TU WIEN Informatics

Change Detection

unter Verwendung des InfiniTAM Frameworks

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Thomas Steinschauer
Matrikelnummer 01426150

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dr. Stefan Ohrhallinger

Wien, 4. Juni 2021

Thomas Steinschauer	Michael Wimmer

TU WIEN Informatics

Change Detection

using the InfiniTAM framework

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Thomas Steinschauer
Registration Number 01426150

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dr. Stefan Ohrhallinger

Vienna, 4th June, 2021

Thomas Steinschauer	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Thomas Steinschauer

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 4. Juni 2021

Thomas Steinschauer

Contents

CHAPTER 1

Introduction

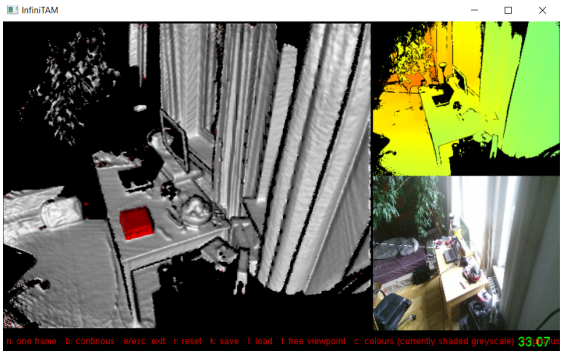

Figure 1.1: Example of a detected change

The automated detection of changes in a 3D space can be a useful tool. [PCBS16] names 3D surface reconstruction, environment monitoring, natural events management, and forensic science as possible application scenarios. In this work, we introduce software that scans an area at two different points in time and detects the changes between these scans. The software is based on InfiniTAM [PKG+17], a framework released under an Oxford University License. InfiniTAM integrates multiple depth images (e.g. recorded with a Kinect-V2-Camera) to a 3D model using volumetric representations. Because of the volumetric representation and the fast GPU computation, the change detection can happen in real-time. This is outstanding because in other approaches, (like [PCBS16]) the change detection can take minutes. Other approaches that detect changes in real-time (like [KMK+19]) use the same representation of data (T-SDF) as we do. Our approach also takes sensor tolerance into account, which leads to a reduction of false change detections. This work can be seen as a starting point for more specific use cases (like [LTW+21]) who specify on scene change detection and overcoming unnecessary changes such as light and seasons.

CHAPTER 2

Background

2.1 InfiniTAM

This work is based on the InfiniTAM [PKG+17] framework, a framework that integrates depth images into a 3D surface data structure in real-time. In this chapter, we will give you a brief overview of all components of InfiniTAM, we used to implement the change detection.

2.1.1 Representation and Storage of Data

Generally, the geometry is represented as a surface by a signed distance function (SDF). The SDF maps a point in space to the distance from the nearest point on the surface. The sign of a SDF determines if a point lies behind or in front of the surface. The data is saved volumetrically so that for every point on the grid there is a SDF-value. To save memory, the framework uses a truncated SDF (T-SDF). That means that only voxels with an absolute SDF-value less than a certain μ value are saved. That area is called truncation band. Because of the truncation, memory must only be allocated for voxels clos to the surface. The voxels are organized in 8x8x8-voxel blocks that store 512 voxels. These voxel blocks are stored in a hash list.

2.1.2 Three Coordinate Spaces

In this work, we will often talk about different spaces. Here I want to give an overview to make the following chapters more understandable.

The Image space

The image space refers to a two-dimensional picture. That can be a depth frame recorded by a camera, but it can also be an output frame created by the rendering engine.

The World space

The world space is the three-dimensional space of the real world.

The Model space

Because the voxel blocks are stored in a hash list, they need to be indexed with integers. Therefore, it is useful to have a space where a voxel block has a length of 1. That can be achieved by dividing the world coordinates by the size of a voxel block. Analogously the trnsformation from model space to world space is simply a multiplication by the voxel block size.

2.1.3 Integration of one Frame Into the Scene

The processing of each depth frame consists of three stages: Tracking, Fusion, and Rendering. In the Tracking stage, the depths-frame is aligned to the scene, in the Fusion stage, the depths frame is integrated into the scene and in the Rendering stage, the scene is rendered.

