2010 First International Conference on Networking and Computing

Softassign and EM-ICP on GPU

Toru Tamaki, Miho Abe, Bisser Raytchev, Kazufumi Kaneda
Hiroshima University, Japan
tamaki@ieee.org

***Abstract*—In this paper we propose CUDA-based implementations of two 3D point sets registration algorithms: Softassign and EM-ICP. Both algorithms are known for being time demanding, even on modern multi-core CPUs. Our GPUbased implementations vastly outperform CPU ones. For instance, our CUDA EM-ICP aligns 5000 points in less than 7 seconds on a GeForce 8800GT, while the same implementation in OpenMP on an Intel Core 2 Quad would take 7 minutes.**

I. Motivation

Registration (alignment) of 3D point sets is one of the most important problems in computer vision and several methods have been developed over the last two decades. The widely used Iterative Closest Point (ICP) algorithm [1] provides quick registration, but requires a good initial alignment in order to prevent local minima and produce a plausible match. Softassign [2] and EM-ICP [3] represent efforts to overcome such limitations: instead of looking for "hard" correspondences between points (each point in one of the sets has to uniquely map to another point in the other set), the latter two algorithms focus on "soft" correspondences (each point in one of the sets corresponds somehow to every point in the other set by some weight). Although these algorithms can handle any initial arrangement, their associated computational cost has been preventing their practical usefulness even for moderately large number of points.

Recent advances in graphics hardware and software [14], [15] have motivated us to implement Softassign and EM-ICP on a GPU and evaluate their corresponding behavior and performance. Our contribution is twofold: we introduce the GPU implementations and also demonstrate that most steps of these algorithms are GPU-friendly, consisting of either vector-matrix multiplications or element-wise operations.

The rest of the paper [1] is organized as follows. In section 2, we review alignment algorithms: ICP, Softassign, and EM-ICP. Also, estimations of optimal transformation using Horn's method are described. Then we discuss how these algorithms are implemented in CUDA in section 3. Performance evaluations are given in section 4 and section 5 concludes the paper.

II. Algorithm review

The goal of alignment algorithms is to find rotation matrix R and translation vector t that align two sets of 3D points $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{n_x}\}$ and $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{n_y}\}$: the rigid transformation minimizes the residual error between X (fixed) and Y (transformed by R and $\boldsymbol{t}$).

Here we describe three alignment algorithms, ICP [1], Softassign [2], and EM-ICP [3]. These methods differ in their strategies for point correspondence establishment: ICP employs *hard* correspondence while Softassign and EM-ICP use *soft* correspondence. Once point correspondences are established, the rigid transformation R and $\boldsymbol{t}$ are estimated by using a rigid transformation estimation method such as Horn's quaternion method [10].

A. ICP

Iterative closet point (ICP) algorithm [1] is an alignment method proposed for free form surface registration. It takes two point sets X and Y and find R and $\boldsymbol{t}$ as following procedure.

Algorithm 1 ICP

1: $R^0 \leftarrow I$, $\boldsymbol{t}^0 \leftarrow \boldsymbol{0}$.
2: **for** $k = 1, \ldots,$ `maxIterations` **do**
3: **for** each point $\boldsymbol{x}_i$ in X **do**
4: find the closest point $\boldsymbol{y}_{i*}$ in Y:

$$i* = \underset{j=1,\ldots,n_y}{\operatorname{argmin}} ||\boldsymbol{x}_i - (R^{k-1}\boldsymbol{y}_j + \boldsymbol{t}^{k-1})||. \quad (1)$$

5: **end for**
6: build the ordered correspondence set $Y^* = \{\boldsymbol{y}_{1*}, \ldots, \boldsymbol{y}_{n_x*}\}$.
7: find the rigid transformation $R^*, \boldsymbol{t}^*$ that minimizes mean squared error between X and Y^*.
8: $R^k \leftarrow R^*$, $\boldsymbol{t}^k \leftarrow \boldsymbol{t}^*$.
9: **end for**

Some implementations of ICP on GPUs [7], [8], [9], however, main problem of ICP is the convergence is not satisfactory.

