Multi-Frame Adaptive Non-Rigid Registration for Markerless Augmented Reality

Antonio C. S. Souza*
Federal University of Bahia, Brazil
Federal Institute of Bahia, Brazil

Márcio C. F. Macedo†
Federal University of Bahia, Brazil

Antônio L. Apolinário Jr.‡
Federal University of Bahia, Brazil

Abstract

Augmented Reality is a technology in which the user's view of a real scene is augmented with virtual information. To provide real-time performance, the major of its applications only support rigid interaction with the fiducial marker (i.e. marker-based augmented reality) or the part of the real scene being used as a natural marker (i.e. markerless augmented reality). In this context, when the natural marker consists of a deformable object (e.g. face, body, hand), it is desirable for the application to support non-rigid interactions between the user and the marker. In this paper we present an adaptive non-rigid surface registration algorithm for markerless augmented reality. It is applied in a multi-frame manner to achieve fast performance. Likewise, it takes advantage from the power of the graphics processing unit to improve application's performance. Based on multi-frame adaptivity, we show that the markerless tracking runs almost in real-time, improving the accuracy of the tracking when compared to the rigid-only solution.

CR Categories: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.7 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages and systems; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities;

Keywords: Non-Rigid Registration, Adaptive Algorithms, Augmented Reality.

1 Introduction

Augmented Reality (AR) is a technology in which an additional virtual information augments a real scene. Accurate tracking and real-time performance are two of the most important technical challenges of AR applications.

In depth-based AR, tracking is performed by registering two surfaces captured from the real scene. Surface registration can be classified into two major types: rigid and non-rigid (i.e. deformable). Rigid registration assumes that the surfaces to be aligned have the same shape and are related by a rigid transformation. On the other hand, non-rigid registration does not assume that the surfaces have the same shape, relating them by multiple rigid transformations distributed along the surface, or in its joints, for articulated structures [Tam et al. 2013].

In general, AR applications can be divided in two groups: marker-based or markerless. Marker-based AR uses a fiducial marker as a point of reference in the field of view to help the system to estimate the camera pose. Markerless AR (MAR) uses a part of the real scene as a natural marker. By using it as a point of reference for tracking, one can expect non-rigid motion of the marker if it consists of a deformable object (e.g. face, body, hand). In scenarios such as on-patient craniofacial medical data visualization [Lee et al. 2012; Macedo et al. 2014], it is specially important for a MAR environment to provide support for non-rigid tracking, which adds one level of interactivity for the user and improves the robustness of the tracking algorithm for rigid and non-rigid patient interactions. The main issue related to this support is that AR requires real-time interactivity and most of the current state-of-the-art works in the field of non-rigid surface registration do not provide such performance.

Despite the real-time techniques which rely on strong priors about a specific scenario (e.g. [Weise et al. 2011; Bouaziz et al. 2013; Li et al. 2013] for face, [Chen et al. 2012] for body), a few methods have been proposed for fast generic non-rigid registration (e.g. [Sumner et al. 2007; Nutti et al. 2014]). Their common characteristic is the way they represent the deformation for a given surface: using a deformation graph. Each node of this graph has a 3D affine transformation which allows source surface to be deformed to a target surface. Deformation is modelled in terms of an energy function and, by using a non-linear optimization algorithm, energy is minimized and the best affine transformation for each node of the graph can be found.

In this paper we present a multi-frame adaptive non-rigid surface registration algorithm for a MAR environment. It is proposed in the context of a medical application for on-patient craniofacial data visualization to allow support for patient's non-rigid interaction. Therefore, the focus of our approach is non-rigid facial registration. For each new frame, the current surface captured by a depth sensor is rigidly aligned to a previous one represented by a 3D reference model being generated. When rigid tracking fails, non-rigid registration is employed to allow current camera pose prediction and deformation estimation between frames. In this case, the final result is used to update the 3D reference model. Deformation space is represented by a graph built on the source surface. To achieve fast performance, we have chosen to use a 3-level adaptive approach in which:

1. The graph is dynamically subdivided according to an energy function that models the deformation between source and target surfaces;

2. An adaptive algorithm is employed to select, for each iteration, which points on the source surface will be used as constraints to guide the non-rigid registration;

3. A multi-frame adaptive approach is used to apply the non-rigid registration only when the rigid registration fails. Therefore, this solution can be applied in an AR environment in real-time.

Moreover, all algorithms (i.e. non-rigid registration and MAR environment) are implemented in parallel by using the graphics processing unit (GPU).

The paper is organized as follows. Section 2 provides a brief review on the recent related works of non-rigid surface registration.

*e-mail:antoniocarlos@ifba.edu.br
†e-mail:marciocfmacedo@gmail.com
‡e-mail:apolinario@dcc.ufba.br

Section 3 introduces an overview of the full pipeline. Section 4 presents the MAR environment proposed in this work. Section 5 presents the non-rigid surface registration algorithm, describes its GPU implementation and how it can be integrated with the MAR environment. Section 6 discusses the experimental results. The paper concludes in Section 7, with a summary and discussion of future work.

2 Related Work

Non-Rigid Surface Registration: Non-rigid surface registration has been driven by different approaches in recent years. In this section, we present the relevant works that are most closely related to our approach. For a more detailed discussion of rigid and non-rigid surface registration, the reader should refer to [Tam et al. 2013].

One of the first works in the field of fast non-rigid registration applied to Computer Graphics is the Embedded Deformation (ED), a real-time deformation algorithm for object manipulation and creation of 3D animation [Sumner et al. 2007]. The goal of this technique is to allow an user intuitive surface editing while preserving the surface's features. The deformation in their approach is represented by a graph. Each node of this graph is associated with an affine transformation that influences the deformation to the nearby space. The great advantage of this approach is that it can be applied to a wide range of objects, articulated or not.

Although its main goal is the user object manipulation, the algorithm proposed by Sumner et al. also can be seen as a non-rigid registration algorithm in which the source and target surfaces are the objects before and after user manipulation. In this sense, many other works have used or improved this approach to the specific problem of non-rigid surface registration.

Li et al. adapted Sumner's algorithm to the registration of partial range scans acquired from a 3D scanner [Li et al. 2008]. They augmented the ED algorithm with a rigid registration and designed an energy function to penalize unreliable correspondences. An extension of [Li et al. 2008] was proposed in [Li et al. 2009], where an algorithm for high-quality template-based non-rigid surface registration and reconstruction using dynamic graph refinement and multi-frame stabilization was presented.

A method for temporally coherent completion of surfaces captured from real-time dynamic performances is presented in [Li et al. 2012]. They extended the non-rigid registration proposed in [Li et al. 2009] by adding texture constraints for the optimization. Dou et al. [Dou et al. 2013] proposed an algorithm to track dynamic objects acquired from real-time commodity depth cameras such as the Microsoft Kinect Sensor. Basically, they have extended the KinectFusion algorithm [Newcombe et al. 2011] to deal with non-rigid registration. Its non-rigid registration algorithm is based on the ED algorithm, however color consistency and dense point cloud alignment were added to the original energy function.

