Supplementary Material: Efficient Subpixel Refinement with Symbolic Linear Predictors

Vincent Lui, Jonathan Geeves, Winston Yii, and Tom Drummond

1 DERIVATION OF ERROR PREDICTION EQUATION

In Sec. 5.4 of the paper, we denote the predicted average error of a Linear Predictor using Eq. (26). Here, we show how this term is derived from Eq. (25), which is re-stated here as

$$\bar{\mathbf{e}}^2 = \mathrm{Tr}\left[(\mathbf{AE}-\mathbf{P})(\mathbf{AE}-\mathbf{P})^T\right], \tag{1}$$

where $\mathbf{A}$ is the Linear Predictor, $\mathbf{E}$ is the error matrix, and $\mathbf{P}$ is the warp matrix. Noting that $\mathbf{A} = (\mathbf{PE}^T)(\mathbf{EE})^{-1}$, and letting $\mathbf{X} = (\mathbf{PE}^T)$, $\mathbf{Z} = \mathbf{EE}$, we substitute $\mathbf{A} = \mathbf{XZ}^{-1}$ into (1) which yields

$$\bar{\mathbf{e}}^2 = \mathrm{Tr}\left[(\mathbf{XZ}^{-1}\mathbf{E}-\mathbf{P})(\mathbf{E}^T\mathbf{Z}^{-1}\mathbf{X}^T-\mathbf{P}^T)\right] \tag{2}$$

where we have used that $\mathbf{Z}$ is symmetric. Expanding the bracket gives

$$\bar{\mathbf{e}}^2 = \begin{array}{l}\mathrm{Tr}\left[\mathbf{XZ}^{-1}\mathbf{ZZ}^{-1}\mathbf{X}^T-\mathbf{X}-\mathbf{XZ}^{-1}\mathbf{X}^T-\mathbf{XZ}^{-1}\mathbf{L}^T+\mathbf{PP}^T\right]\\ \mathrm{Tr}\left[\mathbf{PP}^T-\mathbf{XZ}^{-1}\mathbf{X}^T\right]\end{array}. \tag{3}$$

Finally, substituting $\mathbf{A} = \mathbf{XZ}^{-1}$ and $\mathbf{X} = \mathbf{PE}^T$ back into (3), we obtain

$$\bar{\mathbf{e}}^2 = \mathrm{Tr}\left[\mathbf{PP}^T) - \mathrm{Tr}(\mathbf{A}(\mathbf{PE}^T)\right], \tag{4}$$

which is the result shown in the paper.

2 ADDITIONAL RESULTS FROM SYNTHETIC EXPERIMENT

In this section, we provide additional results from the synthetic data experiment (see Sec. 6.1 in the paper). Fig. 1(a) and (b) provides a deeper illustration on how the computational complexity of our method changes. From Fig. 1, we see that the number of non-zero coefficients to be computed increases very slowly as the number of sample warps increases, leading to an very slow increase in computational complexity. On the other hand, as the number of pixels used increases, the number of non-zero coefficients increases very quickly, leading to an increase in computational complexity. Fig. 1(c) shows a timing breakdown of the learning step on a CPU. We can

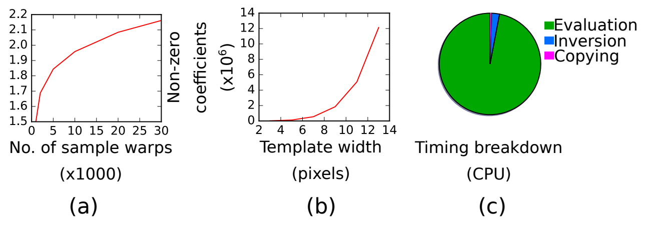

Figure 1: A deeper look at the computational complexity of our proposed method.

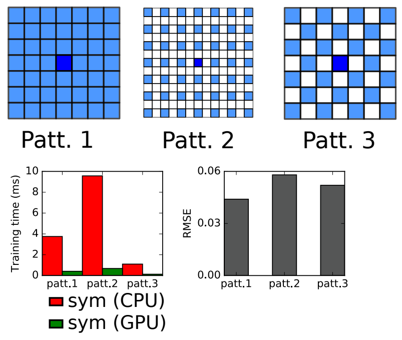

Figure 2: Influence of path sampling patterns on performance.

