3D Scanning Using Consumer-Grade Depth Sensors: Methods and Applications

A Thesis Presented

by

Carlos Orrego

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

May 2012

Stony Brook University

The Graduate School

Carlos Orrego

We, the thesis committee for the above candidate for the

Master of Science degree, hereby recommend

acceptance of this thesis.

Dimitris Samaras
Associate Professor, Computer Science Department

Xianfeng David Gu
Associate Professor, Computer Science Department

Tamara Berg
Assistant Professor, Computer Science Department

This thesis is accepted by the Graduate School

Charles Taber
Interim Dean of the Graduate School

Abstract of the Thesis

3D Scanning Using Consumer-Grade Depth Sensors: Methods and Applications

by

Carlos Orrego

Master of Science

in

Computer Science

Stony Brook University

2012

A 3D scanner is a device that analyzes a real world object and generates a point cloud describing the surface of such object, possibly including color information as well. However, these devices are expensive, fragile, large, and usually require especially adapted facilities to house them.

The advent of inexpensive depth sensors such as Kinect provide new opportunities to bridge the existing gap between systems that offer good scanning quality and systems that are affordable.

The objective of this thesis is to use Kinect as a 3D scanner. We achieve this goal by exploring techniques to generate point clouds from depth maps, and triangulation methods to construct meshes from point clouds. However, depth maps are not noise-free. To deal with this noise, we explore different depth map reconstruction and smoothing techniques. We then measure their effectiveness in reducing the noise and enhancing the quality of the generated model.

The main contribution of this work is an acquisition and processing pipeline that allows for capture and generation of accurate 3D models whose quality is comparable to those generated by expensive scanner devices.

We show that the accuracy of our acquisition system is on par with higher resolution scanners. We also demonstrate applications for our method by capturing a data set of human faces and generating an Active Appearance Model from this data set.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor, Dimitris Samaras, for his guidance throughout the development of this thesis. Without his knowledge and support, this work would have been a much more difficult task to accomplish. I would also like to thank the members of my dissertation committee, Prof. Xianfeng David Gu and Prof. Tamara Berg for their support and invaluable feedback.

Thanks to all my family members, those who are here in the US and those who are back in Chile. It is good to have relatives around, and they certainly made things easier for me with their good will and care.

I would also like to thank my girlfriend Elizabeth. Her patience and support were fundamental to conclude this work. She also had an outstanding role in helping me find subjects willing to participate in my experiments. Thank you!

Finally, I want to thank all professors in the Computer Vision group, and special thanks to the graduate students in the Computer Vision Lab, always available to help. Good luck to all of them with their work. You guys rock!

Chapter 1

Introduction

A 3D scanner is a device that analyzes a real world object and generates a point cloud describing the surface of such object, possibly including color information as well. Applications for 3D scanning are present in many fields, from architecture to topography, art preservation, movies and video games, industrial inspection, object recognition, etc.

Unfortunately, most 3D scanners are large and expensive devices that require to be installed in special facilities. For example, the Cyberware 3030 scanner, currently part of the 3D Scanning Lab, uses a great amount of space and requires a special setup. Even the smaller custom structured-light scanner present in the same laboratory is rather large and requires a dedicated port for data transfer.

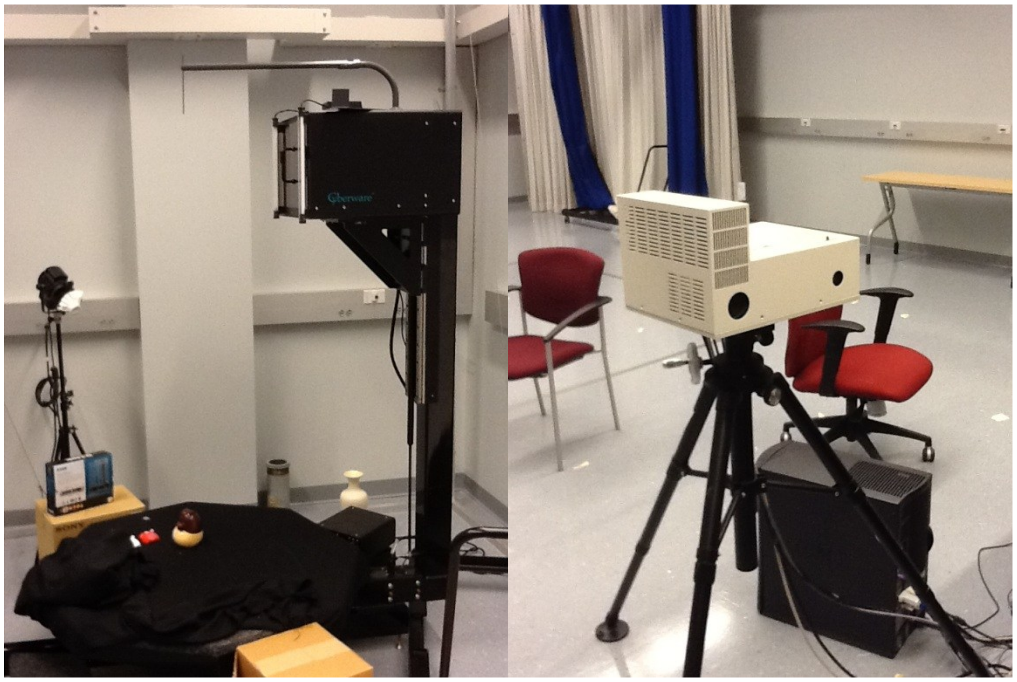


Figure 1: Scanner devices. Cyberware 3030 scanner (left); custom structured-light scanner (right)

While the aforementioned devices offer very good scanning quality, they have some drawbacks. For example, subjects must be taken to the lab for scanning. Also, because of their size these scanners have a limited range of movement, reducing the number of possible scanning targets. Also, relocation of the scanner is not an easy task.

Another issue is price. The Cyberware 3030 scanner is currently priced at \$63,200[1], with additional expenses for setup and maintenance. Clearly these prices are out of reach for individuals or even small institutions.

Ideally, for some applications we would want to have scanners that a) are less expensive than current alternatives; b) are portable; c) can be easily replaced in case of failure at a reasonable cost and d) can acquire 3D models whose quality is comparable to their more expensive counterparts.

The goal of this thesis is to build an acquisition system that addresses these issues using commercially available depth sensors.

The Kinect sensor is the first of a set of commercial-grade depth sensors that appeared in the market (others include for example the ASUS Xtion PRO and the eBox sensor). It is currently priced at \$149.99[2]. In this work we use this device as the cornerstone for an inexpensive 3D acquisition system.

However, while the use of Kinect solves the problems of pricing and portability, it comes with its own set of challenges, such as lower resolution textures and noisy depth maps. In this work we address such difficulties by testing and applying different methods to produce high quality 3D models.

The main contributions of this thesis are:

- Discussion and selection of techniques to filter and segment low resolution and noisy depth data (Chapter 5).
- Quality assessment of point clouds acquired with Kinect with respect to clouds acquired with higher resolution scanners, and techniques to reduce the acquisition error (Chapter 6).
- Discussion and selection of triangulation techniques for the generation of 3D meshes from the acquired data (Chapter 7).

[1] Information retrieved from http://www.cyberware.com/pricing/domesticPriceList.html
[2] http://www.microsoftstore.com/store/msstore/en_US/pd/Kinect-for-Xbox-360/productID.216507400

- An acquisition and processing pipeline for capture and generation of accurate 3D models whose quality is comparable to those generated by expensive scanner devices. The proposed system embodies the best methods and algorithms discussed and tested in this work (Chapter 8).

We also test the proposed capture pipeline in a real application. The application consists of the construction of an active appearance model. For this we acquire and process human faces using our system, we train the model and we test its accuracy in approximating new observations.

The rest of this thesis is structured as follows: Chapter 2 surveys previous and related work in the context of 3D scanning using inexpensive commercially available hardware; Chapter 3 introduces concepts related to 3D scanning techniques and the Kinect sensor; Chapter 4 describes different frameworks and libraries available for Kinect, and our data capture application; Chapter 5 discusses several restoration techniques for noisy depth maps; Chapter 6 compares point clouds from Kinect with higher resolution scanners and proposes smoothing algorithms to reduce the acquisition error; Chapter 7 examines triangulation methods to generate a mesh from a point cloud; Chapter 8 presents applications for our method; Finally, Chapter 9 concludes this thesis, summarizing the work presented in previous chapters and discussing some directions for future research.

Chapter 2
Related Work

Several techniques have been developed over the years for 3D reconstruction. Important to our study are techniques ranging from methods that employ a single image [1], [2]; methods that work with stereo images [3], [4]; methods that use laser range finders [5] to methods that make use of structured light patterns [6].

Methods that use structured light usually require a camera and a projector [6], [7]. Although these methods help to reduce the final price of the acquisition system, cameras and projectors used are generally expensive, when compared to an integrated system that appeared recently: Kinect.

Since its inception, Kinect became instantly popular due to its technical capabilities, accessible price and all its possible applications in the computer vision field. Soon enough, work utilizing this device begun to emerge, the majority of it related to pose detection [8], gesture recognition for new interfaces [9] and human activity recognition [10].

With respect to 3D reconstruction, there are many works in the web that use Kinect to generate point clouds and even mesh generation attempts, but surprisingly there are not many publications that address this topic. Tong *et al.* [11] propose a system to perform full body capture using an array of Kinects. However, this method requires careful arrangement of the devices, and a special mounting platform, suffering from the same issue of space requirement as specialized scanners. Another issue is the quality of the generated model: the distance at which the sensors are located with respect to the subject (1 m) implies that the level of identifiable features in the acquired model is very low, thus rendering this approach unsuitable for applications requiring a higher level of identifiable details.

The work of Smisek and Padja [12] explores the Kinect hardware, geometric model and calibration. A rig with a Kinect and two standard photo cameras is used to capture different views and depth information from a single scene. Depth information is used then to enhance the results of a stereo from motion approach. While this work utilizes depth map as a fundamental input for the proposed method, there is no mention of the noisy nature of the depth map and consequently no actions are taken regarding to that issue.

Cui *et al.* [13], [14] present a superresolution-based approach to perform shape acquisition using time-of-flight cameras and a Kinect device. Registration of different point clouds is performed using the ICP algorithm. Though their approach maintains the overall shape and appearance of the scanned object, the reconstruction error in the obtained model is in the order of centimeters when contrasted with a 3D laser scanner.

Some other works involve scene reconstruction using multiple views. In [15], the authors present a point cloud alignment approach. They first capture and generate point clouds for different views of the same scene. Then they compute SIFT descriptors for each scene and use RANSAC to select the best matching feature points. Their approach does not consider any kind of smoothing or triangulation after the matching step.

In the same line, one of the most recognized works by Newcombe *et al.* is KinectFusion [16], a real-time interactive scene acquisition system using a single device. It can reconstruct dynamic scenes by having the user move the camera around the room. A dense point cloud is created by this movement, and the generated model is refined with every new captured frame. This approach works well with static scenes where exposition time is not an issue, however, the method fails in the presence of large displacements and/or occlusions. While this method generates quality models by leveraging on the high density of points captured over time, scene stability cannot be guaranteed for long periods of time when capturing targets that cannot be still all the time. This is the case of human subjects, for example.

In the industrial field, Geomagic[3], a 3D software provider, presented at CES 2012 a system in early stages of development that "enables an instant 3D image of visitors at the event to be automatically turned into a 3D model and printed out on a 3D printer during the [CES] show." While the accompanying video shows a 3D model being obtained using a Kinect device and then printed using a 3D printer, the company did not mention any further details about the specifics of how the model is generated or any post-processing algorithms employed.

[3] http://www.geomagic.com

Chapter 3

3D Scanning and Kinect

Several techniques exist for the acquisition of the shape of a 3D object. Of such techniques, we focus primarily on two of them: time-of-flight scanners and structured light scanners. We do so because their principles and ideas are very important to understand the fundamentals of the device that we are interested in: Kinect. In this chapter, we first explain the basics of 3D data acquisition from the perspective of the aforementioned methods, to then shift our attention to the Kinect sensor and its technical characteristics.

Time-of-flight Scanners

A time-of-flight scanner is a scanner that uses a light probe to measure distance [17]. The main principle behind it is to measure the time it takes for a laser pulse to travel and return to the sensor. Since the speed of light is a known constant, range (distance) can then be measured using the following formula:

$$R = \frac{ct}{2}$$

Where c is the speed of light and t is the time the light pulse traveled.

Another approach is to use a continuous beam of laser radiation of a known wavelength λ. In this case, distance is measured by measuring the phase shift between the emitted and reflected pulse:

$$R = \frac{(M\lambda + \Delta\lambda)}{2}$$

Where M is the integer number of wavelengths, λ is the known wavelength of the pulse, and $\Delta\lambda = (\frac{\varphi}{2\pi})\lambda$ is the fractional part of the wavelength phased by the phase angle φ.

Since both these approaches only detect one point at a time, a rotating mirror is used to capture scenes. A typical commercial time-of-flight scanner can detect between 10,000 and 100,000 points per second.