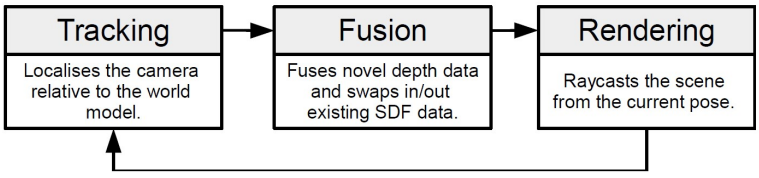

Figure 2.1: The three processing stages, *source: [PKG+17]*

In this section, we will take a brief look at the Fusion stage, because it is one part of the framework that is important for change detection. The first step is to add all voxels captured by the camera to the so-called visible list for further processing. Therefore, there is an iteration over all pixels of the current depth image. In one iteration step, the current pixel is projected from the image space into the model space. That point is called P. In the next step, is an iteration over all points on the line between the camera point and P that lie inside the truncation band. In each step of this inner iteration, the index of the voxel block that includes the current point is saved sin the visible-list. When these iterations are done, the next step is an iteration over all voxels in the visible-list.

In this step, the SDF values of all voxels, captured in the current depth frame, will be updated. Firstly, there is an iteration over all voxels in the visible-list. Then the current voxel is projected from the model space into the image space of the current depth frame to retrieve the depth information. With this information, a surface point Q that is on the same line as the current voxel and the camera point can be calculated. The distance d between Q and the current voxel is the newly calculated SDF value for the voxel. Also, a counter C is incremented every time, the SDF value of a voxel is updated. C is initially set to 0. If the voxel does not have an SDF value jet, the new value is the distance d. If a voxel already has an SDF value, the new value is the average of the previous values and d (see 2.1).

$$\frac{d + SDF_{old} \cdot C}{C + 1} \tag{2.1}$$

2.1.4 Rendering of the Scene

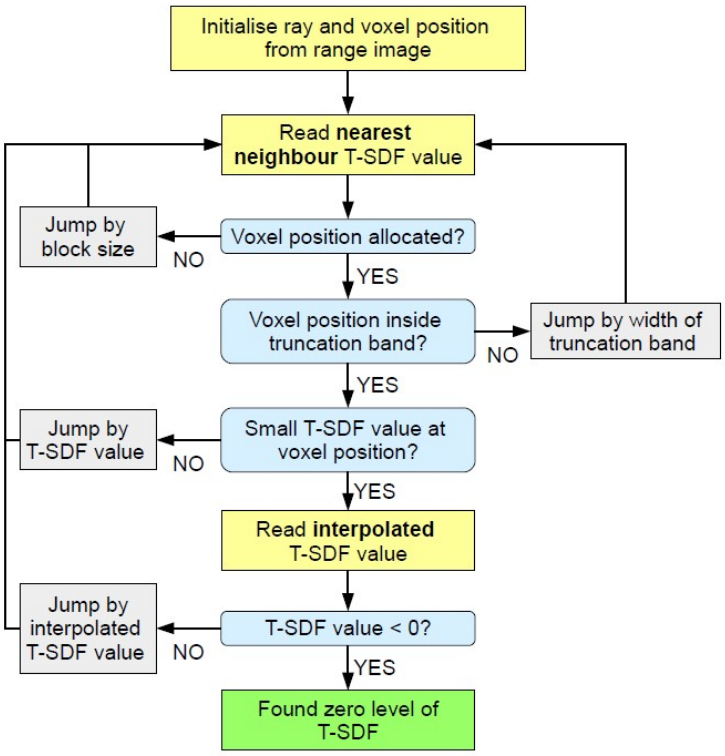

Figure 2.2: Flowchart of the raycasting process, *source: [PKG+17]*

Because the detected changes must be visualized to the user, the Rendering stage is also very important for this work. Generally, the output image is rendered via ray-casts. The first step of this process is an iteration of all pixels of the output image. For each pixel, the corresponding point on the surface must be found. That is realized by jumping on the ray of sight in different step-sizes, depending on the current SDF-value (see Figure 2.2). As long as there is no allocated voxel, the jump size is set to the block size. If the iteration reache the inside of the truncation band, the jump-size will be set to μ, and if the value is less than a certain threshold, the jump distance will be set to the SDF value of the current voxel. When the SDF-value of the current ray point is negative, the surface is reached. Now the grayscale value of the pixel is calculated by approximating a surface normal for that point considering the SDF-values of the adjacent points.

2.2 KinectV2

Figure 2.3: Kinect v2, *source: [MS]*

Although infiniTAM can use various sources as input, we optimized our work for the KinectV2 camera (figure 2.3). Kinect is a series of cameras with depths-sensors from Microsoft. The KinectV2 depth-sensor uses time-of-flight (TOF) technology. For each pixel, the sensor measures the time an artificial light (caused by the camera) takes to travel to a surface and back to the camera. That time is directly proportional to the distance between camera and surface.

