Eurographics Workshop on 3D Object Retrieval (2012)
M. Spagnuolo, M. Bronstein, A. Bronstein, and A. Ferreira (Editors)

Parallelized algorithms for rigid surface alignment on GPU

Aviad Zabatani, Alex M. Bronstein†

School of Electrical Engineering, Tel Aviv University
aviad.zabatani@gmail.com, bron@eng.tau.ac.il

Abstract

Alignment and registration of rigid surfaces is a fundamental computational geometric problem with applications ranging from medical imaging, automated target recognition, and robot navigation just to mention a few. The family of the iterative closest point (ICP) algorithms introduced by Chen and Medioni [YC] and Besl and McKey [PB92] and improved over the three decades that followed constitute a classical to the problem. However, with the advent of geometry acquisition technologies and applications they enable, it has become necessary to align in real time dense surfaces containing millions of points. The classical ICP algorithms, being essentially sequential procedures, are unable to address the need. In this study, we follow the recent work by Mitra et al. *[NJM] considering ICP from the point of view of point-to-surface Euclidean distance map approximation. We propose a variant of a k-d tree data structure to store the approximation, and show its efficient parallelization on modern graphics processors. The flexibility of our implementation allows using different distance approximation schemes with controllable trade-off between accuracy and complexity. It also allows almost straightforward adaptation to richer transformation groups. Experimental evaluation of the proposed approaches on a state-of-the-art GPU on very large datasets containing around* 10^6 *vertices shows real-time performance superior by up to three orders of magnitude compared to an efficient CPU-based version.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

1. Introduction

Alignment and registration of rigid surfaces is a fundamental computational geometric problem with applications ranging from medical imaging, automated target recognition, and robot navigation just to mention a few. Given two surfaces, often referred to as the template and the query, each represented in its own system of coordinates, the goal of registration is to find a rigid transformation (translation, rotation, and, possibly, reflection) optimally aligning one surface with the other. The family of the iterative closest point (ICP) algorithms introduced by Chen and Medioni [YC] and Besl and McKey [PB92] and improved over the three decades that followed (see, e.g., [RL01] for a systematic overview) constitute a classical to the problem. ICP algorithms first establishes the correspondence between the two surfaces by assigning each point on the query to its nearest neighbour on the template. Once the correspondence has been established, the aligning transformation is found by minimizing some error criterion, and is applied to the query. Since this usually changes the closest point correspondences, the above two steps are iterated. Being essentially a sequential procedure, ICP is unable to address the need of real time alignment of dense surfaces containing millions of points, which is becoming necessary with the advent of affordable geometry acquisition technologies and the applications they enable.

In this study, we follow the recent work by Mitra *et al.* [NJM] considering ICP from the point of view of point-to-surface Euclidean distance map approximation. We take advantage of modern programmable graphics hardware and propose parallelized algorithms for efficient surface registration based on distance map approximation. The flexibility of our implementation allows using different distance approximation schemes with controllable trade-off between accuracy and complexity. Experimental evaluation of the pro-

† Research supported by the Israeli Science Foundation and the German-Israeli Foundation.

DOI: 10.2312/3DOR/3DOR12/017-023

posed approaches on a state-of-the-art GPU on very large datasets containing around 10^7 vertices shows real-time performance superior by up to five orders of magnitude compared to a CPU-based version.

This paper is organized as follows: in Section 2 we formulate the rigid registration problem and briefly overview different approximations of the surface-to-surface distances constituting the objective function being minimized. Section 3 is dedicated to a self-contained derivation of a Newton descent-based rigid registration algorithm introduced in [NJM].

In Section 4, we describe a parallel implementation of the algorithm, and in Section 5 show its experimental evaluation. Finally, Section 6 concludes the paper.

2. Distances between surfaces

Let $\mathcal{S}$ and $\mathcal{Q}$ be some two-dimensional surfaces in $\mathbb{R}^3$, to which we refer as to the *template* and the *query*, respectively. We aim at finding a Euclidean isometric $\mathbf{x} \mapsto \mathbf{R}\mathbf{x}+\mathbf{t}$ parametrized by an orthonormal 3×3 rotation matrix $\mathbf{R}$ and a 3×1 translation vector $\mathbf{t}$ that minimizes some measure of misalignment between the template $\mathcal{S}$ and the transformed query $\mathbf{R}\mathcal{Q}+\mathbf{t}$,

$$\min_{\mathbf{R},\mathbf{t}} d(\mathbf{R}\mathcal{Q}+\mathbf{t},\mathcal{S}). \tag{1}$$

Out of a wide variety of possibilities to define a misalignment criterion between two surfaces, a non-symmetric squared distance

$$d^2(\mathcal{Q},\mathcal{S}) \;=\; \sum_{\mathbf{q}\in\mathcal{Q}} w_{\mathbf{q}}\, d^2(\mathbf{q},\mathcal{S}), \tag{2}$$

where the weight $w_{\mathbf{q}}$ represents, for example, the discrete area element of the point $\mathbf{q}$, and $d^2(\mathbf{q},\mathcal{S})$ is the squared *point-to-surface distance* between a query point $\mathbf{q}$ and the template $\mathcal{S}$.

