Computing Homography Transformations

CS 330

Due 11:59 pm, Friday, November 15, 2013

1 Introduction

Figure 1: Projecting the face of a cereal box using a homography transformation.

For this project you will create a module for solving linear equations of the form $A\mathbf{x} = \mathbf{b}$ using *LU decomposition* and then use your module to build an application that determines the parameters of an interesting image transformation involving projections. As an example, your program will discover a transformation that maps the pixels for the face on the cereal box on the left of Figure 1 to the image on the right. This type of transformation is called a *homography* and is described in Section 2. Section 3 specifies the linear systems used for computing the parameters of a homography transformation. The routines you need to implement for your LU decomposition module are listed in Section 4. The specifications of your application program are given in Section 5 and what your are to submit is listed in Section 6

2 Homography

Two images of the same planar surface are related by an image coordinate transformation called a *homography*. The type of homography we are interested in can be described by a 3×3 matrix H. The pixel coordinate (x, y) is represented by a vector $[x\ y\ 1]^T$ which is multiplied by H yielding the vector $[u\ v\ w]^T$:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1}$$

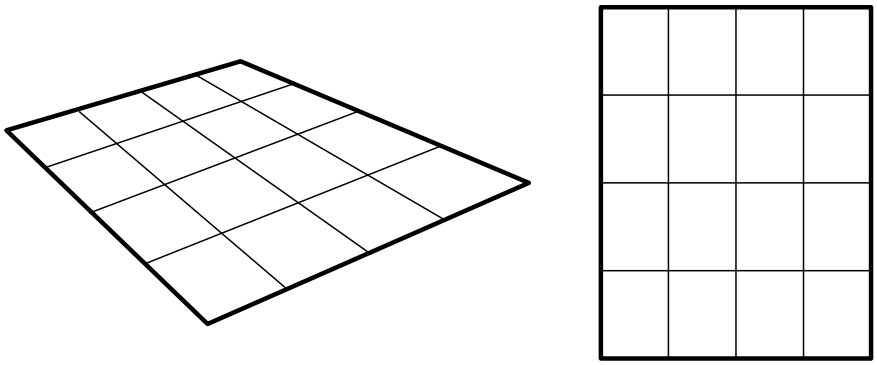

Figure 2: Projections of a grid from two different views.

We divide the resulting components by w yielding the resulting coordinate (x', y') :

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} \mapsto (u/w, v/w) = (x', y'). \tag{2}$$

The division by w warps the coordinates properly to account for perspective foreshortening. Figure 2 shows two different projections of the same grid.

3 Finding the optimal homography

Given $N \geq 4$ source points $\{(x_i, y_i)\}_{i=0}^{N-1}$ and N corresponding target points $\{(x'_i, y'_i)\}_{i=0}^{N-1}$ we have the following $2N$ equations and 8 unknowns:

$$\frac{h_{00} \cdot x_i + h_{01} \cdot y_i + h_{02}}{h_{20} \cdot x_i + h_{21} \cdot y_i + 1} = x'_i, \tag{3}$$

$$\frac{h_{10} \cdot x_i + h_{11} \cdot y_i + h_{12}}{h_{20} \cdot x_i + h_{21} \cdot y_i + 1} = y'_i. \tag{4}$$

We can rewrite these equations as a linear system for $i = 0, 1, \ldots N-1$:

$$h_{00} \cdot x_i + h_{01} \cdot y_i + h_{02} - h_{20} \cdot x_i \cdot x'_i - h_{21} \cdot y_i \cdot x'_i = x'_i, \tag{5}$$

$$h_{10} \cdot x_i + h_{11} \cdot y_i + h_{12} - h_{20} \cdot x_i \cdot y'_i - h_{21} \cdot y_i \cdot y'_i = y'_i. \tag{6}$$

In matrix form we have $A\mathbf{x} = \mathbf{b}$ where A is a $2N \times 8$ matrix and $\mathbf{b}$ is a $2N \times 1$ vector:

$$\begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -x_0 \cdot x'_0 & -y_0 \cdot x'_0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -x_0 \cdot y'_0 & -y_0 \cdot y'_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 \cdot x'_1 & -y_1 \cdot x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 \cdot y'_1 & -y_1 \cdot y'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{N-1} & y_{N-1} & 1 & 0 & 0 & 0 & -x_{N-1} \cdot x'_{N-1} & -y_{N-1} \cdot x'_{N-1} \\ 0 & 0 & 0 & x_{N-1} & y_{N-1} & 1 & -x_{N-1} \cdot y'_{N-1} & -y_{N-1} \cdot y'_{N-1} \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{bmatrix} = \begin{bmatrix} x'_0 \\ y'_0 \\ x'_1 \\ y'_1 \\ \vdots \\ x'_{N-1} \\ y'_{N-1} \end{bmatrix} \tag{7}$$

where $\mathbf{x} = [h_{00} \ldots h_{21}]^T$ is our homography matrix in row-major order. When $N = 4$ (and no three of the source points are co-linear), solving $A\mathbf{x} = \mathbf{b}$ is sufficient.