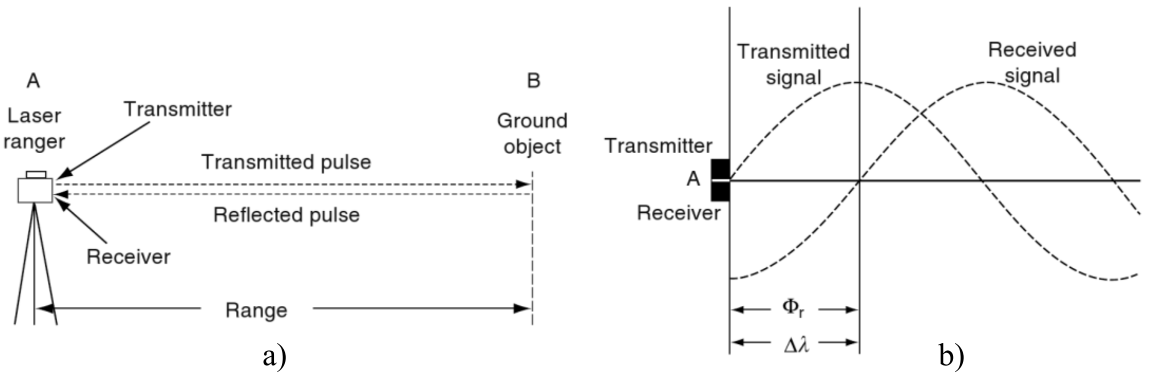


Figure 2: Two kinds of laser measuring. a) pulse laser; b) continuous laser. Image from [17].

Time-of-flight scanners require the subject to be still during the process, as any amount of movement while a frame is captured will result in distortion from motion. This is due to the fact that different points are captured during different time intervals. In consequence, this kind of scanner is usually attached to a rigid surface, limiting its variety of applications.

On the other hand, the range of these devices is really big (in the order of km); therefore they are the obvious choice when it comes to scanning large static targets.

Structured-Light Scanners

Stereoscopic systems work by finding the relationship between two cameras having a different view of the same image. An object point position can be calculated from these two different views if the position of said object is known in both cameras. The problem of making sure that a point in one view corresponds exactly to a point in the second view is known as the correspondence problem.

The correspondence problem can be lessened by replacing the second camera by a structured light source [18]. Structured-light scanners are a type of scanner that works by projecting a predefined light pattern into the scene and acquiring that pattern back using a camera. Distance is then computed by analyzing the distortion of the captured pattern against the projected light pattern [6].

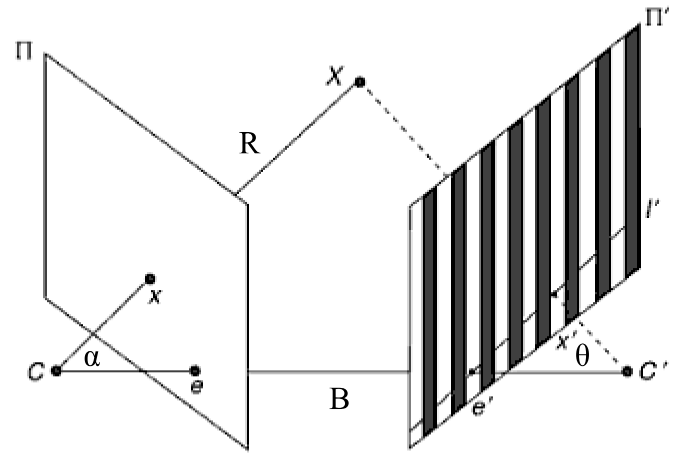


Figure 3: Structured-light scanning. Π is the image plane, Π' is the pattern projected onto the object. X is a point in the object. Image from [19].

Following Figure 3, the range (distance) R of a point X from a camera C can be expressed by the triangulation principle [20] as:

$$R = B\frac{\sin(\theta)}{\sin(\alpha+\theta)}$$

Given α, θ, B the problem then is to determine the correspondence between the projected pattern and the acquired image. Determination of such correspondences depends on the choice of pattern. While a more complete list of pattern kinds can be found in [20] and [21], the most relevant categories and some example patterns are:

- Sequential projections: Binary code, Gray code, phase shift
- Continuous varying pattern: Rainbow 3D Camera
- Stripe Indexing: Color coded stripes, segmented stripes
- Grid Indexing: Pseudo-random binary dots, color coded grid

Multi-shots methods such as sequential projections require multiple frames to compute depth values, thus requiring objects in the scene to remain static. Single-shot methods, on the other hand, can compute depth values using a single frame, at the expense of being less accurate than multi-shots methods; however these methods usually work better in the presence of moving targets.

The accuracy of structured-light scanners is determined by the size of the pattern and –at a lesser level for most applications- by the wavelength of light. Range is short (usually not greater than 10 meters) due to light dispersion and other phenomena that affect the amount of light captured by the camera.

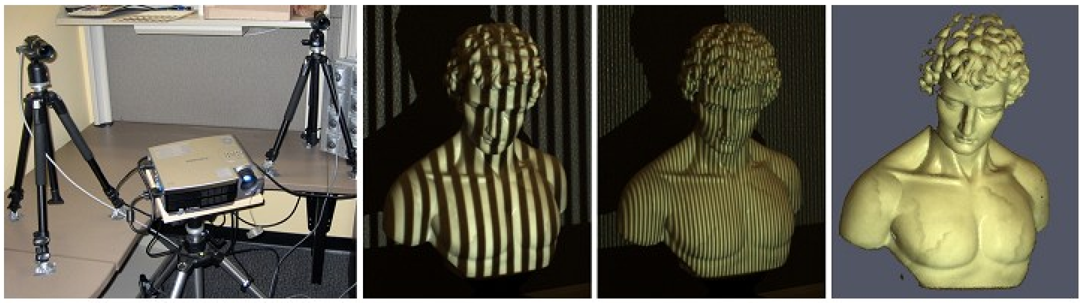


Figure 4: Structured-light scanning setup, process and result. Image from [22].

Microsoft Kinect Sensor

The Microsoft Kinect sensor is the first general consumer-grade structured-light camera to hit the market. It is a game-oriented device which incorporates an RGB camera, an infrared camera and a microphone array.

The RGB camera is a 24-bit camera that supports multiple resolutions (320x240 pixels at 30 Hz, 640x480 pixels at 30 Hz and 1280x1024 pixels at 15 Hz). The infrared camera is an 11-bit camera that supports resolutions of 640x480 pixels at 30 Hz and 1280x1024 pixels at 10 Hz. Depth images however are limited to 320x240 or 640x480 pixels, both at 30 Hz. Frame rate at higher resolutions is limited by the USB port bandwidth.

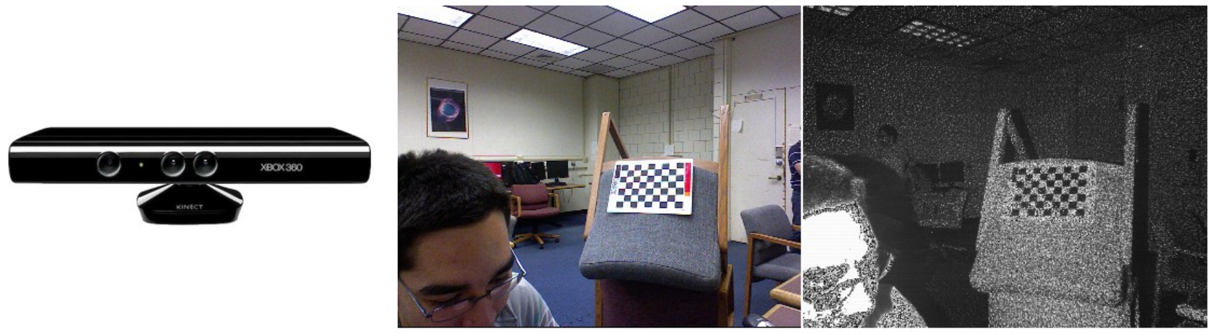


Figure 5: Kinect sensor. a) the device itself; b) RGB camera view; c) infrared camera view. The speckle pattern can be seen in the image.

Depth sensing is achieved by projecting an infrared speckle pattern using an infrared emitter [23]. This is a patented single-shot grid-indexing method similar to pseudo-random binary dots. The use of infrared light avoids the issues produced by structured-light pattern scanners that use light in the visible spectrum, allowing the device to capture RGB and depth scenes at the same time. The projected pattern is captured by the infrared camera and depth is calculated by the sensor's firmware. All processing is done by the device.

Table 1 resumes the technical specifications of the device:

Sensor	• Color and depth-sensing lenses • Voice microphone array • Tilt motor for sensor adjustment
Field of View	• Horizontal field of view: 57 degrees • Vertical field of view: 43 degrees • Physical tilt range: ± 27 degrees • Depth sensor range: 1.2m - 3.5m
Data Streams	• 320x240 11-bit IR @ 30 Hz • 640x480 11-bit IR @ 30 Hz • 1280x1024 11-bit IR @ 10 Hz • 320x240 16-bit depth @ 30 Hz • 640x480 16-bit depth @ 30 Hz • 320x240 24-bit color @ 30 Hz • 640x480 24-bit color @ 30 Hz • 1280x1024 24-bit color @ 15 Hz • 16-bit audio @ 16 kHz

Table 1: Kinect Technical Specifications.

Chapter 4

Building a Kinect 3D Acquisition System

Due to the complexity of the device at hand, it is a daunting task to try and build every part of the system ourselves. Instead, we leverage basic functionality such as device access and raw stream processing to third party libraries built specifically for that purpose, focusing our attention on building higher level functionality into our applications. In this chapter we present and discuss different framework alternatives in existence. We also introduce our data capture application, which is one of the cornerstones of this work.

Kinect Framework Selection

The Microsoft Kinect sensor was initially developed for use with the XBOX 360 Console. However, shortly after the device was released different groups started working to allow the device to be used on a computer. Time has passed since, and nowadays there are three different main frameworks in existence.

OpenKinect[4] is commonly signalled as the first attempt to bring Kinect support to the PC/Mac. It is a community effort with no commercial support, and as such, the amount of supported features is rather limited. Among the supported features are: access to depth and RGB streams, and control of LED and motors. Also, there are unofficial patches to tweak different device parameters, such as the auto-exposure time of the RGB camera. While OpenKinect lacks of any higher-level tracking facilities present in other frameworks, it allows for low-level access to the device, has a very small memory footprint and it is supported in multiple platforms.

OpenNI[5] is the open-sourced version of PrimeSense's original sensor code. It does not support Kinect natively, but there is an independently maintained driver for it. It supports low-level access to depth, RGB and IR streams, and high-level features such as body recognition and skeleton tracking. OpenNI is well supported by community members and also by an industry-led

[4] OpenKinect is available at http://www.openkinect.org
[5] OpenNI is available at http://www.openni.org

consortium of companies. OpenNI provides ports for all major computing platforms. It also supports a variety of sensors besides Kinect, such as the Asus Xtion PRO.

Microsoft Kinect SDK for Windows[6] is the official SDK released and supported by Microsoft. Its first version was released several months after the other frameworks first appeared. The goal of this framework is to support the creation of applications that make use of Kinect's body tracking and voice recognition capabilities. In this context, it offers access to the depth stream, RGB stream and skeleton tracking but no access to the IR stream. Also, the first beta version did not support registration between depth and RGB, situation that changed with the release of version 1.0. This framework is only available on Windows.

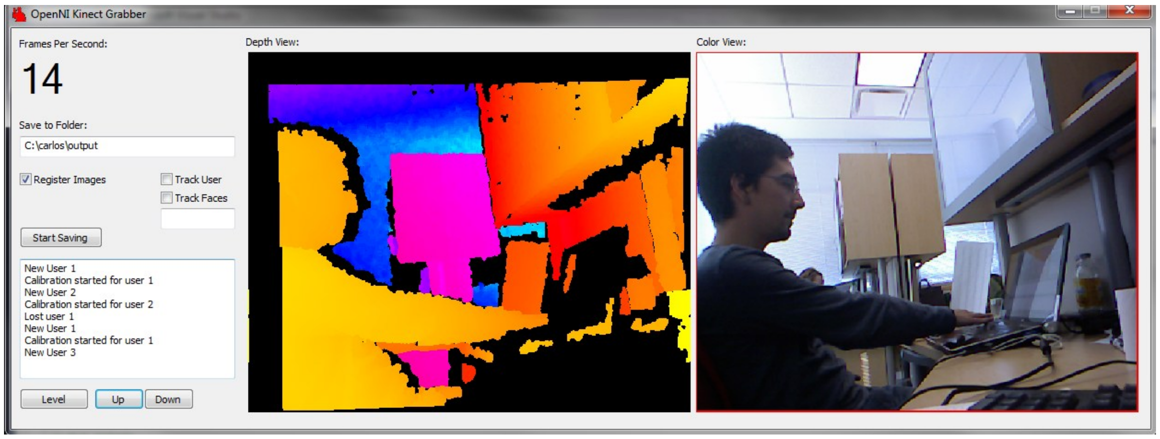
All frameworks were evaluated before and while building the capture and processing application. OpenKinect was deemed not suitable because it did not support any kind of tracking facilities. While Kinect SDK supported body segmentation and skeleton tracking, at the time of evaluation it did not provide registration between the depth camera and the RGB camera, thus rendering this framework not suitable for our purposes. OpenNI was chosen because it supported all of the required functionality needed to build our 3D acquisition application.