All these approaches achieve high accuracy, however need execution time in the order of minutes to register two point clouds. Different from these approaches, the KinEtre has adapted the ED algorithm to deal with incomplete meshes and to align the skeleton of a user in movement [Chen et al. 2012]. The ED algorithm was implemented almost entirely on the GPU and it runs in 30 frames per second (FPS), as it was constrained to align only the skeleton of the user.

By assuming that there is a temporal coherence between the source and target objects, rigid registration can be solved before the non-rigid alignment. To do so, Iterative Closest Point (ICP) algorithm [Rusinkiewicz and Levoy 2001] is used. Thus, instead of comprising the global rigid transformation into the unknowns of the ED algorithm, we deal only with the node's affine transformations as unknowns to be estimated, saving memory space and computational time for the optimization step. Moreover, our algorithm focuses only on the non-rigid registration and does not deal with reconstruction or texture constraints. Finally, our algorithm runs in ≈ 60 milliseconds and it is applied in a multi-frame manner to achieve real-time performance in an augmented reality application.

Markerless Augmented Reality: Despite the recent advances in the field of virtual reality based on real-time non-rigid user interaction [Weise et al. 2011; Chen et al. 2012; Bouaziz et al. 2013; Li et al. 2013], in depth-based AR, from the best of the our knowledge, only one work has been proposed for markerless augmented reality with support to non-rigid tracking.

Nutti and colleagues proposed an application for real-time tumor tracking based on a depth sensor [Nutti et al. 2014]. They adapted the technique proposed in [Li et al. 2009] by using multi-threading and axis-angle rotation-based deformation representation and a Finite Element Method simulation on the patient's body. As result, they could interactively track the tumor position on the patient. Their system uses reprojection for physician visual guidance to adjust the posture of the patient under radiation therapy. While this application provides high accuracy to track the tumor, it runs only in 10 FPS and the visual guidance (i.e. reprojection) is updated in 5 FPS. Therefore, the method proposed in this paper is one of the first for markerless augmented reality based on interactive non-rigid registration.

3 Pipeline Overview

An overview of our entire pipeline can be seen in Figure 1. It requires only an RGB-D sensor (e.g. Kinect) and a computer with GPU. The main goal of the proposed pipeline is to support non-rigid tracking in a MAR environment. First, the natural marker must be segmented from the real scene and reconstructed in real-time. Next, current depth frame (D_t) captured by RGB-D sensor and previous depth frame (D_s) represented by 3D reference model are used for tracking. As long as the deformation of target object increases, the chances for rigid tracking to fail increase as well. To minimize this possibility, the reference model should be updated with the deformation of the target object.

After marker segmentation, and by considering that the background scene is already segmented in the depth maps, the 2D bounding box that contains both source and target objects in D_s and D_t is used to discard from the memory every position outside the xy-axis of the bounding box.

Afterwards, non-rigid registration algorithm builds a deformation graph (G) on the source surface (P_s) to allow its deformation to the target surface (P_t) iteratively. Each node in G has a 3D affine rigid transformation (i.e. a 3D rotation matrix R and a 3D translation vector t) which influences the deformation to the nearby space. Current deformation between P_s and P_t is modelled in terms of an energy function and a non-linear optimization algorithm is applied to minimize this energy based on the affine transformations of G. To reduce computational cost of the non-linear solver, a sub-sample of P_s is selected as constraint to be used during optimization. Next, the algorithm iteratively refines G according to the energy function measured previously. This refinement is based on a quadtree. The registration is stopped when the residual error between deformed P_s and P_t is sufficient low. To achieve a good performance, the full pipeline runs entirely on GPU and non-rigid registration algorithm is applied in a multi-frame manner only when rigid tracking fails.

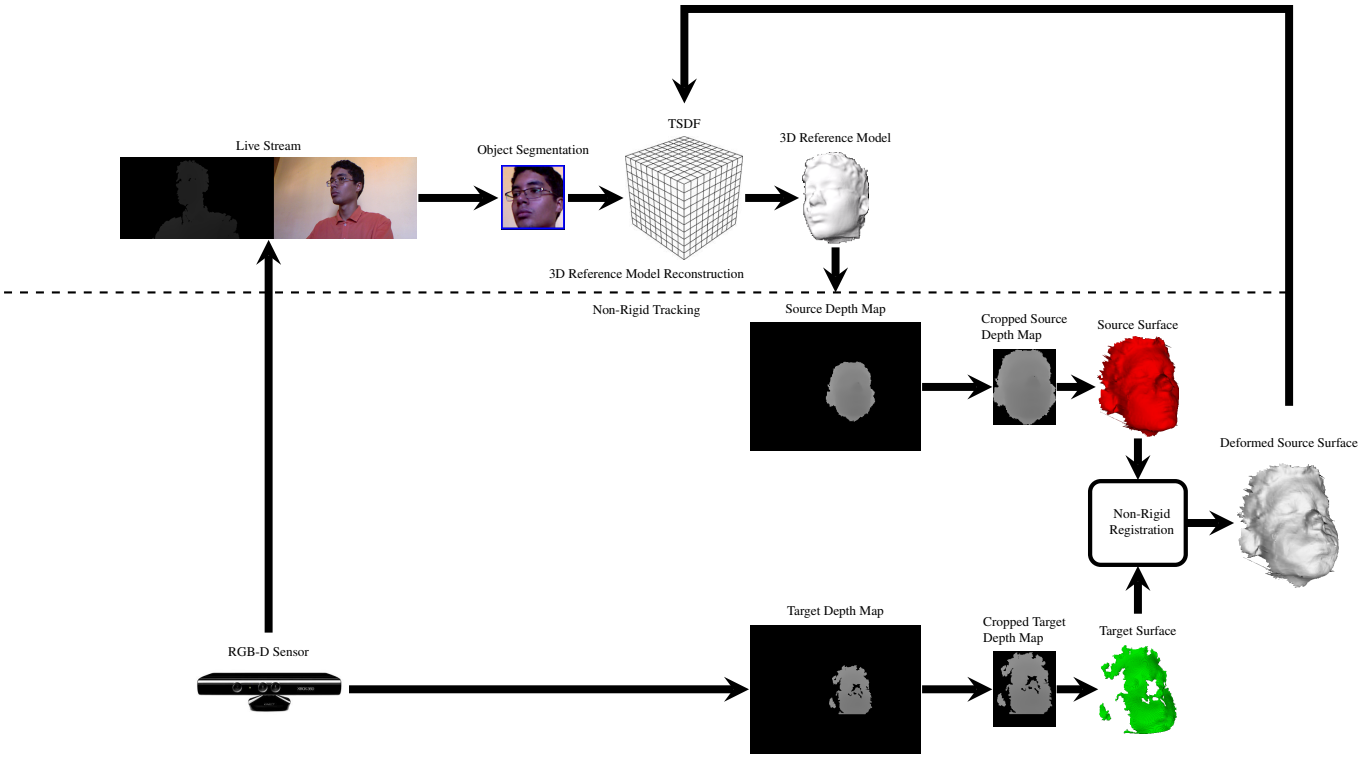

Figure 1: *Overview of the proposed approach from 3D reference model reconstruction to final non-rigid aligned surface.*

