Inoë ANDRE

SICOM
2016-2017

NII(National Institute of Informatics)
2-1-2 Hitotsubashi, Chiyoda-ku
Tokyo 101-8430 Japan

Computer Vision
3D reconstruction using RGBD cameras

from 13/02/17 to 11/08/17

Under the supervision of :

- **Company supervisor : Akihiro, SUGIMOTO, sugimoto@nii.ac.jp**
- **Phelma Tutor : Alice, CAPLIER, alice.caplier@gipsa-lab.grenoble-inp.fr**

Confidentiality : ~~yes~~ no

Ecole nationale
supérieure de physique,
électronique, matériaux

Phelma
Bât. Grenoble INP - Minatec
3 Parvis Louis Néel - CS 50257
F-38016 Grenoble Cedex 01

Tél +33 (0)4 56 52 91 00
Fax +33 (0)4 56 52 91 03

http://phelma.grenoble-inp.fr

Acknowledgements

I thank Professor Akihiro Sugimoto for supervising the project by providing support, materials and time to follow the progress.

I am grateful to Assistant Professor Diego Thomas for sharing his experience in Computer Vision, answering my questions and providing code to optimize some algorithms.

I finally thank Hoai Ngo Nguyen for providing data enabling me to start the project with all necessary material.

Glossary

6 degree of freedom A matrix that enables 3 translations and 3 rotations which means 6 degree of freedom. This matrix is the position of the camera in the 3D space. 12

Depth image is a raw mapping of the space. It contains a 2D array which have value of depth given by a RGBD camera. 12

GPU Graphical Processing Unit. 11

Mesh is common representation of a 3D model because it can be shaped according to the one's will. 12, 13

RGBD Red Green Blue Depth. 9

TSDF Truncated Signed Distance Function. 12, 17

Vertex coordinates of a 3D point. 12, 14

Voxel An elementary 3D space. In 2D, it would be called pixel. 13

Contents

List of Figures

List of Tables

1 NII : National Institute of Informatics

The National Institute of Informatics (NII), established in 2000, has a mission of promoting basic research in computer science from a long-term perspective together with practical research designed to solve issues that society confronts. The technological and research challenges in the 21th century requires to deal with environmental, life and information science issues that impact our lives. That is why the Research Organization of Information and Systems has been created. It manages the link between four national institutes (The National Institute of Polar Research, The National Institute of Informatics, the Institute of Statistical Mathematics and the National Institute of Genetics) to meet today's needs. In this framework, NII provides its resources to support fast and effective development of research in Japan and abroad.

Figure 1: National Institute of Informatics Building

This institute aims at solving advanced integrated research and developing information-based fields of studies such as networking and software. Until now, only the NII does academic research in informatics in Japan. As a consequence it provides infrastructure, academic content and services for the sake of research and academic education. For example, it is involved in the Science Information Network project which provides a network of 100 giga-bits per seconds in the whole nation. As such it is a world-class powerful network. NII takes wholeheartedly the mission of educating and helping research in the academic community. Besides it developed partnerships with universities, research institutions, industries and civilian organizations. In addition, NII keeps furthering international exchanges by hosting conferences with international prominent professors or by enrolling international internship students through the International Internship Program and Memoranda Of Understanding. It has also implemented international cooperations with the Japanese-French Laboratory for Informatics and the special German Academic Exchange Service program.

Nowadays informatics must combine diverse disciplines in order to tackle state-of-the-art challenges. Solving them would contribute to the society. In this respect, NII focuses on four main themes. Therefore it has four division: Principles of Informatics Research Division, Information Systems Architecture Science Research Division, Digital Content and Media Sciences Research Division, and, Information and Society Research Division. The Digital Content and Media Sciences Research Division conducts research on all sort of media such as text, video and applications. In this division, Sugimoto laboratory focuses on computer vision and discrete geometry. Both fundamental theories and application system are covered in Sugimoto's group. The currently explored themes are modeling a dynamic 3D scene using a RGBD camera, gaze detection and navigation, and discrete geometric model fitting. The project explained in this document, treats with computer vision, modeling a dynamic 3D scene using a RGBD camera.

2 Introduction

The use of 3D models grows constantly for advanced technologies such as surgery, game, animation or virtual/augmented reality, scene reconstruction. Since 2010, more and more cheap ranging camera like Time-of-Flight camera or Red Green Blue Depth (RGBD) camera have emerged. These technologies enable to reconstruct a high-quality and geometrically precise 3D model of a scene in real-time. Companies are thus more and more eager to use these technologies for innovation. The demand for a variety of applications ceaselessly increase. Notably, applications to capture user environment or reconstruct moving person in real time. It enables to capture the motion of human activities, to determine human action or reuse the movement for animation purpose. For 3D modeling, there are two steps for reconstruction. The first one already starts to be well mastered. It consists of reconstructing rigid objects. Then, since 2015 with DynamicFusion [8] reconstruction of non-rigid object has become possible. It reconstructs slow moving object or human motions.

However, although many works [9, 5, 3] can do static scene reconstruction in real time, having fast moving or topologically changing objects still remain a challenge. In [8, 7, 4], several solution are proposed to solve these issues. However, they have either topological change limitations or limit detail representation by excluding high frequency details (texture mapping).

This work focuses on 3D reconstruction of moving human body with a RGBD camera. `DynamicFusion` [8] presented the first method of incrementally reconstructing a moving body from a single Kinect sensor in real-time which is also the goal of the presented work here. The project here use skeletal information to create topology of the body and help tracking fast motions. It consists in building a model of each rigid body parts and then stitch these parts to have a complete model the human body.

In this work, a new method which aims at reconstructing human movement is presented. The first contribution is to reconstruct body parts given depth image and skeleton. The second contribution is to be able to create 3D model of a human body by stitching the limbs together. The first part of this work deals with getting the data and how to use it so that the second part, reconstruction of scene, can be achieved. The following section presents how to distinguish rigid and non rigid body parts. The fifth section explains how to use this distinction to model the whole body in 3D and the last section concerns determining the camera pose in order to fuse the data of several images. The model can thus be updated at each new frame.

Related work

Skeleton and stitching :

Using skeleton to chopped the body and then reconstruct a stitched model is a technique used in [10, 12]. However in these papers, models of the body are already comprehensive from the start; a body part is fully reconstructed from the start. It just need to be estimated anew for different pose. In this report, the aim is using the chopping and stitching techniques to reconstruct a model frame by frame.

Kinect Fusion :

The fundamental work to scene reconstruction in real-time by updating a 3D model at each new frame is Kinect Fusion [9]. The update is based on Iterative Closest Point algorithm [2]. Kinect Fusion is able to reconstruct 3D model only for static scene.

Non-rigid fusion :

DynamicFusion [8] approach reconstructs scene geometry whilst simultaneously estimating a dense volumetric 6D motion field that warps the estimated geometry into a live frame. It has thus two models. One static model that is updated as in Kinect Fusion and one that follow the movement of the non-rigid objects. Meshes are made through marching cube [6]. It shows results of great quality, however because the reconstruction is based on mesh correspondence, it cannot properly reconstruct splitting object or topological change.

VolumeDeform [4] improve this approach by using scale invariant feature transform for tracking big movement. It enables to have faster motion compare to dynamic fusion yet it still cannot properly recover all free motion.

Topological change : KillingFusion [7] succeeded in having topological change by doing image correspondence directly in a signed distance field. KillingFusion used the signed distance field to do both tracking and meshes reconstruction. Nevertheless, the quality can be improved; high frequency details are lacking.

3 Data conversion

The purpose of the section is to give a clear idea of the environment of work (data,tools) and how to transform raw data to useful data. First, an overview gives the working conditions and the general view of the process to make 3D models. Then, converting raw data into specific map is the starting point to have useful data.

3.1 Overview

This section describes data and explains general view of the process to do rigid reconstruction.

3.1.1 Tools and data

RGBD cameras are really useful for 3D modelling because it is cheap and provide depth image in a high pace. As a consequence, real-time 3D reconstruction is possible. During the internship, the data from a Kinect V2 was used. It uses optical *time-of-flight* technology to measure time of emission and reception. The standard output of Kinect V2 is depth image, skeleton and color image. The RGBD camera is made with two cameras one classical RGB camera, and one infrared camera that gives depth information. To put it simply, a depth image is an image that give numbers representing the distance from the sensors to each surface. Knowing the speed of the wave emitted, it can deduced the distance. Since waves are emitted, the propagation is done in a spherical way. The skeleton is a list of points set at junctions of the body.

Here, the sequences of images was recorded with a hand-held camera (figure 2) moving randomly from right to left or left to right, up and down,etc. The two cameras get color images and depth images at the speed of 30 fps. It is synchronized so that depth and color images have pixels correspondence. An image (figure 2) of the camera give an idea of the material.

Figure 2: Kinect-v2

The datasets provided contains hundreds depth images. Each image is stored in a 424*512 matrix. Besides the datasets provide noisy binary images of the body, lists of joint positions for each depth or noisy binary images and sometimes color images. The sequences are generally set in a room with a human moving or not with different objects around.

Since RGBD cameras have been cost-effectively made, they have fabrication default. Consequently, it should be calibrated. Here is the intrinsic parameter matrix K of the camera used for calibration :

$$K = \begin{bmatrix} f_x & 0.0 & c_x \\ 0.0 & f_y & c_y \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{1}$$

With : $f_x = 357.32$: focal length x, $f_y = 362.12$: focal length y, $c_x = 250.12$: principal point on x axis and $c_y = 217.52$: principal point on y axis.

Using a factor (generally equals to 1000 with KinectV2), this calibration gives the relation between camera's pixels and real world units such as millimeters.

Another tool for future work is the python code of the project. It is available on GitHub [1]. It partly runs on Graphical Processing Unit, which is abbreviated GPU. The specifications are in 9.3.

The capacities of the GPU were enough for to compute it offline in relatively good time (minutes). The GPU was needed in order to diminish significantly the process time. With better material the aims would be to do this in real-time. This project has not been implemented in real-time, but future work should improve it to have real-time performance. As a consequence some algorithm are implemented aiming real-time performances.

Having defined the framework, the following section explains the process of how to create a 3D model and how to update it for rigid part.

3.1.2 Algorithm pipeline

This subsection presents the general view of rigid reconstruction.