The Data Capture Application

General Description

The data capture application evolved from a simple frame grabber to a 2.5D video (RGB+Depth) recorder. The current version was built in C++ and kept as simple as possible to allow for video capture at 30 fps on medium range hardware. For face capture, it can use OpenNI's body tracking and automatically find and track the user's head position. For other applications, the user can interactively select a region of interest.

[6] Kinect SDK is available at http://www.microsoft.com/en-us/kinectforwindows


OpenNI Kinect Grabber

Frames Per Second: 14

Save to Folder: C:\carlos\output

[✓] Register Images
[ ] Track User
[ ] Track Faces

```
New User 1
Calibration started for user 1
New User 2
Calibration started for user 2
Lost user 1
New User 1
Calibration started for user 1
New User 3
```

Figure 6: The data capture application.

The application shows both RGB image and depth image in a distance-colored scale. It can also access the Kinect motor to tilt the camera up or down as desired.

Capturing Data

Using the application is simple. Upon initialization, both the depth and RGB images are shown. To perform data capture, the user can either use the head tracking feature or select a region of interest by hand. Camera tilt can be adjusted as well if necessary using the buttons located in the last row.

Once a region of interest has been selected by either method, the user can press the "Start Saving" button to start capturing frames. Pressing this button again stops the recording process, and the files are saved in the specified folder.

Data Format

The application saves data in a binary file, and also saves a JPEG file with the RGB information for every frame captured. The kind of data saved and the format of the binary file are shown in Table 2.

Field Name	Size (bytes)	Description
Width	4	Width (in pixels) of the region of interest in the current frame.
Height	4	Height (in pixels) of the region of interest in the current frame.
Top	4	Distance (in pixels) of the region of interest from the top of the image.
Left	4	Distance (in pixels) of the region of interest from the left of the image.
Horizontal Resolution	4	Horizontal resolution (in pixels) of the current frame.
Vertical Resolution	4	Vertical resolution (in pixels) of the current frame.
Projection Parameter (X)	4	Convenience projection parameter. Its value is $X_{res}\alpha_x^{-1}$.
Projection Parameter (Y)	4	Convenience projection parameter. Its value is $Y_{res}\alpha_y^{-1}$.
XYDepth	$4 \cdot 3 \cdot W \cdot H$	Point cloud (x, y, z) values from depth information in world coordinates.
RGB	$3 \cdot W \cdot H$	Image captured by the RGB camera.

Table 2: Description of fields saved by the data capture application.

Chapter 5
From Depth Maps to Point Clouds

Depth maps obtained from a Kinect device are noisy by nature, and contain holes in areas where the sensor cannot reliably determine depth values. Also, captured scenes often contain more than just the object or objects we are interested in.

In this chapter we examine techniques to enhance the quality of acquired depth maps. We also consider segmentation methods to focus on objects of interest, to finally construct a point cloud from the resulting depth map.

Hole-Filling for Depth Maps

Inpainting is a technique that attempts to recover lost or deteriorated parts of images and videos. In the real world, its applications are usually linked to art restoration. In the digital world, inpainting methods attempt to reconstruct damaged (noisy) parts of an image or to remove unwanted objects [24].

Borrowing from the Image Processing and Graphics community, we attempt to apply inpainting techniques to fill holes in the depth map. The use of inpainting as a hole-filling technique is not new, as can be seen in [25], [26]. In this work we tested two different inpainting approaches.

Diffusion-based Inpainting

This is the most common inpainting method. Several approaches such as [24], [27], [28] implement some variation of this technique, whose core ideas are: Starting from an inpainting mask (usually user-selected), the algorithm attempt to reconstruct the region Ω by *smoothly* propagating color information inwards from the region boundary $\partial\Omega$ following a certain direction $\vec{N}$. That is:

$$I(i,j) = \overrightarrow{\delta L}(i,j) \cdot \vec{N}(i,j)$$

Where $\overrightarrow{\delta L}(i,j)$ is a measure of the change in the information in the image, and $\vec{N}$ is the direction of such change. A common smooth measure is the discrete Laplacian operator. The direction $\vec{N}$ is chosen so it propagates linear structures (*isophotes* in the literature). Given a point (i,j), the discretized gradient vector gives the direction of the largest spatial change. Rotated by 90°, it gives the smallest spatial change, which coincides with the direction of the isophotes.

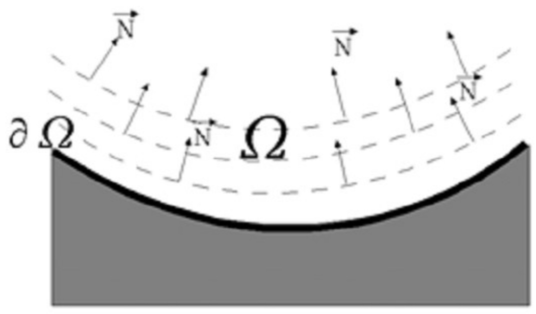
Given $B_e(\boldsymbol{p})$, the neighborhood around a point $\boldsymbol{p}$**,** a first-order approximation of $\boldsymbol{p}$ is given by:

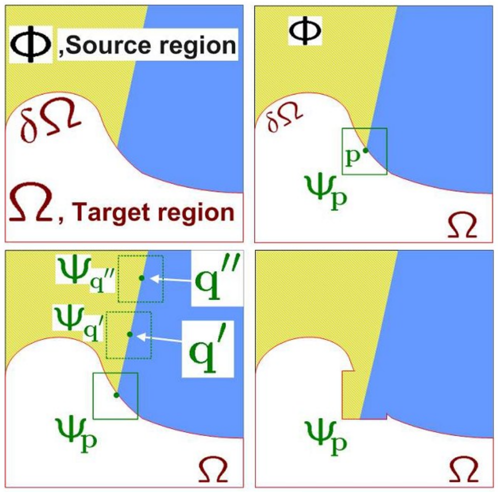
$$I_q(\boldsymbol{p}) = I(\boldsymbol{q}_i) + \nabla I(\boldsymbol{q}_i)(\boldsymbol{p} - \boldsymbol{q}_i)$$

For all $\boldsymbol{q}_i \in B_e(\boldsymbol{p})$. A point $\boldsymbol{p}$ is filled as a function of all points $\boldsymbol{q}_i \in B_e(\boldsymbol{p})$ and a normalized weight function $w(\boldsymbol{p},\boldsymbol{q})$.

$$I(\boldsymbol{p}) = \frac{\sum_{\boldsymbol{q}_i \in B_e(\boldsymbol{p})} w(\boldsymbol{p},\boldsymbol{q_i})[I(\boldsymbol{q}_i) + \nabla I(\boldsymbol{q}_i)(\boldsymbol{p} - \boldsymbol{q}_i)]}{\sum_{\boldsymbol{q}_i \in B_e(\boldsymbol{p})} w(\boldsymbol{p},\boldsymbol{q_i})}$$

Where the weight function $w(\boldsymbol{p},\boldsymbol{q})$ is composed of a directional component that ensures greater contribution from pixels closer to the propagation direction; a geometric distance component that decreases contribution of pixels geometrically farther from $\boldsymbol{p}$; and a level set distance component that increases the contribution of pixels closer to the region contour **Ω**.





a) b)

Figure 7: Inpainting methods. a) FMM Inpainting. The reconstruction process starts at the boundary $\partial\Omega$, following the normal inwards Ω. Image from [24]**; b) Exemplar-based Inpainting. For a patch around pixel p, candidates are selected and blank pixels are filled with pixels from the patch with the highest confidence. Image from** [29]**.**

Exemplar-based Inpainting

Exemplar-based inpainting techniques generate new texture by sampling and copying color values from the source image. The Exemplar-based inpainting algorithm in [29] is a *patch-based* technique (contrary to diffusion-based techniques that are usually *pixel-based* techniques). The algorithm maintains a color value per pixel (the actual pixel color or 'empty', if the pixel has not been filled yet) and a confidence value per pixel. This confidence value becomes constant once the pixel is filled.

The algorithm works by computing a priority queue of patches to be filled. This queue is sorted to prefer patches that are the continuation of strong edges and are surrounded by high-confidence pixels. Formally, priority is defined as:

$$P(\boldsymbol{p}) = C(\boldsymbol{p})D(\boldsymbol{p})$$

Where $C(\boldsymbol{p})$ is the confidence term and $D(\boldsymbol{p})$ is the data term for a given pixel $\boldsymbol{p}$. The confidence term is a measure of the amount of reliable information surrounding a pixel, biased towards patches that have more of their pixels already filled and further preferring patches with pixels filled early on. The data term is a function of the strength of isophotes around the region boundary in each iteration of the algorithm. The idea is to prefer the filling of linear structures first as in diffusion inpainting techniques.

Once a patch has been selected for filling, the whole image is searched for a similar patch that minimizes the sum of squared distances with respect to the patch being filled. After choosing an appropriate patch, pixels that lie inside the region Ω are filled with pixels from the selected patch. This process is repeated until there are no more patches left to fill.

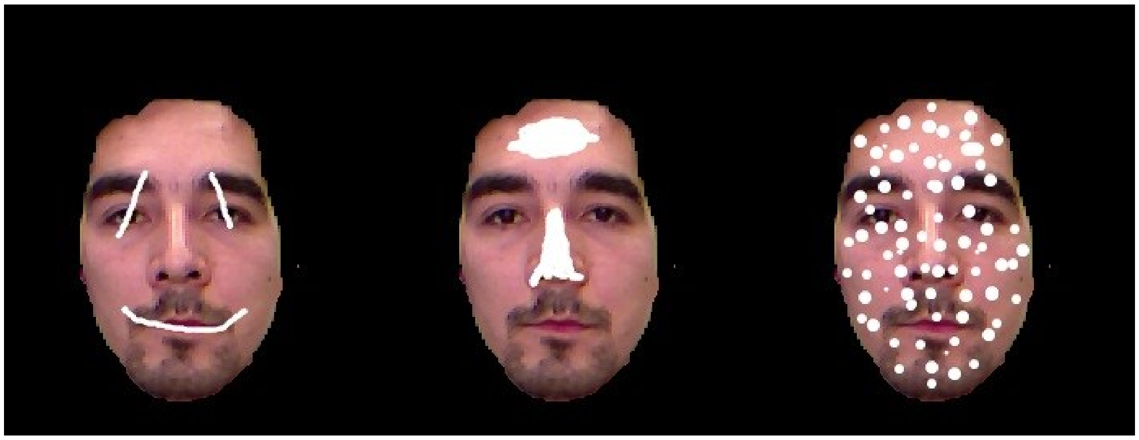
Method Comparison

We compared the methods of Fast Marching Method (FMM) Inpainting [28] and Exemplar-based Inpainting [29] for the reconstruction of holes in depth maps[7]. For this, several adjustments to the algorithms had to be made, especially regarding the treatment of depth map information.

[7] An implementation of Criminisi's method can be found at http://www.cc.gatech.edu/~sooraj/inpainting/. For Telea's method, an implementation is present in the OpenCV library.

Depth maps have discrete values roughly between 0 and 10000, and can be represented as 16-bit grayscale images. Unfortunately, this mode is not standard, and the algorithms tested were written to work with standard image modes (usually, 8-bit grayscale or 24-bit RGB). To overcome this limitation, we modified the algorithms to work with depth information.

The tests involved the reconstruction of hand-made holes in a depth map. Results are compared against a hole-free ground-truth model using mean square error. Figure 8 and Table 3 show the tests and results.



a) b) c)

Figure 8: Inpainting test images. a) simple thin lines, b) more complex and bigger holes and c) holes scattered all over the image.

	MSE		
	Base Error	Exemplar Inpainting	FMM Inpainting
Test 1	3221.067	**0.005**	0.014
Test 2	8809.714	1.429	**1.235**
Test 3	16195.130	0.251	**0.219**

Table 3: Inpainting tests error.

From the tests can be seen that while exemplar-based inpainting shows less error in the first test, it is surpassed by FMM inpainting in the following and more complex tests. This results are probably due to the fact that exemplar-based inpainting works with image patches. While patches can be a good solution for the reconstruction of color images, depth maps represent in this case the shape of relatively smooth surfaces. The use of patches would then introduce unwanted discontinuities, thus increasing the error. On the other hand, FMM inpainting ensures the smoothness of the reconstruction. For this reason this is the method we choose and use in the rest of this work.

Depth Map Segmentation

After reconstructing missing areas in the depth map, the next step is to segment and extract the object(s) of interest from the scene. We made a qualitative assessment of three different segmentation methods.