Since the template is fixed, theoretically, $d^2(\cdot,\mathcal{S})$ can be pre-computed as a Euclidean distance map from $\mathcal{S}$. For example, fast marching algorithms simulate the propagation of a wave-front starting at $\mathcal{S}$; the time of a arrival of the front to a point represents the value of the distance map at that point. However, while the computation and the storage of the exact distance map are relatively expensive, surface registration tasks are known to work well with approximate distance maps. In what follows, we detail three of such commonly used approximations.

Far field approximation. When a query point $\mathbf{q}$ is sufficiently distant from the template surface $\mathcal{S}$, the point-to-surface distance can be approximated by the distance to the closest point $\mathbf{s}^*$ on the surface,

$$\hat{d}^2(\mathbf{q},\mathcal{S}) \;=\; \min_{\mathbf{s}^*\in\mathcal{S}} \|\mathbf{q}-\mathbf{s}^*\|^2. \tag{3}$$

This leads to a positive definite quadratic distance map approximant

$$\hat{d}^2(\mathbf{q},\mathcal{S}) \;=\; \mathbf{q}^{\mathrm{T}}\mathbf{q} - 2\mathbf{s}^{*\mathrm{T}}\mathbf{q} + \mathbf{s}^{*\mathrm{T}}\mathbf{s}^*, \tag{4}$$

where $\mathbf{s}^*$ is the closest point depending on $\mathbf{q}$.

Near field approximation. When a query point $\mathbf{q}$ is close to the template surface, the distance map can be approximated as a point-to-plane distance obtained by projecting the vector $\mathbf{q}-\mathbf{s}^*$ onto the normal $\mathbf{n}$ to $\mathcal{S}$ at the point $\mathbf{s}^*$,

$$\begin{aligned}\hat{d}^2(\mathbf{q},\mathcal{S}) \;&=\; \langle \mathbf{n}, \mathbf{q}-\mathbf{s}^*\rangle^2 \\ &=\; \mathbf{q}^{\mathrm{T}}\mathbf{n}\mathbf{n}^{\mathrm{T}}\mathbf{q} - 2\mathbf{s}^{*\mathrm{T}}\mathbf{n}\mathbf{n}^{\mathrm{T}}\mathbf{q} + \mathbf{s}^{*\mathrm{T}}\mathbf{n}\mathbf{n}^{\mathrm{T}}\mathbf{s}^*.\end{aligned} \tag{5}$$

Second-order approximation. While both the near- and the far-field cases constitute merely a first-order approximation to the true distance map, Pottmann and Hofer [PH03,PLH04] showed the following second-order[†] approximation:

$$\begin{aligned}&\hat{d}^2(\mathbf{q},\mathcal{S}) = \\ &\frac{\delta}{\delta-\rho_1}\langle \mathbf{e}_1, \mathbf{q}-\mathbf{s}^*\rangle^2 + \frac{\delta}{\delta-\rho_2}\langle \mathbf{e}_2, \mathbf{q}-\mathbf{s}^*\rangle^2 + \langle \mathbf{n}, \mathbf{q}-\mathbf{s}^*\rangle^2,\end{aligned} \tag{6}$$

where $\mathbf{e}_1,\mathbf{e}_2,\mathbf{n}$ are the vectors of the principal frame at the point $\mathbf{s}^*$ corresponding, respectively, to the principal curvature directions and the normal; ρ_i are the corresponding principal curvature radii; and $\delta = \pm\|\mathbf{q}-\mathbf{s}^*\|$ is the signed distance with the sign determined by the projection on the normal, $\operatorname{sign}\delta = \operatorname{sign}\langle \mathbf{n}, \mathbf{q}-\mathbf{s}^*\rangle$.

Note that the above expression generalizes the near- and far-field approximations, corresponding to $\delta \ll \rho_i$ and $\delta \gg \rho_i$, respectively. Since the distance approximant may become negative for some query points, the authors proposed a non-negative version,

$$\begin{aligned}\hat{d}^2(\mathbf{q},\mathcal{S}) = \max\left\{\frac{\delta}{\delta-\rho_1},0\right\}\langle \mathbf{e}_1, \mathbf{q}-\mathbf{s}^*\rangle^2 + \\ \max\left\{\frac{\delta}{\delta-\rho_2},0\right\}\langle \mathbf{e}_2, \mathbf{q}-\mathbf{s}^*\rangle^2 + \langle \mathbf{n}, \mathbf{q}-\mathbf{s}^*\rangle^2.\end{aligned} \tag{7}$$