4 Markerless Augmented Reality

The MAR environment used in this work is based on the one proposed in [Macedo et al. 2014] for on-patient medical data visualization. However, we are only interested on the markerless tracking algorithm. To track an object without markers, a 3D reference model of the object must be generated. To do that, the object must be detected and segmented from the scene either by using color, depth or both information. This process can be done by using an appropriate classifier or semi-automatically by the user. Once the object is detected on the scene, the region that contains the object is fixed. Then, the object is constrained to be turned in this region. By denoising the depth map using a bilateral filter [Tomasi and Manduchi 1998] and by converting the filtered depth map into a vertex and normal map, the KinectFusion algorithm is used to reconstruct the 3D reference model in real-time [Izadi et al. 2011]. The KinectFusion is an algorithm that integrates raw depth data captured from an RGB-D sensor into a 3D grid to produce a high-quality 3D reconstruction of the object/scene of interest. The grid stores for each voxel the distance to the closest surface (i.e. TSDF - Truncated Signed Distance Function) and a weight that indicates uncertainty of the surface measurement. These volumetric representation and integration are based on the VRIP algorithm [Curless and Levoy 1996]. The 3D reconstructed model is extracted by detecting zero-crossings on the grid through a ray caster.

This algorithm allows accurate markerless tracking without error accumulation, as the high-quality 3D reference model is used as basis for tracking. In a naive implementation, non-rigid tracking support can be added by applying a non-rigid surface registration algorithm to align the 3D reference model and the current depth frame captured. One way to improve the accuracy of this solution is updating the 3D reference model after the non-rigid registration, as will be shown in Section 5.3.

5 Non-Rigid Surface Registration

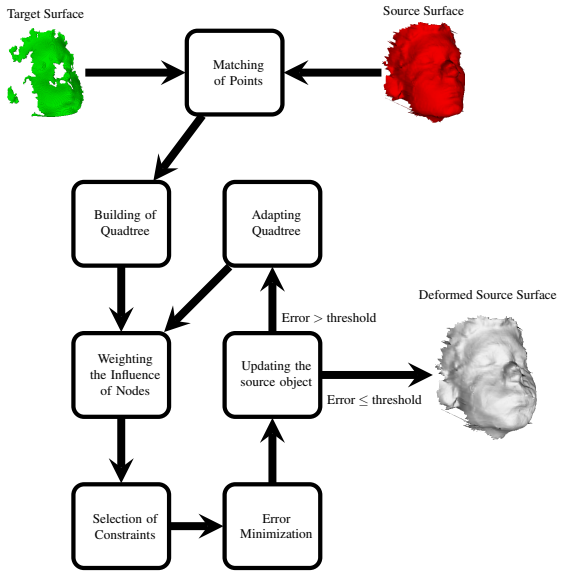

Figure 2: *Overview of the non-rigid registration algorithm.*

In this section we describe the non-rigid surface registration algorithm and how it can be integrated with the MAR environment presented in the previous section. An overview of the proposed approach can be seen in Figure 2.

5.1 Deformation Model

The proposed algorithm consists of the following stages:

- **Matching of points**: After the bounding box segmentation, the source (P_s) and target (P_t) points are associated. As it is assumed that the surfaces are aligned rigidly, it is ensured that the objects are relatively near from each other. Therefore, it is used the projective data association to match the points;
- **Selection of nodes**: It is computed a deformation graph G from P_s. Different from previous approaches, this stage runs on a GPU in a quadtree-based algorithm;
- **Weighting the influence of nodes**: The influence of the k-nearest nodes for each P_s is computed;
- **Selection of constraints**: The points from P_s in which the residual deformation error will be measured are selected adaptively;
- **Error minimization**: The affine transformation $A = [R|t]$, where R is a 3×3 rotation matrix and t is a 3D translation vector, is estimated for each node by a non-linear Gauss-Newton solver using the constraints selected previously;
- **Updating the source object**: The affine transformations computed in the previous step are applied on P_s and the algorithm is reiterated to the second step until the maximum number of iterations is reached or if the error is stabilized.

All these stages take advantage from the parallelism provided by the GPU as it will be described in the Section 5.2.

First, the corresponding points between P_s and P_t are associated by using the projective data association. In this association, each point $p_s \in P_s$ is transformed into camera coordinate space and perspective projected into image coordinates. The corresponding points are that on the same image coordinates.

After the matching of points, the nodes of G are selected. A quadtree is built on GPU to perform the selection of nodes. The influence of a node j with respect to a vertex p can be defined by:

$$p^{'} = \sum_{j=1}^{k} w_j(p)[R_j(p - g_j) + g_j + t_j] \quad (1)$$

where k represents the k-nearest nodes of p and w_j is a weight that measures the influence of each node to the point. The weight w_j can be computed by:

$$w_j(p) = (1 - ||p - g_j||/dist_{max})^2 \quad (2)$$

and then normalized to sum to one. $dist_{max}$ is the distance to the $k+1$-nearest node with respect to p. From the Equation 2, it is guaranteed that the nearest nodes will have more influence in the deformation of p. Also, as the nodes are points of P_s, they are deformed by other nodes of G.

To compute the best affine transformations that align P_s to P_t, we must:

1. Select the constraints (i.e. points from P_s that will be used during the optimization phase);
2. Convert the affine rotations from Euler to the quaternion representation;
3. Compute the energy function E_{tot} that models the constraints to guide the proper registration of the objects;
4. Use a non-linear solver to minimize E_{tot};

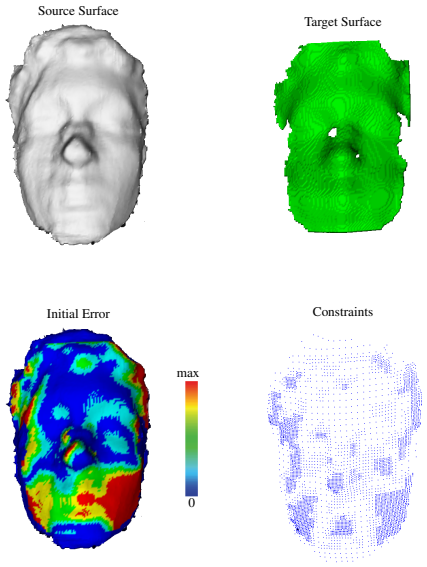

Figure 3: *Constraint selection based on the initial non-rigid error between source and target surfaces.*

Instead of using the full dense point cloud as constraint for the optimization or asking the user to perform this task of constraint selection, we use an adaptive algorithm that performs the selection of constraints based on the residual error previously measured. Given a region on the source surface, the higher the error, the higher the number of points selected as constraints for the optimization, as can be seen in Figure 3.

In the first iteration, where the residual error still was not measured, an uniform sampling is used to select the constraints. To do that, a $n \times n$ mask, with step n, is scanned through the 2D projection of P_s at the xy coordinates. The point at the center of this mask is selected to be a constraint if it exists in P_s (i.e. it is not in a hole). From empirical tests, $n = 4$ produced the best results.