Average-based segmentation

This segmentation method will discard pixels farther away from the sensor than the mean depth value of the selected area. This process works well with faces, where usually the face is closer to the sensor than the rest of the body. However, this method does not take into account the orientation of the face. Segmentation results are therefore distorted when the subject is not facing the sensor.

k-Means Segmentation

This algorithm attempts to find clusters within the depth map using the k-means clustering method [30], with $k = 2$. Before clustering, the background is eliminated from the scene by filtering depth values larger than a threshold defined as the mean depth plus one standard deviation. Then, clustering is performed and the cluster whose centroid is closer to the center of the region of interest (or the head position, if tracking is enabled) is selected as the head cluster.

Grabcut Segmentation

The Grabcut Segmentation method [31] is a semi-automatic segmentation algorithm that requires minimal user interaction. This technique uses both region and boundary information contained in an image in order to perform segmentation by “cutting the graph” following a least-energy path. This energy is defined so that its minimum value corresponds to areas where a good segmentation can be made.

We apply this method by defining the subject as our region of interest and refining by marking foreground and background regions as required. For our tests, we show only the first iteration of the algorithm after the region of interest is defined, without marking any foreground or background regions, but results improve considerably with more detailed user intervention.

Method Comparison

We compared all three methods[8] to segment faces under three different pose situations: facing the sensor, looking to the right and looking up. Our qualitative results in Figure 9 show that different methods work better for certain situations than others. For example, all methods work acceptably well when the subject is facing the sensor, but the results are mixed for when the subject is looking up. Since there is no method that outperforms the rest by an ample margin, we perform segmentation method selection on a case by case basis after analyzing the specific requirements for every application.

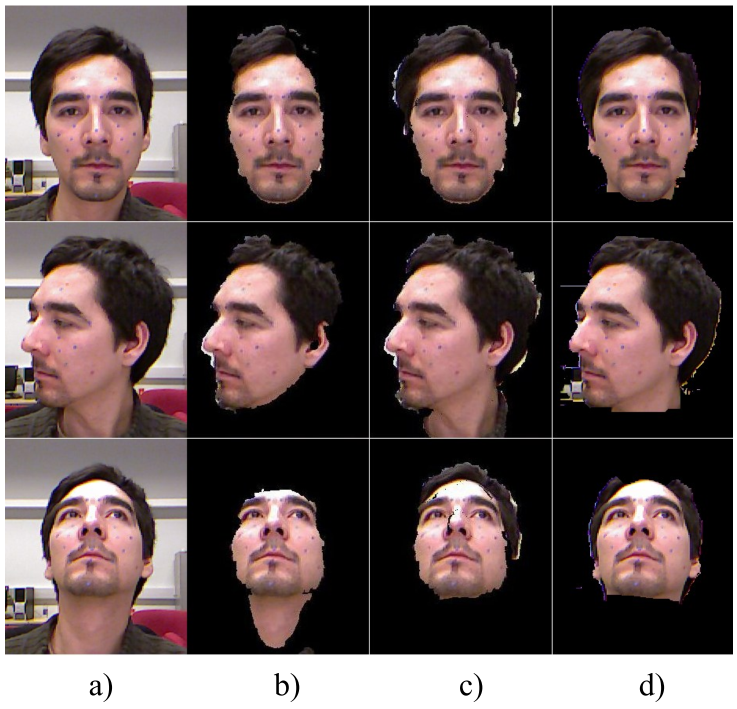


Figure 9: Segmentation Methods. In each row, a) original image; b) Average segmentation; c) k-means segmentation; d) grabcut segmentation

[8]For comparison, we used the implementation of k-means clustering present in the Point Cloud library, and the implementation of the Grabcut algorithm present in the OpenCV library.

Point Cloud Generation

The final step after performing hole-filling and segmentation is to generate a point cloud from the depth map.

Depth and RGB Registration

Cameras that register depth and RGB images are at a certain distance from each other; hence they capture slightly different views of the same scene. Registration of depth and RGB cameras is necessary if we want to extract color information for each point in the depth map. Registration of the depth and RGB cameras and projection of depth points into the RGB image plane is achieved as follows:

- Calculate $R_{IR \to RGB}$, the relative rotation between the cameras and $T_{IR \to RGB}$, the relative translation between the cameras.

$$R_{IR \to RGB} = R_{RGB} \cdot R_{IR}^{T}$$
$$T_{IR \to RGB} = T_{IR} - R_{IR \to RGB}^{T} \cdot T_{RGB}$$

- Back-project depth image points into the IR camera's coordinate system, apply transformation to the RGB camera's coordinate system, and finally project the point onto the RGB camera image.

$$p_{IR} = K_{IR}^{-1} \cdot p$$
$$p_{RGB} = R_{IR \to RGB} \cdot p_{IR} + T_{IR \to RGB}$$
$$p_{Im} = K_{RGB} \cdot p_{RGB}$$

In practice this procedure is implemented by the underlying framework using pre-calculated shift tables where the position of a depth pixel in the RGB image space is indexed according to its position and depth value.

The calibration values used by each framework that provides registration are usually adequate. In [32] a joint stereo calibration approach is presented and registration is compared to that provided by OpenNI. Their results show that the calibration values used by that framework are accurate, as their re-projection error was similar.

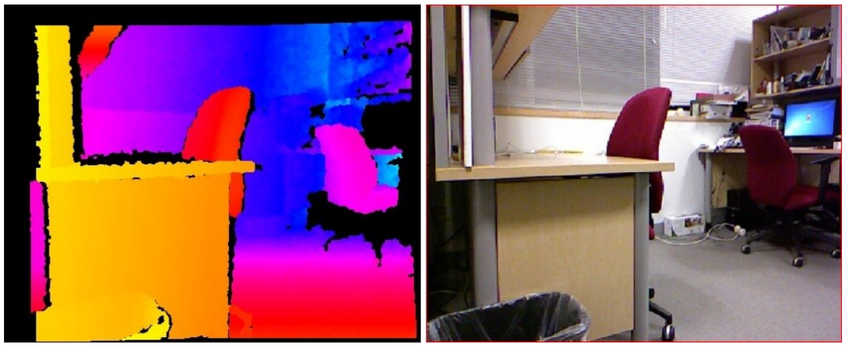


Figure 10: Registration of depth and RGB image.

From the Image Plane to Sensor Coordinates

The final step involves the transformation of a point $p = (u, v, d, c)$ where (u, v) is the position of the point in the image plane, d is the depth of the point and c is the color of the point. The transformation is given by:

$$x_c = \frac{d(u - u_0)}{\alpha_x}$$

$$y_c = \frac{d(v - v_0)}{\alpha_y}$$

$$z_c = d$$

Where $\alpha_x, \alpha_y, u_0, v_0$ are the intrinsic camera parameters. The point is also assigned the color value c. This process is repeated for every valid point in the registered depth map.

Chapter 6

Point Cloud Smoothing

Kinect uses a speckle light pattern to capture depth, which is calculated based on the position and intensity of the reflected infrared light. Because of this, depth measurements are less accurate in areas with poor reflection or where diffraction occurs, such as zones perpendicular to the sensor or around the edges of objects [33].

Given the relative low resolution of the sensor, noisy measurements have a big impact in the overall quality of the generated model. In this chapter, we explore how Kinect compares against a higher resolution 3D scanner. We also examine smoothing methods that aim to reduce the noise in the acquired point cloud, using the aforementioned high-resolution point cloud as a ground truth for our tests.

Kinect vs. Scanner

To test the accuracy of Kinect we captured scans of a face-shaped mask with a custom structured-light scanner. This scanner uses a phase-shift pattern to acquire images, and has a resolution of 640x480 pixels. Specific details about this device can be found in [7]. We also captured and generated a point cloud of the same mask using Kinect, with no further post-processing.

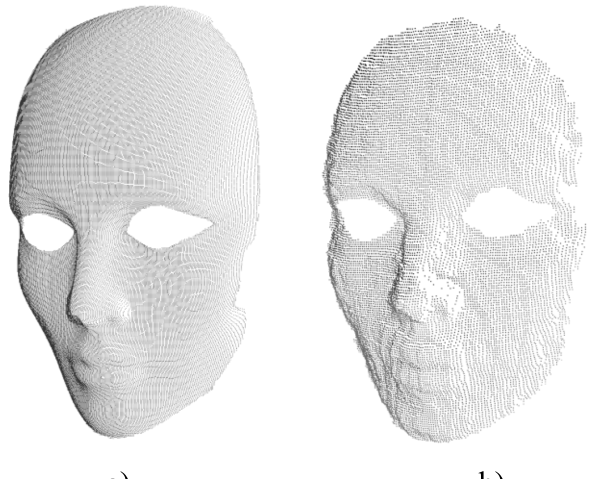


Figure 11: Two scans of a face-shaped mask. a) acquired with a high resolution scanner; b) acquired with Kinect.

Figure 11 shows two scans of the face-shaped mask. It can be seen clearly that there is a difference in resolution between the two scans. Table 4 shows statistics for each device.

Device	Points	Resolution
M.C.L. Scanner	78159	~0.2 mm
Kinect	15472	~1 mm

Table 4: Number of points and resolution for each scan.

If we assume the higher resolution scan as a ground truth, we are interested then in the accuracy error that a raw Kinect scan has with respect to the established baseline. For this, we first align the scans using the ICP algorithm. Then for each point in the smallest cloud, we obtain the distance to its nearest neighbor. With these distances, we can compute the error in terms of the mean squared distance and L_∞ Norm, as Table 5 shows.

MSE (L_2 Distance)	6.70382
Average	1.47027
Std. Dev.	2.1313
L_∞ Norm	24.0844

Table 5: Error for a raw scan.

Figure 12 shows a distance map for the Kinect scan. As can be seen, the greatest differences are around the nose area of the mask, the corner of the eyes and some spurious outliers around the borders.

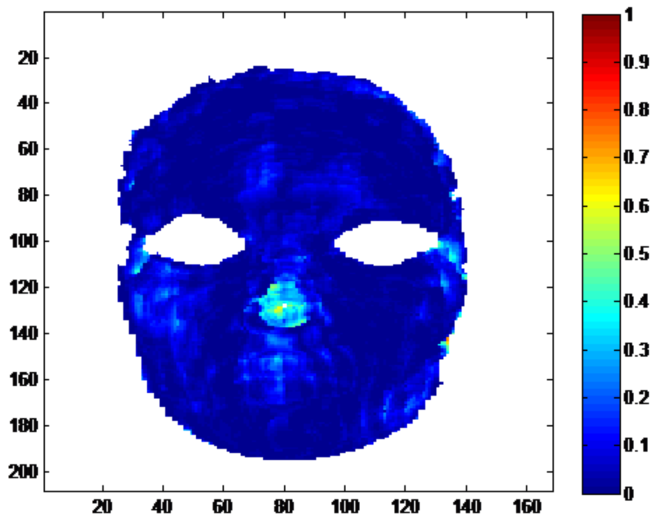


Figure 12: Distance map for raw scan.

Point Cloud Smoothing Methods

We consider two smoothing techniques: Laplacian smoothing and Windowed Sinc smoothing.

Laplacian Smoothing

A common diffusion smoothing method is Laplacian smoothing. This method incrementally moves points in the cloud in the direction of the Laplacian. The Laplacian operator can be linearly approximated at each vertex by the umbrella operator [34]:

$$L(\boldsymbol{v}_i) = \sum_{j \in i^*} w_{ij}(\boldsymbol{v}_j - \boldsymbol{v}_i)$$

Where i^* is the 1-ring neighborhood of the vertex $\boldsymbol{v}_i$, and w_{ij} is the weight of the edge (i, j) corresponding to the vertex $\boldsymbol{v}_i$, with $\sum_{j \in i^*} w_{ij} = 1$.

In our tests, the weighting scheme $w_{ij} = 1/|i^*|$ was used, and three iterations of the algorithm were performed.

Windowed Sinc Smoothing

Taubin proposes the use of a traditional signal processing approach to perform point cloud smoothing [35]. This technique adjusts the position of each point using a windowed sinc interpolation kernel.

The main idea here is to use a low-pass filter to relax the mesh. In traditional filter design, the ideal low-pass filter transfer function is approximated using a trigonometric polynomial approximation by truncating its Fourier series.

A sinc window is used because it reduces the Gibbs overshooting effect produced by directly truncating the Fourier series. The filter used then becomes:

$$f_n = w_0 \frac{\theta_{PB}}{\pi} (T_0\left(1 - {}^{k}/_{2}\right) + w_n \sum_{n=1}^{N} \frac{2\sin(n\theta_{PB})}{n\pi} T_n(1 - {}^{k}/_{2})$$

Where $w_0 .. w_n$ are the window coefficients. The tested implementation uses a Hamming window.

To further improve the algorithm, the shrinkage effect is reduced by first performing a topological and geometric analysis of the cloud, and selecting feature edges. A feature edge occurs when the angle between the two surface normals of a polygon sharing an edge is greater than a certain angle α. In our tests, $\alpha = 45°$. Vertices are then classified as *simple* (no feature edges attached to it), *interior* (exactly two feature edges attached to it) or *fixed*.

Once the classification is known, the vertices are smoothed differently. Fixed vertices are not smoothed at all. Simple vertices are smoothed normally. Interior vertices are smoothed only along their two connected edges, and only if the angle between the edges is less than a certain angle β. In our tests, $\beta = 15°$.