In some scenarios, the TOF technology can produce incorrect results. In [SLK15] the most common sources of errors were listed:

Ambient Background Light:

Because the Kinect sensor works with infrared light, ambient background light can be a problem. This problem becomes more serious outdoors because sunlight contains more infrared light than most artificial light sources. To receive reliable depth data the ambient light must be filtered.

Multi-Device Interference:

Similar to the problem of Ambient light is the interference of multiple devices. Often it can be useful to scan an area with more than one device. The problem is that all devices emit infrared light for depth detection and so disturb each other's measurements. A theoretical solution to this problem would be different modulations for each unit.

Reflecting Surfaces:

Multi-path effects of reflecting surfaces can generate enormous errors in depth measurements. Despite highly reflecting surfaces like mirrors also sharp angles can lead to multi-path effects.

Systematic Errors:

Systematic errors are either introduced by the calibration of the sensor or effects of approximations in the measurement pipeline.

Multi-Path-Effects

Multi-path effects of reflecting surfaces can generate enormous errors in depth measurements. Despite highly reflecting surfaces like mirrors also sharp angles can lead to multi-path effects.

2.3 Noise Models

In their bachelor thesis [Gro16] and [Kö17] examined systematic errors introduced by the Kinect sensor. Their error model considers two types of noise: Axial noise and lateral noise.

Axial noise

The axial noise describes measurement errors along the ray of sight. The error is the difference between the measured depth and the actual depth. In general [Gro16] and [Kö17] measured axial noise by placing a plane in a certain distance and angle to the sensor, then measuring the depth-values and extracting them to a point cloud. Finally, they measured the difference between the measured points and the ground truth model (calculated by manual measurements).

Lateral Noise

The lateral noise describes how much a point is shifted in the directions perpendicular to the ray of sight compared to the real-world object. To measure the lateral noise a plane is positioned perpendicular to the ray of sight. Then the measured border points are compared with the ground truth (again calculated by manual measurements).

As a result of the work, several noise models are introduced so that you can calculate the maximal noise for a measured point. This is the noise model we used for our work

$$\sigma_{z_1}(z,\theta) = a + b * z + c * z^2 + d * z^e * \frac{\theta}{(\frac{\pi}{2} - \theta)^2} \tag{2.2}$$

where θ is the rotation of the pain and z is the distance to the plain. The coefficients a to e are printed in the table 2.1

a	2.094
b	$-1.099 * 10^-3$
c	$4.048 * 10^-7$
d	$6.846 * 10^-7$
e	1.7

Table 2.1: coefficients of the axial noise model *source: [Gro16]*

This model returns the axial noise value for a distance z and an angle θ. *source [Gro16]*

CHAPTER 3

Method

Our contribution to the InfiniTAM [PKG+17] framework is the feature of change detection. That means that a scene can be scanned two times, and all changes are displayed in the user interface. For the realization of this feature, three major changes were introduced to the framework: the introduction of a second scene (3.1), the introduction of sensor tolerance values (3.2) and a new rendering algorithm (3.3).

3.1 Introduction of a Second Scene

The hash table containing the voxel blocks is located in a so-called scene object. To scan an area at two points in time, the system needs to hold two scene objects. So, we added a second scene instance to the system. While scanning an area the user can choose the scene that the depth images will be integrated into via hotkey. We take the currently inactive scene as a reference so that the two scenes are aligned. It is also possible to save a recorded scene if the user wants to detect changes at a later point in time. A consequence of the additional scene object is that the number of voxels a scene can store is cut in half.

3.2 Integration of Sensor Noise Extent

Systematic errors produced by the Kinect sensor can lead to problems in the change detection. That can happen because the rendering algorithm (see 2.3) checks if the difference of the SDF value of the same voxel in the two different scenes exceeds a certain value. Inaccuracies of the SDF values can therefore lead to falsely detected changes. We used the noise model of [Gro16] and [Kö17] to avoid this problem (see 2.2). The noise function takes the camera distance z and the angle θ between the ray of sight and surface as input and returns the noise. Because of lack of data, we assume the angle to be 90° so we only use the distance d as an input parameter.

To add noise values to the system, we altered the integration of frames into the scenes. When iterating over the voxels in the visible list, in each step the distance d between the current voxel and a surface point Q is calculated (see Section 2.1.3). Next, we determine the noise n that may have been introduced by the camera using the distance z. We added a new float value 'noise' to the voxel datatype. If n is greater than the old noise value of the current voxel (or the current voxel has no noise value), n becomes the new noise value. Otherwise, we keep the old value. We always want to work with the greatest noise occurrences when detecting changes. Otherwise, the inaccuracy of the SDF value could be calculated as too low which could lead to incorrect change detection.

