il} = x_p^k (T_k^{3l} - x_p^{''l}T_k^{33})$$

$$b^{il} = x_p^k (x_p^{''l}T_k^{i3} - T_k^{il})$$

Since there is the x_p^k element, we need to have a sum in the loop. Here is the pseudocode:

for *i from 0 to 1* **do**
  for *j from 0 to 1* **do**
    for *k from 0 to 2* **do**
      $B[2*i+j,i] += x^k(T_2^{jk} - x^{''j}T_2^{2k})$
      $b[2*i+j] += x^k(-x^{''j}T_i^{2k} + T_i^{jk})$
      increment k of 1
    end
    increment j of 1
  end
  increment i of 1
end

Algorithm 4: transfert to the second image

We apply the SVD with the method solve(b) on the B matrix in order to get x.

Figure 1: Transfert on the second image after clicking on the first and the third images.

Transfert on the third image

The method is similar to the one for the third transfert. The equation can be written like this :

$$\sum_{k=1}^{3} x_p^k (x_p^{''l} (T_k^{i3} - x_p^{'i} T_k^{33}) + (x_p^{'i} T_k^{3l} - T_k^{il}))$$

Then we have for B and b :

$$B_l^{il} = x_p^k (T_k^{i3} - x_p^{'i} T_k^{33})$$

$$b^{il} = x_p^k (x_p^{'i} T_k^{3l} - T_k^{il})$$

Here is the pseudocode :

for *i from 0 to 1* **do**
  for *j from 0 to 1* **do**
    for *k from 0 to 2* **do**
      $B[2 * i + j, j]$ += $x^k T_i^{2k} - x^{''i} T_2^{2k}$)
      $b[2 * i + j]$ += $x^k (T_i^{jk} - x^{'i} T_2^{jk})$
      increment k of 1
    end
    increment j of 1
  end
  increment i of 1
end

Algorithm 5: transfert to the third image

Use the right transfert

Once the transfert working for three pictures, we had to put conditions on the click event to calculate the right transfert, depending on the two first points clicked.

2.3 Saving the clicked points

At first, the lists were lists of 7 points and when clicking, the first point of the list was changed. It was not a great idea. Then we add an eighth point, calculated with the transfert in order to keep the list of the seven points needed to build the tensor. The problem was that we did not save all the points and we were writting in the files with std::ifstream. So we changed the saveMatrix function in order to save as many points as wanted. But it still changed the initial file. We decided to save the point int tmp/list.list with a new method : " << ". A row is just added into the matrix and the file saved for each click. Finally, we created a function (updateMatrix) in order to do the resizing of the matrix and the saving in a file.

2.4 Save or load a list in a file

Saving

At the beginning, the lists of the clicked points were saved in the initial file. This was a probleme since the user may not want to modify his files. So we decided to save them directly into the repository tmp/. Finally, we had an option for the user. He can specify a repository when he launches the program. The lists are then saved in this repository.

All the files are saved with the format ask in the subject (row and col at the beginning). In order to do that, we activated the header to true in saveMatrix()in mathIO.

Loading

The lists are loaded at the beginning of the program, according to the options that the user chooses. In order to manage the empty list or the list containing a row or column equal to zero, we modified the function loadMatrix() and readMatrixHeader() in mathIO.

2.5 Launch the program with arguments

We have done a program which can be launched with or without arguments. When the program is launched, there is a test that checks if there are some arguments, and how many. With *argc*, we have the number of arguments written in the command, and with $argv[i]$, we can pick up the argument to use it in a fonction for example. Be careful to not forget that when we wrote for example ./bin/trifocal, it counts like an argument. When the user does not give arguments, the default program is executed. It's a program with the three default images, and the three default lists with already the seven points in it.

Figure 2: Program launched without arguments

If the user wants, he can give three more arguments, which are the three images he wants to load. If he loads some images, there will be no list with points. He has to clic the corresponding points on the three images.

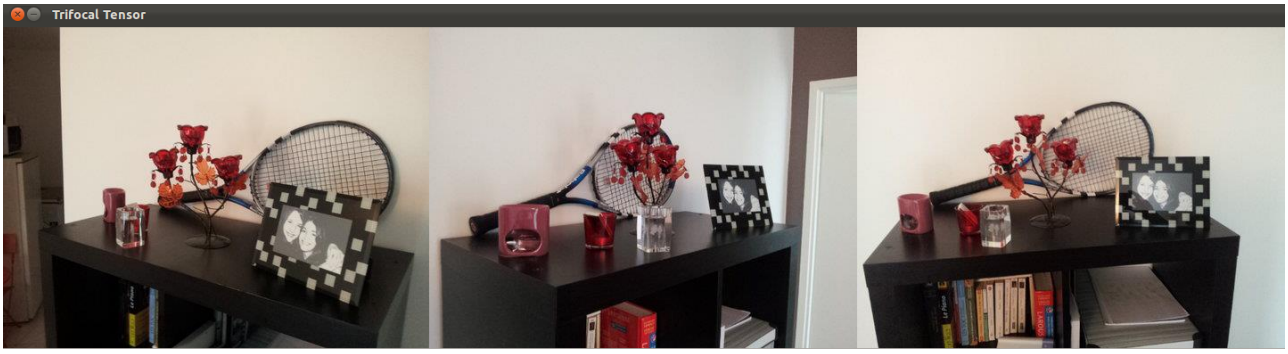

Figure 3: Program launch with user's pictures

If the user wants to have his own images and lists of points, he can give six arguments, first the three images, then the three lists. The user can also specify the path of the repository in which he wants the list to be saved, when he gives some arguments.

2.6 Undo/Redo

We wanted to make undo-redo in the program because currently, if you click a wrong point, you have to start again from the beginning if you don't want to modify the files. In order to do that, we created an undo function that delete the last point. We succeed in modifying the last point clicked of the last list modified. This function is working, the point disappear of the list, but it is still displayed on the screen. So we did not go further. The code is in commentary in the files.

3 Improvement

This program can still be improved.

3.1 Resize

Currently, when we load pictures which are to big, the program works but does not adapt the size of the images to the size of the screen. So we can't see them on our screen and work with it. We want to resize these pictures in order to have a window contained in the screen surface.

3.2 Zoom

When we don't load a list of point, launching the program, we have to click on the seven matching points to calculate the tensor and use the transfert. But it is difficult to click precisely. So we thought about creating a zoom with SDL rotozoom. We did not had time to implement it, but it is a way to click without to much error.

3.3 Options

The options in the console are working, but are not very precised. We still can work on it in order to manage all the configuration that the user may use.

3.4 Undo/Redo

As the undo function is not displayed, we can improve the project with a functionnal undo/redo function. We could delete the points clicked and redo them in case of error. This function is very used in all the programs and enable a better user experience.

Conclusion

This project was quite absorbing since he will never be completely finished. Actually, the tricky part was to pass from mathematics function to code. Using the Eigen Library, it forced us to search the functions we could use in the documentation. We learn to work with code that is not ours and not explained by someone.

We decided to write all the project in english in order to pratice. We usually use english name for our functions so we can also write the commentaries in english.

Even if we did not success in implementing all the functions we wanted, the program is functionnal and the transferts are working.