Method Comparison

We compared[9] the two aforementioned smoothing methods against the ground-truth higher resolution scan of a face mask previously taken with the Motion Capture Lab scanner.

As we did at the beginning of this chapter for the raw Kinect scan, we measured the mean squared error and L_∞ Norm in point correspondences for each smoothed point cloud.

[9] We used the implementation of the Windowed Sinc filter present in the VTK library.

	Smoothing Method		
	None	Laplacian	Windowed Sinc
MSE (L_2 Distance)	6.70382	7.9046	**6.10081**
L_∞ Norm (Distances)	24.0844	33.1252	**24.0844**

Table 6: Smoothing comparison results.

As can be extracted from the results in Table 6 and seen in Figure 13, windowed sinc smoothing presents the best visual response while maintaining the most prominent features of the face mask. This method reduces the overall mean squared error and also prevents shrinkage, a known effect when using Laplacian smoothing (and the most likely cause of the error increase using this method).

Another interesting result is that while the overall error is reduced by windowed sinc, we observe that the L_∞ distance remains the same. This is due to the vertex classification part of the algorithm, where border vertices are not smoothed to reduce the shrinking effect previously mentioned.

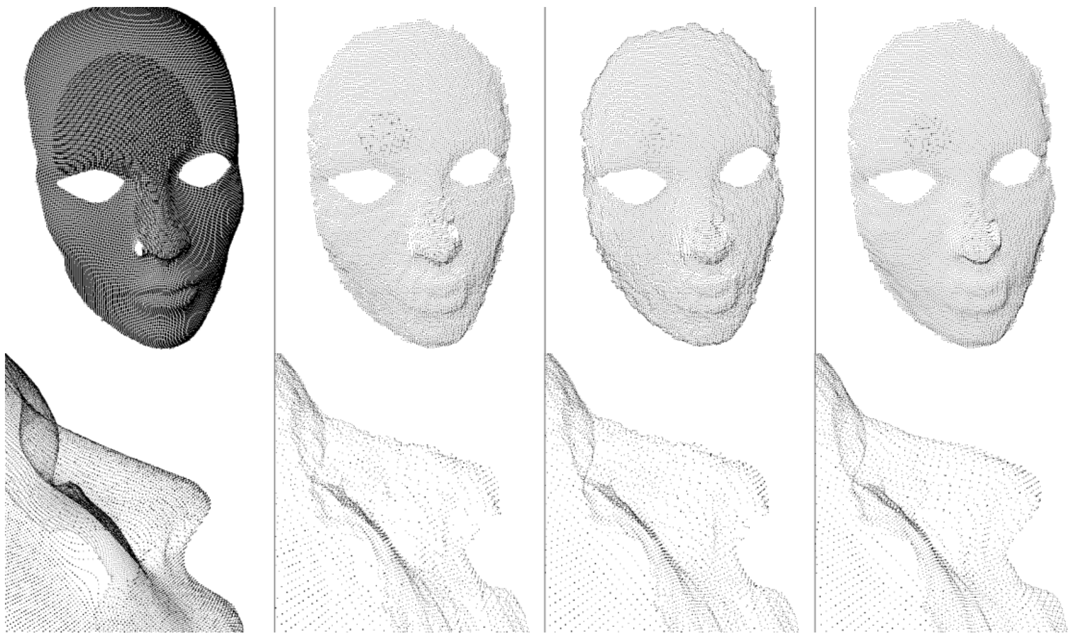


Figure 13: Visual comparison of smoothing methods. a) high-resolution scan; b) low-resolution Kinect scan, no smoothing; c) low-resolution scan, Laplacian smoothing; d) low-resolution scan, Windowed Sinc smoothing.

The distance map in Figure 14 shows how both smoothing methods work. While in the case of Laplacian smoothing the error is actually increased around the borders by the shrinking effect, windowed sinc avoids this problem. Also, by maintaining features, windowed sinc isolates and reduces zones with high error. In the rest of this work, we use windowed sinc smoothing.

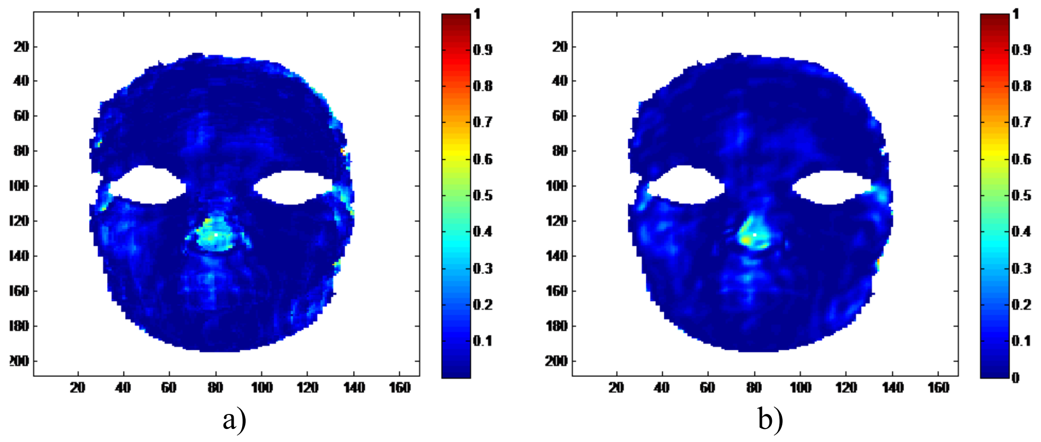


Figure 14: Distance maps for smoothing methods. a) Laplacian smoothing; b) windowed sinc smoothing

Chapter 7

From Point Clouds to 3D Models

After the acquisition and processing of a point cloud, the final step required to obtain a 3D structure is to generate a triangulation from this cloud. The final reconstruction step is to generate a triangle mesh from our point cloud.

Triangulation of a point set is an important problem with applications that go beyond graphics. The quality of the generated mesh is important as well. For example, applications such as finite element simulation in CAD/CAM require triangular meshes of bounded aspect ratio to avoid ill-conditioning and discretization errors [36]. Several different triangulation methods have been proposed in the past as surveyed in [37], but none of them can guarantee quality triangulations for any given point set. In this chapter we present, discuss and compare three different triangulation techniques.

Mesh Generation

For this work we considered three triangulation algorithms, two popular methods and a proposed simple triangulation method for organized point clouds.

Marching Cubes with Algebraic Point Set Surfaces

The Marching Cubes Algorithm [38] is a well-known algorithm for surface reconstruction, which solves the reconstruction problem by forming a facet approximation to an isosurface through a scalar field sampled on a rectangular 3D voxel [39]. In the tested implementation, the surface is approximated at each voxel by using a moving least squares spherical fit to the point cloud data in the context of the APSS framework presented in [40], [41]. Then, the voxel intersection points are calculated and triangles are generated from the approximated surface. Quality and number of triangles are determined by the chosen voxel size.

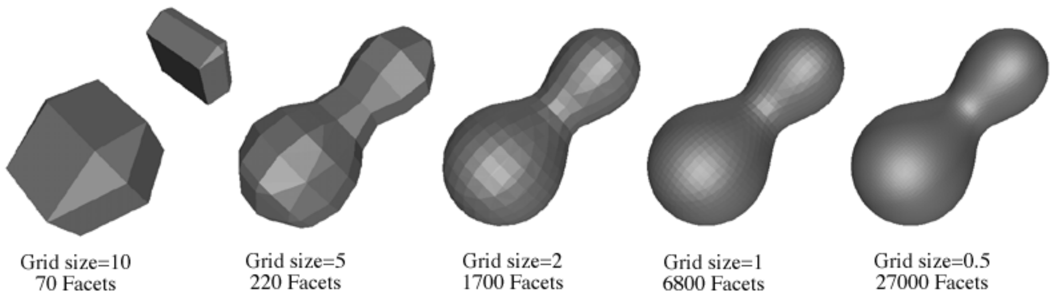


Figure 15: Example of generated polygons (facets) as a function of voxel size. Image by Paul Bourke.

Ball Pivoting Algorithm

The Ball Pivoting Algorithm [42] generates a triangle mesh from a point cloud. Given the point cloud and the estimated normal at each point, the algorithm begins finding a seed triangle such that a ball of radius ρ touches its vertices and no other point lies inside this ball. The algorithm then adds points by rotating the ball around a pivot point and connecting those points touched by the ball. Triangles are generated when three points become connected. While the algorithm has no problem connecting points whose distance is smaller than ρ, for points farther than ρ the algorithm will generate holes. This issue can be solved by applying multiple passes of the algorithm increasing the ball radius after each pass, but our experiments showed that this require extreme fine tuning to avoid creating artifacts. Quality in this case is determined by the ball radius ρ.

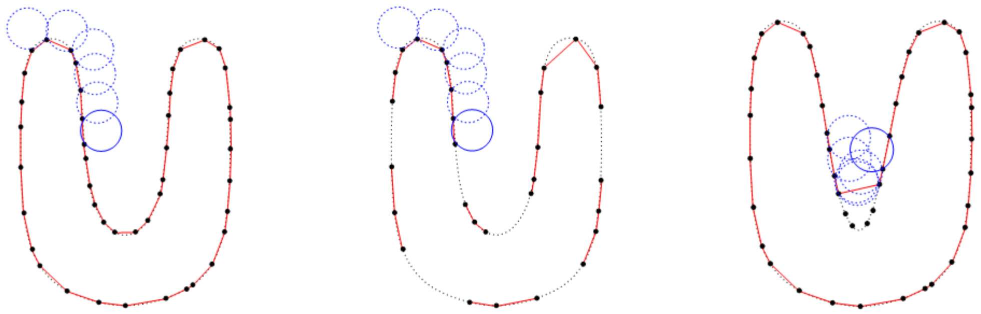


Figure 16: Ball pivoting in two dimensions. Points are connected by pivoting a ball around nearby points. The second image shows how a small ball radius creates holes in the reconstructed surface. The last image shows how a large ball radius will lead to artifacts in the reconstructed mesh. Image from [42].

Simple Organized Point Cloud Triangulation

The proposed triangulation method takes advantage of the fact that point clouds captured from a Kinect device are organized as a two-dimensional array of points. Because the perspective projection transformation to obtain world-space coordinates for each point in the depth map is homographic, resulting (x, y) coordinates for any point $\boldsymbol{p}$ preserve the spatial relation to its neighbors with respect to the original depth map.

This algorithm generates triangles considering only the closest neighbors of a point $\boldsymbol{p}$ as follows: for each valid point $\boldsymbol{p}_{i,j}$, its neighborhood $B_p = \{\boldsymbol{p}_{i+1,j}, \boldsymbol{p}_{i,j+1}, \boldsymbol{p}_{i+1,j+1}\}$ is considered. From this neighborhood, two triangle configurations are possible. We define pivot points $\boldsymbol{v}_1 = \boldsymbol{p}_{i,j+1}, \boldsymbol{v}_2 = \boldsymbol{p}_{i+1,j}$ if points $\boldsymbol{p}_{i,j+1}, \boldsymbol{p}_{i+1,j}$ are valid points. This configuration generates triangles $t_a = \{\boldsymbol{v}_2, \boldsymbol{v}_1, \boldsymbol{p}_{i,j}\}$ and $t_b = \{\boldsymbol{v}_2, \boldsymbol{p}_{i+1,j+1}, \boldsymbol{v}_1\}$ if and only if all the points in t_a, t_b are valid. If points $\boldsymbol{p}_{i,j+1}, \boldsymbol{p}_{i+1,j}$ are not valid then a second possible configuration $\boldsymbol{v}_1 = \boldsymbol{p}_{i+1,j+1}, \boldsymbol{v}_2 = \boldsymbol{p}_{i,j}$ is tried. This configuration generates triangles $t_a = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{p}_{i+1,j}\}$ and $t_b = \{\boldsymbol{v}_1, \boldsymbol{p}_{i,j+1}, \boldsymbol{v}_2\}$ if the points are valid. A valid point is a point whose z coordinate is defined. If no pivot configuration is valid, no triangles are generated.

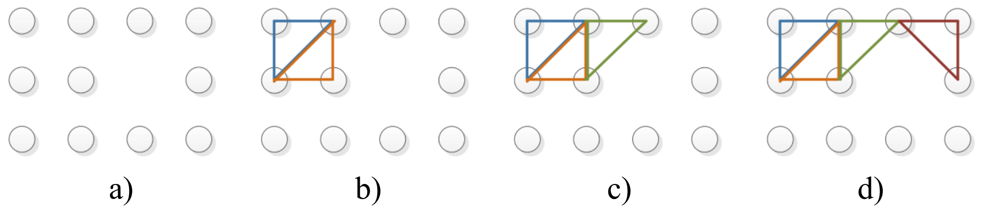


Figure 17: Triangles generated by the Simple Organized algorithm. After three iterations: a) initial point cloud; b) two triangles generated using the first pivot rule; c) one triangle generated using the first pivot rule; d) a triangle generated using the second pivot rule.