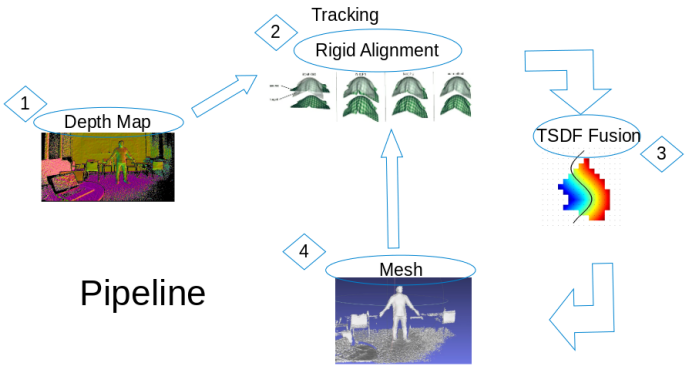


Figure 3: Pipeline of the rigid reconstruction

Standard rigid reconstruction is initialized at a frame with the creation of a 3D model. Then it globally runs on two steps. The first step consists in following the pose of the camera. It means continually updating 6 degree of freedom (defined in section 3.2.4,equation 5) pose of the camera at each new frame because each new frame is taken from a different position of the camera. In the second step, the depth data of the current camera pose are fused with the 3D model. That is to say the system updates the model of the first camera pose at each new position. Figure 3 introduce a simplified pipeline of the process done at each new current frame (or position of the camera).

Figure 3 have four images representing four steps to reconstruction a rigid scene in 3D and potentially in real time.

1. The first step consists in converting Depth image into Vertex and normal map, that is to say computing 3D points positions called *vertices* (plural of Vertex)and their *normales*. In the case of real-time processing, the image treated is the most recent images of the video stream. This image is called current image or current frame.

2. The second step have two inputs: the current depth map and a 3D representation of the scene, a 3D model called Mesh. A mesh is defined through triangles, vertices and normales. This step is called tracking because it determines the transformation matrix to get the new position of the camera. The purpose is to minimize positions of the two inputs in order to determine the transformation matrix between the two inputs. In order to do that, vertices and normales are aligned. This is also called rigid alignment because it only works for fixed scenes.

3. In the third step a function called Truncated Signed Distance Function (TSDF) updates a 3D volume of the scene. The transformation found in the former step aligns this volume with the

3d model. The volume is divided into Voxel in which the distance to surfaces of the 3D model are stocked. A Voxel is an elementary 3D space (in 2D, it would be a pixel).

4. In the last step, a Mesh is updated from the TSDF using marching cubes algorithm.

This pipeline is limited because it only presents how to do rigid reconstruction. It works when, only the camera is moving. If a body moves reconstruction spread on the whole motion. Having described it in block, the first step is explained in the following sub-sections.

3.2 Data conversion

Before exploiting raw data, camera should be calibrated and raw data should be pre-processed because of noise and hardware limitations. This section aims at explaining how to get rid of hardware limitations and how to get useful data. In this section, from raw data, the conversion process to create a map of distance homogenized in the same direction is described.

3.2.1 Depth map

This subsection present how the data provided by the depth sensor is made. As explained earlier, KinectV2 uses Time-of-flight technology. One sensor emits a wave. This wave is reflected by a surface and then a receptor of the sensor receive the wave. The time between the emission and the reception is measured. Given this time and the propagation speed of the wave, the distance camera to surface is obtained.

Figure 4 shows a depth image given by the depth camera.

Figure 4: Image representing the depth of each surface in a gray scale. The scene is a room containing a standing static human in a middle of chairs and tables.

In the gray scale scene, the more the surface is far, the more the depth is big. As a consequence, pixel is white are representing furthest surfaces. However, depth image can be noisy and have holes. That is why, Figure 4 have many black pixels even in the background. As the body is globally at a fixed distance from the camera, the body is uniformly colored in dark gray.

The depth images are scaled by a factor to convert the depth value in meter, i.e., for a factor equal to 1000, a pixel value of 1000 in the depth image corresponds to a distance of 1 meter from the camera, 2000 to 2 meter distance, etc. A pixel value of 0 means missing value or no data.

Having clearly defined the input, the following section consists in converting this input depth image into 3D position map called $Vertex\ Map$.

3.2.2 Vertex map

After receiving a depth image, to be able to do 3D processing, a vertex map can be created. Knowing that a Vertex is the position of a point in a 3D space, each pixel of the 2D image should receive a vertex. Therefore, creating a vertex map means creating a big matrix that has a 3D position stored in each of his index. In the case of Kinect v2, it is a 424*512*3 matrix because each depth image is a 424*512 sized image.

As the raw depth image noted $\mathbf{R}_i(\mathbf{u})$ is noisy, a pre-processing operation on the raw depth image improve the result of the vertex map and the normal map. As the edges should be preserved the bilateral filter operation is used here (defined in [9]):

$$\mathbf{D}_i(\mathbf{u}) = \frac{1}{W_p} \sum_{\mathbf{q} \in frame} N_{\sigma_s}(||\mathbf{u} - \mathbf{q}||_2) N_{\sigma_r}(||\mathbf{R}_i(\mathbf{u}) - \mathbf{R}_i(\mathbf{q})||_2) \mathbf{R}_i(\mathbf{q}) \tag{2}$$

W_p: Normalized constant and $N_\sigma(t) = \exp(-t^2\sigma^{-2})$
$\mathbf{q}$ is the projection of 3D coordinates into 2D coordinates. $\mathbf{u}$ is the 2D coordinates of the depth image. This non-linear filter enables to replace an intensity value of pixel by a weighted average value of nearby pixels. This process preserves edges and reduces noise.

The conversion from the depth map to the vertex map at an image `i` means converting a 2D representation of space to a 3D representation of space. Given a pixel $\mathbf{u} = (x, y)$ in the depth map $\mathbf{D}_i(\mathbf{u})$ of the current image, the corresponding Vertex $\mathbf{v}_i(\mathbf{u})$ is given by the inverse perspective projection (defined in [9]). Therefore, the conversion from the 2D images to 3D point clouds works as follow:

$$\mathbf{v}_i(\mathbf{u}) = \mathbf{D}_i(\mathbf{u}) \mathbf{K}^{-1} [\mathbf{u}, 1] \tag{3}$$

In order to have homogeneous coordinates, $[\mathbf{u}, 1]$ is used instead of $\mathbf{u}$. The inverse intrinsic matrix $\mathbf{K}^{-1}$ enable to project pixels into the frame of the color image which means that there is a 1:1 correspondence between pixels in the depth map and the color image.

Having the position in the 3D space the direction of surfaces should be known to comprehensively define a surface in 3D. This requirement is achieved by computing the normal map.

3.2.3 Normal map

The normales of surfaces are estimated from neighbouring pairs of 3D points by exploiting the regular grid structure of the matrix image. Here the normal map is only computed through the vertices. Given the vertex map at an image `i`, the normal $\mathbf{n}_i$ at a vertex $\mathbf{u}$ is computed as follow (defined in [9]):

$$\mathbf{n}_i(\mathbf{u}) = (\mathbf{v}_i(x+1, y) - \mathbf{v}_i(x, y)) \times (\mathbf{v}_i(x, y+1) - \mathbf{v}_i(x, y)) \tag{4}$$

Normalized it to unit length : $\frac{\mathbf{n}_i}{||\mathbf{n}_i||}$

In equation 4, given a vertex, the distance vector between neighboring vertices is computed. The normal is obtained by computing the normalized cross product of these distances. The next subsection explains how these maps are useful for camera pose computation and how to visualized the results of these computations mainly for checking whether it is correct.

3.2.4 Visualization and representation

When the scene is reconstructed, the camera is moved from one point to another. It changes its position in the real world space. It is important to have an efficient way of computing the camera pose. This is done by updating a transformation matrix. Second, since it is not obvious to get 3D representation, visualizing a 2D image of the 3D space can help to debug.

Transform coordinate system:
As each frame i is represented by a global 6DOF $\mathbf{T}_{pose,i}$ camera pose, vertices and normales should be updated according to the camera pose. It is updated simply by multiplying by $\mathbf{T}_{pose,i}$ defined in [9] as follow:

$$\mathbf{T}_{pose,i} = \begin{bmatrix} \mathbf{Rot}_i & \mathbf{t}_i \\ \mathbf{0}^\intercal & 1 \end{bmatrix} \tag{5}$$

$\mathbf{Rot}_i$ is a 3x3 rotation matrix. $\mathbf{t}_i$ is a 3x1 translation matrix. Therefore computing $[\mathbf{v}_i(\mathbf{u}), 1] = \mathbf{T}_{pose,i}[\mathbf{v}_0(\mathbf{u}), 1]$ and $\mathbf{n}_i(\mathbf{u}) = \mathbf{Rot}_i\mathbf{n}_0(\mathbf{u})$ transform the vertices and normales of the first frame 0 into the current global vertices and normales.

Perspective projection:
The transformation $\mathbf{q} = \pi(\mathbf{p})$ from 3D space to 2D space is called perspective projection. Let X,Y,Z be the 3D coordinates of p (a vertex), and x,y the corresponding coordinate of the point q in the 2D image. $\frac{X}{Z}, \frac{Y}{Z}$ compute the angular distance from the center of the camera. This center is given in equation 1. Besides the 2D image have all the pixel in a fixed depth. This depth is chosen as the focal distance and the 2D image coordinates are found. It leads to compute([11]):

$$x = f_x \frac{X}{Z} + c_x, y = f_y \frac{Y}{Z} + c_y \tag{6}$$

Then each normal can be used as if it was color channels. The colors of object give direction of surface. This is how the figure 5 is obtained

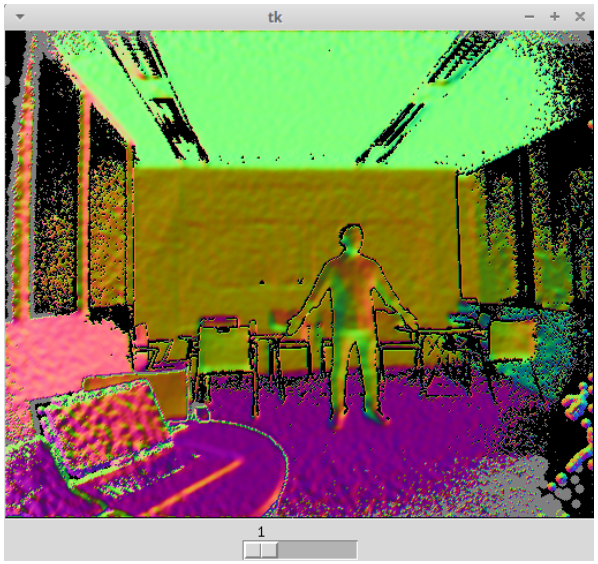


Figure 5: Scene reconstruction from 3D points and normales. The unnatural colors are given by the values of normales and shapes are given by projected vertices.