3.3 Rendering Algorithm

We implemented the actual change detection in the new rendering algorithm. It is very similar to the original InfiniTAM-rendering-algorithm discussed in Section 2.1.4. The challenges are that two scenes need to be rendered and that the differences between these two scenes must be detected and visualized.

The basic idea is to cast a ray against two surfaces instead of one. We remember which surface is hit first by the ray. That is the surface that will be rendered. If the other surface is very near we do not assume that there is a change. Otherwise, we do. With this approach, we can do both, rendering and change detection at once. However, only changes within the current view will be detected.

Like in Section 2.1.4 the first step is an iteration over all pixels of the output image. In each iteration step, a ray is cast from the camera position along the ray of sight. As long as the voxel on the current position is not allocated, we jump per block size. When a voxel is allocated in at least one of the two scenes, we know that we have reached the truncation band. In that case, we jump by the width of the truncation band to get as close to the surface as possible. Now as we have small SDF values, we jump by the amount of the smaller SDF value.. If the SDF value of one scene falls below 0 we have found the first surface point of the two scenes that is hit by the ray. (see Figure 3.2) In that case, we save which scene had the negative SDF value. Now we must determine if this point is close enough to the surface of the other scene or if the difference is big enough to consider this pixel as changed.

For this task, we examine the SDF Value of the scene that was not hit by the ray. If the difference between the two SDF values falls below a certain threshold, we assume that there was no change. Otherwise, we define the pixel as changed. If the systematic error of the camera (discussed in 2.2) was the only source of inaccuracy, we could just use the noise value saved in the voxels as a threshold. But experiments with that solution have shown that that approach leads too many falsely detected changes. That suggests that there is another source of inaccuracy in addition to the errors of the camera. These inaccuracies are most likely introduced in the integration of a frame into the scenes (see 2.1.3).

To avoid these falsely detected changes we introduce a fixed threshold value (t) additionally to the noise value (n). The final theshold is calculated by adding the noise value to the fixed threshold. If we would only use one fixed threshold value we could not guarantee that the final threshold is as small as possible, which could lead to not detected changes. We evaluated the perfect value for the fixed threshold by performing different experiments. The results showed that a too-small value does not solve the problem as to many falsely detected changes remain in the rendered image (see Figure 3.2). A too-great value on the other hand introduces another problem. The greater the threshold, the more false negatives will be produced by the algorithm (see Figure 3.3). So we have to find a compromise. Our experiments have shown that 3,5mm is a good value for the fixed threshold. We cannot prevent false negatives with this method but we can reduce the occurrences drastically by choosing a small enough number as the fixed threshold.