In the remaining iterations, we use the same $n \times n$ mask to perform a scan on the 2D projection of P_s and its residual error E_{tot}. First, the algorithm evaluates the average residual error at the $n \times n$ region being scanned. Based on the average error E_{avg} and a pre-defined threshold th_c, the number of points selected at that region will be defined. In this case, we have three situations:

- $E_{avg} > th_c$, all the n^2 points are selected;
- $E_{avg} >= th_c/2$ and $E_{avg} <= th_c$, n points uniformly distributed over the mask are selected;
- $E_{avg} < th_c/2$, only the point at the center of the mask is selected;

Therefore, we select more constraints in the regions where the deformation is high and must be minimized, but we still consider the regions where the deformation is small or none, by selecting a small number of constraints to represent them. From empirical tests, th_c equals to the half of the averaged root mean squared error measured for the dataset produced the best results.

Next, we need to convert the affine rotations from Euler to quaterion representation. The motivation is related to our non-linear solver, that operates faster with quaternions (3 unknowns) than the Euler-form rotation matrix (9 unknowns).

To compute E_{tot}, Sumner et al. proposed 3 energy functions - $E_{rot}, E_{reg}, E_{con}$ [Sumner et al. 2007]:

- **Energy function for rotation (E_{rot})**: In order for a 3×3 rotation matrix to represent a rotation in SO(3), it must satisfy six conditions: each of its three columns must be unit length, and all columns must be orthogonal to one another [Grassia 1998]. The squared deviation of these conditions is given by the function Rot(R):

$$\begin{aligned} Rot(R) &= (c_1 \cdot c_2)^2 + (c_1 \cdot c_3)^2 + (c_2 \cdot c_3)^2 + \\ &\quad (c_1 \cdot c_1 - 1)^2 + (c_2 \cdot c_2 - 1)^2 + \\ &\quad (c_3 \cdot c_3 - 1)^2 \end{aligned} \tag{3}$$

  where c_1, c_2 and c_3 are the column vectors of a given rotation matrix.

  The term E_{rot} is defined by the sum of the rotation error over all affine transformations of G:

$$E_{rot} = \sum_{j=1}^{m} Rot(R_j) \tag{4}$$

- **Energy function for regularization (E_{reg})**: In order to apply a deformation sufficiently smooth, we must ensure that the affine transformations of adjacent nodes in G must be consistent. E_{reg} is the sum of the squared distances between each node's transformation applied to its neighbors and the actual transformed neighbor positions:

$$E_{reg} = \sum_{j=1}^{m} \sum_{k \in N(j)} ||R_j(g_k - g_j) + g_j + t_j - (g_k - t_k)||_2^2 \tag{5}$$

  where N_j consists of all nodes connected with the node j.

- **Energy function for constraints (E_{con})**: This energy function deals directly with P_s and P_t. It measures how distant they are from each other. E_{con} is the sum of the Euclidean distances between the deformed source points and its correspondents on the target object:

$$E_{con} = \sum_{i=1}^{n} ||p_i^{'} - q_i||_2^2 \tag{6}$$

  q is the target point correspondent to p_i, $p_i^{'}$ is p_i after deformation (Equation 1). n is the total of points in P_s.

The total energy function E_{tot} is defined by the following equation:

$$E_{tot} = w_{rot}E_{rot} + w_{reg}E_{reg} + w_{con}E_{con} \tag{7}$$

with $w_{rot} = 1$, $w_{reg} = 10$ and $w_{con} = 100$ as suggested in [Sumner et al. 2007].

Once with E_{tot}, we must solve the optimization step to obtain the affine transformations that align P_s to P_t. To achieve this goal we use the Gauss-Newton solver [Madsen et al. 2004]. Our objective is to solve the normal equation $J^tJ\Delta = J^tr$. We compute the residual r, that consists in the computation of E_{tot} for each coordinate x, y and z of each constraint and the Jacobian J, that is the first-partial derivative of E_{tot} for each one of the parameters. Δ represents the unknown parameters that we want to find to minimize E_{tot}. To compute J efficiently we compute only the partial derivative for the parameters that affect the constraint in which the derivation is being computed. Once with J and r, we reduce the normal equation to the linear system $A\Delta = -b$ and compute the products $A = J^tJ$ and $b = J^tr$. After solving the linear equation, we add Δ to the array of parameters (i.e. quaternions + translation vectors) and reiterate the optimization algorithm until the maximum number of iterations or if the error is stabilized (does not change more than 5%).

After the optimization step, the Equation 1 can be applied for every point in P_s. Afterwards, we reiterate the algorithm to the selection of nodes until the maximum number of iterations is reached (we use 3 iterations) or if the error is stabilized in the optimization step.

5.2 GPU Processing

This subsection describes the adaptations performed on the deformation model in order to achieve best performance using the parallelism provided by the GPU.

For the matching of points, each GPU thread transforms a point P_s into image coordinate and associates it with the point P_t at the same image coordinate.

To select the nodes, we use an algorithm in GPU that does the node selection based on the 2D parametrization of G. As the nodes of G are also points in P_s, we can convert them from world to image coordinates by using the same process used to reproject P_s into D_s. As discussed before, P_s can be an object with holes distributed along the surface. In this case, the selection of nodes only based on the 2D space may cause the nodes to be selected in regions where there is no depth data. To solve this problem, we take advantage from what we call virtual nodes to represent the space where there is no depth data. Virtual nodes favor the expansion of the quadtree in regions where naturally we have depth data, however in the node position not. It is worthy to mention that virtual nodes do not have affine transformation, they are just leaves of the quadtree that can be refined to generate real leaf nodes if necessary. Therefore, we restrict the use of virtual nodes in the first 2 levels of the quadtree.

To build the quadtree, four additional GPU buffers are required:

1. A buffer to store whether in a given position exists a node in G;
2. A buffer to store the level for each node in G;
3. A buffer to store whether in a given position G has children (is a parent node);
4. A buffer to store whether in a given position exists a virtual node in G;

The algorithm can be divided in two steps: the building of the quadtree and the adaptive refinement/collapse of nodes in G.

We build the quadtree in the first iteration of our algorithm. The GPU kernel that will select the nodes is called iteratively. We iterate from the first level to the level required by the user to build the quadtree. Each GPU thread, in parallel, is uniformly assigned to a position on the 2D parametrization of P_s to select a node. In the first level, if the point assigned to the GPU thread is visible, then, it will be a new node in G. In the opposite case, it can be a new virtual node. Therefore, we allow the quadtree to be refined even in regions where there are just a few points. On the next levels, if the node is selected, the GPU thread removes the parent node from G, being it a real or virtual node, and inserts it into a parent list, indicating that it has already been expanded.

For collapsing of nodes, the GPU kernel is iteratively called from the last to the first level of the quadtree in order to collapse the nodes in a bottom-up fashion. Each GPU thread, in parallel, is assigned to a node in G. Then, the GPU thread checks if the node has children and if it is at the current level that is being iterated. If the thread passes from these conditions, given a region C around the 2D position of the node, it computes the average of the energy function E_{con} for each $p_s \in C$. If the average error is below a certain threshold, the children nodes in C must be collapsed. To collapse the nodes, a thread checks if exists child nodes and they are leaf nodes. In this case, they are collapsed and the region C is represented by the old parent node.

For refinement of nodes, the GPU kernel is iteratively called from the first to the last non-empty level of the quadtree in order to refine the nodes in a top-down fashion. Each GPU thread, in parallel, is assigned to a node in G. Then, the GPU thread computes the average error around a region C, as defined before. If the average is above a certain threshold, the node must be refined. After the refinement (i.e. creation of 4 new child nodes), the GPU thread removes the parent node from G and inserts it into a parent list, indicating that it has already been expanded.