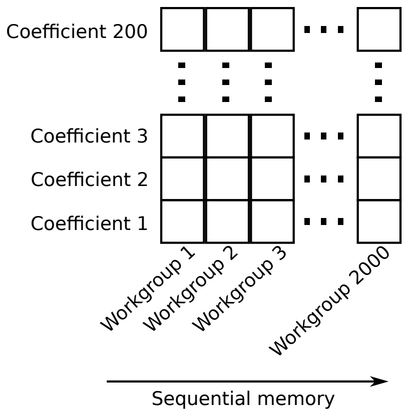

Figure 3: Memory layout of workgroup model for GPU calculation with 2000 workgroups and 200 entries per workgroup.

see that for a small image patch, most of the computational resources goes towards computing the linear predictor, and very little time goes towards matrix inversion.

Further, we also investigated the trade-off between accuracy and computational efficiency via different patch sampling methods. We experimented with three different patch sampling patterns as shown in Fig. 2. The result shows that while pattern 3 provides a good trade-off between computational complexity and accuracy.

3 GPU ACCELERATION

In this section we provide more details on the GPU implementation of our proposed method. Our proposed method involves computing the linear term $\mathbf{PE}^T$ from the tensor $\mathbf{Y}$ and the quadratic term $\mathbf{EE}^T$ from the tensor $\mathbf{Q}$, where each column along the depth dimension of the tensor consists of pixel indices and coefficient values. We store the image patch in constant memory, a small amount of memory, usually 64kB on newer cards, which allows faster access times compared to global memory. It is also possible to store the image patch in shared memory, and we found that the choice of either constant or shared memory is dependent on the specific card used. The pixel indices and their corresponding coefficient values are stored in global memory as each pixel index and coefficient value is only accessed once.

As each entry in the linear and quadratic term consists of variable sized linear and quadratic combinations respectively, we divide the computation into a number of workgroups, where each workgroup is responsible for the same number of computations. As the number of terms in the combination is not likely to be integrally divisible by the workgroup size, there will be one workgroup with more computations required. This is accounted for by just setting the associated coefficients in the workgroup to zero. Further, in

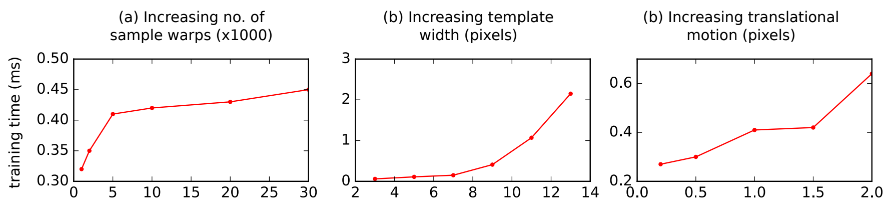

Figure 4: GPU timings for different training settings on an NVIDIA 1080 card.

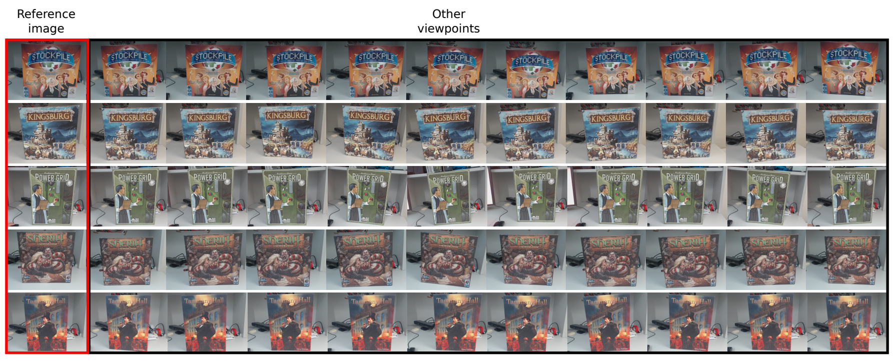

Figure 5: Thumbnails of data set used for error prediction experiment.

order to ensure efficient memory access, we design the GPU implementation such that threads executing in parallel are accessing sequential locations in memory. This means that sequential memory accesses the same coefficient and pixel value, but for different workgroups. An illustration of the memory layout is shown in Fig. 3.

Similar to the CPU implementation, the time required to train a Linear Predictor on the GPU depends on the number of non-zero coefficients. Fig. 4 shows the timings for the synthetic data experiment in the paper, obtained using an NVIDIA 1080 card with 10 workgroups for the linear term and 100 workgroups for the quadratic term. We see that the training time does not increase much with the number of sample warps, but it increases with the number of pixels used as well as the translational motion. This is in line with the timings of the CPU implementation.

4 ERROR PREDICTION DATASET

Finally, for completeness, here we provide the thumbnails of the images used for the final experiment in Sec. 6.3 of the paper, shown in Fig. 5 where the images on the left column were used to extract the keypoints and the image patches for training the Linear Predictors, and the other images were used to obtain point correspondences with the first image in the data set.