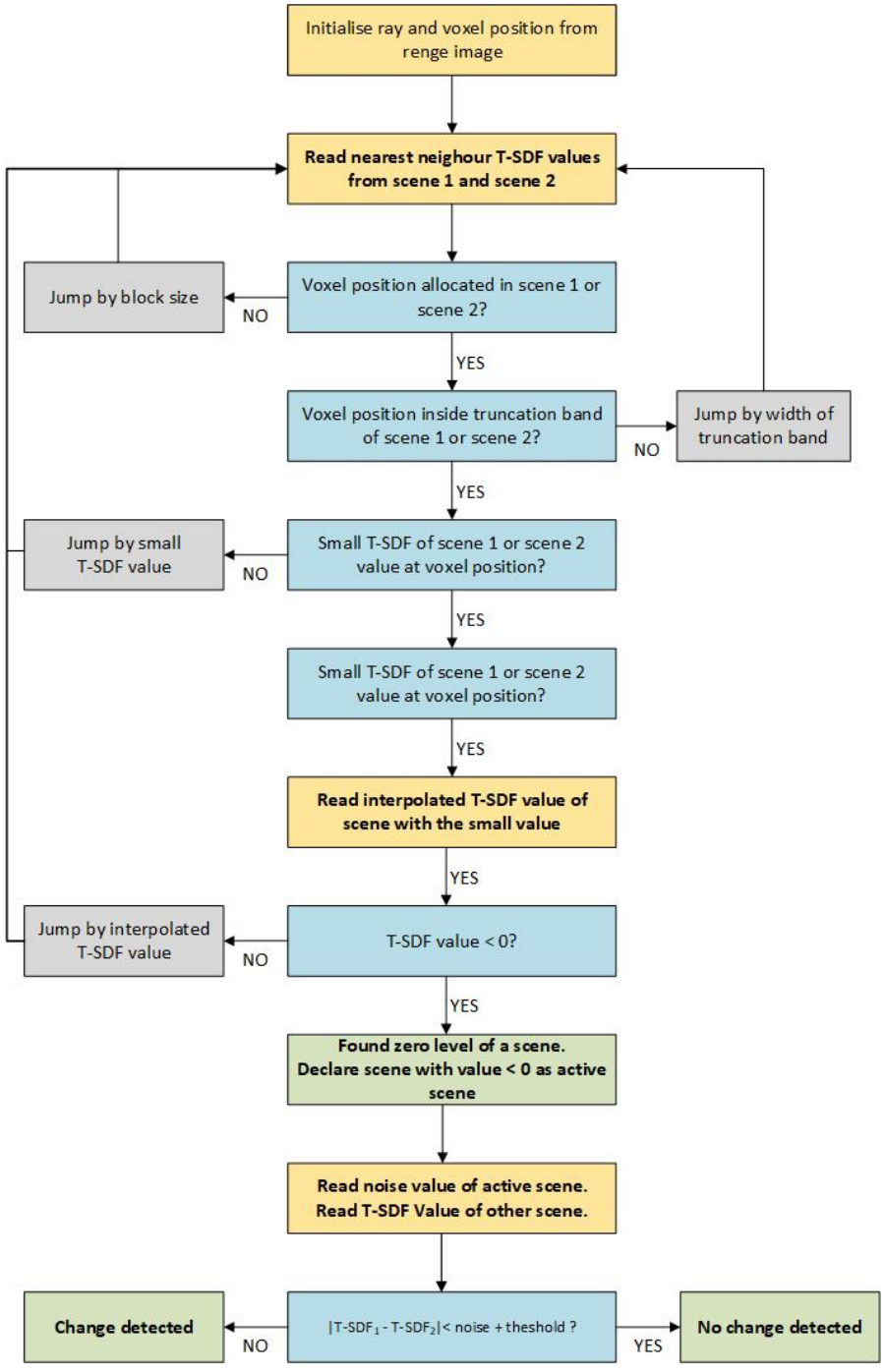

Figure 3.1: Flowchart of the change detection algorithm *adapted from: [PKG+17]*

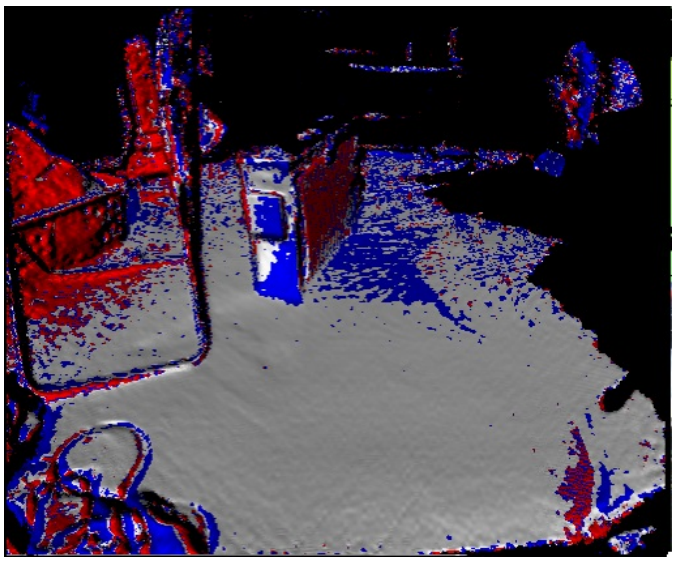

Figure 3.2: too low threshold-value, many false positives

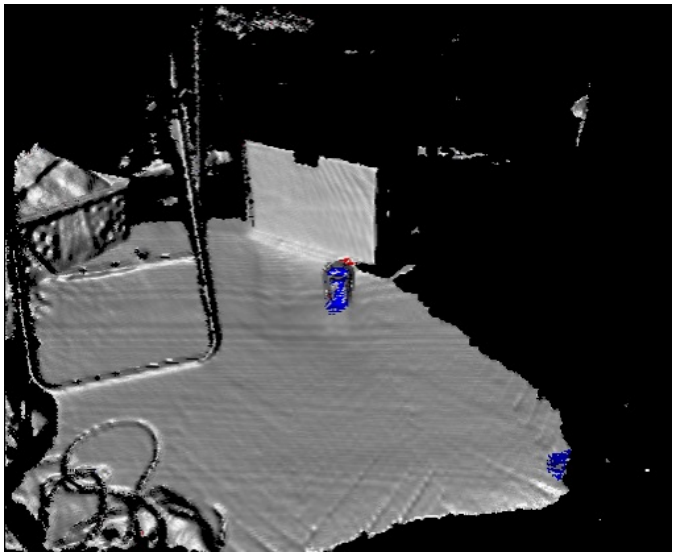

Figure 3.3: too great threshold-value, many false negatives

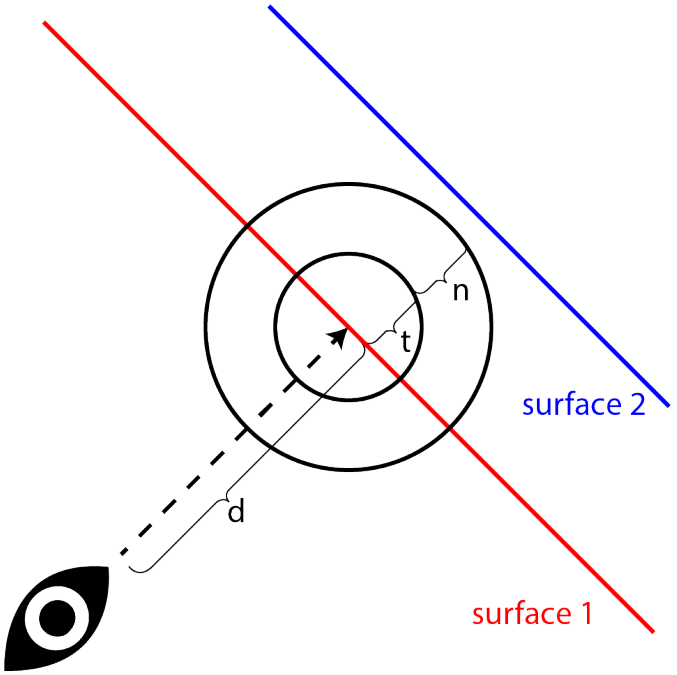

Figure 3.4: example of a detected change; d: distance, t: threshold, n: noise

CHAPTER 4

Results, Evaluation and Limitations

In this chapter, we will discuss the tool that emerged from our studies. First, we will take a look at the user interface, then we will discuss the strengths and weaknesses.

4.1 User Interface

The user interface of the change detection tool extends the user interface of InfiniTAM. The main window is divided into three sections (see figure 4.1). If there is a detected change the surface is not visualized in grayscale but in red or blue, depending which scene is hit first by a ray in the rendering algorithm.

The tool can be used with the three hotkeys n, b, and d. By pressing the n-key the user can record a new depth frame, integrate it into the scene and trigger a rerender of the updated scene. The b-key activates the continuous