As in the case of the first-order approximants, the second-order counterpart is also a positive definite quadratic function in $\mathbf{q}$,

$$\hat{d}^2(\mathbf{q},\mathcal{S}) \;=\; \mathbf{q}^{\mathrm{T}}\mathbf{A}\mathbf{q} - 2\mathbf{s}^{*\mathrm{T}}\mathbf{A}\mathbf{q} + \mathbf{s}^{*\mathrm{T}}\mathbf{A}\mathbf{s}^* \tag{8}$$

with

$$\mathbf{A} = \max\left\{\frac{\delta}{\delta-\rho_1},0\right\}\mathbf{e}_1\mathbf{e}_1^{\mathrm{T}} + \max\left\{\frac{\delta}{\delta-\rho_2},0\right\}\mathbf{e}_2\mathbf{e}_2^{\mathrm{T}} + \mathbf{n}\mathbf{n}^{\mathrm{T}}.$$

We conclude that the point-to-surface distance can be expressed as the positive definite quadratic function

$$\hat{d}^2(\mathbf{q},\mathcal{S}) \;=\; \mathbf{q}^{\mathrm{T}}\mathbf{A}(\mathbf{q})\mathbf{q} - 2\mathbf{b}^{\mathrm{T}}(\mathbf{q})\mathbf{q} + c(\mathbf{q}), \tag{9}$$

[†] The approximation is first-order at the points of the medial axis of $\mathcal{S}$, where the distance map is not $\mathcal{C}^1$.

with the coefficients $\mathbf{A}(\mathbf{q})$, $\mathbf{b}(\mathbf{q})$, and $c(\mathbf{q})$ depending on the closest point $\mathbf{s}^*$.

3. Rigid registration

In order to solve the minimization problem (1), one has to minimize the approximate surface-to-surface distance over rotation matrices and translation vectors. Since imposing the unitarity constraint $\mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I}$ on the rotation matrix results in second-order constraints that are difficult to deal with, a common alternative is the parametrization of rotation using Euler angles. However, such a parametrization implies a complicated relation between the optimization variables (the angles) and the result of the transformation. Following [NJM], we linearize the rotation matrix under the small rotation assumption, obtaining the skew-symmetric matrix

$$\mathbf{R} \approx \begin{pmatrix} 1 & -\theta_1 & \theta_2 \\ \theta_1 & 1 & -\theta_3 \\ -\theta_2 & \theta_3 & 1 \end{pmatrix} \tag{10}$$

where $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)^{\mathrm{T}}$ is the vector of the Euler angles parametrizing the rotation. The application of the transformation to a point $\mathbf{q}$ can be therefore expressed as

$$\begin{aligned} \mathbf{R}\mathbf{q}+\mathbf{t} &\approx \begin{pmatrix} -q_2 & q_3 & 0 & 1 & 0 & 0 \\ q_1 & 0 & -q_3 & 0 & 1 & 0 \\ 0 & -q_1 & q_2 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{\theta} \\ \mathbf{t} \end{pmatrix} + \mathbf{q} \\ &= \mathbf{Q}(\mathbf{q})\mathbf{x} + \mathbf{q}, \end{aligned} \tag{11}$$

where $\mathbf{x}^{\mathrm{T}} = (\boldsymbol{\theta}^{\mathrm{T}}, \mathbf{t}^{\mathrm{T}})$ is the combined six-dimensional vector of the optimization variables.

3.1. Objective function

Our next goal is to express the objective function (2) with respect to the selected problem parametrization. Let us first examine an individual squared point-to-surface distance term. The influence of the transformation on the quadratic squared distance approximant (9) translates into the complicated relation

$$\begin{aligned} \hat{d}^2(\mathbf{R}\mathbf{q}+\mathbf{t}, \mathcal{S}) &= (\mathbf{R}\mathbf{q}+\mathbf{t})^{\mathrm{T}}\mathbf{A}(\mathbf{R}\mathbf{q}+\mathbf{t})(\mathbf{R}\mathbf{q}+\mathbf{t}) \\ &-2\mathbf{b}^{\mathrm{T}}(\mathbf{R}\mathbf{q}+\mathbf{t})(\mathbf{R}\mathbf{q}+\mathbf{t}) + c(\mathbf{R}\mathbf{q}+\mathbf{t}), \end{aligned} \tag{12}$$

in which the coefficients of the quadratic function depend on the transformed query point $\mathbf{q}$. However, under a small transformation assumption, we abandon this dependence obtaining a simpler approximant $\tilde{d}^2 \approx \hat{d}^2$,

$$\begin{aligned} &\tilde{d}^2(\mathbf{R}\mathbf{q}+\mathbf{t}, \mathcal{S}) = \\ &(\mathbf{R}\mathbf{q}+\mathbf{t})^{\mathrm{T}}\mathbf{A}(\mathbf{q})(\mathbf{R}\mathbf{q}+\mathbf{t}) - 2\mathbf{b}^{\mathrm{T}}(\mathbf{q})(\mathbf{R}\mathbf{q}+\mathbf{t}) + c(\mathbf{q}), \end{aligned} \tag{13}$$