The colors are not natural colors but they give an idea of the surface that are differently oriented from one object to another. For example, the ceiling and the chin have similar color. This looks right because both of those surface are oriented towards the ground. The edges of the depth map 4 are preserved here. The projection is done correctly.

In section 3, all fundamental tools have been define to tackle the issue of moving human reconstruction. Vertex map and normal map are the tools that contain all necessary 3D information and the 6DOF enables to update these maps to the position of the RGBD camera.

Converting the depth map gives sparse information of the space. There is only a cloud of points that contains information to reconstruct surfaces. However surfaces themselves are represented. The following section consists in creating a 3D model from the cloud of points.

4 3D Modelling

Through a cloud of points a 3D model is generated. At the first image, a 3D model is created. For all following images, the model is only updated. This sections explains in two steps how to create or update a 3D model. The first part describes how to represent a surface thanks to a distance function called TSDF. The second step shows how to get the wanted surface.

4.1 Surface representation

By using the information of the cloud of points of the current image i, the volume created at the first frame will be updated. Updating the information in the volume from the cloud of points is called fusion. To achieve fusion, the volume is transformed to be aligned with the current image. Then distances of voxel to the surface contained in cloud of points are associated to the volume (thus voxels). The following subsection details the processes.

4.1.1 Volume transform

The first step to have a 3D model starts by creating a volume big enough to contain the scene. The volume is divided into elementary 3D space called voxel. When the volume is created, it is formed in a local space. Therefore, there are no links between the current image coordinates and the volume coordinates. That is why the volume should be transform so that it can be linked with the current image.

The volume is created as a voxel grid of 512*512*512 voxels. In the given discrete setting, all points in space that belong to a certain voxel obtain the same properties. Thus an index $(X, Y, Z) \in \mathbb{N}^3$ refers to the whole voxel. Once the 6DOF camera pose estimated $\mathbf{T}_{pose,i}$ is updated for the current image i (see 7), it is used to transform each voxel into the frame of the current image (in the same way as the vertices and the normales in 3.2.4. Here are the equations based on Kinect Fusion [9] to transform the volume voxel by voxel :

$$\mathbf{q}' = \mathbf{T}_{pose,i}\mathbf{K}^{-1}\mathbf{p} \tag{7}$$

The equation can be read in two steps. It uses the perspective projection 6 on $\mathbf{p}$,which gives $\mathbf{q}$, the projection in the current image. $\mathbf{q}$ is then transformed in the current frame giving birth to $\mathbf{q}'$. Therefore, voxels of the volume are aligned with the vertices of the current image.

Having all voxels aligned with the cloud of points, the next step is to compute the distance between the center of voxels and the points representing the surface.

4.1.2 Truncated signed distance function

The signed distance function (SDF) is a function that associate a distance to each voxel of the volume. This distance is the voxel to surface distance. Therefore the main idea is to do difference between the center of the voxel and the depth position of the vertex that correspond following the depth axis.

Figure 6 is an illustration of how the distance to a surface is computed in 2D. The volume created by the TSDF (TSDF volume) is the square containing the whole scene. From the camera view, the visible part is positive and once the ray cross the surface the value of distance given to a voxel is negative. Voxels that are too far from the surface are clamped to 1 or -1.

As a consequence, in the global volume containing all the voxels associated to a distance, the zeros cross will give the position of the surface. Besides, since there is an uncertainty in the position of the surface, the function is defined to saturate the value to 1.0 or -1.0 outside the uncertainty. In the zone of uncertainty , that is to say, when the voxels are really close to the surface, the values are between -1.0 and 1.0. Let [-μ,μ] be the uncertainty zone, $F_{R_{pose,i}}(\mathbf{p})$ the current truncated signed distance value at the coordinate $\mathbf{p}$ and Ψ the function that operates the truncation, the following equation compute the TSDF (based on Kinect Fusion [9]):

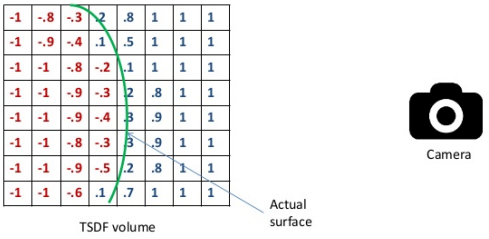


Figure 6: Illustration of the TSDF in 2D

$$F_{R_{pose,i}}(\mathbf{p}) = \Psi(\mathbf{q}_c - R_i(\mathbf{x})) \tag{8}$$

$$\Psi(\eta) = \begin{cases} \min(1, \frac{\eta}{\mu})\ sgn(\nu) & \eta \geq -\mu \\ \text{null} & \text{otherwise} \end{cases} \tag{9}$$

This is the process for one frame. However, to avoid computation, for each new image, updating the volume instead of computing it again increases the speed of computation and diminishes the error in the estimate of the surface.

4.1.3 Weight update

Updating the TSDF is convenient to gain processing time. Global fusion of all depth maps in the volume is formed by the weighted average of all individual TSDF volume. Since each image is noisy, this weighted average is seen as a de-noising filter. The final update of the TSDF volume is thus more accurate. The solution adopted here is simply using a weighted running average, defined for each point as follow (defined in [9]):

$$\mathbf{F}_i(\mathbf{p}) = \frac{W_{i-1}(\mathbf{p})F_{i-1}(\mathbf{p}) + W_{R_{pose,i}}(\mathbf{p})F_{R_{pose,i}}(\mathbf{p})}{W_{i-1}(\mathbf{p}) + W_{R_{pose,i}}(\mathbf{p})} \tag{10}$$

$$W_i(\mathbf{p}) = W_{i-1}(\mathbf{p}) + W_{R_{g,i}}(\mathbf{p}) \tag{11}$$

In practice, choosing $W_{R_{pose,i}} = 1$ results in a simple average that provide good results. Moreover, the weighted average can be truncated to a value W_η in order to get a moving average aiming reconstruction of moving objects (defined in [9]).

$$W_i(\mathbf{p}) = min(W_{i-1}(\mathbf{p}) + W_{R_{pose,i}}(\mathbf{p}), W_\eta) \tag{12}$$

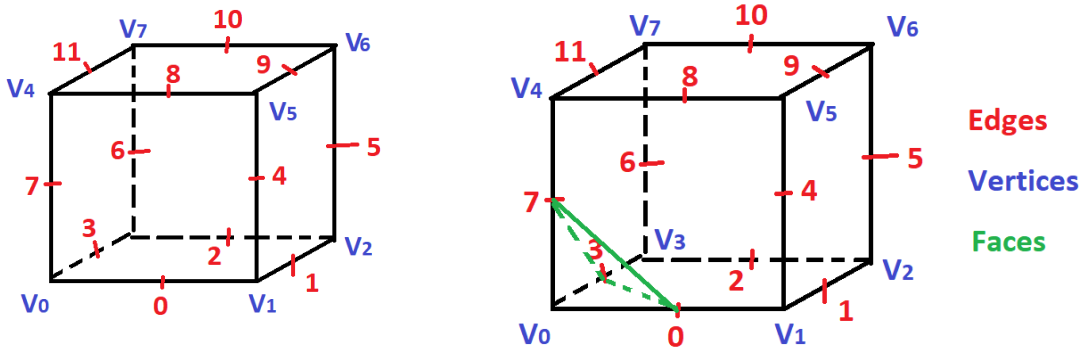
With this, the surface is measured in the 3D space. However this representation of the surface is not made for visualization and it is not used to register to a new current image defined by vertices and normales. Therefore, from this TSDF representation, the marching cubes algorithm enable to render a surface through triangulation using vertices and normales.

4.2 Marching cubes

Marching cubes algorithm here enables to pass from a volumetric implicit function representation (TSDF volume) to triangulated mesh representation of the surface. Here, the volumetric implicit function is a three dimensional scalar field that is called TSDF volume. Marching cubes algorithm uses the iso-surface and value of vertices and normales to create triangular patches surface. In the case of this project the iso-surface has the value 0 in the TSDF volumetric representation. It is preferred from the ray-tracing method like it is used in [9] which generate high end 3D graphics because a mesh is require to do non-rigid reconstruction as it is done in Dynamic Fusion [8].

4.2.1 Principle and convention

It is called marching cubes because it uses a cube that runs through all the voxels of the TSDF volume in order to create triangular surfaces in each voxel according to the value of the distance voxel-surface. For each voxel the 8 cube's corners are associated with a binary number. Either one if the value of the vertex is bigger than the iso-surface (which means positive) or 0 otherwise (the vertex is negative). Here the sign of the vertex is thus identified to know its position according to the surface. Whether the surface cross the voxel and according to the position of the summits of the cube compared to the surface, some triangle will be created. Since there are 8 corners on the cube, it gives $2^8 = 256$ combinations of vertex disposition. For each combination, a specific surface is created. See a simple example on Figure 7b.



(a) V0 to V7 : vertices, 0 to 12 : edges

(b) Example of facets

Figure 7: Cube edges and vertices convention

In the Figure 7a, a drawing shows the convention chosen in this project to compute the surface. Figure 7b shows an example of how the surface is created. In Figure 7b , the green lines stand for the surface created in the case only V0 was negative and all other vertices were positive. Therefore, the number $00000001_{(2)} = 1$ is the representation of the vertex combinations. In this representation each bit correspond to a summit. Besides, in that case, since V0 and V1 are two adjacent vertices and have different signs, the edges 0 is cut by the surface. Accordingly 3 and 7 edges are also cut by the surface. The intersections of cut edges and the surfaces give points. Linking these points create a triangle. From this triangle a surface is created.

Applying the process of creating a surface for all voxels, the whole surface will be reconstructed.

Since the surface is based on triangles, in order to get a more realistic rendering, the normales are used. This is called smooth rendering. The approach used in the project consists in computing for each vertex, the average normal of the triangle sharing the vertex. It can be done by running through all the vertex and accumulating the normal of the current face/triangle in the three vertices composing the triangle. Finally, all normales are normalized.

Each combination correspond to a known surface. In practice, all combinations are stocked in tables. Consequently, once a vertex combination is deduced, this combination is used as an input of a table containing edge combinations.