mode so that depth images will be integrated into the scene continuously in real-time. The user stops the continuous mode by pressing the n-key. These features were all included in the original InfiniTAM [PKG+17] framework. As mentioned in Section 3.1 we introduced a second scene for change detection. To switch between scenes, the user can press the d-key.

4.2 Qualitative Analysis

First, we want to give an outline of basic conditions for the InfiniTAM [PKG+17] framework to work properly. Then we will look at conditions explicitly for change detection.

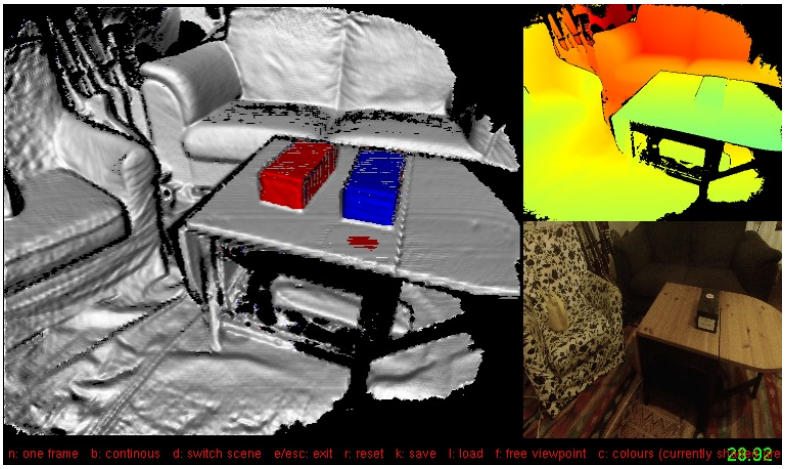

Figure 4.1: Top right image: the current depth image. (coler-coded values from green to red); Bottom right image: the current image of the color camera; Left image: the scenes with the detected changes.

4.2.1 Requirements for a Good InfiniTAM Scan

For the reconstruction, the scanned surface must not have many small or thin features. Examples of such complex surfaces could be hair, plants, and small wires. What is also important is that there need to be features on the surface that the tracking engine of InfiniTAM [PKG+17] can refer to. Featureless surfaces like a flat wall will most likely not be scanned correctly.