Substituting (11) into the above result yields a quadratic function in the vector of optimization variables $\mathbf{x}$,

$$\begin{aligned} \tilde{d}^2(\mathbf{R}\mathbf{q}+\mathbf{t}, \mathcal{S}) &\approx \mathbf{x}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{A}\mathbf{Q}\mathbf{x} - 2(\mathbf{Q}^{\mathrm{T}}\mathbf{b} + \mathbf{Q}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{q})^{\mathrm{T}}\mathbf{x} + \\ &(\mathbf{q}^{\mathrm{T}}\mathbf{A}\mathbf{q} - 2\mathbf{b}^{\mathrm{T}}\mathbf{q} + c) = \mathbf{x}^{\mathrm{T}}\mathbf{F}(\mathbf{q})\mathbf{x} - 2\mathbf{g}^{\mathrm{T}}(\mathbf{q})\mathbf{x} + h(\mathbf{q}), \end{aligned} \tag{14}$$

where the coefficients are given by

$$\begin{aligned} \mathbf{F}(\mathbf{q}) &= \mathbf{Q}(\mathbf{q})^{\mathrm{T}}\mathbf{A}(\mathbf{q})\mathbf{Q}(\mathbf{q}) \\ \mathbf{g}(\mathbf{q}) &= \mathbf{Q}^{\mathrm{T}}(\mathbf{q})\mathbf{b}(\mathbf{q}) + \mathbf{Q}^{\mathrm{T}}(\mathbf{q})\mathbf{A}^{\mathrm{T}}(\mathbf{q})\mathbf{q} \\ h(\mathbf{q}) &= \mathbf{q}^{\mathrm{T}}\mathbf{A}(\mathbf{q})\mathbf{q} - 2\mathbf{b}^{\mathrm{T}}(\mathbf{q})\mathbf{q} + c(\mathbf{q}). \end{aligned} \tag{15}$$

Finally, plugging in the latter result into the squared surface-to-surface distance (2) yields

$$\hat{d}^2(\mathbf{R}\mathcal{Q}+\mathbf{t}, \mathcal{S}) \approx \mathbf{x}^{\mathrm{T}}\mathbf{F}(\mathcal{Q})\mathbf{x} - 2\mathbf{g}^{\mathrm{T}}(\mathcal{Q})\mathbf{x} + h(\mathcal{Q}), \tag{16}$$

with the coefficients

$$\begin{aligned} \mathbf{F}(\mathcal{Q}) &= \sum_{\mathbf{q}\in\mathcal{Q}} w_{\mathbf{q}}\,\mathbf{F}(\mathbf{q}) \\ \mathbf{g}(\mathcal{Q}) &= \sum_{\mathbf{q}\in\mathcal{Q}} w_{\mathbf{q}}\,\mathbf{g}(\mathbf{q}) \\ \mathbf{h}(\mathcal{Q}) &= \sum_{\mathbf{q}\in\mathcal{Q}} w_{\mathbf{q}}\,h(\mathbf{q}). \end{aligned} \tag{17}$$

Note that the above coefficients depend on the surface $\mathcal{Q}$ and have to be recomputed each time it is transformed. Furthermore, we emphasize that (16) approximates the true objective (12) only for small transformations.

3.2. Numerical optimization

The approximate objective function (16) is a positive definite quadratic function in six variable, whose minimizer is given by the solution of the linear system

$$\mathbf{F}(\mathcal{Q})\mathbf{x} = \mathbf{g}(\mathcal{Q}). \tag{18}$$

The corresponding transformation can be described in homogeneous coordinates as

$$\mathbf{T}(\mathbf{x}) = \begin{pmatrix} \mathbf{R}(\boldsymbol{\theta}) & \mathbf{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \tag{19}$$

where $\mathbf{0}^{\mathrm{T}} = (0,0,0)$, and $\boldsymbol{\theta}$ and $\mathbf{t}$ are the components of the vectors $\mathbf{x}$.

However, since the approximation made in (16) is valid only for small transformation, applying $\mathbf{T}$ to $\mathcal{Q}$ can actually increase the value of the true objective function. Mitra *et al.* [NJM] proposed to regard the solution $\mathbf{z}$ merely as a descent direction, in which a fractional step $\mathcal{Q}' = \mathbf{T}^{\alpha}\mathcal{Q}$ is performed with α selected to be small enough to guarantee sufficient decrease of the objective. The resulting optimization algorithm is usually referred to as damped or safe-guarded Newton descent. While it is possible to use exact line search in order to establish the optimal value of α minimizing the objective over the ray $\{\mathbf{T}^{\alpha}\mathcal{Q} : \alpha \geq 0\}$, inexact line search is often preferred due to its lower computational complexity. Algorithm 1 outlines a variant of inexact line search known as Armijo rule. Note that while the descent direction is established from the approximate objective (16), the line search is performed on the exact one in order to guarantee its decrease.