[1] The developed software was presented at CVPR2010 Demo [13].

978-0-7695-4277-5/10 $26.00 © 2010 IEEE
DOI 10.1109/IC-NC.2010.60

B. Softassign

Softassign [2] takes two point sets X and Y' and find R and $\boldsymbol{t}$ that minimizes the error function:

$$E = \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} m_{ij}||\boldsymbol{x}_i - (R^{k-1}\boldsymbol{y}'_j + \boldsymbol{t}^{k-1})|| - \alpha\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} m_{ij} - \frac{1}{\beta}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} m_{ij}(\log m_{ij} - 1), \quad (2)$$

where m_{ij} is the i-jth element of a weight correspondence matrix M, and α and β are constants. Note that M has additional row and column for dealing with outliers in correspondences.

The algorithm of Softassign is as follows:

Algorithm 2 Softassign

$R^0 \leftarrow I$, $\boldsymbol{t}^0 \leftarrow \boldsymbol{0}$, $k \leftarrow 0$.
for all i, $m_{i,n_y+1} \leftarrow$ `moutlier`, for all j, $m_{n_x+1,j} \leftarrow$ `moutlier`.
for $\text{iteration}_J = 1, \ldots,$ `JMAX` **do**
  for $\text{iteration}_I = 1, \ldots,$ `I0` **do**
    for each i, j compute

$$m_{ij} \leftarrow \exp\left(-\beta\left(\frac{\partial E}{\partial m_{ij}} - \alpha\right)\right), \quad (3)$$

    where

$$\frac{\partial E}{\partial m_{ij}} = \boldsymbol{x}_i - (R^{k-1}\boldsymbol{y}_j + \boldsymbol{t}^{k-1}). \quad (4)$$

    for $\text{iteration}_{\text{Shinkhorn}} = 1, \ldots,$ `I1` **do**
      normalize rows i:

$$m_{ij} \leftarrow \frac{m_{ij}}{\sum_{j=1}^{n_y+1} m_{ij}} \quad (5)$$

      normalize columns j:

$$m_{ij} \leftarrow \frac{m_{ij}}{\sum_{i=1}^{n_x+1} m_{ij}} \quad (6)$$

    end for
    find the rigid transformation $R^*, \boldsymbol{t}^*$ that minimizes E error weighted by M between X and Y.
    $R^k \leftarrow R^*$, $\boldsymbol{t}^k \leftarrow \boldsymbol{t}^*$, $k \leftarrow k+1$.
  end for
end for

C. EM-ICP

EM-ICP [3] takes two point sets X and Y and find R and $\boldsymbol{t}$ that minimizes the error function:

$$E = \sum_{j=1}^{n_x}\sum_{i=1}^{n_y} \alpha_{ij} d_{ij}^2,\ d_{ij} = ||\boldsymbol{x}_j - (R\boldsymbol{y}_i + \boldsymbol{t})||, \quad (7)$$

where α_{ij} is the probability that $\boldsymbol{x}_j$ matches to $\boldsymbol{y}_i$ and given by

$$\alpha_{ij} = \frac{1}{C_i}\exp\left(\frac{-d_{ij}^2}{\sigma_p^2}\right), \quad (8)$$

$$C_i = \exp\left(\frac{-d_0^2}{\sigma_p^2}\right) + \sum_{k=1}^{n_x+1}\exp\left(-\frac{-d_{ik}^2}{\sigma_p^2}\right), \quad (9)$$

where σ_p and d_0 are constants.

The error function can be rewritten as [4]:

$$E = \sum_{i=1}^{n_y} \lambda_i^2 ||\boldsymbol{x}'_i - (R\boldsymbol{y}_i + \boldsymbol{t})||, \quad (10)$$

where

$$\lambda_i = \sum_{j=1}^{n_x}\sqrt{\alpha_{ij}},\ \boldsymbol{x}'_i = \frac{1}{\lambda_i}\sum_{j=1}^{n_x}\sqrt{\alpha_{ij}}\boldsymbol{x}_j. \quad (11)$$

By following [5] the algorithm of EM-ICP is as follows:

Algorithm 3 EM-ICP

$R^0 \leftarrow I$, $\boldsymbol{t}^0 \leftarrow \boldsymbol{0}$, $k \leftarrow 0$. $\sigma_p \leftarrow$ `sigam_p`. $\sigma_{\text{recon}} \leftarrow$ `sigam_inf`. $d_0^2 \leftarrow$ `d_02`.
while $\sigma_p > \sigma_{\text{recon}}$ **do**
  for each i, j compute α_{ij} with $R^{k-1}, \boldsymbol{t}^{k-1}$.
  find the rigid transformation $R^*, \boldsymbol{t}^*$ that minimizes E error weighted by λ_i.
  $R^k \leftarrow R^*$, $\boldsymbol{t}^k \leftarrow \boldsymbol{t}^*$, $k \leftarrow k+1$.
  $\sigma_p \leftarrow \sigma_p \times$ `sigma_factor`
end while

III. Estimation of R and $\boldsymbol{t}$

Given correspondences of points with or without weights, the rigid transformation is usually computed by Horn's method [10] using quaternions. Readers who are interested in other methods, refer a review [12]. First we describe the original version that does not take weights and requires *hard* correspondences.

A. Horn's method

Assume that two corresponded point sets $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ and $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n\}$ are given. Then the optimal rigid transformation minimizes the following function:

$$\sum_{i=1}^{n} ||\boldsymbol{x}_i - (R\boldsymbol{y}_i + \boldsymbol{t})||. \quad (12)$$

Usually $\boldsymbol{t}$ is estimated by the difference between centers of point sets. R is then estimated separately.

Horn [10] proposed a method for estimating R between $\hat{X}$ and $\hat{Y}$ by using quaternions.

$$K = \begin{pmatrix} S_{xx}+S_{yy}+S_{zz} & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ S_{yz}-S_{zy} & S_{xx}-S_{yy}-S_{zz} & S_{xy}+S_{yz} & S_{zx}+S_{xz} \\ S_{zx}-S_{xz} & S_{xy}+S_{yz} & S_{yy}-S_{xx}-S_{zz} & S_{yz}-S_{zy} \\ S_{xy}-S_{yx} & S_{zx}+S_{xz} & S_{yz}-S_{zy} & S_{zz}-S_{xx}-S_{yy} \end{pmatrix}. \quad (15)$$

Algorithm 4 Estimation of R and $\boldsymbol{t}$

1: Let $\hat{X} = \{\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_n\}$ and $\hat{Y} = \{\hat{\boldsymbol{y}}_1, \hat{\boldsymbol{y}}_n\}$. Here, $\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i - \bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$ is the center of gravity of the point set.
2: Estimate R between $\hat{X}$ and $\hat{Y}$.
3: Let $\boldsymbol{t} \leftarrow \bar{x} - R\bar{y}$.

Algorithm 5 Horn's quaternions method

1: Let

$$S = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix}, \quad (13)$$

where

$$S_{ab} = \sum_{i=1}^{n} x_{ia} y_{ib}, \quad a, b \in \{x, y, z\}. \quad (14)$$

Here, x_{iz} means the z coordinate of point $\boldsymbol{x}_i$, and so on.
2: Construct the following real-symmetric matrix K in Eq.(15).
3: Compute the eigenvector $\boldsymbol{q}$ corresponding to the maximum eigenvalue of K. This is a unit quaternion.
4: Convert the unit quaternion $\boldsymbol{q}$ to rotation matrix R.

B. Weighted version

Horn's method can be extended to handle with weights for each correspondences of difference size of point sets.