3.1 Overdetermined System ($N > 4$)

When $N > 4$ this is an overdetermined system which we solve using the *method of least-squares.* The optimal solution $\mathbf{x}^*$ is found by solving the system

$$A^T A\mathbf{x}^* = A^T\mathbf{b}. \tag{8}$$

$A^T A$ is a 8×8 symmetric (positive semi-definite) matrix and $A^T\mathbf{b}$ is an 8×1 vector. The simplest approach is to construct the $2N \times 8$ matrix A in Equation 7 and use this to construct the upper triangular portion of $A^T A$:

$$\left(A^T A\right)_{ij} = \sum_{k=0}^{2N-1} A_{ki}A_{kj} \quad \text{for } i = 0\ldots 7; \text{ and } j = i\ldots 7. \tag{9}$$

We then copy the upper portion of the matrix into the corresponding values below the diagonal:

$$\left(A^T A\right)_{ij} = \left(A^T A\right)_{ji} \quad \text{for } i = 0\ldots 7; \text{ and } j = 0\ldots i-1. \tag{10}$$

We compute $A^T\mathbf{b}$ as follows:

$$\left(A^T\mathbf{b}\right)_i = \sum_{j=0}^{2N-1} A_{ji}\mathbf{b}_j \quad \text{for } i = 0\ldots 7. \tag{11}$$

Now we use our LU decomposition engine to solve Equation 8. When $N = 4$ this approach should give us the exact same result as solving Equation 7 directly.

4 LU decomposition module

You will create a reusable LU decomposition module in C99 with the following interface:

```
typedef struct { ... } LUdecomp;
LUdecomp *LUdecompose(int N, double **A);
void LUdestroy(LUdecomp*);
void LUsolve(LUdecomp *decomp, double *b, double *x);
```

`LUdecomp` : an opaque data structure that holds the necessary LU decomposition information.

`LUdecompose()` : this routine performs the decomposition (see course notes for details) and allocates, fills, and returns a `LUdecomp` object. If the matrix A is singular return `NULL`. Note that the matrix A is stored as a vector of row-pointers.

`LUdestroy()` : deallocates the data allocated in `LUdecompose()`.

`LUsolve()` : solves the system $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$.

5 Homography program

Your program will read a text file from from `stdin`. The first value will be the integer N representing the number of source and target points. The pixel xy-coordinates (column, row) of the N source points and N target points are listed next (they may be floating point numbers even though the example list only integer values). Here is a sample input file that maps 4 source coordinates to 4 target coordinates.

```
4
0 0
500 0
500 650
```

```
0 650
10 107
362 7
789 189
318 401
```

Your program should write out the nine components on the homography matrix to `stdout` in row-major order; here is a sample run using the above dataset:

```
./homography < boxtop.in
0.8170656923 0.2809752103 10.0000000000
-0.1978136468 0.2090962195 107.0000000000
0.0003123362 -0.0006065124 1.0000000000
```

You are required to at least solve for the case when $N = 4$ (*e.g.,* by just solving Equation 7 directly. As a bonus, you can make your program more general purpose and allow for overdetermined systems by solving Equation 8 when $N > 4$.

I will provide you with some test data and a program for actually transforming an image using your homography. The image on the right of Figure 1 was created by extracting the pixels from the image on the left as follows:

```
./homography < boxtop.in | ./hmap cheerios-small.ppm 500 650 > face.ppm
```

6 What to submit

You will archive the source code and supporting files for your solution and submit it electronically. Your archive should contain at least the following files.

- `README` : a text file that contains your contact information, a brief description of the project, and how to build and run your application.
- `LUdecomp.h, LUdecomp.c` : source code for LU decomposition module (this should be an independent, separately compiled module);
- `homography.c` : source code for homography computation program.

You may also include any test data that you have used. Submit by midnight on the due data. Have fun.