input : descent direction $\mathbf{x}$; parameters $\beta, \sigma \in (0,1)$
output: step size α
Set $\rho = 2(\mathbf{x}^{\mathrm{T}}\mathbf{F}(\mathcal{Q})\mathbf{x} - \mathbf{g}^{\mathrm{T}}(\mathcal{Q})\mathbf{x})$
Construct $\mathbf{T}$ according to (19)
Set $\alpha = 1$
while $\hat{d}^2(\mathbf{T}^\alpha\mathcal{Q},\mathcal{S}) - \hat{d}^2(\mathcal{Q},\mathcal{S}) > \sigma\alpha\rho$ **do**
  $\alpha = \beta\alpha$
end

Algorithm 1: Armijo rule for fractional step selection.

input : template surface $\mathcal{S}$; query surface $\mathcal{Q}$
output: query surface $\mathcal{Q}$ registered to the template
repeat
  foreach $\mathbf{q} \in \mathcal{Q}$ **do**
    Construct or fetch distance approximation coefficients $\mathbf{A}(\mathbf{q}), \mathbf{b}(\mathbf{q})$, and $c(\mathbf{q})$
    Construct coefficients $\mathbf{F}(\mathbf{q}), \mathbf{g}(\mathbf{q})$, and $h(\mathbf{q})$ according to (15)
  end
  Construct quadratic approximant coefficients $\mathbf{F}(\mathcal{Q}), \mathbf{g}(\mathcal{Q})$, and $h(\mathcal{Q})$ according to (17)
  Find Newton direction $\mathbf{x} = \mathbf{F}^{-1}(\mathcal{Q})\mathbf{g}(\mathcal{Q})$
  Construct transformation $\mathbf{T}$ according to (19)
  Using Armijo rule, find step size α
  Transform query surface $\mathcal{Q} = \mathbf{T}^\alpha\mathcal{Q}$
until convergence;

Algorithm 2: Rigid surface registration algorithm [NJM].

The entire rigid registration algorithm is outlined in Algorithm 2. As a stopping criterion, either the change in the transformation $\|\mathbf{T} - \mathbf{I}\|$ or the value of the objective itself can be used. Note that the algorithm returns the transformed query surface $\mathbf{T}\mathcal{Q}$ optimally registered to the template. If the transformation itself is sought after, it can be composed from the iterates $\mathbf{T}_k$ as $\mathbf{T} = \mathbf{T}_n^{\alpha_n}\mathbf{T}_{n-1}^{\alpha_{n-1}} \cdot \cdots \cdot \mathbf{T}_1^{\alpha_1}$.

3.3. Distance approximation coefficients

Despite its apparent departure from the classical ICP schemes iterating nearest neighbour search with optimal alignment of sets of corresponding points, Algorithm 2 still conceals nearest neighbour search, as the construction of the coefficients $\mathbf{A}(\mathbf{q}), \mathbf{b}(\mathbf{q})$, and $c(\mathbf{q})$ is based on the closest point $\mathbf{s}^*$.

Since a point $\mathbf{s} \in \mathcal{S}$ constitutes the closest point to all query points belonging to the Voronoi region

$$V(\mathbf{s}) = \{\mathbf{q} : \|\mathbf{q} - \mathbf{s}\| \leq \|\mathbf{q} - \mathbf{s}'\| \forall \mathbf{s}' \in \mathcal{S}\}, \quad (20)$$

the coefficients $\mathbf{A}, \mathbf{b}$, and c are constant in each such $V(\mathbf{s})$. Representing the Voronoi decomposition of the space induced by the points of the template surface and storing the approximation coefficients corresponding to each region effectively undoes the need of explicit nearest neighbour search.

In [NJM], Mitra *et al.* proposed to use an octree structure to store and efficiently fetch the quadratic approximant coefficients. In this study, we adopt a more flexible data structure resembling a k-d tree and describe its parallel implementation on graphics hardware.

4. Parallel implementation

We propose four parallel algorithms taking advantage of the GPU to perform the closest point correspondence search or evaluate $\hat{d}^2(\mathcal{Q},\mathcal{S})$ in (2), accelerate the application of the transformation $\mathbf{R}, \mathbf{t}$ to the query, and evaluate the objective function (16) at every iteration. The CPU is used to build a generalized k-d tree data structure utilized for the efficient point-to-surface distance approximation including the calculation of the quadratic approximant coefficients in (15); to solve the the 6×6 Newton system for the transformation parameters; and to control the flow of the algorithm (Figure 2). The template itself is stored in the GPU memory, since it is static throughout the entire process, while the query shape changing at every iteration is stored in the CPU and copied to the GPU for computation. To overcome the GPU's inability to work efficiently with irregular memory access patterns, we propose a parallel implementation of the k-d tree optimizing the access to the nodes storing the template vertices in subsection (4.2). Also, the use of simple instruction and frequent storage of intermediate results in the GPU registers were used to avoid branching and complex instruction that are known to be less efficient on the GPU. This led to an efficient parallel GPU implementation of the k-d tree structure, the main ingredient of ICP.