4.2.2 Tables

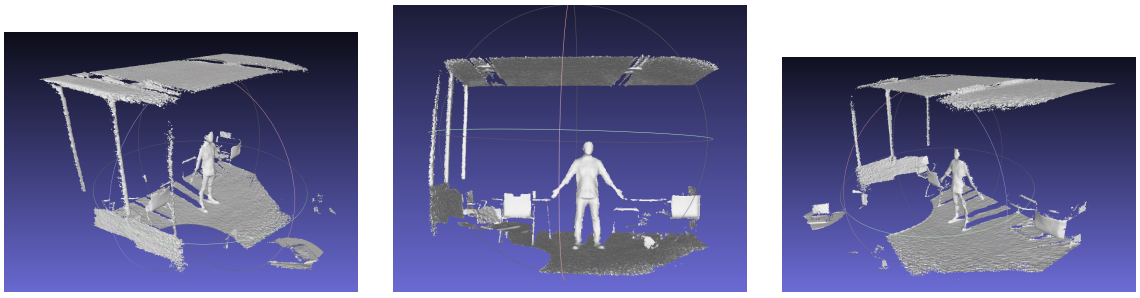
From the binary vertex combination, an index *indx* is defined. From this index, two types of information can be obtained.

First, a table can stock the combination of cut edges according to *indx* (a vertex combination). Since there are 12 edges, a table called $IndexVal$ return a 12 binary array for a particular index. One binary number equals 1 if the corresponding edge is cut, 0 otherwise. For example, in the Figure 7b case: $IndexVal[indx] = 000010001001_{(2)}$ because the edges 0, 3 and 7 are cut. The bit in position 0, 3 and 7 are equals to 1. Knowing a cut edge, its corresponding vertex s_1,s_2 and values v_1 v_2 , the vertex of the intersection $v_{intersection}$ of the cut can be computed as a mean of the two adjacent vertices:

$$v_{intersection} = \frac{s_1 v_1 + s_0 v_0}{v_1 + v_0} \tag{13}$$

The second table *Config* contains all the triplets that define configuration of triangular facets. There are maximum 5 facets per voxel. Consequently, for an index *indx*, *Config* return an array containing 16 values. The last value is always -1 because -1 means there are no more facets in the current voxel. In the case of Figure 7b : $Config[indx] = \{0, 3, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1\}$ Since there is only one face, there are only -1 from the fourth position.

TSDF and marching cubes algorithm are applied on this project dataset. Figure 8 is the mesh result of the whole scene taken from the dataset. To visualize it, the vertices and the faces created by the marching cube algorithm were written and converted into a *.ply* file that MeshLab is able to read.



(a) Left view (b) Front view (c) Right view

Figure 8: Mesh of the scene

Figure 8 shows a reconstruction of the scene using just one RGBD image. The view from only one image is incomplete. Therefore, it cannot contain full reconstruction object (forward and backward).

Many libraries have already implemented the marching cubes algorithm. However, using these libraries results in having closed volume covering of the whole uncertainty zone. For this project, just an open surface is necessary for the tracking; otherwise the tracking 7 is shifted by the volume of the uncertainty zone which breed false tracking results. That is why, the body seems flat. See figure 22 for a closer look. On the other hand, the surface are smooth thanks to the computation of normales.

In order to create a 3D model, two main algorithms are used: TSDF and marching cubes. TSDF is an efficient 3D representation of surface for computation and update. However it cannot be visualize or be used as a 3D model. That is why is marching cubes is used and give meshes. Having operational tool for reconstruction, the aims of the project is to reconstruct human in motion. To reach this goal the idea is to create model of body parts separately and then stitch it together. For that purpose, segmenting the body is necessary to separate the reconstruction.

5 Segmentation

Segmentation is an important step to reconstruct 3D model of moving body. Indeed, making body part moving individually enables to compute motion for each body part separately. A body is made out of bones. Since bones remains rigid, most of the body can be considered as rigid. As a consequence, to reconstruct a human body, most of it can be treated as it is rigid even when the body moves. Distinguishing different rigid part in the human bodies lies in a process that can automatically define the interesting parts : segmentation.

Instead of processing the whole body directly, repeating 3D modelling on several little part would not only enable to have faster computation and less memory use but it also avoids unnecessary computations of dynamic change on rigid part. Besides, in standard non-rigid reconstruction, when there is topological change such as passing from joined hands to separate hands, the model creates a surface between the separated hands. Doing segmentation avoid that kind of issues. Such process is mainly done through the use of human skeleton and images segmentation. In this part, to get simple data the human body should not be moving, the scene is static.

In the section, the aim is to segment the body into different parts in order do the fusion on each different part independently. As the RGBD camera can provide some data concerning only the body it will be used to do the segmentation.

5.1 Skeleton

The RGBD camera provide the skeleton of the human body. This is the based data to do the segmentation. A skeleton is defined by a list of point which are located in the junctions such as wrists, elbows, shoulders, knees. The segmentation is actually done based on the skeleton. By knowing the joints see Section 9, it is possible to define areas that correspond to rigid part. The depth sensor can actually directly give the information of the list of point defining the skeleton. Here, the list is composed of 25 joints. To get the drawing of the skeleton, pairs of joints is given by a list to know which joint should be connected. Figure 9 shows a skeleton on a body.

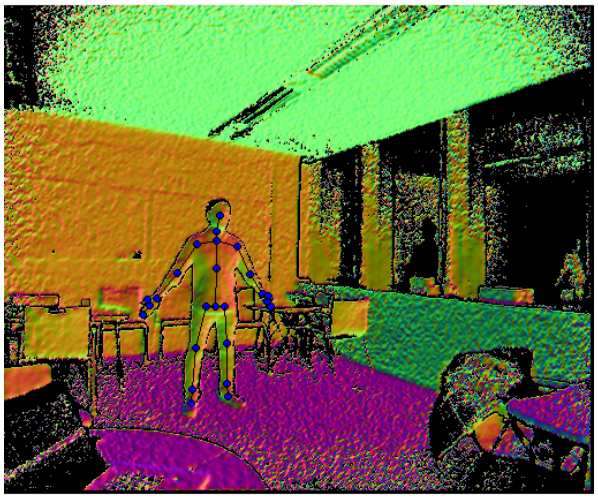


Figure 9: Depth map image of a room with a person. It comes from the projection of the vertices in 2D and the colors are made with normales

On the depth image in Figure 9 skeleton links are drawn in black while joints are drawn in blue disk. It fits perfectly with the body. Consequently, it is used to do image processing or human motion tracking. First it is used to compute body part regions and thus segmenting the body.

5.2 Slopes

Defining regions of each body part is done by using the slope between two extreme joints of the body part. For the limbs (arms, legs), a body part is generally defined by two adjacent joints. Since the parts are rigid, this slope is actually similar to the slopes of the edges between a limb and the background. Then for most limbs, computing the perpendicular to this slope give all other borders or segment that delimit the body part. This section explains how to find the points that define the contour of each body parts.

Before getting slopes, every elements of image that are not the body must be erased. As the body's depth dwells between two particular depths, a double threshold kill the background and the foreground. Then only small components remain. Selecting the biggest connected component (except the background) keep the body in the image. The second step, is to compute line equation between two joints of a limb, compute its perpendicular slope to finally get the crossing points between two slopes. A perpendicular slope pass by a junction of the limb. All points, found by crossing slopes, define the contour of the limbs. Here is the equation of the line $d : (y_B - y_A)(x - x_A) - (x_B - x_A)(y - y_A) = 0$ and the equation of the line d_n that goes through a point $C(x_C, y_C)$ and is perpendicular to AB $d_n : (x_B - x_A)(x - x_C) + (y_B - y_A)(y - y_C) = 0$

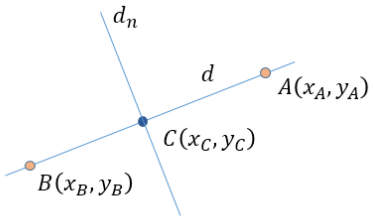


Figure 10: Line Equation

Using these equations, perimeters or polygons are defined. Nevertheless, the purpose of segmentation is to get labels for each body parts which means having filled uniform region for each body parts.

5.3 Polygon

Having found the slopes, polygons are defined. However, to get a label for a body part, the area of the polygon should be known and uniform. This section shows the way to fill polygons and get the label for the segmented part of the body. Polygons can be classified into concave and convex. Therefore, two ways are introduced to process the polygons. Since the human body is a known form, the type of polygon treated is known before starting the computation.

In case of convex polygon:
Each pixel is treated in the image to check whether this pixel is inside the polygon or not. For example, in Figure 11 there is a 4-sided polygon and the point $O(x_O, y_O)$ is inside the polygon. Note that once N is inside the polygon, both O and N are in the same side with the lines AB,BC,CD,DA. In opposite, when M is outside the polygon, the point O and M are in the different side of the line AB. A particular point is inside the polygon if and only if that point and point O are on the same side with all the polygon's lines.

To determine whether point P and point O are in the same side of a line $AB : ax + by + c = 0$, the coefficient alpha is computed :

$$alpha = (ax_0 + by_0 + c)(ax_P + by_P + c)$$

If $\alpha > 0$, both P and O are in the same side of AB
If $\alpha < 0$, both P and O are in different side of AB
If $\alpha = 0$, both P is on line AB.

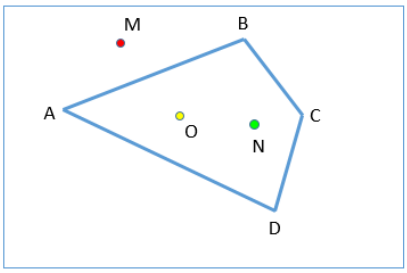


Figure 11: Convex Polygon

In case of concave polygon:

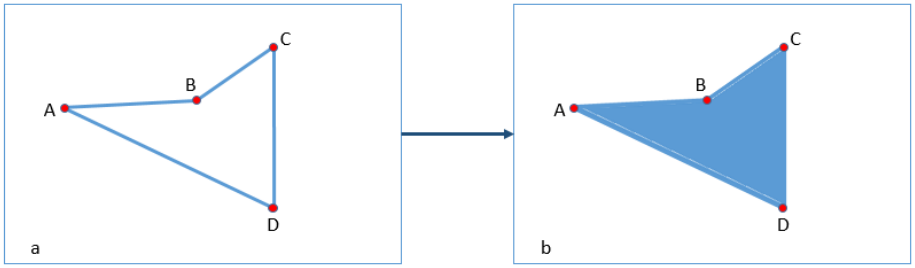


Figure 12: Filling concave polygon.
a. Input : borders of the polygon. *b*. Output : the polygon is filled

From the created contour as in Figure 12, flood-filling algorithm from the OpenCV library is used to get the segmented body. This algorithm determine the connected area that is connected to a point inside the polygon.

These two methods create areas. The areas are labelled with a number and each label can be associated with a colors and a number. Then the segmentation is done. Now, the concept used to do the segmentation are define and just by using those methods all parts can be found.