Assume that two corresponded point sets $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ and $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n\}$ are given, and also given weights λ_i for each correspondence $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$. Then the Horn's method is modified [11] in following two points:

1) centers of point sets: computing the center of X now takes into the weights as

$$\bar{\boldsymbol{x}} = \frac{1}{\sum_{i=1}^{n} \lambda_i} \sum_{i=1}^{n} \lambda_i \boldsymbol{x}_i, \quad (16)$$

and so as $\bar{y}$.
2) elements in S: now $S_{xx}, \ldots$ are computed as

$$S_{ab} = \sum_{i=1}^{n} \lambda_i x_{ia} y_{ib}. \quad (17)$$

Modification is similar even when two point sets has different number of points. Assume that two corresponded point sets $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{n_x}\}$ and and $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{n_y}\}$ are given, and also given weights m_{ij} for each correspondence $\{\boldsymbol{x}_i, \boldsymbol{y}_j\}$. Then the Horn's method is modified in following two points:

1) centers of point sets:

$$\bar{\boldsymbol{x}} = \frac{1}{N} \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} m_{ij} \boldsymbol{x}_i, \; \bar{\boldsymbol{y}} = \frac{1}{N} \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} m_{ij} \boldsymbol{y}_i, \quad (18)$$

where

$$N = \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} m_{ij}. \quad (19)$$

2) elements in S:

$$S_{ab} = \frac{1}{N} \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} m_{ij} x_{ia} y_{jb}. \quad (20)$$

IV. GPU IMPLEMENTATION: KEY IDEA

Softassign and EM-ICP involve a lot of loops and summations. The key idea to achieve the best performance of implementations Softassign and EM-ICP on GPU is separate vector-matrix computation and element-wise computation in the original algorithms. In our implementation, CUDA Basic Linear Algebra Subprograms (CUBLAS) [17], an implementation of BLAS[16] on top of CUDA, is used to accelerate the vector-matrix computation part, while CUDA kernel computation is employed for element-wise computation.

In the following subsections, we describe how matrix S is computed by Softassign and EM-ICP with CUDA.

A. Softassign with CUDA

The most computationally intensive parts of Softassign algorithm is the Shinkhorn iteration as well as the computation of M.

1) Computing M: Elements m_{ij} of M can be computed in parallel since it just involves the distance between $\boldsymbol{x}_i$ and $\boldsymbol{y}_j$.

To maximize the use of the shared memory on a GPU, two points for implementations should be taken into account. First, R and $\boldsymbol{t}$ are first loaded on a shared memory because they are common for computations of all elements. Second, $\boldsymbol{x}_i$ are fixed when computing m_{ij} for all $\boldsymbol{y}_j$ (or vice versa). Therefore, $\boldsymbol{x}_i$ is loaded and stored in shared memory once for each i in a block, which is a unit where streaming processors on a GPU run in parallel.

2) Shinkhorn iteration: Shinkhorn iteration repeats normalization of rows and columns in turn. In terms of vector-matrix computation, the normalization can be seen as the multiplication of M and a vector whose elements are all 1.

We divide the normalization of rows i in M:

$$m_{ij} \leftarrow \frac{m_{ij}}{\sum_{j=1}^{n_y+1} m_{ij}} \tag{21}$$

as follows.

1) $M\mathbf{1} \rightarrow \boldsymbol{R}_M$, where $\mathbf{1}$ is a column vector with all elements 1, and $\boldsymbol{R}_M$ is a column vector whose elements are the sums of each row of M. This can be easily performed by `sgemv` in BLAS.
2) $\boldsymbol{R}_o + \boldsymbol{R}_M \rightarrow \boldsymbol{R}_M$, where $\boldsymbol{R}_o$ is a column vector which corresponds to outliers. This can be performed by `saxpy` in BLAS.
3) divide row i of M by i-th element of $\boldsymbol{R}_M$. This can be implemented by CUDA kernel to compute in parallel.

The column normalization is also implemented in the same way with the row normalization.

3) Centering point sets: To compute translation $\boldsymbol{t}$, point sets X and Y are centered by subtracting weighted centers of gravity. This can be also done by using results of the row/column normalizations.