Method Comparison

We compared[10] the three previously exposed triangulation techniques measuring the quality of the generated triangles. For applications that involve numerical methods, the error bound is kept low if the triangles are as close as equilateral triangles [43]. Thus, it makes sense to measure triangle quality with respect to the properties of the equilateral triangle. It is also desirable that the generated triangle meshes are a visually plausible representation of the scanned

[10] We used the implementation of Marching Cubes and Ball Pivoting present in MeshLab.

surface. For this, we expect the triangulation to be smooth with respect to the surface and that it contains no holes in the viewing direction of the sensor.

Several triangle quality measures have been developed over the past years as surveyed in [44]. For this work we selected measures that include different properties of a triangle such as the inner angles, side length, inradius and circumradius. We also measured angle distribution in the generated mesh. Table 7 shows the selected measures. All mea sures are normalized with respect to the ideal values of an equilateral triangle. This means that values closer to 1 are better.

Measure	Description
$q_{\alpha_{min}} = \frac{3\alpha_{min}}{\pi}$	α_{min} is the smallest angle of a triangle.
$q_{Ll} = \frac{l_{min}}{l_{max}}$	l_{min}, l_{max} are the length of the shortest and longest edge of a triangle, respectively.
$q_{ALS} = \frac{4\sqrt{3}A}{l_1^2 + l_2^2 + l_3^2}$	l_1, l_2, l_3 are the edges of a triangle; A is the area of the triangle.
$q_{Rr} = \frac{2r}{R}$	R is the circumradius of a triangle;r is the inradius.
$q_{Lr} = \frac{2\sqrt{3}r}{l_{max}}$	r is the inradius of a triangle; l_{max} is the longest edge.

Table 7: Quality measures for triangles.

Table 8 shows the results for each method. Figure 18 shows the distribution of angles in the generated triangles for each method.

Quality Measure	Triangulation Methods					
	Marching Cube APSS		Ball Pivoting		Simple Organized	
	$\overline{X}$	σ	$\overline{X}$	σ	$\overline{X}$	σ
$q_{\alpha_{min}}$	0.491	0.491	**0.648**	**0.163**	0.591	0.193
q_{Ll}	0.497	0.497	**0.642**	**0.128**	0.589	0.169
q_{ALS}	0.651	0.651	**0.815**	**0.150**	0.760	0.182
q_{Rr}	0.620	0.620	**0.779**	**0.181**	0.723	0.199
q_{Lr}	0.278	0.278	**0.347**	**0.074**	0.322	0.085

Table 8: Results for triangle quality measurement.

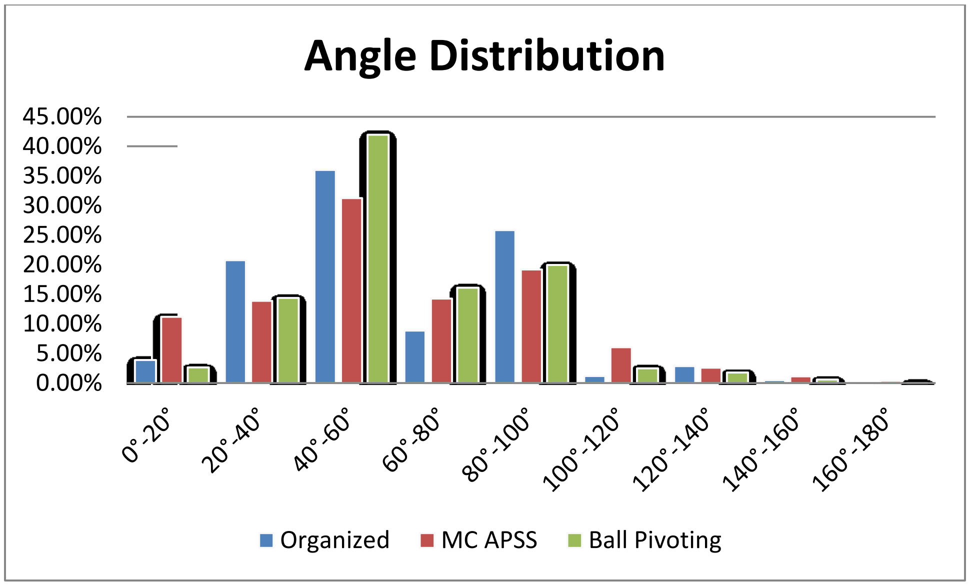


Figure 18: Angle distribution for each triangulation method.

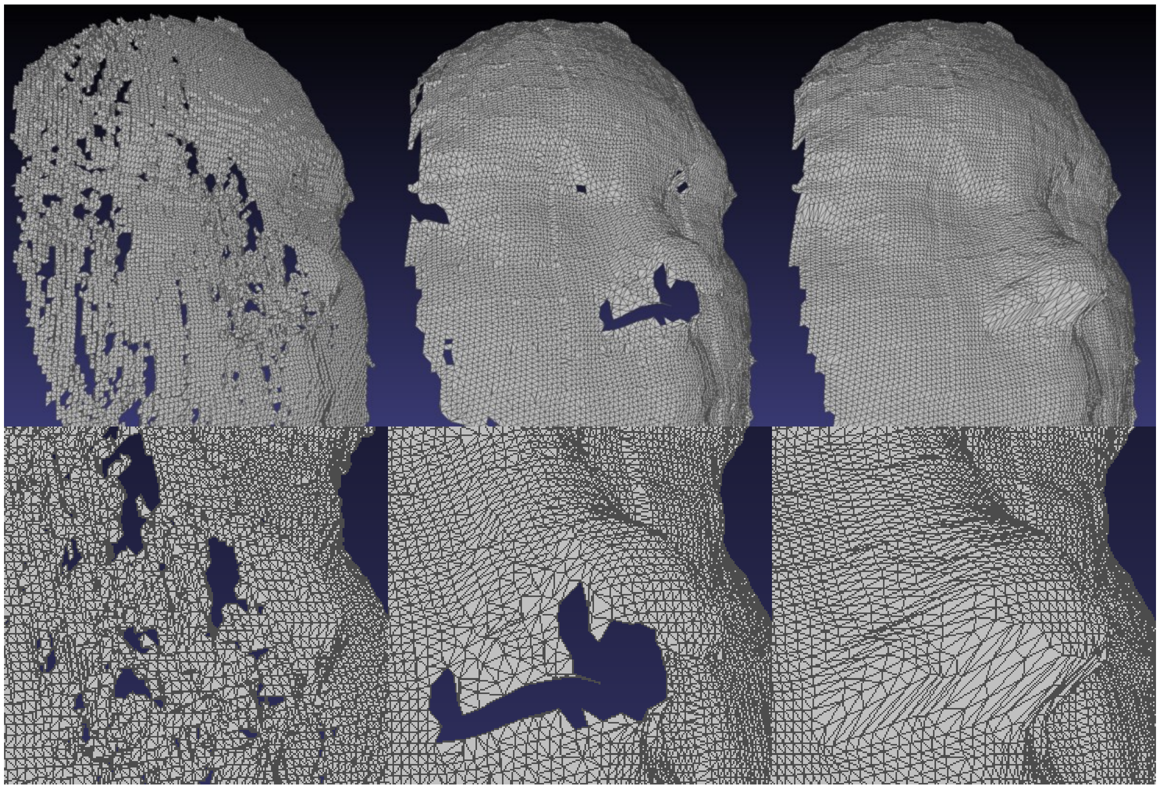


Figure 19: Triangulation methods. a) Marching Cube (APSS); b) Ball Pivoting; c) Simple Organized. Top row: full view of a generated mesh. Bottom row: a close-up of the nose zone.

According to our triangle quality measurement, the method that generates the best triangles is the Ball Pivoting Algorithm. However, in terms of visual quality, both Marching

Cubes and Ball Pivoting generate meshes with holes in them. This can be seen in Figure 19. On the other hand, our Simple Organized Point Cloud method generates a good mesh with no holes. While holes can be closed using mesh editing software, our method requires no user intervention or supervision.

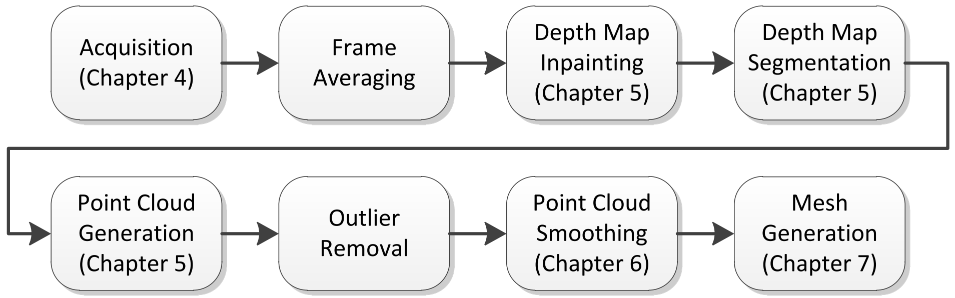
Our method also generates triangles with quality similar to Ball Pivoting. Similarly as Figure 18 shows, there is a similar distribution of angles in both Ball Pivoting and Simple Organized Point. We decided to use our method because it generates good quality and hole-free meshes automatically.

Chapter 8
Applications

After reviewing, discussing, testing and selected methods and algorithms for 3D scanning, the next logic step is to show the effectiveness of those techniques in an application. In this chapter we introduce a face capture and processing pipeline which utilizes all of the techniques discussed so far in this work. We then use this processing pipeline to acquire a dataset of human faces. We finally generate an active appearance model from this dataset and test its generalization capability to approximate new observations.

A Face Capture and Processing Pipeline

Our proposed pipeline is composed of sequential instances of the methods and algorithms presented so far in this work, along with new techniques devised to help enhance the quality of static sequences. Figure 20 shows a diagram of the proposed system.



Acquisition (Chapter 4)
Frame Averaging
Depth Map Inpainting (Chapter 5)
Depth Map Segmentation (Chapter 5)
Point Cloud Generation (Chapter 5)
Outlier Removal
Point Cloud Smoothing (Chapter 6)
Mesh Generation (Chapter 7)

Figure 20: Face capture and processing pipeline

The first step in our processing pipeline is to acquire color and depth information of the face from a static subject. For this, we use the application presented in Chapter 4 to acquire a short video sequence.

After the sequence has been acquired, we average the color image and depth map where it contains valid values. With this we attempt to extract the greatest amount of information from the depth map, exploiting the fact that we are working with static sequences.

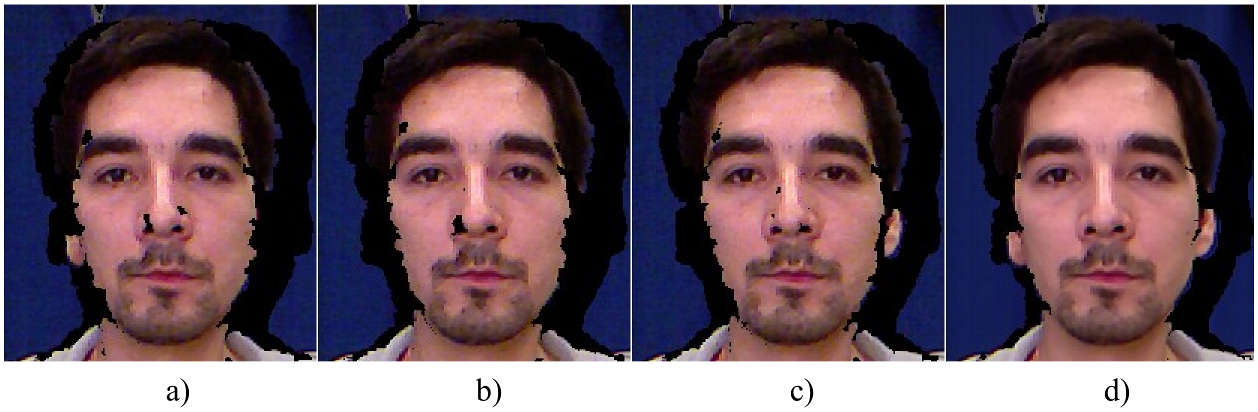


Figure 21: RGB and depth averaging. a), b), c) individual noisy frames from a sequence. Black pixels indicate values where depth values are invalid; d) reconstructed frame from a sequence.

On average, every individual frame of our sample sequence contains 50733 valid pixels, while the reconstructed frame contains 53252 valid points. This represents a 5% increment in the number of available pixels. Given the low resolution and limitations of the Kinect sensor this increase in the amount of usable data is significant.

Once we have the recovered depth map, we apply the FMM inpainting technique described in Chapter 5. We then perform semi-automatic segmentation by means of the Grabcut segmentation algorithm also portrayed in Chapter 5.