In the accompanying implementation, we supply a simple interface allowing to specify the distance function used in (2) by changing the calculations of the stored distance coefficients. The resolution of the tree is also controllable, allowing to trade off accuracy with execution time.

4.1. Algorithms

The following algorithms were implemented:

CPU ICP [CCICP]. A serial implementation of the classical ICP as introduced by Chen and Medioni [YC] using the ANN library for efficient nearest neighbour search. This implementation is presented as a baseline for measuring the performance of the parallel algorithms.

GPU ICP [GCICP]. An implementation of the classical

ICP as introduced by Chen and Medioni [YC], using the GPU to perform the correspondence search and to apply the transformation in each iteration.

Approximate GPU ICP [GCAICP]. An implementation of the classical ICP with approximation of the using the k-d Tree cell to store the geometric mean mass point of all the vertices in that cell, thus avoiding the in-vertex search and the back search,and as result reducing the iteration time the. Uses the GPU to perform the correspondence search and to apply the transformation in each iteration.

Revised GPU ICP. An implementation of the registration algorithm proposed by Mitra *et al.* [NJM] on GPU with the far-field distance map approximation (**[GRICPPT]**) and the near-field distance map approximation (**[GRICPPL]**).

4.2. Parallel k-d Tree

The computation of the point-to-surface distance map is the core of the registration algorithm and is the most time consuming operation. The brute force approach is based on evaluating the distance function from each of the template vertices followed by selecting the vertex with the minimal distance. Such a naïe approach has the complexity of $\mathcal{O}(m \times n)$ with m being the template size and n the query size. In order to decrease the distance evaluation time, we used a variant of the k-d tree data structure. k-d tree is a binary space partitioning tree in which each node splits the space along a plane normal to one of the axes (yet, having an arbitrary offset). Leaf nodes create a partition of $\mathbb{R}^3$, with each leaf corresponding to a single cell of the partition. In the traditional k-d tree, each such cell is linked to a template vertex; given a query vertex, the tree is traversed from the root down to the leaf into whose cell the query vertex belongs, thus establishing the closest point correspondence. Instead of associating each cell merely to a template vertex, we associate it to the quadratic approximation coefficients, such that all query vertices belonging to a particular cell will use the same $\mathbf{Q}$, $\mathbf{b}$, and c to evaluate the approximate squared distance map. The refinement of the tree cell size (by increasing the number of levels) improves the accuracy of the distance map approximation at the expense of higher computational complexity. The coefficients are calculated one time on tree construction; by storing the coefficient at intermediate levels allows trading off between accuracy and speed in run-time rather than at initialization time.

Distance computation using the k-d tree has the time complexity of $\mathcal{O}(n)$ with an accuracy-dependent constant; when the computation is parallelized for p query vertices, the process is accelerated by the factor p. The main difficulties in parallelizing the k-d tree are the random access patterns of tree search due to data-dependent branching, as well as efficient storage of the tree. As there is no way to change or know ahead of time the access patterns, we construct the tree off-line and serialize its nodes to the GPU constant LFU-cached memory ordered by their level and position in the tree (Figure 3). This makes the root and the first tree levels cached and allows fast access to them, while the least accessed nodes are not cached an have slower access. This arrangement also reduces the probability of bank conflicts for typical access patterns, thus further accelerating the memory access. The serialization process also reduces the storage complexity by extracting the template vertices and arranging them in an array referred by index only. Since the approximation coefficients rather than the actual vertex coordinates are used for distance computation, a significant amount of space is saved.

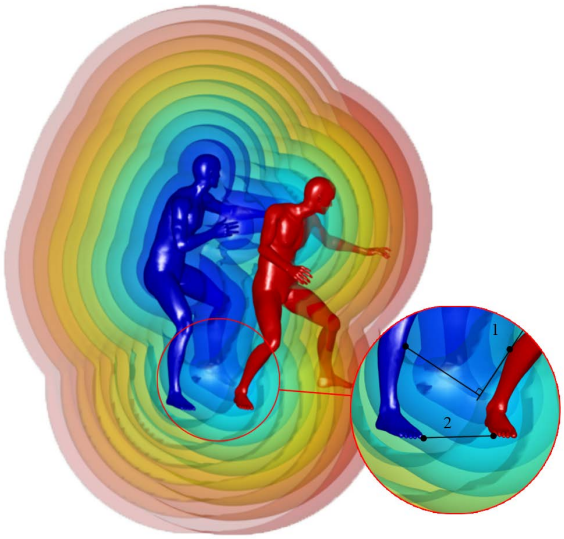

Figure 1: *Visualization of the quadratic distance map from the template shape (left) to the query shape (right). 1 represents point to plane distance and 2 represents point to point distance*

5. Results

We evaluated the proposed approach on a state-of-the-art Tesla C2070 GPU on inputs ranging from 10^5 to 10^6 vertices. For reference, results of an optimized sequential ICP algorithm using ANN library executed on modern Intel Xeon E5620 2.4 GHz are given.

