# Errata list:

## Page vi, figure 2.6

**Original text:** 2.6 Illustration a K-D tree in $R^2$ space where the nodes are inserted in alphabetical . . . . . . . . . . . . . . . . . . . . . . . . 11

**Corrected text:**

2.6 Illustration of a K-D tree in $R^2$ space where the nodes are inserted in alphabetical order . . . . . . . . . . . . . . . . . . . 11

## Page vii, figure 5.5, 5.6, 5.7, 5.8

**Original text:**

5.5 Shows time in seconds needed to complete all necessary cublasHgemm calls needed to multiply $10^6$ query points with $10^1$0 reference points, where the number of query points per call is in the range 40 - 400000 . . . . . . . . . . . . . . . . . . 60

5.6 Shows time in seconds needed to complete all necessary cublasHgemm calls needed to multiply $10^6$ query points with $10^1$0 reference points, where the number of query points per call is in the range 40 - 29729 . . . . . . . . . . . . . . . . . . . 61

5.7 Shows time in seconds needed to complete all necessary cublasHgemm calls needed to multiply $10^4$ query points with $10^1$0 reference points, where the number of query points per call is in the range 40 - 10500 . . . . . . . . . . . . . . . . . . . 61

5.8 Shows time in seconds needed to complete all necessary cublasHgemm calls needed to multiply $10^4$ query points with $10^1$0 reference points, where the number of query points per call is in the range 1500 - 2000, and we average run time over 100 runs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62

**Corrected text:**

5.5 Shows time in seconds needed to complete all necessary cublasHgemm calls needed to multiply $10^6$ query points with $10^4$ reference points, where the number of query points per call is in the range 40 - 400000 . . . . . . . . . . . . . . . . . . 60

5.6 Shows time in seconds needed to complete all necessary cublasHgemm calls needed to multiply $10^6$ query points with $10^4$ reference points, where the number of query points per call is in the range 40 - 29729 . . . . . . . . . . . . . . . . . . . 61

## Page 13, 14

**Original text:**

To take advantage of the cuBLAS library, the calculation of the squared Euclidean distance $p^2$ from point x to y is done:

$$p^2(x, y) = (x - y)^T(x - y) = ||x||^2 + ||y||^2 - 2x^T y$$

where the T is the transpose. If we rewrite it for two matrices, R which is a $d \times m$ matrix and Q which is a $d \times n$ matrix, we get the equation:

$$p^2(N_R, N_Q) = N_R + N_Q - 2R^T Q$$

here N stands for the norm, and the resulting matrix $p^2(N_R, N_Q)$ would be a $m \times n$ matrix. The GPU brute force algorithm which uses 8 kernels with cuBLAS and CUDA, goes as follows:

1. Kernel 1 calculates $N_R$ using CUDA (coalesced read/write)
2. Kernel 2 calculates $N_Q$ using CUDA (coalesced read/write)
3. Kernel 3 uses cuBLAS to calculate $-2R^T Q$ we call the resulting matrix A
4. kernel 4 adds all elements in $N_R$ to A we call the new matrix B. this is done with CUDA threads and shared memory.
5. kernel 5 sorts the B matrix in parallel with n threads. giving us the matrix C
6. kernel 6 adds the $j^{th}$ element of $N_Q$ to the k first elements of the $j^{th}$ column of the matrix C. this is done using CUDA (coalescedread/write). We call the new matrix D
7. Kernel 7 computes the square root of the first k elements in D, this gives us the k smallest distances, this is done using CUDA (coalesced read/write). We call the new matrix E

8. The last kernel extracts the $k \times n$-submatirx from E. This is the matrix of the distances for each query point.

**Corrected text:**

To take advantage of the cuBLAS library, the calculation of the squared Euclidean distance $d^2$ from point x to y is done:

$$d^2(x, y) = (x - y)^2(x - y) = ||x||^2 + ||y||^2 - 2xy$$

If we rewrite for two matrices, R which is a $d \times m$ matrix and Q which is a $d \times n$ matrix, we see that we can find the squared distance between any point in R and Q by doing:

$$d^2(R_x, Q_y) = ||R_x||^2 + ||Q_y||^2 - 2R_xQ_y$$

We see that the squared distance is calculated from the squared norm of the two points, and the dot product between them multiplied by 2. To do the latter we can perform the matrix multiplication $-2R^TQ$. This can be done with one call to cuBLAS. The GPU brute force algorithm which uses 8 kernels with cuBLAS and CUDA, goes as follows:

1. Kernel 1 calculates the squared norm of every vector in R using CUDA (coalesced read/write)

2. Kernel 2 calculates the squared norm of every vector in Q using CUDA (coalesced read/write)

3. Kernel 3 uses cuBLAS to calculate $-2R^TQ$ we call the resulting matrix A

4. kernel 4 adds the squared norms of every vector in R (calculated in kernel 1) to the corresponding values in A we call the new matrix B. This is done with CUDA threads and shared memory.

