Parallel Augmented Reality with Planar Homography

Jiayu Bai, Xingsheng Wang

November 2020

1 Summary

We are going to implement a parallel version of augmented reality with planar homography. The program uses planar homography to project a video on top of another surface in another video. An example output video could be found on the project website:

`https://checkraiseoncloud.github.io/ParallelPlanarHomography/`

2 Background

2.1 Input

The input to the program includes the original video, an image of the target surface, and the target video on which we want to project the original video.

Figure 1: original video

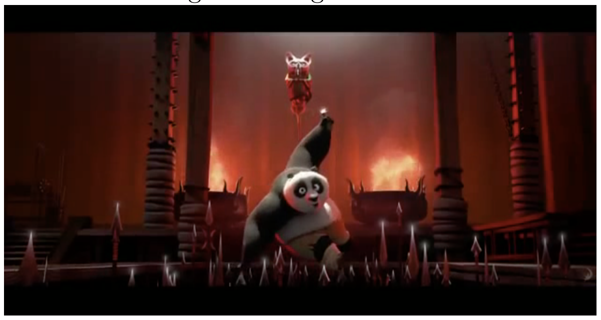


Figure 2: target surface

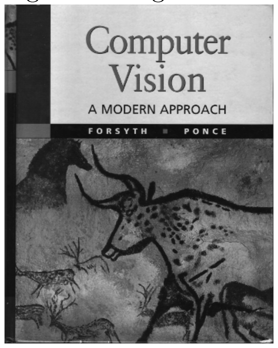
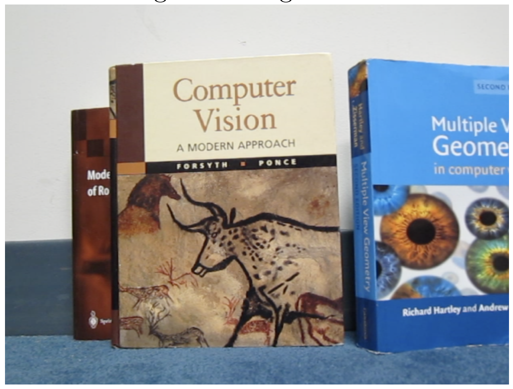


Figure 3: target video



2.2 Planar Homography

In order to project the original video onto the target surface of the target video, we need to perform planar homography on every frame of the video. For each frame, we need to find a homography matrix H that projects the target surface onto the target video frame. This homography matrix is a 3*3 matrix and it is the mathematical representation of warping the surface on to its location in the frame. For every point in the surface target, we can obtain its location the target video frame by the following equation:

$$\begin{bmatrix} X_{video} \\ Y_{video} \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} * \begin{bmatrix} X_{surface} \\ Y_{surface} \\ 1 \end{bmatrix} \tag{1}$$

Once we have the homography matrix H, we can use it to project the target video frame to the target video frame.

2.3 Points of Correspondence

In order to find the homography matrix, we need to first find points of correspondence between the target surface and the target video. We first use FAST (Features from Accelerated Test) detector to extract feature points in the target surface. Then, we use BRIEF (Binary Robust Independent Elementary Features) descriptor to summarize the feature point and match them using Hamming distance. This will give us multiple point correspondence between the surface image and the target video frame. For each feature points in the surface image, we obtain where that point is located on the target video frame.

2.4 RANSAC

Once we have multiple point correspondence, we can move on to solve the homography matrix. Since there are miss matches and noise in the point correspondence, we use an algorithm called RANSAC (Random Sample Consensus) to select the best homography matrix. For every iteration, we first select four points to calculate H. Then we evaluate H by warping the point and checking how many points actually gets warped to its corresponding location. The best homography matrix is the one where we have the largest number of in-liers: points whose warped location is close enough to its corresponding location on the target video.

2.5 Homography Matrix

For every iteration of the RANSAC algorithm, we derive the homography matrix with the four corresponding points using the following equation:

$$A * \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = 0 \tag{2}$$

where

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' & -x_1{}' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' & -y_1{}' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' & -x_2{}' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' & -y_2{}' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' & -x_3{}' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' & -y_3{}' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' & -x_4{}' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' & -y_4{}' \end{bmatrix} \tag{3}$$

and $(x1, y1)$ and $(x1', y1')$ represents the first pair of corresponding points. We can then derive H by using Singular Value Decomposition of matrix A.