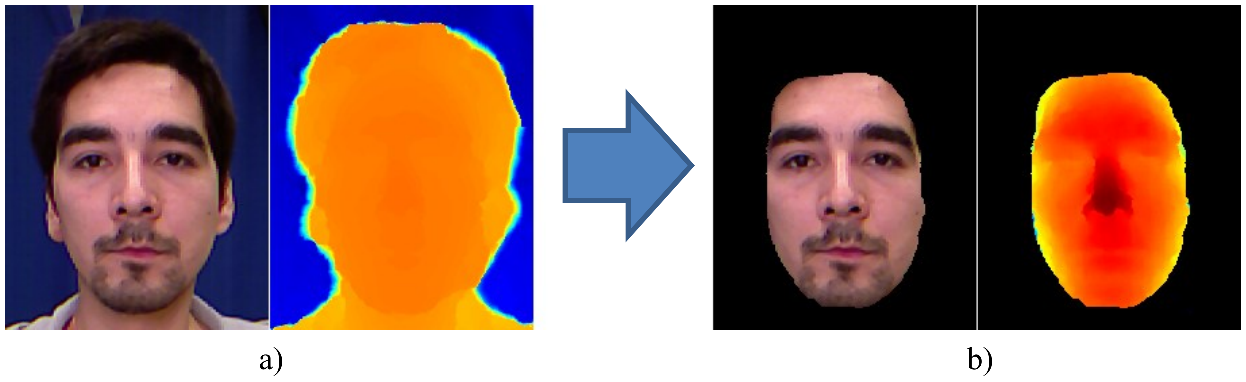


Figure 22: Inpainting and segmentation. a) RGB and depth images after inpainting and before segmentation; b) RGB and depth images after Grabcut segmentation.

After segmentation, we construct a point cloud employing the method presented in Chapter 5. Before generating a mesh, we apply a statistical outlier removal filter. This filter calculates for each point the average distance and standard deviation using the first 500 nearest-neighbors. It then deletes points that are farther away than three times the standard deviation from its closest neighbor.

Finally, we apply the windowed sinc smoothing method depicted in Chapter 6, and we obtain our final model by generating a mesh applying the simple organized triangulation algorithm illustrated in Chapter 7.

Building an Active Appearance Model

Active Appearance Models

The concept of an Active Appearance Model (AAM) was first proposed in [45]. Active Appearance Model are non-linear, generative and parametric models of a certain visual phenomenon [46]. While there are several classes of AAMs, in this work we focus on Independent AAMs.

An Independent AAM models the shape and appearance of a class of objects independently from each other. We represent shape of a face with a vector $S = (X_1, X_2, \dots, X_n, Y_1, \dots, Y_n, Z_1, \dots, Z_n) \in \mathbb{R}^{3n}$ as the concatenation of the (X, Y, Z) coordinates of all n vertices. We also represent appearance with a texture vector $A = (R_1, \dots, R_n, G_1, \dots, G_n, B_1, \dots, B_n) \in \mathbb{R}^{3n}$, the concatenation of the (R, G, B) color information for all n vertices. For a model containing m faces, we can then express a new face as a linear combination of these faces as in [47]:

$$^{m}\boldsymbol{S}(a) = \sum_{i=1}^{m} a_i \boldsymbol{S}_i$$

$$^{m}\boldsymbol{A}(b) = \sum_{i=1}^{m} b_i \boldsymbol{A}_i$$

With $\sum a_i = \sum b_i = 1$. In practice, data compression and dimensionality reduction is achieved by performing Principal Component Analysis (PCA) [48] on the training data set. We can then express our model in terms of the average face and the eigenvectors of the covariance matrix:

$$^{m}\boldsymbol{S}(\alpha) = \overline{\boldsymbol{S}} + \alpha \boldsymbol{\lambda}_s$$

$${}^{m}\mathbf{A}(\beta) = \bar{\boldsymbol{A}} + \beta \boldsymbol{\lambda_a}$$

Model Training

Our data set consists of 35 face scans of Caucasian females acquired using our face acquisition pipeline.

Before training can be performed, we have to bring the faces into correspondence. For this, we first correct all faces for rotation and translation using the ICP algorithm.

To perform registration, we select an arbitrary face as a temporary reference and register every other sample against this face. Since every scan has a rather different configuration (different number of points, different boundaries, etc.) it is difficult to perform non-rigid registration in $\mathbb{R}^3$. All of our samples are genus zero surfaces, therefore we can perform surface uniformization by computing a conformal map $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ [49], [50] from the surface to the unit disk. We then center this map on the tip of the nose and correct for rotation by applying Mobius transformations.

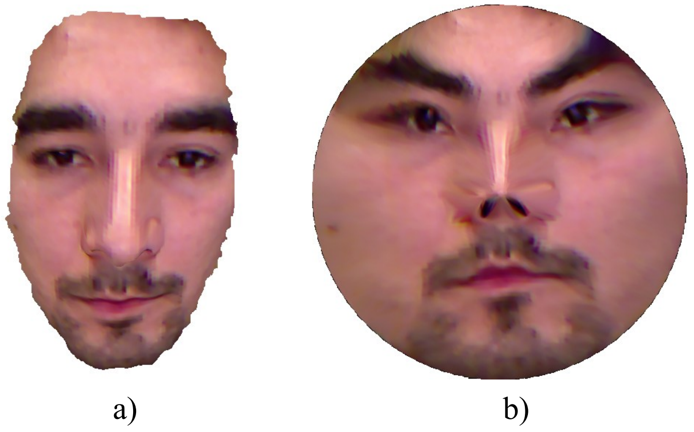


Figure 23: Conformal mapping. a) original face; b) conformal map to the unit disk. Irregular boundaries in a) become regular in b).

,

Once we have conformal maps for every face, we find shape and appearance correspondences as follows: given a chosen reference face F_r, for every new face F_n shape correspondences $F_n \rightarrow F_r$ are found by performing a nearest neighbor search for every point in F_r.

Appearance is computed with respect to the reference face F_r by first calculating a Delaunay triangulation for F_r. Then we calculate barycentric coordinates α, β, γ for every corresponding point from F_n with respect to this triangulation. We can finally compute the interpolated color with respect to the color of the vertices of the surrounding triangle ${}^1C,\ {}^2C,\ {}^3C$ as:

$$C_{n\to r} = \alpha\ {}^1C_n\ + \beta\ {}^2C_n\ + \gamma\ {}^3C_n$$

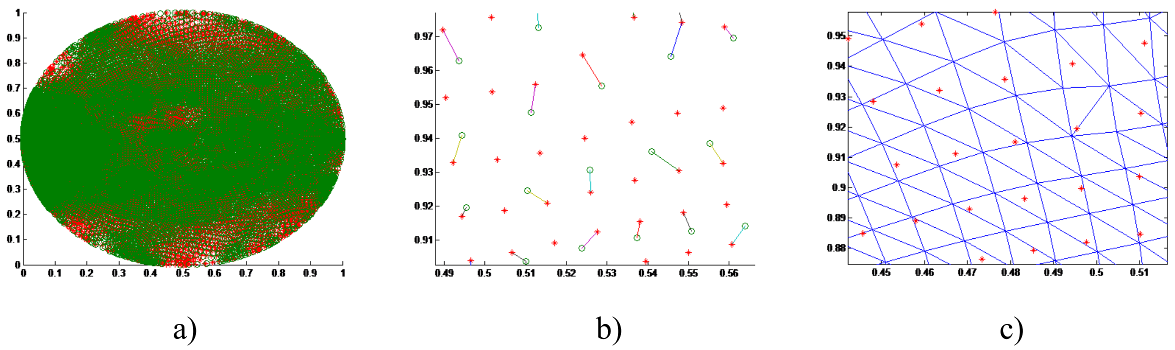


Figure 24: Registration of two different face scans. a) points mapped to the unit disk; b) nearest-neighbor selection of correspondent points; c) Delaunay triangulation for color interpolation.

The final step is to compute the mean shape and mean appearance $\overline{\boldsymbol{S}}, \overline{\boldsymbol{A}}$ and to apply PCA to our shape and appearance sample matrices to obtain the covariance eigenvectors $\boldsymbol{\lambda_s}, \boldsymbol{\lambda_a}$. With a complete model, we can compute the average face as shown in Figure 25.

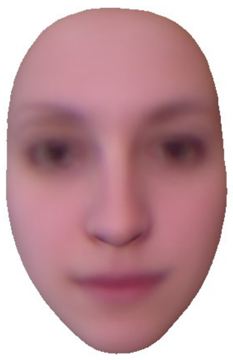


Figure 25: Average face.

Fitting the model to a new face

To fit a new captured face, first we follow the same process as we did for our training data. Then, to generate an approximate representation of the new face using our model, we find the minimum error between the new and the generated face in terms of shape and appearance:

$$E_S = \left|\ {}^*\boldsymbol{S} - {}^m\boldsymbol{S}(\alpha)\right|$$

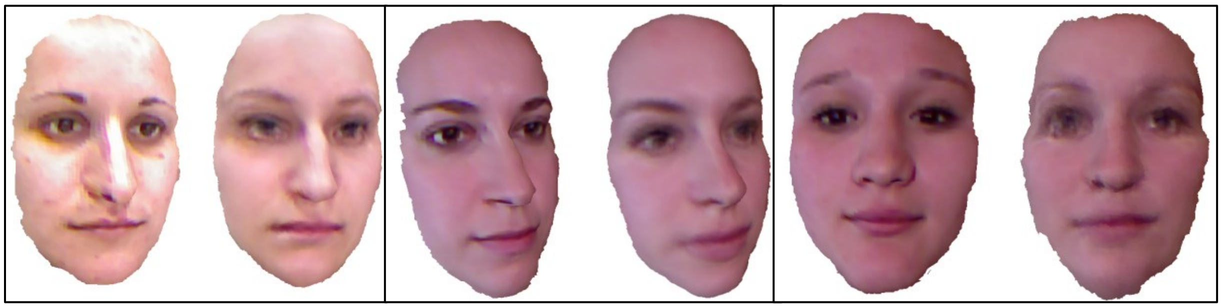
$$E_A = \left| {}^{*}\boldsymbol{A} - {}^{m}\boldsymbol{A}(\beta) \right|$$

Where E_S, E_A are the shape and appearance errors respectively. We define our error metric as the sum of squared distances. We find the parameters $\vec{\alpha}, \vec{\beta}$ that minimize the quantities:

$$\arg\min_{\alpha} \left\| {}^{*}\boldsymbol{S} - {}^{m}\boldsymbol{S}(\alpha) \right\|^2$$

$$\arg\min_{\beta} \left\| {}^{*}\boldsymbol{A} - {}^{m}\boldsymbol{A}(\beta) \right\|^2 \ s.t.\ A_i \in [0,1]$$

We performed leave-one-out cross-validation to test how well our model generalizes to fit new data. Figure 26 shows cases of successful fitting and a failure case. It is worth noting how the model can generalize for different skin colors and illumination variation, as can be seen from the first two images of Figure 26.



a) b) c)

Figure 26: Approximation of new faces. a) and b) show good approximations; c) a failure case. Left: original scan. Right: Fitted model.

Exploring the effect of data

We tested the effects of the number of samples on the overall fitting error. For this, we trained our model several times, each time modifying the number of available samples. We then computed the mean fitting error by performing leave-one-out cross-validation for the number of available samples. The following graph shows the normalized fitting error versus the number of samples in the model.

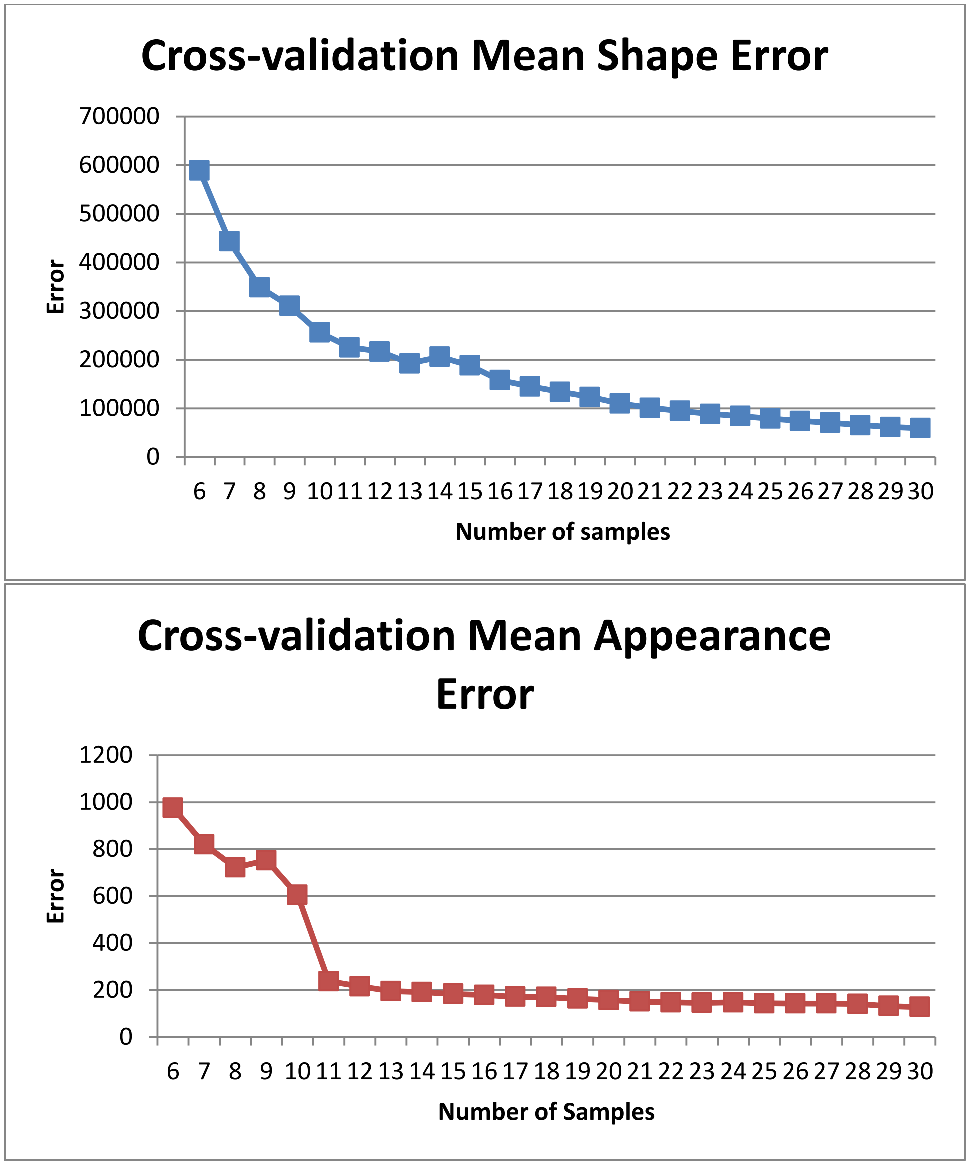


Figure 27: Fitting error vs. Number of samples.