By definition,

$$\bar{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} m_{ij}\boldsymbol{x}_i, \tag{22}$$

$$= \frac{1}{N}\sum_{i=1}^{n_x}(\boldsymbol{x}_i \sum_{j=1}^{n_y} m_{ij}) = \frac{1}{N}\sum_{i=1}^{n_x} \boldsymbol{x}_i R_{Mi}, \tag{23}$$

where R_{Mi} is the i-th element of $\boldsymbol{R}_M$. Also,

$$N = \sum_{i=1}^{n_x} R_{Mi}. \tag{24}$$

Hence, $\bar{\boldsymbol{x}}$ can be computed by re-using $\boldsymbol{R}_M$ that is stored at the normalization step. Eq.(23) can be performed by using element-wise computation with CUDA kernel ($\boldsymbol{x}_i R_{Mi} \rightarrow \boldsymbol{x}_i$) followed by summing up the elements by inner product ($\boldsymbol{x}^T\mathbf{1}$ with `sasum` in BLAS). N can be also computed by inner product ($\boldsymbol{R}_M^T\mathbf{1}$).

Centered point sets $\hat{X}, \hat{Y}$ are then performed by subtracting each element in X (or Y) by $\bar{x}$ (or $\bar{y}$) with CUDA kernel.

4) Computing S: If we arrange the way how the point sets are stored in memory, computing matrix S is very simple.

Elements in S are of the form as:

$$S_{ab} = \frac{1}{N}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} m_{ij}x_{ia}y_{jb}. \tag{25}$$

Therefore, S consists of the following matrix and vectors:

$$S = \hat{X}^T M \hat{Y}, \tag{26}$$

where X and Y are matrices of 3D points:

$$\hat{X} = \begin{pmatrix} \hat{\boldsymbol{x}}_1^T \\ \hat{\boldsymbol{x}}_2^T \\ \vdots \\ \hat{\boldsymbol{x}}_{n_x} \end{pmatrix}, \quad \hat{Y} = \begin{pmatrix} \hat{\boldsymbol{y}}_1^T \\ \hat{\boldsymbol{y}}_2^T \\ \vdots \\ \hat{\boldsymbol{y}}_{n_y} \end{pmatrix}. \tag{27}$$

This format means that x coordinates of all points are stored in the first (left most) column, then y and z coordinates stored in the second (middle) and third column. Since it facilitate the vector-matrix multiplication, S can be computed by the use of BLAS `sgemm` twice: $M\hat{Y} \rightarrow D$ then $\hat{X}^T D \rightarrow S$, where D is a matrix temporary used.

5) Estimating R and $\boldsymbol{t}$: Once S is computed, R is computed by Horn's method, then $\boldsymbol{t}$ is obtained as $\bar{x} - R\bar{y}$. Here, $\bar{x}$ and $\bar{y}$ are centers weighted by M.

B. EM-ICP with CUDA

The implementation of EM-ICP algorithm is similar to that of Softassign.

1) Computing d_{ij}: Elements d_{ij} can be computed in parallel since it just involves the distance between $\boldsymbol{x}_i$ and $\boldsymbol{y}_j$. In the following, we denote a matrix $A = (\alpha_{ij}) = (\exp(-d_{ij}^2/\sigma_p^2))$. A can be computed with CUDA kernel in the same way of computing M for Softassign.

2) Computing C_i: Coefficients C_i involve the computation of summing rows of A and adding a constant. This can be done in two steps below. First, rows of A are summed by BLAS `sgemv`: $A\mathbf{1} \rightarrow \boldsymbol{C}$, where $\boldsymbol{C} = (C_1, \ldots, C_{n_y})^T$. Second, BLAS `saxpy` is used to compute $\exp(\frac{-d_0^2}{\sigma_p^2})\mathbf{1} + \boldsymbol{C} \rightarrow \boldsymbol{C}$.

Then, each row of A is divided by $\boldsymbol{C}$ and square-rooted for performing $A = (\sqrt{\alpha_{ij}/C_i})$ with CUDA kernel.

3) Pseudo correspondence $\boldsymbol{x}'$: Unlike Softassign, EM-ICP do not need row/column normalization of A. Instead, correspondence to $\boldsymbol{y}_i$ is established to all $\boldsymbol{x}_i$ with weights A.

First, $A\mathbf{1} \rightarrow \boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{n_y})^T$ is computed with BLAS `sgemv`. Also $N = \sum_{i=1}^{n} \lambda_i$ is computed here with BLAS `sasum`. Second, $\boldsymbol{x}'_i$ is computed by multiplication of A and X: $AX \rightarrow X'$ with BLAS `sgemm`. Here points are stored in X as described in "computing S" for Softassign. Then, each element $\boldsymbol{x}'_i$ in X' is divided by λ_i with CUDA kernel.