4.2.2 Areas That Were not Scanned

One problem with the change detection method we use is that it cannot distinguish between voxels that are not initialized because they have never been scanned and voxels that are not initialized because they are far away from the surface. This can lead to scans like 4.2.

This example was created by scanning the same surface twice but, one time a larger area. On the figure, you can see that the area that was only scanned once is marked as changed because the corresponding voxels are not initialized in the other scene. A more sophisticated solution would be to check for visibility and mark such voxels as "undefined" because we do not know if there was a change.

4.2.3 Noise Near the Changed pixel

Another issue with the program is, that if changes are detected, there might occasionally be small areas of false positives. (detected changes where there are no real changes). In figure 4.1 you can see a red spot right under the blue object and in 4.3 there is a red area around the blue marked guitar.

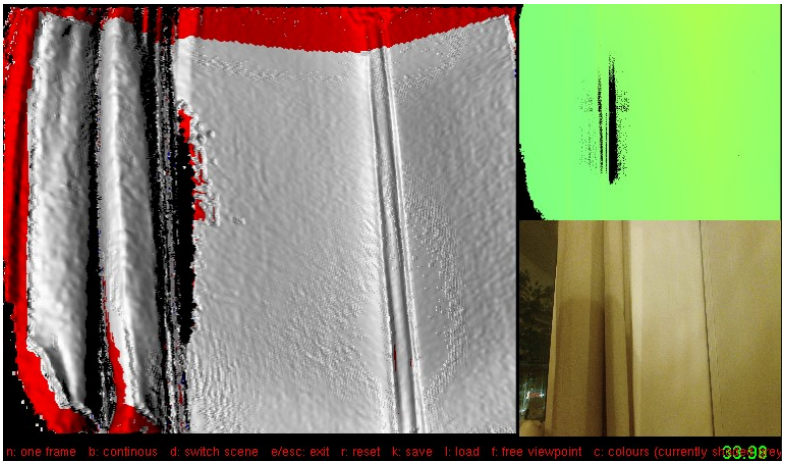

Figure 4.2: Unscanned areas marked as changed

4.2.4 Flaws of the Noise Calculation

For the calculation of sensor noise we used formula 2.2 taken from [Gro16] and [Kö17]. This choice leads to other limitations of the framework. This formula takes camera distance and an angle as input parameters. In these theses, there are other - more sophisticated - models to calculate the noise introduced by the Kinect sensor, which provide more accurate results. We decided to use this model because it is simple and the calculation is fast.

Also, we assumed the value of the angle θ in the formula to always be 90°. It could lead to more accurate results if the angle between the ray of sight and the surface was calculated using SDF values of voxels adjacent to the currently processed voxel. However, this could also lead to longer computation times and can be investigated in future research.

Figure 4.3: false positives: the red area around the blue marked guitar

4.3 Quantitative Analysis

4.3.1 Theoretical Analysis

A problem in our change detection algorithm is that changes smaller than the threshold are interpreted as noise. For theoretical analysis, we examined how big changes need to be so that thay are not interpreted as noise. As described in Section 3.2 we save a noise value within every voxel to take the sensor tolerance into account. These noise values are added to a fixed threshold. If changes are smaller than this sum, this can lead to not detected changes. Below you can see the sum of noise and threshold for a selection of distances.

distance(m)	noise + threshold (mm)
0.5	5.17
1	4.99
1.5	5.03
2	5.30
2.5	5.79
3	6.50
3.5	7.43
4	8.58
4.5	9.96

Table 4.1: sensor noise for different distances. Calculated using the findings of [Gro16] and [Kö17]

Sensor noise is something that cannot be corrected in later steps. Therefore, these values can be considered as a lower bound for the inaccuracy of the change detection.

4.3.2 Analysis by Simulation

To observe the real behavior of the tool concerning such small values, we used blensor [GKUP11] (an addon for blender) to simulate scans. In the scene, we created for that purpose (see 4.4) the distance between the camera and the surface was exactly one meter.

To find out how big a change needs to be so that the tool can identify it, we varied the distance between the back wall and the protruding square. For every distance, we recorded the scene two times. One time containing the protruding square, the other time without it. Then we imported the frames into the tool to observe if the change was detected. We found out that for one meter distance between the camera and the surface, the minimal change that can be detected must have a size of 17mm.

