Assignment 3: Projective Transformation

Sudheer Nadella

***Abstract*—Projective Transformation allows us to project an image in another image within the selected region and it works by replacing the selected region of the target image with the source image. It first finds p and set the initial homogeneous matrix and keep on updating the homogeneous matrix and calculate the residuals. With the homogeneous matrix the source image coordinates are calculated in the target image and the finally the our selected region in the target image will be replaced with the source image.**

A. Algorithm

For better understanding the algorithm can be broken down to 3 parts. In the first part we calculate the P and assign the values to initial H. In this step: we calculate ΔX, set $P_1 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$ and $P_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$ and calculate J_1 and J_2 matrices as $J_1 = X^T P_1$ and $J_2 = X^T P_2$ and calculate $J = \begin{bmatrix} J_1 \\ J_2 \end{bmatrix}$ $A = J^T J$ $b = J^T \Delta X$ and $P = A^{-1} b$. Update the P values to H. In the second step: continue calculating P for number of boundaries considered and update H but the J calculation is different from the one we did in the first step $J = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -1.0 * x * \widehat{x} & -1.0 * y * \widehat{x} \\ x & y & 1 & 0 & 0 & 0 & -1.0 * x * \widehat{y} & -1.0 * y * \widehat{y} \end{bmatrix}$. In the third step: we now have H, so check whether the points fall under our region and update the points and then update the pixel values over our target region.

B. Related Work

In the second step, I have used dampling factor to compute A^{-1} as if the determinant of A matrix results zero, then singular matrix error will be raised. In order to prevent singular matrix condition I have updated the diagonal values of A with the product of dampling factor with diagonal values of A. so $A = A + d * diag(A)$. In the third step, to check whether the point lies within our selected region of the target I have used $within(poly)$ by importing $shapely$.

C. Examples

I ran projective transformation on three images In which I have rectangular region for 2 images and hexagon for 1 image. For hexagon I have considered more than 4 points to perfectly fit the image in the region. I have performed transformation on hallway.png, pepsiTruck.png and stopSign.jpg.

Source Images:

Projection Images:

As you can clearly see the image with spectacles is projected on to the screen in hallway.

1) : The nerdy image is actually a rectangular image, but our region of projection in the hallway is not a rectangular region, so the hallway image is first transformed to corresponding target region and then mapped to our region of projection.

2) : Both cocacola and coke are rectangular images, but our region of projection on the pepsi truck is not a rectangular region, so the both the images are first transformed to corresponding target region and then mapped to our region of projection.

3) : This case is different from the above ones. Here our source image is a rectangular image and our area of projection is a hexagon. You can't just map 4 points with 8 points i.e, the corners. So when in cases like these: make the number of points of source image equal to number of points considered in the target image. If points of source are more than target then reduce them by considering appropriate points that map to target region and if less than target consider more points that map to target region. Once you're done with equalising the number of points, then do the same by mapping them over our region of projection in the target image.

D. Lessons Learned

While computing Inverse of matrix check whether the determinant of the matrix which you are computing inverse is zero, if it's zero try to update the diagonal values in that matrix by a marginal value so that the determinant should not be zero.

For checking the point whether it lies in the region, we can use 'within' by importing shapely.

For appending values one after other in a vertical manner we can use numpy.vstack.