5.4 Body parts and results

Having specified methods and algorithms, this part clears up how to use it for the different part of the body. According to the way of inferring the body parts, the limbs can be classified into 5 groups:

1. Lower arms and legs. Points are all inferred with just two joints by using the perpendicular slopes of the line passing by the two joints. Since the background is put as zero, starting from a joint, by running through the perpendicular slopes in both direction, the first point to reach

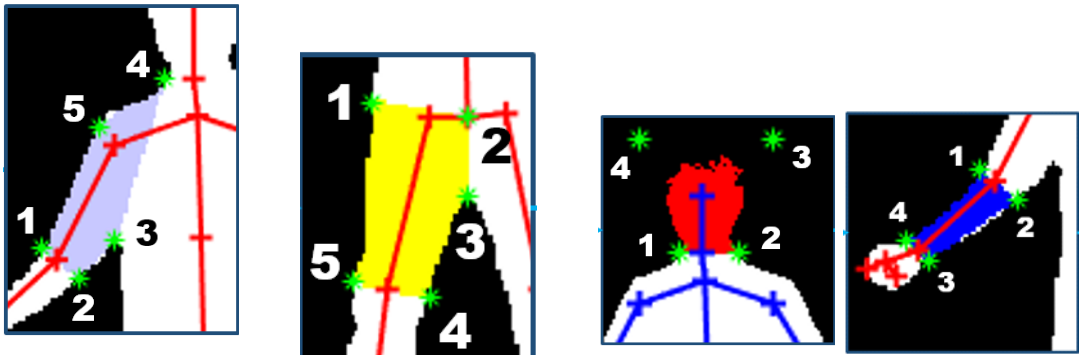
0 are chosen as corner of the polygon. It corresponds to when slopes meet an edge (namely background). See green points in Figure 13d.

2. Upper arms and upper legs. For the upper arms, the points 1 and 2 in Figure 13a are found exactly as in the group 1 with the joints of elbows and shoulders. The points 5 and4 also used the same methods with the joints of shoulders and *SpineShoulder* but the lower points are retrieved. Considering only one side of the body (left or right), point 3 is found as the first point that meet the background in the vertical axis with the *SpineShoulder* as reference. For the upper leg, in Figure 13b, point 3 is inferred similarly as the point 3 in Figure 13a but adapted to the leg and the point 2 is directly the joint called *SpineBase*.

3. Head. Two points 1 and 2 in Figure 13c are induced such as in group one. 3 and 4 are defined by using the abscisse of joints in shoulders and having a distance in the vertical axis from the joints $Neck$ of two times the distance between $Neck$ and $Head$

4. Hands and Feet. Two points are induced such as in group one defining limitation from other body parts. By retrieving the arms and legs, hands and feet are independent connected components. Knowing the joints of these body parts, the connected component are located and then labelled.

5. Torso. Having deduced all the others parts, the torso is found as the biggest connected component of the image when all other parts are retrieved from the body.



(a) Upper arm (b) Upper leg (c) Head (d) Lower arm

Figure 13: Particular body part segmentation

Although algorithms used here are not robust to all situations, to test the segmented fusion on a standing and moving body, this segmentation is enough. However this segmentation should be improved to have robust segmentation. As a matter of fact, testing topological change can be done by touching objects and in this case, the segmentation of the hands should be reviewed. Having noted this remarks, in the main dataset used for this project, the man is standing without touching anything apart from the floor. In these condition, Figure 14b shows the results of the segmentation and the labelling.

(a) Body labels (b) Body labels error

Figure 14: Results of body segmentation

As shown in Figure 14a, this segmentation provide good segmentation and labelling. However for a few images the feet are difficult to separate from the ground like in Figure 14b. The use of color image could easily improve these cases since there are edges around the body in the color image. Yet in the main dataset provided, the color images were not saved. On the other hand, having enough correct images and enough segmented body parts, the project could proceed to the next stage. To minimize this default of segmentation, feet and hands cannot be larger than a circle (or a sphere) that center on the joint of the feet or the hands and have a radius bigger than feet. That is why in Figure 14b, there are circles around the feet.

The segmentation thus consists in defining polygons and filling it uniformly thanks to slopes and joints. Knowing how to operate rigid fusion and once the segmentation done, these two processes can be used to do segmented fusion, that is to say fusion on each body parts separately.

6 Segmented 3D reconstruction

Segmented fusion means reconstructing 3D models body part by body part. This process handles topological change and fast motion. Previously, the segmentation in 2D on the human body could be done. Now, it will be used to compute the 3D volume of each part and to transform it locally so that the movement of the body can reconstructed correctly. First, the bounding boxes of each part should be computed in order to get the local position and the volume of each part. They are thus reconstruction in a 3D model. Adjacent body parts are stitched together.

Having the segmentation done, Figure 3 presents the pipeline of the segmented fusion:

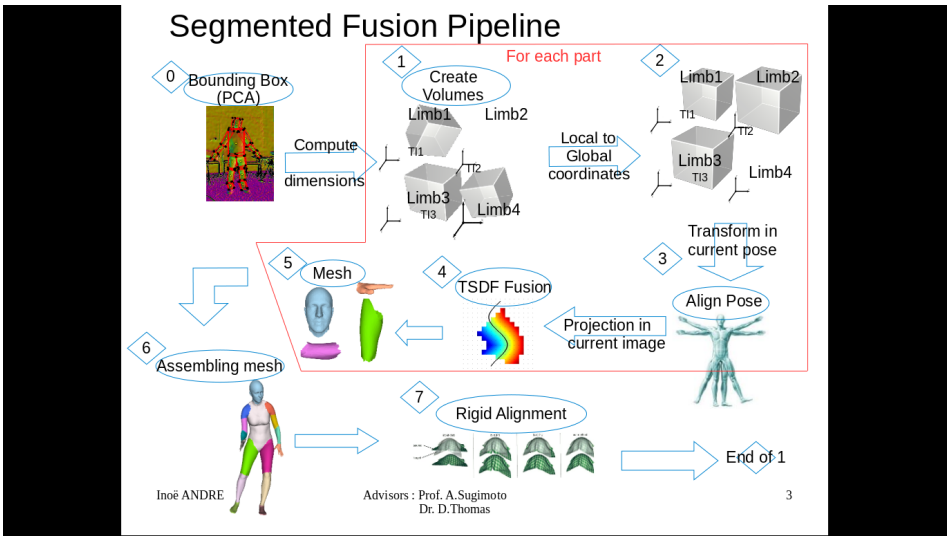


Figure 15: Pipeline of the project. The pipeline starts by considering that segmentation is done. It has 8 steps.

1. The first step consists in computing the 3D bounding boxes of each body parts. It provides the local coordinate systems of a part but also its coordinates in the global image. The local system of a part is the coordinate system that contains only the body part and where the orthonormal coordinate system axis have the minimum weight for each points.

2. Having the bounding boxes, the local volume of each body part is deduced using corners of bounding boxes.

3. To compare the volume with the correct surface, it is transformed in the global system.

4. Since the current image have a different pose from the global image, the volume is transformed with the current pose estimation to the current image.

5. Compare each voxel of the body part volume to the surface of the current image in order to do TSDF Fusion on the body part.

6. Compute the meshes of the body part through marching cubes algorithm.

7. Stitch all the body parts to get the mesh of the whole body.

8. Do the rigid alignment of this mesh with the new current image.

9. There is no need to create the volume to continue but the volume should be transformed. Therefore, the iterations restart at the end of the first step.

As described by the pipeline above, the first step to do segmented fusion is to get the oriented boxes of each rigid part called bounding boxes.

6.1 Bounding boxes

As the Figure 15 shows, the first step of the segmented fusion is to compute the bounding boxes. That is to say, get a 3D representation using the 2D segmentation. The segmentation of the body done, the known components are the label of each body part, the skeleton and the position of the joints. From vertex maps, clouds of points are made using the labels as a mask. The clouds of points are transformed into lists of vertices. The center $ctr3D$ of a cloud of points is computed as a mean of all the points in the cloud.

In order to deduce the principal orthogonal axis, a principal component analysis (PCA) in 3D is applied on the clouds of points of each body part. The eigen vectors given by the PCA,e_1,e_2,e_3 are the principal axis. These axis e_1,e_2,e_3 and the center ctr of a cloud of points are used to create the passage matrix from the local to global coordinate systems $T_{loc\to glo}$. Here is how the matrix is constructed.

$$T_{loc\to glo} = \begin{bmatrix} e_{11} & e_{21} & e_{31} & ctr_1 \\ e_{12} & e_{22} & e_{32} & ctr_2 \\ e_{13} & e_{23} & e_{33} & ctr_3 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{14}$$

This matrix is a 4×4 matrix composed of a rotation matrix R_e^{-1} given by the three principal axis and a translation vector $\mathbf{t}_e$ given by the center of the cloud of points. The first column contains the three components of the first principal axis, respectively for the second and third column.

Defining the bounding boxes requires to find the corners. It can be found by searching the maximum and the minimum on each axis and defining corners through these extremum. Nevertheless, since the clouds of points are in the global coordinate system, finding the extremum would lead to create bounding boxes that do not fit perfectly the body parts and will not be oriented. Therefore, the clouds of points should first be transform in the local coordinate system in which the principal axis are oriented horizontally, vertically and in the depth axis.

$$T_{glo\to loc} = \begin{bmatrix} R_e^{-1} & -R_e^{-1}\mathbf{t}_e \\ \mathbf{0}^\intercal & 1 \end{bmatrix} \tag{15}$$

Having found the corners locally, these points should be transform back in the global coordinate system using $T_{loc\to glo}$.

Applying these transformations on all body parts, and drawing the bounding boxes would give the Figure 16.

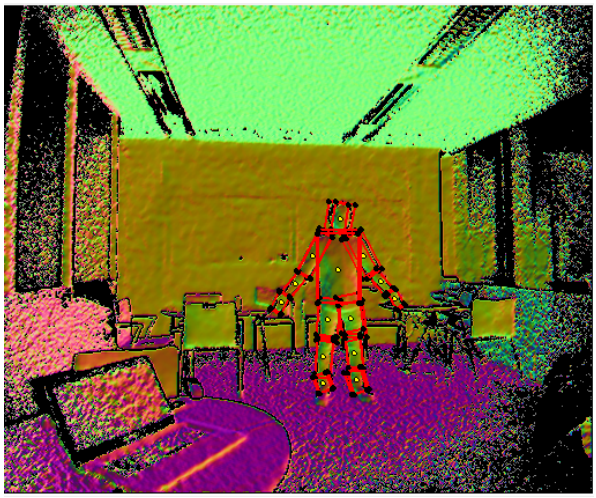


Figure 16: Bounding boxes

Figure 16 shows the scene in the converted depth map explained in section 3.2.4. On top of this converted depth map, the center of clouds of points of each body parts are drawn in yellow. Moreover, the bounding boxes edges are drawn in red while the corners are in black.