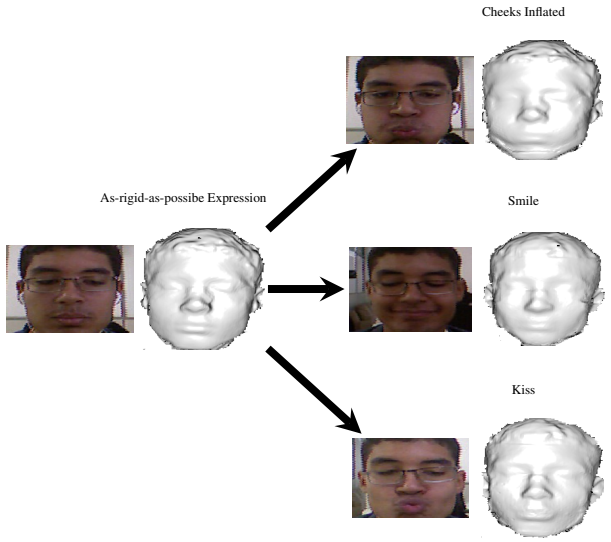

Figure 4: *Neutral and deformed reference models based on user's facial expression.*

To compute the weights efficiently in GPU, we create an array that contains only the nodes selected. The direct access to this array prevents us from checking explicitly on the surface whether a point is also a node. Then, each GPU thread computes the influence for a specific node in G.

To store the affine transformations that will be estimated, we create two arrays: one array to store 6 parameters (i.e. 3 from quaternion and 3 for translation) for each node, and another array that is a hash relating a node in G to where are its parameters in the first array. We compute the array and the hash elements using atomic operations on the GPU.

In the optimization step, r and J are computed in parallel. A and b are computed by using the matrix-matrix and matrix-vector multiplication from CUBLAS[1] library. The linear system is solved by using a GPU implementation of the LLT decomposition proposed in [Henry 2009] together with a linear solver *Strsm* from the CUBLAS library.

[1] http://docs.nvidia.com/cublas/index.html

5.3 Integration into the MAR Environment

To add support for non-rigid tracking, one solution is to apply it whenever the rigid tracking fails, enhancing the robustness of the MAR environment. However, to apply non-rigid tracking for every frame has a computational cost which does not make it suitable for real-time applications. Therefore, if rigid tracking keeps failing consecutively, non-rigid tracking will be used more frequently, reducing user interactivity.

To solve this problem, we take advantage from the volumetric representation of KinectFusion algorithm to update the 3D reference model in real-time based on the current deformation measured. When the rigid tracking fails (i.e. error measured is above a certain threshold), non-rigid registration is applied and the 3D reference model deformed surface is sent to KinectFusion's grid with a high weight. 3D reference model is updated in the grid representation by the TSDF computation and then the grid is ray casted to generate a new source surface for the next iteration. High weight is used for fast adaptation of the previously stored 3D reference model into the new deformed one. As consequence, by deforming the 3D reference model, non-rigid tracking converges faster and with higher accuracy in the next iterations than the rigid-only solution (i.e. in which only rigid tracking is applied and KinectFusion's volume is not updated).

6 Results and Discussion

In this section we describe the experimental setup used and analyse performance and accuracy of the proposed environment. For all tests, we ran our algorithm on an Intel(R) Core(TM) i3-3220 CPU @3.30GHZ, 4GB RAM, NVIDIA GeForce GTS 450. We have implemented the algorithms in GPU by using the NVIDIA CUDA[2] architecture [Kirk and Hwu 2010]. We used the open source C++ implementation of KinectFusion released by the PCL project [Rusu and Cousins 2011]. 3D reference model was reconstructed with the KinectFusion using a grid with volume size of $70cm \times 70cm \times 140cm$ and resolution of 512^3.

We have tested the approach in a scenario where the user's head is the natural marker. To detect it from the scene, we have used the Viola-Jones face detector [Viola and Jones 2004] on the color map captured by the Kinect sensor. For rigid tracking, we have used the ICP algorithm because it is the only depth-based tracking algorithm which has a variant with real-time performance, therefore, being suitable for augmented reality applications. For non-rigid interactions, we asked the user to perform three different facial expressions after 3D reconstruction: inflate his cheeks, smile and simulate a "kiss" expression, shown in Figure 4.

In all tests we have used k = 4 (i.e. the number of nodes which influences each point on the source surface, Equation 1), three iterations of the optimization and three levels of the quadtree to limit the computational cost of the non-rigid registration, while achieving good accuracy. E_{con} was used as a measure for refinement/collapse of nodes. Although Li and colleagues [Li et al. 2009] suggested the use of E_{reg} for this measurement, our tests show that we do not have accuracy lost by using E_{con} instead of E_{reg}.

As explained in the previous section, we need to update the 3D reference model to minimize the use of the non-rigid registration algorithm. To accomplish that, one solution is to re-send the 3D deformed reference model into the grid with high weight. As explained in Section 4, the KinectFusion algorithm integrates raw depth data into a grid based on TSDF computation and a weight which indicates uncertainty. The higher the weight, the faster the

[2] http://www.nvidia.com/cuda

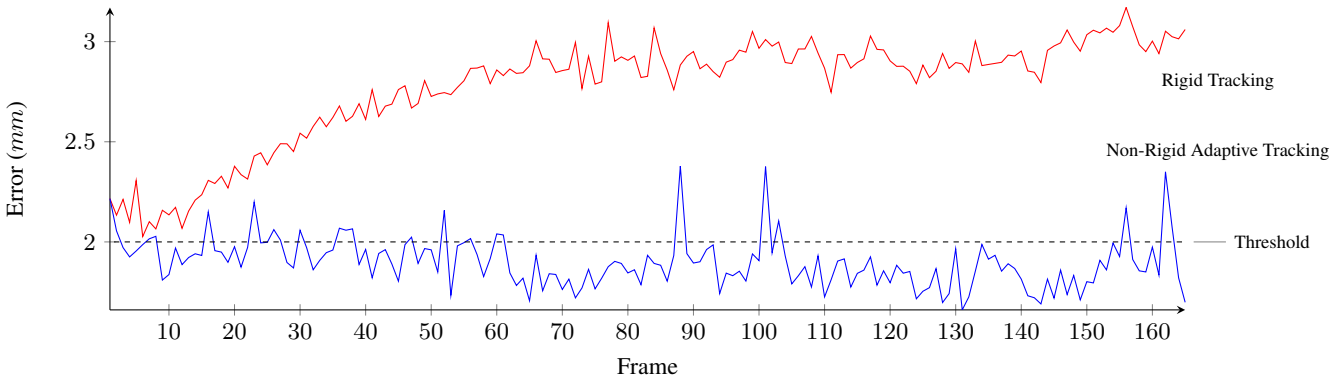

Figure 5: *Cheeks tracking error measured for both rigid and rigid + non-rigid solutions.*