4) Centering point sets: Centering of point sets for computing translation $\boldsymbol{t}$ is similar to the way for Softassign.

The weighted center is now given as

$$\bar{\boldsymbol{x}}' = \frac{1}{N}\sum_{i=1}^{n_y} \boldsymbol{x}'_i\lambda_i, \quad \bar{\boldsymbol{y}} = \frac{1}{N}\sum_{i=1}^{n_y} \boldsymbol{y}_i\lambda_i, \tag{28}$$

implemented with BLAS `sgemv` followed by the division by N with BLAS `sscal`.

Centered point sets $\hat{X}', \hat{Y}$ are then performed by subtracting each element in X' (or Y) by $\bar{x}'$ (or $\bar{y}$) with CUDA kernel.

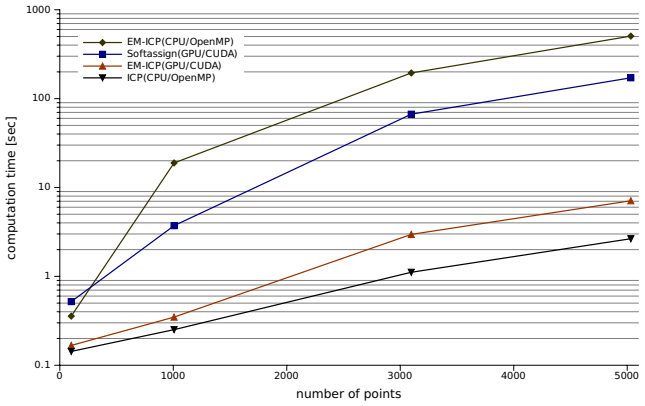

Figure 1. Computing time over different number of points.

5) Computing S: Since EM-ICP uses weights λ_i for correspondence between $\boldsymbol{y}_i$ and $\boldsymbol{x}'_i$, computing S becomes easier than that for Softassign.

If we would compute S in the same way for Softassign, the following matrix multiplication is needed:

$$S = \hat{X}^T \mathrm{diag}(\boldsymbol{\lambda}) \hat{Y}, \tag{29}$$

where $\mathrm{diag}(\boldsymbol{\lambda})$ is the diagonal matrix whose diagonal elements are $\lambda_1, \ldots, \lambda_{n_y}$. This is valid, but not efficient.

Instead, we compute S in two steps. First, $\hat{X}'$ is weighted by $\boldsymbol{\lambda}$: $\hat{\boldsymbol{x}}'_i \lambda_i \rightarrow \hat{x}'_i$. This is done by CUDA kernel. Second, S is computed by $\hat{X}'^T \hat{Y} \rightarrow S$ with BLAS `sgemm`.

6) Estimating R and $\boldsymbol{t}$: Once S is computed, R is computed by Horn's method, then $\boldsymbol{t}$ is obtained as $\bar{x} - R\bar{y}$. Here, $\bar{x}$ and $\bar{y}$ are centers weighted by $\boldsymbol{\lambda}$.

V. Performance evaluation

We have tested our CUDA-based implementations of Softassign and EM-ICP [13], and compared to CPU-implementation of ICP and EM-ICP. We have used OpenMP for CPU-implementation to compute element-wise operations on multiple threads in parallel. Two sets of points randomly taken from one of the bunny dataset [6] are aligned to measure the computing time. Movies of the convergence of these registrations are available at http://home.hiroshima-u.ac.jp/tamaki/study/cuda_softassign_emicp/. Code is also available there.

As shown in Fig.fig:computingtime, our CUDA EM-ICP aligns 5000 points in less than 7 seconds on a GeForce 8800GT, while the same implementation in OpenMP on an Intel Core 2 Quad would take 7 minutes. ICP takes less than 2 or 3 seconds, however, the alignment result is not satisfactory.