This Figure 16 prove that each body parts can have its own bounding boxes. Therefore, it is possible to create a volume corresponding to each limbs. From the maximum length of each boxes, its dimensions are used to create volume representing each body part. When the volume are created their coordinate system is $e_1 = (1.0, 0.0, 0.0)$,$e_2 = (0.0, 1.0, 0.0)$,$e_1 = (0.0, 0.0, 1.0)$ and the center is the origin. That is why, $T_{loc \to glo}$ is needed to transform the volume in body part position and direction as it is demonstrated in Figure 16 (the arms for instances).

From the bounding boxes it is possible to extract each body part volume thus starting part fusion, namely fusing each body part separately and not together as in section 4.

6.2 Part rigid fusion

In section 4, fusion was largely explained for a complete scene. Using it on small volumes improve the performances of the fusion. This section suggests how to achieve fusion for body parts using tools defined in section 3 and bounding boxes 6.1. Rigid fusion is still used here because it is applied on small rigid part. Non rigid parts are treated when adjacent parts are stitched together.

However rigid parts have differences in size. There are several parts and there are small. In that respect, each rigid body part should first be located correctly in the 3D place then the fusion can be operated. The location for the fusion is found using transformation for each body part.

6.2.1 Volume and transformation

It is important to transform volume so that each body part is well placed compared to other body parts. Otherwise, since they are created in a local space, they overlays with all other body parts. Before changing the coordinate system of a volume of a body part, this volume should be created. This subsection covers the steps 1 and 2 in Figure 15. Figure 17 illustrates how to create the volume.

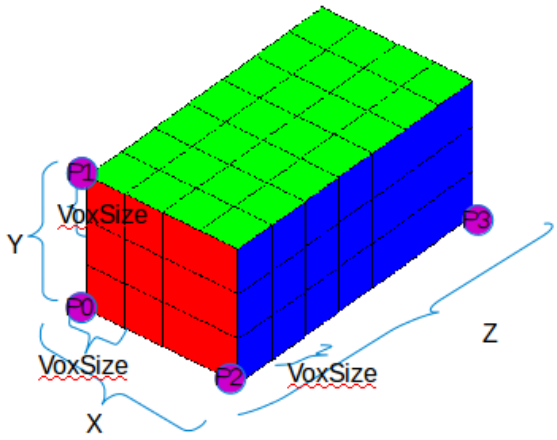


Figure 17: Representation of a body part volume

Figure 17 represents a volume sliced by voxels in shape of cubes. X,Y,Z are the length of each dimension of the volume. P0,P1,P2,P3 are corners of the volume computed in section 6.1 and VoxSize represent the resolution of the volume and gives the size of voxels. It is important to have the same resolution for all body parts because voxels of two different body parts are meant to be compared. The parameter VoxSize is chosen at 0.005 so that there are enough points to create a correct mesh but not too much to avoids unnecessary computations. Knowing that, here is how to compute the length of the volume:

$$\begin{cases} X = \lceil (||\mathbf{P2} - \mathbf{P0}||_2)/VoxSize \rceil, \\ Y = \lceil ((||\mathbf{P1} - \mathbf{P0}||_2)/VoxSize \rceil, \\ Z = \lceil ((||\mathbf{P3} - \mathbf{P2}||_2)/VoxSize \rceil \end{cases} \tag{16}$$

Ceiling in each length assure that body parts have no incomplete voxels or vertices going outside of the volume. In practice, to get correct mesh the volume is created using $X' = max(X, Z)$ and $Z' = max(X, Z)$ otherwise, memory issues are bred and the code cannot be fully executed.

The volume is created in a local coordinate system. It must be transformed so that the volume is located correctly in the frame coordinate systems. This transformation $T^{bp}_{loc \to glo}$ is given by the PCA as explained in 6.1. The index *bp* means *body part* since each body parts has its own transformation. Applying this transform recenters and orients the volume in the global coordinate system. This aligns body parts and the corresponding cloud of points. However, to avoid that a volume overtake on other body parts, the cloud of points is computed to be only the segmented parts and not the whole body.

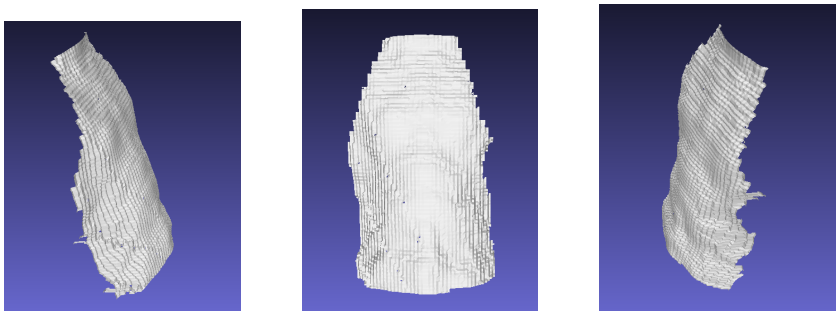
In the case of just one image, creating the volume and transforming it in the frame coordinate system is enough to align it with the corresponding vertices of the current image. Thus generating 3D model of the body part can be processed.

6.2.2 Part reconstruction

This subsection covers the steps 4 and 5 in Figure 15. It explains how to reconstruct body parts independently. It is important to be able to reconstruct 3D body part model because it enables to have topological separation.

Each body part can be reconstructed if the volume and the body part in the current image are

aligned in the depth axis. Extracting only the corresponding limbs in the current image enables to avoid body part overlays in meshes after the reconstruction is computed. Since body parts are also rigid, applying TSDF on the transformed volume and applying marching cube on the result of the TSDF breeds the 3D model of the body part. Figure 18 shows the result of the reconstruction of the trunk (torso plus belly).



(a) Left view (b) Front view (c) Right view

Figure 18: Representation the 3D reconstruction of trunk of the body in three different views.

The mesh of the trunk have clean cut at the top and at the bottom because the segmentation is horizontal for these parts. The separations with the arm create stairs cut in the mesh. As there is only one image, the model lack in smoothness and only the surface viewed from the camera point of view can be reconstructed. However the shape of the trunk is recognizable. Besides since the camera is quite far from the body whereas the resolution of voxel is high, the quality of the reconstruction is limited. There are too many 3D points in the volume compare to the current image maps which creates a few holes that can be seen by zooming the mesh. Other parts have same properties.

As a consequence, the part body reconstruction lacks some quality because camera resolution is big. However to prove that stitching and tracking work, great quality is not mandatory. What is more, fusing several frame improves the quality. Therefore, if stitching and tracking are possible, quality can be improved to a certain extend. All parts being reconstructed and contained in a local coordinate system, stitching limbs together in the global system create a model of the whole body.

6.3 Stitching part

Stitching part consists in manipulating body parts so that a model of the whole body is generated. Stitching aims at recovering junction between limbs even when there is deformation due to motion. A simple recovery of junction can be done with the transformation described in 6.2.1. Here is just an explanation of the way to create the model in the simplest case : only the camera is supposed to move in the scene but mostly the stitching is adapted for one frame. The part stands for the step 6 in the Figure 15

Each body parts are stocked in three lists : triangle faces, vertices and normales. In order to get the model of the whole body, three big lists of faces, vertices and normales are created. These lists contain the information of all body parts. For the simple case there is no need to add constrains between two adjacent body parts. That is why concatenating all body part lists is sufficient to create a human body model. Nevertheless, lists are transformed before concatenation because body parts are

created in a local space. For one body part, face values are numbers of vertices. Consequently, when two face lists are concatenated, the second list should be upped by the maximum value of the first list. The concatenated face list have the sum of the number of the two lists. Concerning vertices and normales, there are created in a local space. Therefore they should be transformed with the local to global transformation (section 6.2.1).

Figure 19 is the result of such process.

(a) Left view (b) Front view (c) Right view

Figure 19: Representation a naive stitching for a simple case of chopped body reconstruction.

All body parts are placed in the right order, they have correct adjacent body parts. There are holes between them marking out the limbs. The quality and shapes of meshes are the same as before stitching.

The model is a chopped model of the body as there are edges in junctions between body parts. These holes are voluntary in this representation. They show that the segmentation was used to reconstruct a chopped model. The concatenation works correctly since all body parts keep its shape, no links are created between two parts and all parts are placed correctly according to their adjacent parts.

Reconstruction of a chopped body is the first step to have a 3D model of a moving object. Volume are created using bounding boxes and 3D body parts models are generated through TSDF and marching cube algorithms. Stitching each part through concatenation fulfills this first step. Concatenation is done by adding the faces and transforming the vertices and the normales in the global space. Nonetheless, only if there are several frame, movement can be reconstructed. But from one frame to another the position of the camera and the pose of the human can change. That is why, some computation, tracking or alignment should be done to predict these positions and changes. In the case of the chopped body model some more transformations are needed to get alignment.

7 Tracking

Tracking means following pose of the camera. Tracking is needed to do coherent fusion 4. To achieve it, the transformation from the first pose to the last pose of the camera is needed. In this section, the seventh step of the pipeline is explained. However, first the general case without any segmentation is explained and then tracking is studied in the case of segmented fusion. This step requires three inputs: the mesh, the 6DOF pose camera and the new current image. Here the 6DOF represents the previous camera pose. Figure 20 illustrate tracking as an iterative process.

Figure 20 shows how surfaces of the mesh and of the current image evolve during the tracking process. A column represent an iteration, the upper part shows two surfaces, a green one and a white one, that should perfectly overlay after enough iterations. The lower figures are the result of the overlay at the corresponding iteration.

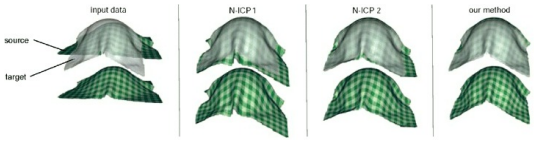


Figure 20: Tracking

Tracking, or here rigid alignment, consists in finding the transformation which transform the first image to the current incoming image. It takes the mesh and the new RGBD image as input to find the incremental transformation between the two inputs and update the 6DOF camera pose transformation with this increment. Using an increment aims at achieving real-time performances.