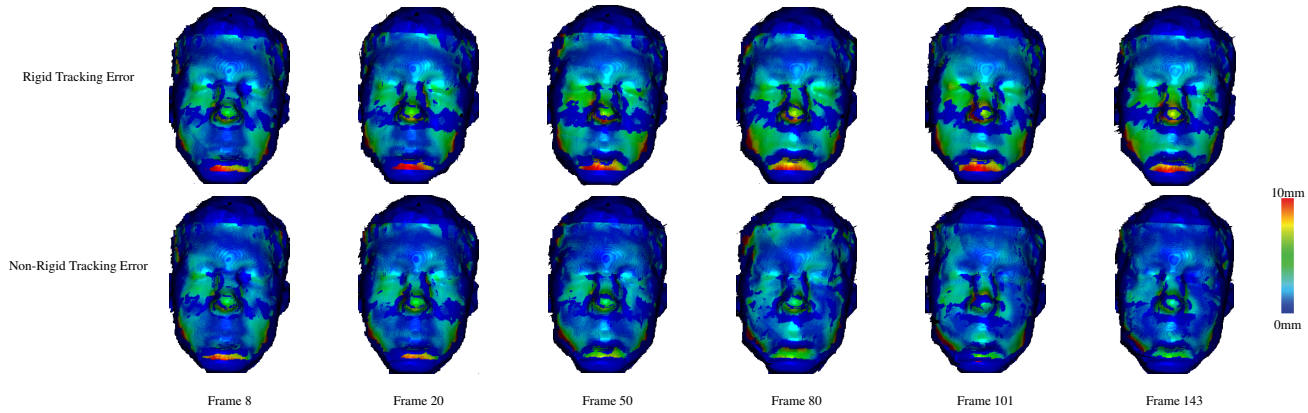

Figure 6: *Color-coded cheeks tracking error measured for both rigid and non-rigid solutions.*

3D reference model shape is updated based on the current measurement. Therefore, to accommodate the current deformation and to stabilize the tracking faster, a high weight must be used to update the 3D reference model. We have tested the influence of such updating on the tracking accuracy. This test can be seen in Table 1. While weight 1 does not result in fast update on 3D reference model shape, stabilization in terms of accuracy is achieved with weight between 8 and 16. We have used weight 8 for all the other tests performed in this section because it provides more stable results than weight 16 (vide standard deviation measurements in Table 1).

User Deformation	Cheeks		Smile		Kiss	
TSDF Weight	A	SD	A	SD	A	SD
1	2.4	0.15	2.23	0.19	1.90	0.17
2	2.2	0.14	2.09	0.16	1.83	0.16
4	2	0.1	2.03	0.14	1.81	0.16
8	**1.9**	**0.1**	**1.99**	**0.12**	**1.75**	**0.15**
16	1.9	0.1	1.99	0.13	1.75	0.16

Table 1: *Average accuracy (A, given in mm) and Standard Deviation (SD, given in mm) results according to the weight used to update the 3D reference model.*

From tests conducted on the three cases mentioned at the beginning of this section, we estimated an average accuracy of $1.5mm$ for rigid tracking during 3D rigid reference model reconstruction, which is performed at 30 frames per second (FPS). On average, non-rigid tracking requires $60ms$ per frame. The step which takes most time to be completed for every frame is the non-linear optimization, which demands $45ms$ per frame.

As can be seen in Table 2, when non-rigid user interaction is present, the average accuracy decreases for rigid tracking. We have tested different scenarios for non-rigid registration in order to evaluate the best multi-frame strategy to balance accuracy and performance. While skipping a specific number of frames (i.e. NR4, NR8) is a good strategy, to apply it for almost every frame reduces the performance while being, sometimes, unnecessary (i.e. NR1, NR2). Likewise, to apply it between a large number of frames (i.e. NR16, NR32, NR64) improves slightly average tracking accuracy while application's performance keeps almost the same when compared to the rigid solution. However, if high deformation occurs in-between these frames, the tracking will fail (i.e. error measured will be above a pre-defined threshold used to detect rigid tracking failure). To apply the non-rigid registration whenever the rigid tracking fails (i.e. NRAdaptive) is the best idea in order to solve every deformation which occurs between frames, while maintaining good accuracy (below $2mm$) and real-time performance (above 20 FPS). It is worthy to mention that, in this case, the algorithm is not applied almost for every frame as the 3D reference model is updated based on the present deformation, reducing the chances for rigid tracking fail in the next iterations. As can be seen in the plots of the Figures 5, 7 and 9, the algorithm is applied 21 times ($\approx$ for every 8 frames) for cheeks deformation, 66 times ($\approx$ for every 2,5 frames) for smile deformation and 16 times ($\approx$ for every 10 frames) for kiss deformation.

When the 3D reference model is continuously updated for a case in which there is a small region of deformation, it will become in-

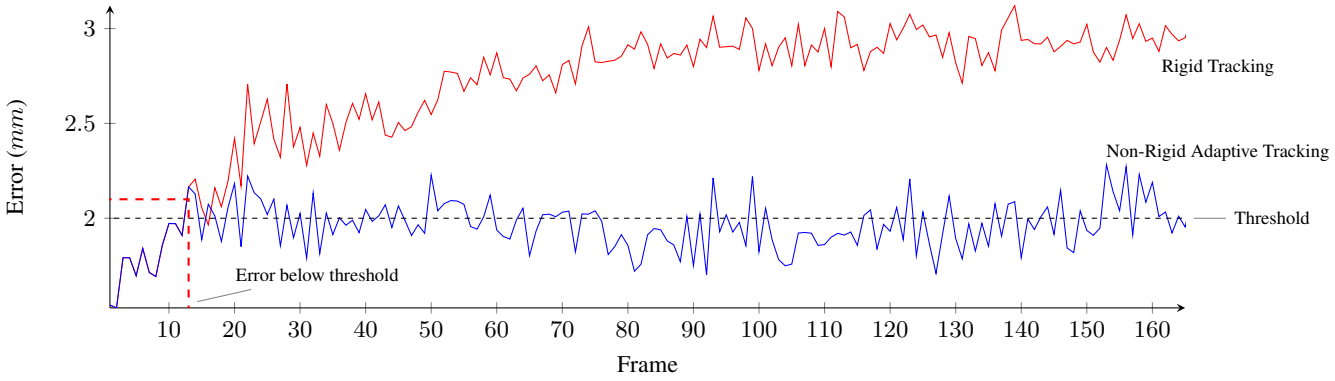

Figure 7: *Smile tracking error measured for both rigid and rigid + non-rigid solutions.*