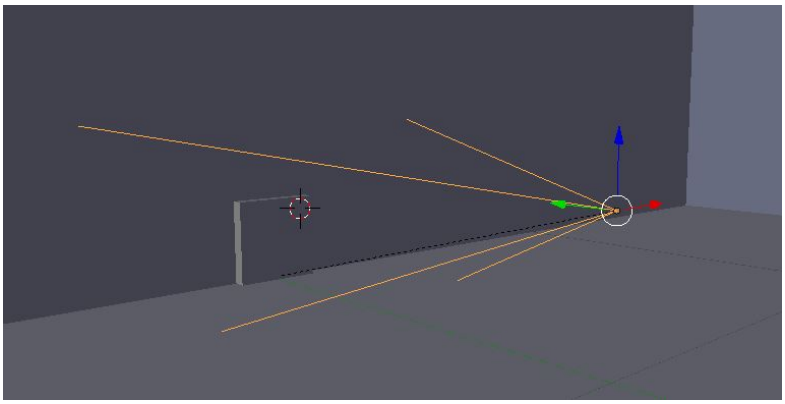

Figure 4.4: test scene created in blensor [GKUP11]

CHAPTER 5

Conclusion and Future Work

In this bachelor's thesis, we implemented the detection of changes in a volumetrical data structure. As a starting point, we used the InfiniTAM [PKG+17] framework. What distinguishes our tool from other solutions to this problem is that our solution can work in real-time. This makes our approach much more interesting for application in the real world when changes need to be found immediately (for example in safety-critical scenarios).

However, there are components of the tool that could be approved in the future. One problem is that the tool can not distinguish between unscanned and empty areas (see Section 4.2.2). This issue could be fixed by marking all scanned voxels - also the empty ones - as "scanned". This is not trivial to achieve because of the sparse way the volume is represented in. Therefore, saving the "scanned" information in the volume would erase the benefit of the way the surface is represented in. A way of fixing this issue could be to introduce a new data structure that is better suited for that purpose.Another problem is the noise of pixels that are marked as changed (see Section 4.2.3). A possible solution for this problem could be a noise-filtering method of the changed pixels in the output frame. This could also cause problems because real changes could be filtered too.One reason why our solution works so fast is that changes are detected during the rendering stage. This causes the problem that changes are only detected that are located within the view frustum and also depend on the camera point. An approach to fix that issue would be the introduction of a new change detection stage between the fusion stage and the rendering stage. Therefore another efficient change detection algorithm would be necessary.

List of Figures

Glossary

image space Two-dimensional coordinte system of a (depth-)image. 3, 4

model space Three-dimensional Coordinte system used for indexing. 3, 4

pixel Picture Element, One point in a (depth-)frame. 3, 5, 9, 15, 19

scene Scene Object, Object where the surface is stored. vii, 3, 4, 8, 9, 14, 15, 17, 18, 20

truncation band all points that have less distance to the surface than μ. 2, 3, 5, 9

voxel Volume Element, One point on the volume grid. 2–5, 8, 9, 15–17, 19

voxel block a three-dimensional block of 8x8x8 voxels. 2, 3, 8

world space Three-dimensional Coordinte system of the real world. 3

Acronyms

SDF signed distance function. 2, 4, 5, 8, 9, 16

TOF time-of-flight. 5

Bibliography

[GKUP11] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. Blensor: Blender sensor simulation toolbox. In *International Symposium on Visual Computing*, pages 199–208. Springer, 2011.

[Gro16] Nicolas Grossmann. Extracting sensor specific noise models. 2016.

[Kö17] Thomas Köppel. Extracting noise models – considering x / y and z noise. 2017.

[KMK+19] Ukyo Katsura, Kohei Matsumoto, Akihiro Kawamura, Tomohide Ishigami, Tsukasa Okada, and Ryo Kurazume. Spatial change detection using voxel classification by normal distributions transform. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2953–2959. IEEE, 2019.

[LTW+21] Jianjun Li, Peiqi Tang, Yong Wu, Mian Pan, Zheng Tang, and Guobao Hui. Scene change detection: semantic and depth information. *Multimedia Tools and Applications*, pages 1–19, 2021.

[MS] Mircrosoft web page. `https://news.microsoft.com/kinect-for-windows-v2-sensor-2/`. Accessed: 2022-01-06.

[PCBS16] Gianpaolo Palma, Paolo Cignoni, Tamy Boubekeur, and Roberto Scopigno. Detection of geometric temporal changes in point clouds. In *Computer Graphics Forum*, volume 35, pages 33–45. Wiley Online Library, 2016.

[PKG+17] Victor Adrian Prisacariu, Olaf Kähler, Stuart Golodetz, Michael Sapienza, Tommaso Cavallari, Philip HS Torr, and David W Murray. Infinitam v3: A framework for large-scale 3d reconstruction with loop closure. *arXiv preprint arXiv:1708.00783*, 2017.

[SLK15] Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb. Kinect range sensing: Structured-light versus time-of-flight kinect. *Computer vision and image understanding*, 139:1–20, 2015.