The tracking used here is iterative. The principle is to compare the vertices and normales of the mesh with the current image. At each iteration, the mesh should come closer to the current image. Each iteration spawns incremental pose transformation and a residual matrix. While the increment pose transformation should tend to the desire transformation, the residual matrix should tend to identity at each iteration. The idea is to minimize a point-tangent plan distance. The tangent plan comes from the mesh whereas the point is from the current image.

To achieve it, the first part is to the find and associate close enough vertices and normales between the inputs by projection of the mesh on the new image. Then compute the distance between the vertices to the tangent plan in order to minimize it.

7.1 Associate vertices and normales

To compare the image with the mesh, both should be in the same 6DOF camera position or at least as close as possible. At this moment of the algorithm the 6DOF camera pose is the one from one former image. However it can be used to get closeness. Since the mesh is created in the first camera pose, applying the 6DOF matrix pose to the mesh, transform it as close as possible in the current image frame. Besides, to have correspondence between the vertices, an index that links two corresponding vertices should be computed. This index is computed by transforming and projecting the vertices of the mesh in the 2D space. Let say $\mathbf{u}$ is a vertex, or a 3D position, then $\dot{\mathbf{u}}$ is the projection of the vertex in the 2D space. $\tilde{\mathbf{u}} = \pi(KT_{g,i}\mathbf{V}_{mesh}(\mathbf{u}))$ is the index in the 2D space corresponding to a vertex of the mesh. Then this index is used to compare vertices and normales as follow:

$$V_{diff} = \mathbf{V}_i(\tilde{\mathbf{u}}) - \mathbf{V}_{mesh}(\mathbf{u}) \tag{17}$$

$$N_{diff} = \mathbf{N}_i(\tilde{\mathbf{u}}) - \mathbf{N}_{mesh}(\mathbf{u}) \tag{18}$$

If $\begin{cases} V_{diff} < \epsilon_v \\ N_{diff} < \epsilon_n \end{cases}$ then the vertices are close enough to be used in the distance minimization section. Otherwise, the vertices considered are just ignored but it might be used in the next iteration. For global fusion of the dataset used in the project $\epsilon_v = 0.01$ and $\epsilon_n = 0.5$ would give good tracking and reconstruction result. These ϵ are found empirically. By each iteration, since the mesh come closer to the new image the number correspondence should grow.

Here is an illustration of what has been explained (presentation of [9]).

In Figure 21, there is a camera position seeing the surface in red and another camera position seeing the surface in green. Associating vertices and normales in Figure 21 is the fact that the red point and the green point are compared. In subsection 7.2, it explains how the red and the green surface are then coming closer to each other.

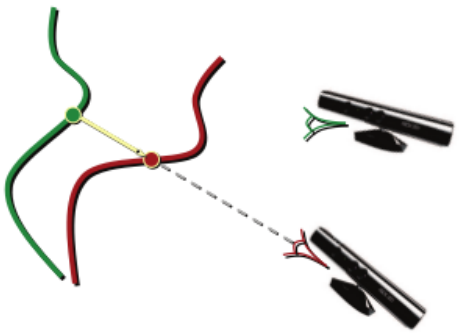


Figure 21: Projective data association

If there are no correspondence candidate the distance minimization would not work. That is why, two following images should have a small change of the view. That is to say the algorithm cannot work if the camera pose change too much suddenly. However due to the 30 frame per second taking by the camera, a hand-held movement is considered to have only small pose transformations between two following images. The second step to track the camera pose is distance minimization. It consists in using an gradient descent algorithm to minimize distance to the tangent plan.

7.2 Distance minimization

Using the correspondence candidate, distance minimization is the algorithm that estimate the incremental pose that aligns surfaces of the mesh and of the current image. It computes the distance of a point to the tangent plan. The difference of the two associated vertices can be seen as a residual and is computed as follow:

$$\mathbf{b} = N_i(\tilde{u})V_{diff} \tag{19}$$

Let $T_{inc,i}$ be the incremental matrix between two images. It can be expressed as follow (defined in [9]):

$$T_{inc,i} = \begin{bmatrix} 1 & \alpha & -\gamma & t_x \\ -\alpha & 1 & \beta & t_y \\ \gamma & -\beta & 1 & t_z \end{bmatrix} \tag{20}$$

With α, β and γ the rotation coefficients and t_x, t_y and t_z translation coefficients. However to update the coefficient values, it will written as follow (defined in [9]):

$$\mathbf{x} = (\beta, \gamma, \alpha, t_x, t_y, t_z)^T \tag{21}$$

This form is preferred because it enables to summoned a linear algebra library that solve the equations. Indeed at each iteration, the least square problem is solved through the descending gradient method. The partial derivative vector, the residual and this $\mathbf{x}$ are all used in the descending gradient method in order the find the coefficient of $\mathbf{x}$. The equation to solved can be written as follow (see [9]) :

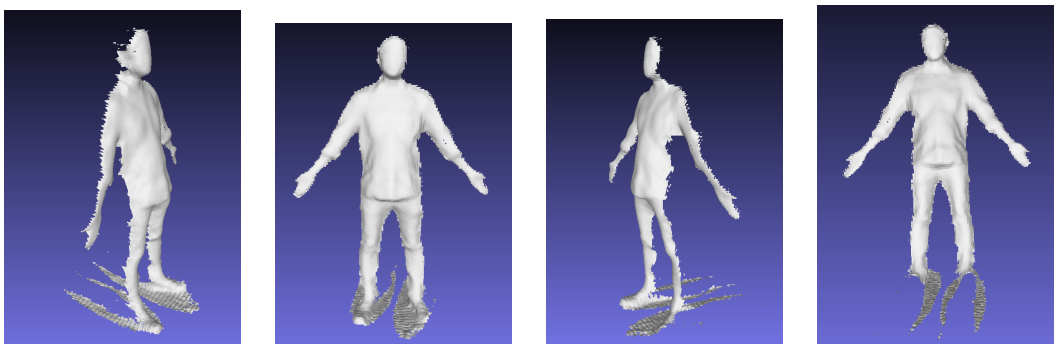
$$\sum_{correspondence} (\mathbf{J}^T\mathbf{J})\mathbf{x} = \sum \mathbf{J}^T\mathbf{b} \tag{22}$$

Where $\mathbf{J}$ is a Jacobian matrix

Through tracking the passage matrix from the first image to the current image is updated to a passage matrix from the first image to the image following the current image. Two following images have a smooth change of point of view which enable to get some results using the functions introduced in this subsection 7.2.

7.3 Rigid alignment results

Testing whether the tracking works or not on few images is enough. In the case it is not working, the 3D models will be deformed or will have duplicate. In case it works correctly, the updated model do not change much from the model with just one image because the movement of the camera are smooth. There are not enough new details to add much more information on the model. In Figure 22, the result of fusion and tracking for 20 following images are shown.



(a) Left view (b) Front view (c) Right view (d) Behind view

Figure 22: Mesh of the body after tracking 20 camera pose

Figure 22 the 3D model of the standing human in four different view to explore the 3D aspect of the model. The reconstruction of feet contains also part of the ground due to a segmentation issue explained in 5.4 Figure 14b. In fact, in the set of 20 images the first images have segmentation default around the feet which thus enables the floor around the feet to be reconstructed in Figure 22.

Apart from this default of segmentation, since the body is reconstructed without duplicate or more noise than in a single image reconstruction and is very similar to the body in Figure 8, the tracking has been correctly computed. This similarity is due to the fact that in the 20 images, the user do not move much the camera but rather do small camera rotation. Because of these small change, there is not so much new information.

Global case of tracking has been studied. It consists of an iterative process that minimize distance

between corresponding surfaces. It should now be applied to body part.

7.4 Part alignment

Aligning a part volume created in the first frame with its corresponding part in the current image enable to accumulate and update the 3D model semi-independently of others body parts. Therefore, moving parts and not moving parts can be processed separately. Visualizing the update of each frame gives the dynamic 3D model. In the case of the chopped body model, each body part model should require three transformations to obtain tracking. Due a to lack of time, the result with only one transformation are shown here. Nevertheless, the necessity of the other two are explained in this section.

Here are the explanation of the three transformations needed to align body part volumes with their corresponding points in the current image.

1. The first necessary alignment is made by knowing the pose of the camera. It can be found by doing a tracking on the stitched body and the body extracted from the current image. It gives the 6DOF pose representation $T_{pose,i}$. That is to say, it transforms the position of the initial camera pose to the current image camera pose. That is the rigid alignment.

2. When the body is also moving, the body parts do not align correctly just with the first transformation. In this case, joints of the skeleton of the body have different position. Therefore the parts correctly follow the motion by tracking the skeleton. This transformation deals with fast human motions. It is a transformation that follows the skeleton movement. These skeletal movement can be big and should be the major contributor for dynamic alignment.

3. The last alignment is done through tracking part by part. It is a refinement that deals small movement, or twisting movement. However it is has not been implemented here.

These three alignments are updated at each new frame.

Figure 23 shows the 3D model of the stitched body by using just the alignment of the camera pose on ten following frames. These 10 frames have a very small camera position change and are supposed to have no body movement.

(a) Left view (b) Front view (c) Right view

Figure 23: 3D model of the stitched body for 10 images fusion

The body is reconstructed and there is no wrong positioning of each body parts. However body parts could be said to mistily expand until covering the holes between body parts. This expansion is more highlighted for the arms.

In this respect, even though the body parts are well placed in the camera pose point of view, the body parts are not perfectly aligned. In fact, it is really hard for a human to stay perfectly still. This misty expansion is supposedly be explained with the fact the third alignment with tracking part by part is missing. Each little movement of the arms or the change in the skeleton blurs the edges since third alignments is missing. The second transformation do not change the result because its purpose is to get fast motion transformation which does not exist here.

The 10 first images used in the fusion, do not translate but just does very little rotation. Therefore, the body moves just a little in one direction to another. That is why the quality of the reconstruction is still good inside the body parts. With this dataset, the contribution of a skeleton alignment for each body parts would not be so visible because the movement are not noticeable. In this case, just the local tracking contribution should be enough to perfect the model.

Tracking camera pose (or images) is done through vertices correspondence and minimizing points to plan distance. Having defined tracking of camera pose is not enough to get part body alignment. There should be two more transformations. One should be defined to track skeleton movement and the other one should do the correspondence locally for each body parts separately. It can be deduced using tracking for the camera pose but applied on each body parts. The result of tracking ten images shows that global rigid alignment is not enough for alignment of the stitched model.