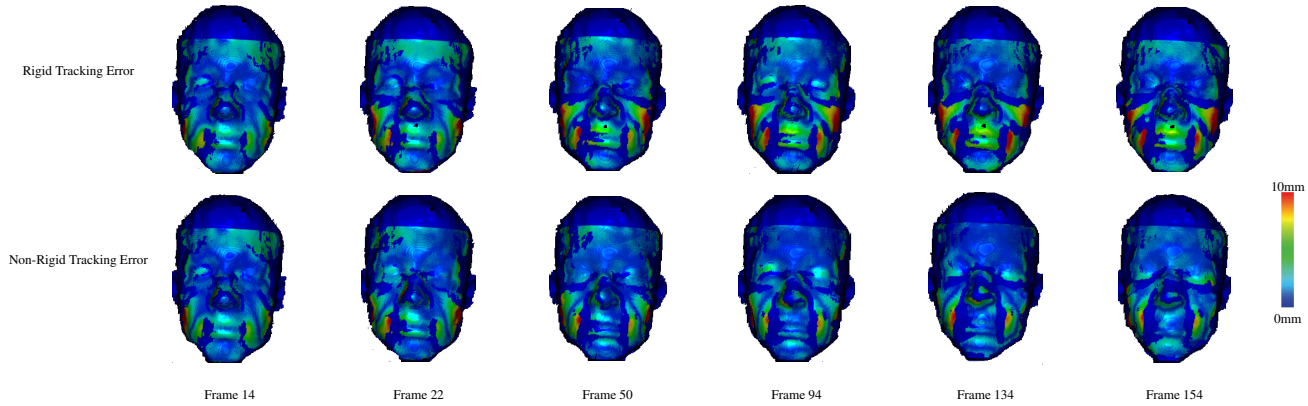

Figure 8: *Color-coded smile tracking error measured for both rigid and non-rigid solutions.*

User Deformation	Cheeks			Smile			Kiss		
Tracking/Measurement	Avg.	Std. Dev.	Perf.	Avg.	Std. Dev.	Perf.	Avg.	Std. Dev.	Perf.
Rigid	3	0.34	30	2.72	0.33	30	2.18	0.15	30
NR64	2.63	0.2	30	2.66	0.29	30	2.1	0.16	29
NR32	2.5	0.16	28	2.53	0.25	28	1.99	0.18	27
NR16	2.48	0.17	26	2.42	0.25	27	1.9	0.14	26
NR8	2.11	0.21	22	2.17	0.21	24	1.75	0.2	24
NRAdaptive	**1.9**	**0.1**	**20**	**1.99**	**0.12**	**20**	**1.75**	**0.15**	**27**
NR4	1.87	0.17	18	2.11	0.15	21	1.67	0.14	20
NR2	1.73	0.15	14	1.99	0.13	15	1.75	0.14	15
NR1	1.7	0.19	10	1.96	0.28	10	1.92	0.14	10

Table 2: *Average accuracy (Avg., given in mm), Standard Deviation (Std. Dev., given in mm) and Performance (Perf., given in FPS) results for each one of the tracking algorithms tested in presence of specific user deformation. NRn: Non-Rigid Registration applied for every n frames (independent of rigid tracking fail); NRAdaptive: Non-Rigid Registration applied whenever the rigid algorithm fails.*

creasingly smooth for each frame. In this case, this solution may be not the most accurate, as the 3D reference model will lose information in regions where there is no deformation. In Table 2, we can see this scenario from the tests conducted on the "kiss" expression, where non-rigid registration applied for every 1 or 2 frames does not produced the best results. This issue can be solved by using the adaptive approach.

Tracking error evolution can be seen in Figures 5, 7 and 9. When there is sufficient non-rigid user interaction, error grows considerably and the non-rigid solution minimizes it. 3D reference model is updated to stabilize the tracking based on the current deformation. Non-rigid registration and 3D reference model updating are done only when the deformation changes in intensity (i.e. error above the threshold, shown as a dashed line) and the rigid tracking fail. When using rigid tracking only, the error grows and stabilizes. This behaviour is motivated by the use of the face as deformable object, which has a limit in terms of deformation. With other objects, the error could grow even more without achieving stabilization.

A test to analyse the best threshold to detect rigid tracking fails was performed and the results can be seen in Table 3. As mentioned before, rigid tracking has average accuracy of $1.5mm$. Therefore, by using this value as threshold, the algorithm applies non-rigid tracking for almost every new frame. On the opposite case, by using threshold of $3mm$, the algorithm uses almost rigid tracking only.

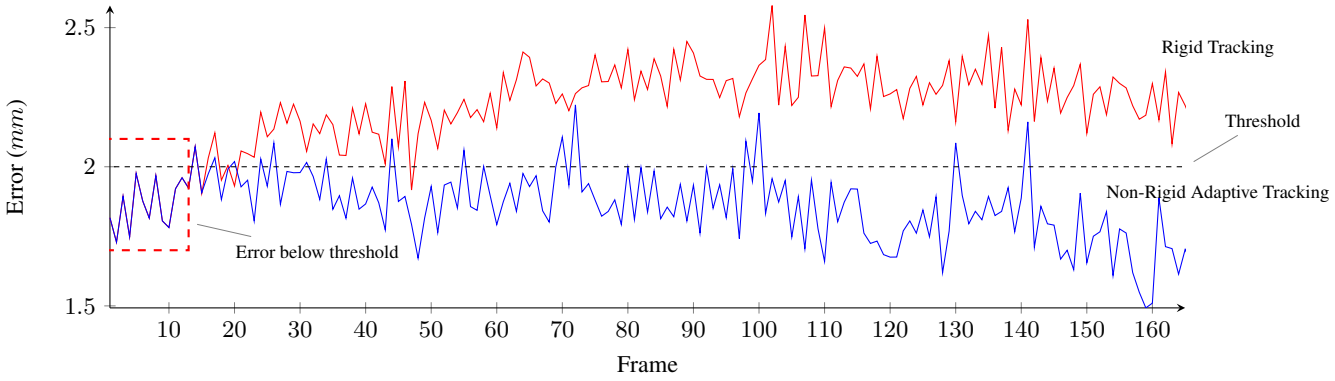

Figure 9: *Kiss tracking error measured for both rigid and rigid + non-rigid solutions.*

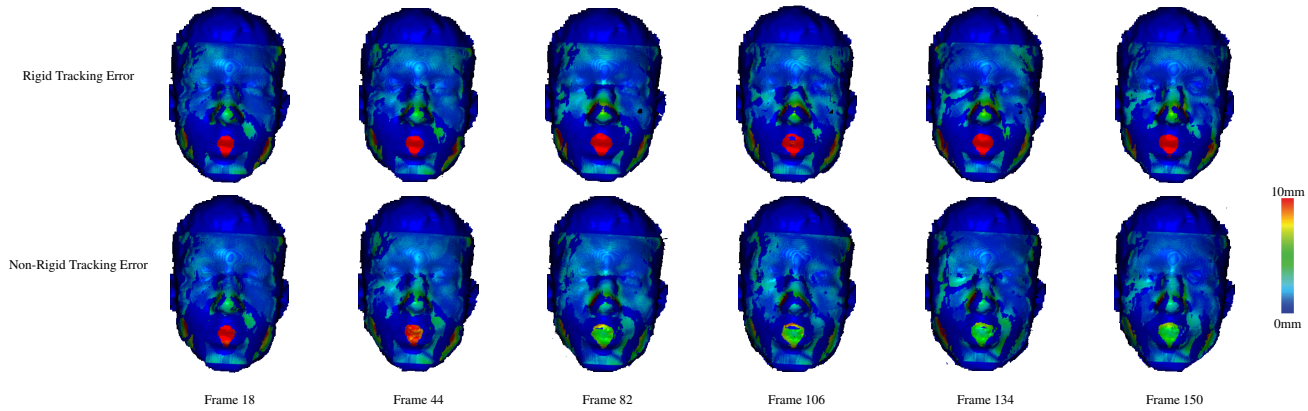

Figure 10: *Color-coded kiss tracking error measured for both rigid and non-rigid solutions.*

User Deformation	Cheeks			Smile			Kiss		
Threshold/Measurement	Avg.	Std. Dev.	Perf.	Avg.	Std. Dev.	Perf.	Avg.	Std. Dev.	Perf.
3	2.8	0.21	28	2.62	0.33	30	2.18	0.15	30
2.5	2.3	0.15	21	2.28	0.36	26	2.15	0.16	30
2	**1.9**	**0.1**	**20**	**1.99**	**0.12**	**20**	**1.75**	**0.15**	**27**
1.5	1.8	0.19	12	1.96	0.28	10	1.92	0.14	10

Table 3: *Average accuracy (Avg., given in mm), Standard Deviation (Std. Dev., given in mm) and Performance (Perf., given in FPS) results for each one of the thresholds used to detect rigid tracking fail.*

The best threshold is $2mm$, which provides fast and accurate tracking.