5.1. Iteration time

In many applications, the surfaces are roughly registered and ICP is used to refine their alignment. In such cases, the number of iterations required to converge is small. To evaluate the performance of different alignment algorithms, we benchmarked the execution time of a single iteration for different input sizes. Execution times are summarized in Fig-

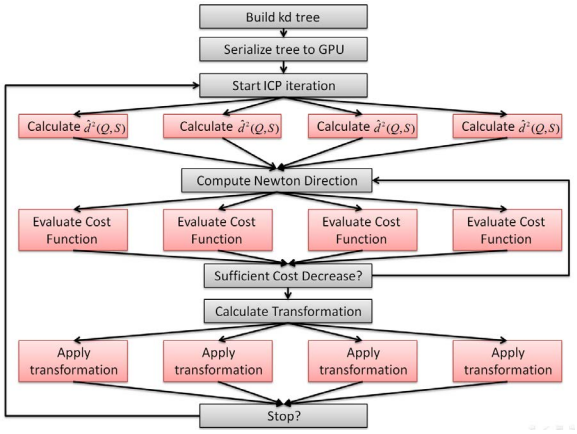

Figure 2: *The flow of the paralleled ICP algorithm, taking advantage of the graphic hardware to compute the independent steps of the algorithm. The Gray steps are done in serial, while the Pink steps are done in parallel.*

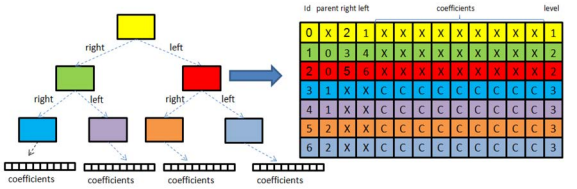

Figure 3: *Visualization of the memory layout of the tree in the GPU memory*

ure 4 (left); the right-hand plot depicts the speed-up factor of the parallel implementations relative to the serial counterpart. For large data, acceleration of up to 300 times is observed, with the parallel algorithm spending from 80 to 300 milliseconds per iteration. This enables real-time performance. In this experiment, sufficiently deep trees were used to make the approximate distance function within 1% error compared to its exact value. Lowering the accuracy further reduces the execution time.

5.2. Convergence time

For the completeness of the execution time benchmarks, we also measured to time different algorithms take to convergence given the same input. Algorithms based on Newton steps (GRICPPT and GRICPPL) require a significantly smaller number of iterations compared to the standard ICP, as shown in Table 1. This results in even higher speed-up factors than those observed for a single iteration.

Figure 5 depicts the convergence time and relative speed-up (compared to the serial version) of different alignment algorithms. We observe an acceleration of up to 1000 times

CCICP	GCICP	GCAICP	GRICPPT	GRICPPL
62	35	47	17	5

Table 1: *The average number of iteration in each of the algorithms (subsection 4.1) for all input sizes*

compared to the serial version. In absolute figures, alignment of surfaces containing one million vertices takes less than 1.5 seconds. Again, we note that lowering the distance computation accuracy further improves performance.

5.3. Accuracy

In order to test the accuracy of different algorithms, we evaluated the alignment error according to

$$\varepsilon = \|\mathbf{R}^{-1}\mathbf{R}^* - \mathbf{I}\|_{\mathrm{F}} + \|\mathbf{R}\mathbf{t} - \mathbf{t}^*\|, \qquad (21)$$

where $\mathbf{R}$, $\mathbf{t}$ is the computed transformation, and $\mathbf{R}^*$, $\mathbf{t}^*$ is the groundtruth transformation. Table 2 summarizes the relative errors (compared to the exact ICP) averaged over various input sizes. Note that accuracy degradation due to the distance approximation is insignificant; further improvement in accuracy can be achieved by increasing the tree depth.

Algorithm	GCICP	GCAICP	GRICPPT	GRICPPL
Error	99.98%	99.83%	99.3%	98.87%

Table 2: *Average relative alignment errors of different algorithms computed for various input sizes*

5.4. Tree construction

The construction of the k-d tree is performed offline on the CPU and then serialized to the GPU. This initialization process limits the application of our proposed method to templates that do not change frequently. We note that construction time depends on the template size and the tree depth. The construction of the tree with 17 levels for a template containing 5×10^5 vertices takes less than 2.5 seconds. The GPU acceleration of this process will be explored in future work.