5. kernel 5 sorts the B matrix in parallel with n threads. giving us the matrix C

6. kernel 6 adds the $j^{th}$ element of the vector we got from kernel 2 to the k first elements of the $j^{th}$ column of the matrix C. this is done using CUDA (coalesced read/write). We call the new matrix D

7. Kernel 7 computes the square root of the first k elements in D, this gives us the k smallest distances, this is done using CUDA (coalesced read/write). We call the new matrix E

8. The last kernel extracts the $k \times n$-submatirx from E. This is the matrix of the distances for each query point.

## Page 32

**Original text:**

$$d(X,Y)^2 = (X-Y)^2(X-Y) = ||X||^2 + ||Y||^2 - 2X^TY$$

Here $||\cdot||$ represents the Euclidean norm, and $X^T$ is the transpose of X. To rewrite for sets of vectors, we think of Q and R as two sets of vectors, e.g. $Q = \{v_1,..,v_i\}$ and $R = \{w_1,..,w_j\}$. $v_1,...,v_i$ and $w_1,..,w_j$ are vectors in a D-dimensional Euclidean space. Dist is the vector of squared distances between any vector in Q to any vector in R. We can then write Dist as:

$$Dist = ||Q||^2 + ||R||^2 - 2Q^TR$$

### 3.1.2 cuBLAS for SIFT vector matching

Here an important thing to note is that we are dealing with SIFT vectors, not arbitrary data. SIFT vectors are normalized, in our case to 1. This means that we already know both $||Q||^2$ and $||R||^2$ as this it the distance from the zero vector to the vector. For matrices of size $D\times i$ like Q, the norm $||\cdot||$ is defined as $||Q|| = Q^TQ$, which is an i-dimensional vector. The definition means that each row $v_n$ in $Q^T$ is multiplied with the same column $v_n$ in Q, which is the same as $v_n \cdot v_n = ||v_n||^2$, which we know is 1. We can therefore reduce our problem even more giving us the equation

$$Dist = 2 - 2Q^TR$$

$-2Q^TR$ is a matrix multiplication and can be done extremely well with one call to the cuBLAS library. We only need the 2NN, meaning we can simplify what we do into 2 steps:

1. Calculate $-2Q^TR$
2. Find the two smallest values for each query vector

**Corrected text:**

1. $d(X,Y)^2 = (X-Y)^2(X-Y)$
2. $d(X,Y)^2 = (x_1^2 + x_2^2 + ... + x_D^2) + (y_1^2 + y_2^2 + ... + x_D^2) - 2(x_1y_1 + x_2y_2 + ... + x_Dy_D)$
3. $d(X,Y)^2 = ||X||^2 + ||Y||^2 - 2X\cdot Y$

Here $||\cdot||$ represents the Euclidean norm. To rewrite for sets of vectors, we think of Q and R as two sets of vectors, e.g. $Q = \{v_1,..,v_i\}$ and $R = \{w_1,..,w_j\}$. $v_1,...,v_i$ and $w_1,..,w_j$ are vectors in a D-dimensional Euclidean space. The squared distance between any vector in Q to any vector in R can then be written as:

$$d(Q_x, R_y)^2 = ||Q_x||^2 + ||R_y||^2 - 2Q_x \cdot R_r$$

The distance is calculated by adding the squared norm of the 2 vectors, and subtracting the dot product multiplied by 2. The dot product between every vector in 2 matrices can be calculated with one matrix - matrix multiplication e.g

$QR^T = \{\{v_1 \cdot w_1, v_1 \cdot w_2, ..., v_1 \cdot w_j\}, \{v_2 \cdot w_1, v_2 \cdot w_2, ..., v_2 \cdot w_j\}, ..., \{v_i \cdot w_1, v_i \cdot w_2, ..., v_i \cdot w_j\}\}$ This can be done with one call to cuBLAS.

### 3.1.2 cuBLAS for SIFT vector matching

Here an important thing to note is that we are dealing with SIFT vectors, not arbitrary data. SIFT vectors are normalized, in our case to 1. This means that we already know both $||Q_x||^2$ and $||R_y||^2$ for every value of x and y, as $1^2 = 1$.

We can therefore reduce our problem even more giving us the equation:

$$d(Q_x, R_y)^2 = 1 + 1 - 2Q_x R_y$$

We see that to find the distance between every point in Q and every point in R we need to perform the matrix multiplication $-2Q^T R$, this can be done extremely well with one call to the cuBLAS library. We only need the 2NN, meaning we can simplify what we do into 2 steps:

1. Calculate $-2Q^T R$
2. Find the two smallest values from each row of the matrix from step 1

## Page 75, Table 5.14

**Original text:**
**Results**

| Time in seconds used on the ANN_SIFT1M data set to achieve 0.8 recall | | |
|---|---|---|
| | Recall @ 1 | Recall @ 100 |
| Total time used | 12.728130s | 1.897476s |
| Short-list brute-force | 10.543828s, 0.82% | 1.427292s , 0.75.2% |
| Thrust sort index array | 1.435962s, 0.11% | 0.286857s, 0.15.1% |
| Setting bits in hash value | 0.242682s, 0.019% | 0.04392s, 0.023% |
| Dot product with cuBLAS | 0.207879s, 0.016% | 0.037911s, 0.019% |
| CPU and data movement/allocation | 0.297779s, 0.023% | 0.101496s, 0.053% |

Table 5.14: Shows approximately the time used by the different parts of the LSH algorithm

**Corrected text:**

**Results**

| Time in seconds used on the ANN_SIFT1M data set to achieve 0.8 recall | | |
|---|---|---|
| | Recall @ 1 | Recall @ 100 |
| Total time used | 12.728130s | 1.897476s |
| Short-list brute-force | 10.543828s, 82% | 1.427292s , 75.2% |
| Thrust sort index array | 1.435962s, 11% | 0.286857s, 15.1% |
| Setting bits in hash value | 0.242682s, 1.9% | 0.04392s, 2.3% |
| Dot product with cuBLAS | 0.207879s, 1.6% | 0.037911s, 1.9% |
| CPU and data movement/allocation | 0.297779s, 2.3% | 0.101496s, 5.3% |

Table 5.14: Shows approximately the time used by the different parts of the LSH algorithm