2.6 Parallel Implementation

We can see that there are several axis of parallelism that exist in this program. The first one is that the homography matrix needs to be calculated for every frame. Thus, we can parallelize over all the frames in the video. The second one is the RANSAC algorithm on each frame, we can parallelize over the number of iterations to find the best fitting homography matrix. Moreover, for each iteration, we need to calculate the number of in-liers, so we can parallelize over all the point correspondences.

3 Challenge

The major challenge in this project is that there are multiple axis of parallelism that we can exploit. We need to determine how to define parallel work and how to assign these work the the hardware. As described in the previous section, we can parallelize over the video frames, each iteration of the RANSAC algorithm and the point correspondences. How to map these different levels of parallelism to Cuda cores and how to synchronize between cuda threads are all questions that we need to answer.

For each individual frames, there is no dependency between frames, and the memory access pattern is good: we only access the target video frame and the target surface image when calculating the homography matrix. For the RANSAC algorithm, since the points are generated randomly, how do we ensure that each thread work on different set of points and the same set of points are not repeated? If we do not have many points correspondences, we might need to dynamically adjust RANSAC to reduce the number of iterations to run, how does this impact the parallel implementation? For frames with huge number of point correspondences, we can use separate threads to run on separate sets of points. In this case, parallelizing over all the point correspondences could also benefit. For frames with only a few points correspondences, parallelism might be an overkill. How to parallelize RANSAC and how to integrate multiple levels of parallelism are problems that make mapping the workload difficult.

4 Resources

We will be using the project assignment from the Computer Vision course (16-720) as our base for the project. In this project, we implemented the seqential version of the code that transforms one image into another image and using this planer homography technique, we projected a video onto a surface of another video.

5 Goals and Deliverables

PLAN TO ACHIEVE: We plan to achieve a 30x speedup for the project compared to the original implementation. Since we have multiple levels of parallelism, we are expecting a very good speed up from parallelizing frames and parallelizing matrix generation and multiplication. Currently the sequential implementation takes 20 minutes to complete given a sample video of length 6s.

HOPE TO ACHIEVE: What we hope to achieve is to have a higher speedup (60x) if possible so that we can get much closer to real time processing.

During our demo, we will show a pre-recorded video that will describe the what our project is and the final product of our project. We will then show the speedup results of our project. Running the comparison during the demo may not be possible as the sequential version of the program takes 20 minutes to complete and given our speedup goal, it may still take minutes to run the parallel version of the program. However, we will show small demos for doing one frame or two to show the optimizations that we made in this project.

6 Platform Choice

We are going to use python as our programming language. While python is a slow language, there are a lot of computer vision libraries provided in python that we will need to utilize. Those libraries help reduce the amount of work that we need to do when implementing the sequential version of the program and since the point of this project is to parallelize the existing program we want to start out with the sequential version (written in python) that we implemented before and do parallel optimization instead of finding libraries in other languages and re-implement the whole sequential project.

We will be using a computer with GPU CUDA support to implement this project. The base project we worked on only utilized the CPU to process the whole image. Once we find the homography matrix, we will be doing matrix multiplication on the whole image. There are a lot of pixels in an image and computations are simple multiplication and add so GPU CUDA will be very useful when parallelizing the work in this case.

7 Schedule

1. Week 11.2 - 11.6
    - Discuss project idea with professor.
    - Finish project proposal.
2. Week 11.9 - 11.13

- Review sequential implementation.
- Environment and dependency setup to run project with Cuda.

3. Week 11.16 - 11.20
    - Parallelize over individual frames.
    - Finish project checkpoint report.
4. Week 11.23 - 11.27
    - Parallelize RANSAC: parallelize over each iteration to calculate homography matrix.
    - Parallelize RANSAC: parallelize over point correspondences to evaluate homography matrix.
5. Week 11.30 - 12.4
    - Further optimization.
    - Evaluate performance and speedup.
    - Start writing project report and poster.
6. Week 12.7 - 12.11
    - Finalize project.
    - Finish project report and poster for poster session.