8 Conclusion

This report presented a work in progress that aims at reconstructing a dynamic human scene (moving camera and moving human) with topological change using only one RGBD camera. Noticing that most of human body parts cannot have topological change, the idea is to separate the workload in different little region of the human body that are topologically stable and process it as in Kinect fusion [9] for static scene. First the depth map is converted into vertex and normal maps. The segmentation of the body can be done through slopes consideration and filling polygons. Through the segmentation bounding boxes can be computed. From bounding boxes, the volume of each body part is generated and used to have the TSDF volumetric representation. Each mesh is created by aligning the volumetric representation with corresponding part of the body in the current frame. The model of the body is computed with the marching cube algorithm. In the last part, tracking each segmented part could not be finished due to a lack of time. It misses the computation of two transformations.

The work shows good promises in tackling the non rigid reconstruction but since it has not been terminated it will be continued. Besides even though some algorithms are not optimized, the computation for one image of the stitched body is faster than the computation without segmentation, of the global body directly. Therefore, the programs can potentially be run in real-time. The stitched model created in this project combines Kinect Fusion [9] and Stitching Puppet [12]. It uses the idea of in puppet of [12] but not only to recover poses but also to reconstruct a 3D model with Kinect Fusion [9] techniques. Therefore, the first contribution of the project is to reconstruct body parts given depth image and skeleton. The second contribution is to be able to create 3D model of a human body by stitching the limbs together. To continue the project two transformations of body parts should be added. The non-rigid alignment used in Dynamic Fusion [8] can also be added to improve alignment. Then in order to have realistic motions, some constraint between body parts should be conceived and added to the project.

References

[1] Inoe Andre, Thomas Diego, and Akihiro Sugimoto. "https://github.com/InoeAndre/NIIComputerVision.git". In: 2017.

[2] PJ Besl andN and D McKay. "A method for registration of 3-d shapes". In: *IEEE Trans. Patt. Anal. Mach. Intell* (1992), pp. 14–2.

[3] Ivan Dryanovski et al. "Large-scale, real-time 3D scene reconstruction on a mobile device". In: *Autonomous Robots* (2017), pp. 1–23.

[4] Matthias Innmann et al. "VolumeDeform: Real-time volumetric non-rigid reconstruction". In: *European Conference on Computer Vision*. Springer. 2016, pp. 362–379.

[5] Shahram Izadi et al. "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM. 2011, pp. 559–568.

[6] William E Lorensen and Harvey E Cline. "Marching cubes: A high resolution 3D surface construction algorithm". In: *ACM siggraph computer graphics*. Vol. 21. 4. ACM. 1987, pp. 163–169.

[7] Slavcheva Miroslava et al. "KillingFusion: Non-rigid 3D Reconstruction without Correspondences". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

[8] Richard A Newcombe, Dieter Fox, and Steven M Seitz. "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 343–352.

[9] Richard A Newcombe et al. "KinectFusion: Real-time dense surface mapping and tracking". In: *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE. 2011, pp. 127–136.

[10] Leonid Sigal et al. "Loose-limbed people: Estimating 3D human pose and motion using non-parametric belief propagation". In: *International journal of computer vision* 98.1 (2012), pp. 15–48.

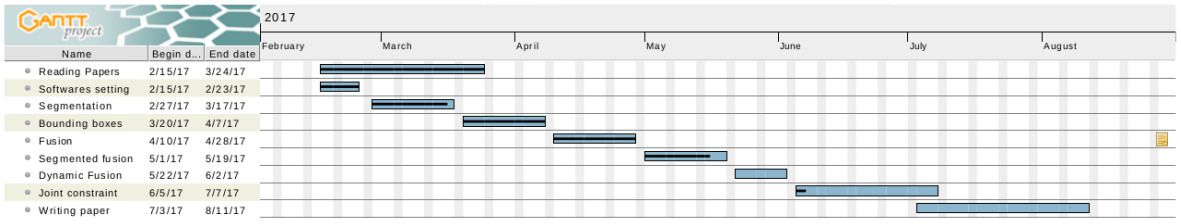
[11] Lembit Valgma. "3D reconstruction using Kinect v2 camera". PhD thesis. Tartu Ülikool, 2016.

[12] Silvia Zuffi and Michael J Black. "The stitched puppet: A graphical model of 3D human shape and pose". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3537–3546.

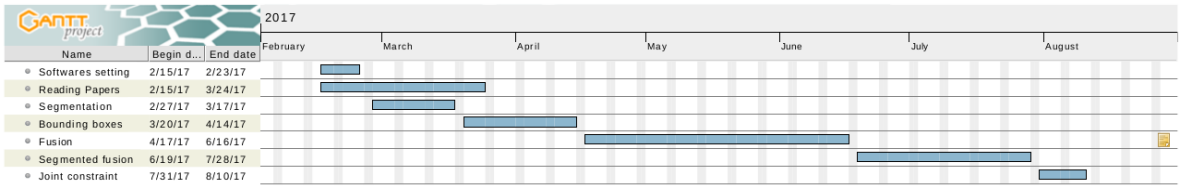
9 Annexe

9.1 Gantt

Here is the explanation of how the time was used during the internship.



(a) Estimate research project



(b) Effectively fulfilled tasks

Figure 24: Figure 24a is an estimation of how the time would be ideally used. Figure 24b is a chart of how tasks has actually been done. Each row stands for a task. This chart divide the time in months from February to August. The blue boxes is how much time is used for the task. In Figure 24a, black lines in the blue boxes are the progression of tasks. Fusion refers to implement some part of [9] and dynamic fusion refers to implement some part of [8]. Others tasks are based on the thoughts of the research team.

In Figure 24b, some task have disappeared compare to Figure 24a because they were simply no time to start the tasks. Until the bounding boxes each task was done on time. However the bounding boxes was prolonged by 2 weeks, fusion (or 3D modelling) was tripled in time and segmented fusion doubled. The main reason was theory misunderstanding which made debugging more difficult. Some time was lost because of material capabilities. Some algorithms that should take a few seconds on GPU, took few minutes on CPU. The GPU algorithms was written but the GPU issued lack of memories. Besides these non optimized algorithms are used dozen of times in the full code. Therefore, what should take few minutes took hours to be computed.

9.2 KinectV2

Depth sensor performances parameters	
Pixel pitch	10u*10u
Pixel array	424*512 pixels
Chips size	8.2mm*14.2mm
FOV	70(H)*60(V) degrees
Number of joints	until 25
Depth uncertainty	$< 0.5\%$ of range
Distance range	0.8-4.2m
Operating wavelength	860nm
Frame rate	max 60fps (typical 30fps)
ADC	2GS/s
Chip power	2.1W
Responsivity @ 860nm	0.144A/W
Readout noise	320uV differential
ADC resolution	10
Color camera	1080p RGB

Table 1: KinectV2 (depth sensor) properties

9.3 GPU specifications

Platform - Name: NVIDIA CUDA
Platform - Vendor: NVIDIA Corporation
Platform - Version: OpenCL 1.2 CUDA 8.0.0
Platform - Profile: FULL_PROFILE

Device - Name: GeForce GT 640
Device - Type: GPU
Device - Max Clock Speed: 901 MHz
Device - Compute Units: 2
Device - Local Memory: 48 KB
Device - Constant Memory: 64 KB
Device - Global Memory: 2 GB
Device - Max Buffer/Image Size: 512 MB
Device - Max Work Group Size: 1024

9.4 Joints indexation

These are the order of joints returned by the kinect adaptor on matlab (on python substract 1 to have correct index). In SkeletonConnectionMap.mat there is an array containing skeleton connection map to link the joints.

```
SpineBase = 1;
SpineMid = 2;
Neck = 3;
Head = 4;
ShoulderLeft = 5;
ElbowLeft = 6;
WristLeft = 7;
HandLeft = 8;
ShoulderRight = 9;
ElbowRight = 10;
WristRight = 11;
HandRight = 12;
HipLeft = 13;
KneeLeft = 14;
AnkleLeft = 15;
FootLeft = 16;
HipRight = 17;
KneeRight = 18;
AnkleRight = 19;
FootRight = 20;
SpineShoulder = 21;
HandTipLeft = 22;
ThumbLeft = 23;
HandTipRight = 24;
ThumbRight = 25;
```

```
SkeletonConnectionMap =
[ [4 3]; % Neck
  [3 21];% Head
  [21 2];% Right Leg
  [2 1];
  [21 9];
  [9 10]; % Hip
  [10 11];
  [11 12];% Left Leg
  [12 24];
  [12 25];
  [21 5];  % Spine
  [5 6];
  [6 7]; % Left Hand
  [7 8];
  [8 22];
  [8 23];
  [1 17];
  [17 18];
  [18 19];  %  Right Hand
  [19 20];
  [1 13];
  [13 14];
  [14 15];
  [15 16];
];
```

Abstract

Reconstruction of moving object in 3D is an expanding and more and more competing area of research since it can be used for game motion, virtual or augmenting reality and even medical surgery. This works aims at generating 3D models of human movement using only one RGBD camera. Here fusion for rigid object is achieved using TSDF volumetric representation, marching cubes algorithm (breeding meshes), and model to frame tracking with a 6DOF pose representation of the camera displacement. However, the way to treat human movement dwells in chopping the body on junctions so that only rigid body parts remains. Rigid parts are confined in bounding boxes using PCA and their displacement is estimated with ICP algorithm to also track body movements. Fusion is operated on each body parts. Finally, stitching these transformed body parts enable to estimate a model of moving human.

Abstract

Le domaine de la reconstruction d'objet 3D amovible s'expand et devient de plus en plus competitif dans la recherche car les applications - qui naissent de ce domaine telles que l'animation dans les jeux videos, la realité augmentée, la realité virtuelle ou encore les opérations chirurgicales - progressent grâce au faible coût que demande la technologie. Le travail presenté dans le rapport vise à générer un modèle d'être humains en mouvement en utilisant seulement une camera RGBD. Ici, la fusion pour les parties rigides du corps est réalisée en utilisant la représentation volumetric générée par la TSDF, l'algorithme des "marching cubes" donnant la représentation des surfaces en 3D, et un traquage de modèle à image fondé sur une représentation matricielle de la position de la caméra. Cette représentation est permet d'avoir 6 degrée de liberté. Cependant, pour traiter les mouvements humains, la méthode présentée dans le rapport réside dans le découpage du corps au niveau des articulations laissant juste les parties rigides du corps à traiter. Ces parties du corps sont confinées dans des bounding boxes grâce à l'utilisation d'une analyse en composantes principales (ACP). Estimer le déplacement des parties rigides grâce à une transformation de système de coordonnées par l'algorithme ICP (Iterative Closest Point) puis coudre ces parties transformées, permet d'estimer un modèle 3D simple des mouvements humains.