In terms of visual quality and accuracy, from Figures 6, 8 and 10, it is visible that the algorithm captures the main deformation present on the deformed expressions through the sequence of frames, improving accuracy in regions where only rigid registration cannot solve the tracking. In this sense, our main contribution is that the non-rigid registration algorithm runs near real-time, allowing its application for an augmented reality environment.

7 Conclusion and Future Work

We have presented a marker-free augmented reality approach with support to fast non-rigid surface registration. Fast non-rigid tracking is performed by an adaptive technique inspired by the Embedded Deformation algorithm. The main goal of the proposed algorithm is to improve tracking robustness for augmented reality applications, such as those for on-patient craniofacial data visualization. Therefore, tests were realized using user's face as natural marker and user's facial expressions as non-rigid interactions. From the tests conducted, we have shown that the non-rigid registration, applied in a multi-frame manner, is capable to run in real-time. Moreover, it improves the tracking accuracy of the augmented reality environment when compared to the rigid-only solution.

For future work, we intend to evaluate the performance of the proposed approach in both GPU and multi-threaded in CPU, which has already proven to be efficient in this context [Nutti et al. 2014]. In this sense, we must investigate how the processing could be balanced between these two processing units. Further tests must be done in order to evaluate the performance of the non-rigid registration algorithm in objects with more degrees-of-freedom than human's head.

Acknowledgements

We are grateful to Thibaut Weise for making his dataset available on the web. We are also grateful to the PCL project for providing the open-source implementation of the KinectFusion algorithm. This research is financially supported by FAPESB and CAPES.

References

BOUAZIZ, S., WANG, Y., AND PAULY, M. 2013. Online modeling for realtime facial animation. *ACM Trans. Graph. 32*, 4 (July), 40:1–40:10.

CHEN, J., IZADI, S., AND FITZGIBBON, A. 2012. Kinetre: Animating the world with the human body. ACM, New York, NY, USA, UIST '12, 435–444.

CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. ACM, New York, NY, USA, SIGGRAPH '96, 303–312.

DOU, M., FUCHS, H., AND FRAHM, J.-M. 2013. Scanning and tracking dynamic objects with commodity depth cameras. IEEE Computer Society, ISMAR '13.

GRASSIA, F. S. 1998. Practical parameterization of rotations using the exponential map. *J. Graph. Tools 3*, 3 (Mar.), 29–48.

HENRY, S. 2009. Parallelizing cholesky's decomposition algorithm. *INRIA Bourdeaux Technical Report*.

IZADI, S., KIM, D., HILLIGES, O., MOLYNEAUX, D., NEWCOMBE, R., KOHLI, P., SHOTTON, J., HODGES, S., FREEMAN, D., DAVISON, A., AND FITZGIBBON, A. 2011. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. ACM, USA, UIST '11, 559–568.

KIRK, D. B., AND HWU, W.-M. W. 2010. *Programming Massively Parallel Processors: A Hands-on Approach*, 1st ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

LEE, J.-D., HUANG, C.-H., HUANG, T.-C., HSIEH, H.-Y., AND LEE, S.-T. 2012. Medical augment reality using a markerless registration framework. *Expert Syst. Appl. 39*, 5, 5286–5294.

LI, H., SUMNER, R. W., AND PAULY, M. 2008. Global correspondence optimization for non-rigid registration of depth scans. *Computer Graphics Forum (Proc. SGP'08) 27*, 5 (July).

LI, H., ADAMS, B., GUIBAS, L. J., AND PAULY, M. 2009. Robust single-view geometry and motion reconstruction. *ACM Transactions on Graphics (Proceedings SIGGRAPH Asia 2009) 28*, 5 (December).

LI, H., LUO, L., VLASIC, D., PEERS, P., POPOVIĆ, J., PAULY, M., AND RUSINKIEWICZ, S. 2012. Temporally coherent completion of dynamic shapes. *ACM Transactions on Graphics 31*, 1 (January).

LI, H., YU, J., YE, Y., AND BREGLER, C. 2013. Realtime facial animation with on-the-fly correctives. *ACM Transactions on Graphics 32*, 4 (July).

MACEDO, M., APOLINARIO, A., SOUZA, A. C., AND GIRALDI, G. A. 2014. A Semi-Automatic Markerless Augmented Reality Approach for On-Patient Volumetric Medical Data Visualization. In *SVR*.

MADSEN, K., BRUUN, H., AND TINGLEFF, O., 2004. Methods for non-linear least squares problems.

NEWCOMBE, R. A., IZADI, S., HILLIGES, O., MOLYNEAUX, D., KIM, D., DAVISON, A. J., KOHLI, P., SHOTTON, J., HODGES, S., AND FITZGIBBON, A. 2011. Kinectfusion: Real-time dense surface mapping and tracking. IEEE Computer Society, Washington, DC, USA, ISMAR '11, 127–136.

NUTTI, B., KRONANDER, A., NILSING, M., MAAD, K., SVENSSON, C., AND LI, H. 2014. Depth sensor-based realtime tumor tracking for accurate radiation therapy. *Proc. of Eurographics 2014 Short Papers* (April).

RUSINKIEWICZ, S., AND LEVOY, M. 2001. Efficient variants of the ICP algorithm. In *3DIM*.

RUSU, R., AND COUSINS, S. 2011. 3d is here: Point cloud library (pcl). In *ICRA*, 1 –4.

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3 (July).

TAM, G. K. L., CHENG, Z.-Q., LAI, Y.-K., LANGBEIN, F. C., LIU, Y., MARSHALL, D., MARTIN, R. R., SUN, X.-F., AND ROSIN, P. L. 2013. Registration of 3d point clouds and meshes: A survey from rigid to nonrigid. *IEEE Transactions on Visualization and Computer Graphics 19*, 7, 1199–1217.

TOMASI, C., AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. In *ICCV*, 839 –846.

VIOLA, P., AND JONES, M. J. 2004. Robust real-time face detection. *Int. J. Comput. Vision 57*, 2 (May), 137–154.

WEISE, T., BOUAZIZ, S., LI, H., AND PAULY, M. 2011. Realtime performance-based facial animation. *ACM Transactions on Graphics 30*, 4 (July).