It can be observed that the error decreases as the number of samples increases, as expected; however, the number of samples available is enough to make this error converge. This means that there is still room for great improvement for this model adding a relatively small number of new samples.

Chapter 9
Conclusions

In this thesis we explored the possibilities of acquiring 3D models with Kinect. We proposed and developed a system to acquire image and depth information. After exploring the characteristics of the Kinect sensor, we examined the challenges associated with it.

The core of this work was devoted to deal with such challenges. We proposed the use of image inpainting techniques to fill holes in depth maps. After testing different methods, we selected the fast-marching method inpainting technique. We also tested different depth map segmentation methods.

For point cloud smoothing, after testing different algorithms we selected windowed sinc smoothing. Finally, we proposed a triangulation method for organized point clouds and tested it against established triangulation methods, showing its superiority for our needs.

We compared the accuracy of generated scans against a higher resolution scanner device. While the resolution of Kinect is low, our results showed that it is enough to observe fine detail in most applications. Quality-wise, it is on par with higher resolution devices.

For this work we developed tools for data capture and processing. We organized such tools in a processing pipeline that makes use of all the tools and techniques discussed throughout this work.

As a way to test the effectiveness of our methods, we captured a dataset of human faces and created an active appearance model of such faces. We showed that the scanning resolution achieved by our system is enough to capture distinctive detail in faces.

There are different areas where improvement can be made in future research. For example in the case of scanning static objects, the acquisition system could be augmented by allowing the device to be moved around the target, a technique already in use in other systems. This would increase the amount of available points. If this were the case, our triangulation method would need to be revised to deal with unorganized clouds.

To improve scanning, multiple Kinects could be used. This can be useful in the presence of occlusions, moving targets, or small targets where a single device is insufficient to acquire enough detail. The use of multiple devices involves a calibration step that would go against the goal of portability, but can left as an option.

With respect to our active appearance model, better conformal registration could be achieved by the use of constrained Mobius transformations. For this, landmarks need to be specified on each face. This can be realized either manually or automatically using one of the many techniques in existence.

New depth sensors are appearing in the market. For now, the technical characteristics of those sensors are expected to be similar to Kinect. However, we expect to see higher resolution sensors to hit the market in the near future. The presented scanning system can benefit almost immediately of the availability of such sensors.

References

[1] D. Hoiem, A. A. Efros, and M. Hebert, "Automatic photo pop-up," *ACM Transactions on Graphics*, vol. 24, no. 3, p. 577, 2005.

[2] M. Wilczkowiak, E. Boyer, and P. Sturm, "Camera calibration and 3D reconstruction from single images using parallelepipeds," *Proceedings Eighth IEEE International Conference on Computer Vision ICCV 2001*, vol. 1, no. C, pp. 142-148, 1999.

[3] A. Fusiello, E. Trucco, and A. Verri, "A compact algorithm for rectification of stereo pairs," *Machine Vision and Applications*, vol. 12, no. 1, pp. 16-22, 2000.

[4] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms," *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Volume 1 CVPR06*, vol. 1, no. c, pp. 519-528, 2001.

[5] P. Axelsson, "Processing of laser scanner data—algorithms and applications," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 54, no. 2–3, pp. 138-147, 1999.

[6] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," pp. 195-202, Jun. 2003.

[7] Y. Wang et al., "High Resolution Acquisition, Learning and Transfer of Dynamic 3-D Facial Expressions," *Computer Graphics Forum*, vol. 23, no. 3, pp. 677-686, Sep. 2004.

[8] V. Ganapathi, C. Plagemann, D. Koller, and S. Thrun, "Real time motion capture using a single time-of-flight camera," *IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR)*, 2010.

[9] L. Gallo, A. P. Placitelli, M. Ciampi, and V. P. Castellino, "Controller-free exploration of medical image data: Experiencing the Kinect," *Performance Computing*, pp. 1-6, 2011.

[10] A. S. Jaeyong Sung, Colin Ponce, Bart Selman, "Unstructured Human Activity Detection from RGBD Images," *International Conference on Robotics and Automation (ICRA)*, 2012.

[11] J. Tong, J. Zhou, L. Liu, Z. Pan, and H. Yan, "Scanning 3D Full Human Bodies Using Kinects," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 4, pp. 643-650, Apr. 2012.

[12] J. Smisek, M. Jancosek, and T. Pajdla, *3D with Kinect*. IEEE, 2011, pp. 1154-1160.

[13] Y. Cui, S. Schuon, D. Chan, S. Thrun, and C. Theobalt, “3D shape scanning with a time-of-flight camera,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 1173-1180.

[14] Y. Cui and D. Stricker, “3D shape scanning with a Kinect,” in *ACM SIGGRAPH 2011 Posters on - SIGGRAPH ’11*, 2011, p. 1.

[15] P. Joubert and W. Brink, “Scene Reconstruction from Uncontrolled Motion using a Low Cost 3D Sensor,” in *Proceedings of the Twenty-Second Annual Symposium of the Pattern Recognition Association of South Africa*, 2011, pp. 13-18.

[16] R. A. Newcombe et al., *KinectFusion: Real-time dense surface mapping and tracking*, vol. 7, no. 10. IEEE, 2011, pp. 127-136.

[17] J. Shan and C. K. Toth, *Topographic Laser Ranging and Scanning: Principles and Processing*. CRC Press, 2009, pp. 3-10.

[18] J. Batlle, J. Batlle, E. Mouaddib, and J. Salvi, “Recent progress in coded structured light as a technique to solve the correspondence problem: a survey,” *NCE PROBLEM: A SURVEY, PATTERN RECOGNITION*, vol. 31, no. 7. pp. 963 - 982, 1998.

[19] Z. Liu and Z. Zhang, *Face Geometry and Appearance Modeling: Concepts and Applications*. Cambridge University Press, 2011, pp. 45-47.

[20] J. Geng, “Structured-light 3D surface imaging: a tutorial,” *Advances in Optics and Photonics*, vol. 3, no. 2, p. 128, 2011.

[21] J. Salvi, J. Pagès, and J. Batlle, “Pattern codification strategies in structured light systems,” *Pattern Recognition*, vol. 37, no. 4, pp. 827-849, Apr. 2004.

[22] D. Lanman and G. Taubin, “Build your own 3D scanner,” in *ACM SIGGRAPH ASIA 2009 Courses on - SIGGRAPH ASIA ’09*, 2009, pp. 1-94.

[23] A. Shpunt and Z. Zeev, “Three-dimensional sensing using speckle patterns,” U.S. Patent US 2009/0096783 A12007.

[24] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, “Image inpainting,” *Proceedings of the 27th annual conference on Computer graphics and interactive techniques SIGGRAPH 00*, vol. 35, no. 4, pp. 417-424, 2000.

[25] S. Salamanca, P. Merch, A. Adan, C. Cerrada, and E. Pérez, “Filling Holes in 3D Meshes using Image Restoration Algorithms,” in *International Symposium on 3D Data Processing Visualization and Transmission*, 2008.

[26] K.-jung Oh, S. Yea, and Y.-sung Ho, “Hole-Filling Method Using Depth Based In-Painting For View Synthesis in Free Viewpoint Television (FTV) and 3D Video,” *Proceedings Picture Coding Symposium PCS Chicago USA*, pp. 1-4, 2009.

[27] M. Bertalmio, M. Bertalmio, A. L. Bertozzi, and G. Sapiro, “Navier-Stokes, fluid dynamics, and image and video inpainting,” *PROC. IEEE COMPUTER VISION AND PATTERN RECOGNITION (CVPR)*, vol. 1, pp. 355 - 362, 2001.

[28] A. Telea, “An image inpainting technique based on the fast marching method,” *Journal of Graphics Tools*, vol. 9, no. 1, pp. 23 - 34, 2004.

[29] A. Criminisi, P. Perez, and K. Toyama, “Region Filling and Object Removal by Exemplar-Based Image Inpainting,” *IEEE Transactions on Image Processing*, vol. 13, no. 9, pp. 1200-1212, Sep. 2004.

[30] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006, pp. 423-430.

[31] C. Rother, V. Kolmogorov, and A. Blake, “‘GrabCut’: interactive foreground extraction using iterated graph cuts,” *Computer*, vol. 23, no. 3, pp. 309–314, 2004.

[32] D. H. C, J. Kannala, and J. Heikkil, “Accurate and Practical Calibration of a Depth and Color Camera Pair,” *CAIP*, vol. LNCS 6855, pp. 437-445, 2011.

[33] G. Borenstein, *Making Things See: 3D Vision with Kinect, Processing, Arduino, and MakerBot (Google eBook)*. O’Reilly Media, Inc., 2012, pp. 52-55.

[34] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel, “Interactive multi-resolution modeling on arbitrary meshes,” *Proceedings of the 25th annual conference on Computer graphics and interactive techniques SIGGRAPH 98*, vol. 98, no. Annual Conference Series, pp. 105-114, 1998.

[35] G. Taubin, “A signal processing approach to fair surface design,” in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH ’95*, 1995, pp. 351-358.

[36] T. K. Dey, C. L. Bajaj, and K. Sugihara, “On good triangulations in three dimensions,” *Proceedings of the first ACM symposium on Solid modeling foundations and CADCAM applications SMA 91*. ACM Press, pp. 431-441, 1991.

[37] S. J. Owen, “A Survey of Unstructured Mesh Generation Technology,” *7th International Meshing Roundtable*, vol. 3, no. 6, p. 25, 1998.

[38] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3D surface construction algorithm,” *Computer*, vol. 21, no. 4, pp. 163-169, 1987.

[39] P. Bourke, “Polygonising a scalar field.” [Online]. Available: http://local.wasp.uwa.edu.au/~pbourke/geometry/polygonise.

[40] G. Guennebaud and M. Gross, “Algebraic point set surfaces,” *ACM Transactions on Graphics*, vol. 26, no. 3, p. 23, Jul. 2007.

[41] G. Guennebaud, M. Germann, and M. Gross, “Dynamic Sampling and Rendering of Algebraic Point Set Surfaces,” *Computer Graphics Forum*, vol. 27, no. 2, pp. 653-662, Apr. 2008.

[42] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, G. Taubin, and S. Member, “The Ball-Pivoting Algorithm for Surface Reconstruction,” *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, vol. 5, pp. 349 - 359, 1999.

[43] T. K. Dey, “Good triangulations in the plane,” in *Proc 2nd Canad Conf Comput Geom*, 1990, pp. 102-106.

[44] J. Sarrate, J. Palau, and A. Huerta, “Numerical representation of the quality measures of triangles and triangular meshes,” *Communications in Numerical Methods in Engineering*, vol. 19, no. 7, pp. 551-561, Apr. 2003.

[45] G. J. Edwards, C. J. Taylor, and T. F. Cootes, “Interpreting face images using active appearance models,” *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*, pp. 300-305, 1998.

[46] I. Matthews and S. Baker, “Active Appearance Models Revisited,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 135-164, Nov. 2004.

[47] V. Blanz and T. Vetter, “A morphable model for the synthesis of 3D faces,” in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’99*, 1999, pp. 187-194.

[48] J. Shlens, “A Tutorial on Principal Component Analysis,” *Measurement*, vol. 51, no. 10003, p. 52, 2005.

[49] X. Gu and S.-T. Yau, “Computing conformal structures of surfaces,” *Communications in Information and Systems*, vol. 2, no. 2, pp. 121-145, 2002.

[50] X. Gu, Y. Wang, T. F. Chan, P. M. Thompson, and S.-T. Yau, “Genus zero surface conformal mapping and its application to brain surface mapping.,” *IEEE Transactions on Medical Imaging*, vol. 23, no. 8, pp. 949-958, 2003.