VI. Conclusions

In this paper we have proposed CUDA-based implementations of Softassign and EM-ICP for 3D point sets alignment. EM-ICP on CUDA is 60 times faster than OpenMP-based implementations on a multi-core CPU.

It is worth to note two limitations of our implementations. First, the number of points is limited due to the small amount of memory on a consumer-price GPU. Even if a GPU has 512MB memory, two sets of 10000 points are not aligned. However, this is not problematic in practice because randomly reducing the number of points allows the implementations work fine. Second, the stopping condition is not implemented. Softassign algorithm iteratively estimate M as well as R and $\boldsymbol{t}$, hence, the iteration can be stopped when M converges. However, it requires storing the entire M and checking whether the current estimate of M is close to the previous M stored. This is obviously not efficient, therefore we omit in our implementation.

Acknowledgment

We thank Marcos Slomp at Hiroshima University for his nice implementation of interactive visualization of 3D points.

References

[1] Paul J. Besl, Neil D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 2, pp. 239-256, 1992.

[2] Steven Gold, Anand Rangarajan, Chien-Ping Lu, Suguna Pappu, Eric Mjolsness, "New algorithms for 2D and 3D point matching: pose estimation and correspondence," *Pattern Recognition*, Vol. 31, No. 8, pp. 1019-1031, 1998.

[3] Sebastien Granger, Xavier Pennec, "Multi-scale EM-ICP: A Fast and Robust Approach for Surface Registration," *in Proc. of 7th European Conference on Computer Vision (ECCV2002)*, Vol. 4, pp. 69-73, 2002.

[4] Guillaume Dewaele, Frederic Devernay, Radu Horaud, "Hand motion from 3D point trajectories and a smooth surface model," *in Proc. of 8th European Conference on Computer Vision (ECCV2004)*, Vol. 1, pp. 495-507, 2004.

[5] Yonghuai Liu, "Automatic registration of overlapping 3D point clouds using closest points," *Image and Vision Computing*, Vol. 24, No. 7, pp. 762-781, 2006.

[6] The Stanford Bunny, Stanford University Computer Graphics Laboratory, The Stanford 3D Scanning Repository.

[7] Deyuan Qiu, Stefan May, Andreas Nüchter, "GPU-accelerated Nearest Neighbor Search for 3D Registration," *in Proc. of The 7th International Conference on Computer Vision Systems (ICVS 2009)*, p. 194-203, 2009.

[8] Sung-In Choi, Soon-Yong Park, Jun Kim, Yong-Woon Park, "Multi-view Range Image Registration using CUDA," *in Proc. of The 23rd International Technical Conference on Circuits/Systems, Computers and Communications*, pp. 733-736, 2008.

[9] Marcel Germann, Michael D. Breitenstein, Hanspeter Pfister, In Kyu Park, "Automatic Pose Estimation for Range Images on the GPU," *in Proc. of the Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM2007)*, pp. 81-90, 2007.

[10] Berthold K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Journal of the Optical Society of America*, Vol. 4, pp. 629–642, 1987.

[11] Kenichi Kanatani, "Analysis of 3-D rotation fitting," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 16, No. 5, pp. 543–549, 1994.

[12] Toru Tamaki, "Pose Estimation and Rotation Matrices," *IEICE Technical Report SIS2009-23*, Vol. 109, No. 203, pp. 59-64, 2009.

[13] Toru Tamaki, Miho Abe, Bisser Raytchev, Kazufumi Kaneda, Marcos Slomp, "CUDA-based implementations of Softassign and EM-ICP," *in Proc. of the 23rd IEEE Conference on Computer Vision and Pattern Recognition (CVPR2010) CD-ROM*, 2009.

[14] Justin Hensley, Oleg Maslov, Nicolas Pinto, "Computer Vision on GPUs," *Course at IEEE CVPR2009*, 2009.

[15] "GPGPU: general-purpose computation on graphics hardware," *ACM SIGGRAPH2007 Course*, 2007.

[16] BLAS (Basic Linear Algebra Subprograms) http://www.netlib.org/blas/

[17] nVIDIA, CUDA CUBLAS Library, 2010.