Final Project for CS 372

*Riley Kopp*

*Jonathan McKee*

# Algorithm, Application, Language Choice

* *Direct Linear Transform (Homography Warping)*
* *Image Stitching*
* *Python 3.6.8*

# Where It Is Used

*Homography warping is accomplished by taking matching points from two images and creating a homography matrix. Image warping works by creating a 3x3 matrix that allows any point from the first image to be translated to the correct (x, y) coordinates in the second image. The DLT algorithm is one of main methods of generating this matrix.*

## Other applications

* *Calibrating camera*
* *Placing videos or still images on a green screen.*
* *Implanting an image on a surface inside of another image*
* *Creating a 3d image representation from 2d images.*

## Alternative algorithms

* *Conic Correlation*
* *Line Correlation*

*All 3 algorithms are used to create the 3x3 homography matrix to warp the first image to be drawn into the second image. In addition, all 3 algorithms benefit greatly from normalizing the data before calculating the matrix. Normalizing the data before calculation allows the homographic algorithms to be more tolerant of noise and shifts in the planes. It is possible to use more than the minimum number of pairs, but the improvement to accuracy is gained by eliminating outliers instead of making the conversion process more accurate.*

*The DLT method works on corresponding points and relies on the matching of the points to be accurate. Because each correspondence generates two points of data in the homography matrix, we need at minimum four pairs of corresponding points.*

*The Line method on the other hand uses the cross product of two lines to determine the transformation matrix. This means we need a minimum of two pairs of corresponding points. It is also possible if we do not have the points we need in the image to generate some of the missing data using the line equations for the two matching lines.*

*The conic method*

## Reason for choice

*[explain why you picked the algorithm you did]*

*[1/2 page is typically sufficient for this section]*

# How Your Project Works

*[How it works. Pretend you are explaining how your project works to a Programming II freshman. 1-3 paragraphs will likely be sufficient, but ask if you are unsure This is mostly to make sure you understand what you did and you didn’t just copy and paste code from somewhere.]*

*(team projects are expected to have more detail. You must “chunk” the algorithm and explain each part.)*

## Correctness (team only)

*[Formally, justify why your algorithm is correct or why it works. You must formally use loop invariant, pre-and post conditions, etc. Any and all properties for a given algorithm, if applicable, are required. Assume you are speaking to someone who has completed this course]*

# Run time

*[Name and explain why the project has this big-O run time as the theoretical run time* **for your implementation***]*

*[A* ***fully*** *labeled graph* runtime graph for varying n with a minimum of 10 points goes here. It **must clearly show** the run time. If you have more than 1 value that affects input, you may just vary one for the graph, but be clear which one you used. Two graphs would be ideal, and you may use a minimum of 5 points for each in this case)

(team only): Formally prove the run time through instruction counting, probability, or recursion analysis depending on your problem. You **will need** to use psudeocode to prove this.

# Built-in Code Correctness Tests

(team only, you must have a minimum of 6 rather than 3 built-in tests)

|  |  |  |  |
| --- | --- | --- | --- |
| ***Test Case*** | ***Description*** | ***Input*** | ***Actual output*** |
| *[test 1]* |  |  |  |
| *[test 2]* |  |  |  |
| … |  |  |  |

Alternatively, you may have this in the following format:

## Test 1; name

Description

### Input

The input

### Output

The output

# Program usage or README

[OPTIONAL, and only used if needed]

# References

[Where did you find the explanation of your code]

[Tutorials]

<https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html>

E. Dubrofsky, R. Woodham, Combining Line and Point Correspondences for Homography Estimation. ISVC ’08 Proceedings of the 4th International Symposium on Advances in Visual Computing, Part II, 2008. 202 -213

E. Dubrofsky. Homography estimation. PhD thesis, UNIVERSITY OF BRITISH COLUMBIA (Vancouver, 2009)

J. Kannala, M. Salo, and J. Heikkilä. Algorithms for computing a planar homography from conics in correspondence. In British Machine Vision Conference, 2006. 118