6. Conclusion

We presented a parallel implementation of the rigid surface alignment recently introduced by Mitra *et al.* [NJM], optimized for graphical hardware. Our implementation is based on a variant of the k-d tree data structure, which is more efficient than the originally proposed octree. Experimental evaluation of the proposed algorithm on a modern GPU applied to very large datasets containing millions of vertices shows real-time performance superior by up to three orders of magnitude compared to an efficient CPU-based version. The paper is accompanied by code, in which a simple interface allows to control the accuracy-complexity trade-off

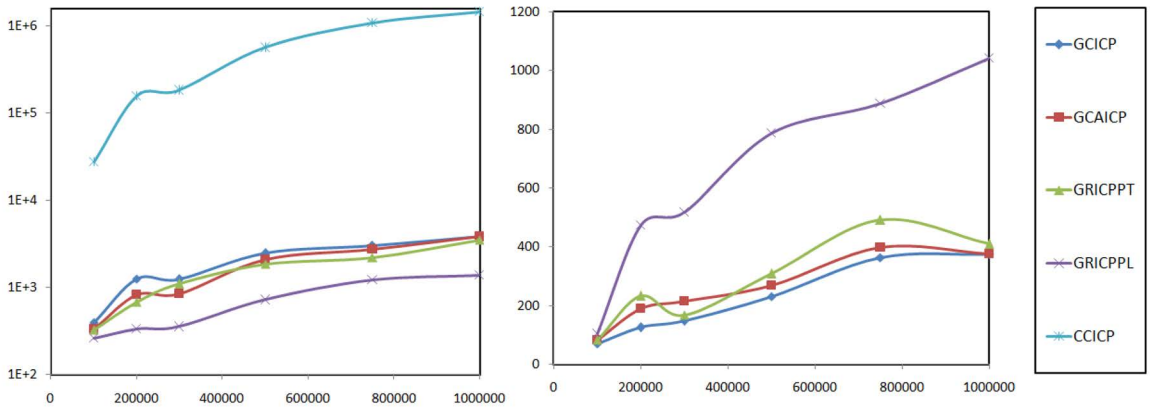

Figure 4: *Execution time in milliseconds of a single ICP iteration (left) and relative speed-up compared to the CPU version as a function of the input size(right).*

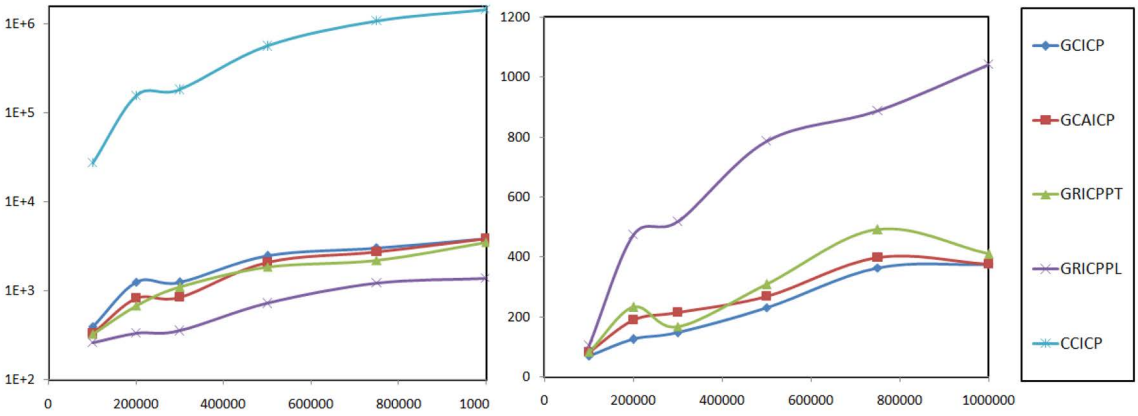

Figure 5: *Execution time in milliseconds until convergence of ICP (left) and relative speed-up compared to the CPU version as a function of the input size(right).*

and supply user-defined distance function. In future studies, we are going to extend the approach to richer families of transformations including local and global affine transformations.

References

[NJM] N. J. MITRA N. GELFAND H. P. L. G.: Registration of point cloud data from geometric optimization perspective. 1, 2, 3, 4, 5, 7

[PB92] P. BESL N. M.: A method for registration of 3-d shapes. 1

[PH03] POTTMANN H., HOFER M.: Geometry of the squared distance function to curves and surfaces. *Visualization and Mathematics III, Springer* (2003), 221–242. 2

[PLH04] POTTMANN H., LEOPOLDSEDER S., HOFER M.: Registration without icp. *Computer Vision and Image Understanding 95*, 1 (2004), 54–71. 2

[RL01] RUSINKIEWICZ S., LEVOY M.: Efficient variants of the ICP algorithm. In *Proc. 3D Digital Imaging and Modeling* (2001), IEEE, pp. 145–152. 1

[YC] Y. CHEN G. M.: Object modeling by registration of multiple range image. 1, 4, 5