DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

A GPU Based Approach for As-Planned versus As-Built Comparison in the Scope of Automated BIM Based Construction Progress Monitoring

Helge Hecht

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

A GPU Based Approach for As-Planned versus As-Built Comparison in the Scope of Automated BIM Based Construction Progress Monitoring

Ein GPU-basierter Ansatz für den Soll-Ist-Abgleich im Rahmen der automatisierten BIM-basierten Baufortschrittskontrolle

Author:	Helge Hecht
Supervisor:	Prof. Dr. Rüdiger Westermann
Advisor:	Alexander Braun, Msc.
	Julian Amann, Msc.
Submission Date:	15. September 2016

I confirm that this bachelor's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, 15. September 2016 Helge Hecht

Abstract

The delay of construction projects and the resulting cost increase is a common issue in civil engineering projects. These difficulties are often related to poor site supervision and project performance management [66]. The advances in computer science and 3D geometry scanning allow the construction of a dense point cloud which can be used to automatically identify the presence and absence of objects on the construction site. The automation of construction site progress monitoring is still subject to research and includes time-consuming computations. These problems include the alignment of two data sets in a common coordinate system and object recognition which both offer great acceleration potential through parallel data processing [8]. The automatic progress detection pipeline is analysed and an approach for the application of graphics processing units (GPU) to accelerate the processes of rigid point cloud alignment and building component classification is proposed. Implementations based on NVIDIA Cuda originally proposed by Tamaki et al. [58] for point cloud alignment and a new method for progress detection implemented with Microsoft's C++ AMP are compared with current automated progress tracking procedures. The evaluations are performed using data provided by Braun et al. [13] which was acquired on two different construction sites in Munich.

Contents

1 Introduction

Recent trends in the Architecture-Engineering-Construction (AEC) industry are mostly related to the application of Building Information Models and computer science for the construction process. These include methods towards automatic detection and evaluation of progress since manually monitoring a construction site is time-consuming and error-prone [1, 7]. The degree of automation varies from fully automated systems to small gadgets available for on-site personnel [43]. State of the art fully automated progress tracking methods rely on Building Information Models and 3D point clouds to compare the as-planned and as-built state of a construction site since the access to accurate data about on-site events and progress is crucial for the effective and successful execution of a construction engineering project [12, 51].

This thesis analyses the needs and advantages of automated progress tracking and inducts into the technologies deployed in current state of the art methods. The steps and processes ranging from point cloud alignment to building component classification are described. The computations required for automatic progress detection are still time-consuming and can take up to several hours [22]. The element-wise execution of operations on large point clouds possibly containing multiple millions of points consume most of the time required for accurate progress detection [8]. A new approach for the acceleration of fully automated progress tracking systems using robust point set registration algorithms implemented for parallel hardware instead of traditional sequential algorithms is proposed. The NVIDIA Cuda based implementations of two registration algorithms originally proposed by Tamaki et al. [58] are compared with sequential implementations of the classic Iterative Closest Point algorithm developed by Besl et al. [6] and another registration algorithm formulated by Jian et al. [29]. The actual progress detection procedure is implemented using Microsoft's C++ Accelerated Massive Parallelism, a framework for parallel programming in a native C++ environment. The proposed algorithms are tested with point clouds and Building Information Models provided by Braun et al. [13].

2 BIM Based Automated Progress Tracking

2.1 Building Information Modelling

A Building Information Model (BIM) is the digital representation of a real building or civil engineering project. This digital representation includes a three dimensional geometric description of the individual building components as well as further information addressing the whole lifecycle of the building [7, 47]. They are therefore often considered 4D models. The advantage of these digital models is the reusability by all parties involved with constructing, maintaining or destructing the building since the processes of recreating this data are very time-consuming and susceptible to mistakes. The BIM serves as the central knowledge and communication base with real time accessibility for everything related to the building. The data contained in a BIM which is relevant towards tracking the progress during the construction process includes the work breakdown schedule, the dependencies between individual building components as well as details about costs or materials associated with them.

2.2 Automated Progress Tracking

The availability of an up-to-date knowledge base about the progress and the performance of a construction site is an important part of quality management. Nonconformity of performed work or of the working process itself with the schedule can lead to variations regarding the desired quality and the completion of the project in time. It is important to address problems occurring during the construction process before the completion of the task since the resulting delay and the correcting rework can be more expensive than on-site quality control [39, 51, 66]. Insufficient supervision of a construction site is considered the primary or secondary cause of delay in civil engineering projects in Hong Kong and Japan respectively [2, 3, 42] but it is, in fact, a problem of global scale since reports of failures in civil infrastructure projects can be found in most industrialized countries [17, 44, 66]. The reason why a lot of money is still spent on reworking instead of better on-site quality control is the tedious nature of manual data collection and evaluation. Additionally, this work is not only labour intensive but the data provided is also unreliable since it depends on the skills of a

certain individual and the overall competence of the management staff [1, 39, 52]. The procedure of measuring project performance is described by Al-Jibouri [30] as a control cycle with the following stages:

1. Make a plan.
2. Implement the plan.
3. Monitor actual output and record it.
4. Report actual planned parameters and their variations.
5. Take action.

The stages 3 to 5 are also stated by Leung et al. [39] as "input"(3), "process"(4) and "output"(5). A system which automatically reports the daily progress should therefore repeatedly capture as-built data on the construction site as input, compare it with the as-planned schedule as process and then report critical variations from the schedule to the staff as output [24, 46]. The experimental simulation of a highway construction site conducted by [1] shows that the time spent on work not contributing to actual progress can be reduced with automatic performance tracking systems.

2.2.1 State of the Art Methods

The technologies used for automatic data collection have improved over the last decades and are applicable for construction site monitoring. The most common techniques rely on the use of Radio Frequency Identification (RFID), Global Positioning Systems (GPS), Video Detection and the use of laser scanners for Laser Detection and Ranging (LADAR) [15, 39, 46]. The current state of the art methods differ regarding their degree of automation and the focus on data collection, processing or evaluation as well as their integration into existing systems. There exist a variety of semi-automatic systems utilizing mobile devices to manually collect data regarding the project performance and to provide easy access to the project schedule as well as a collaboration platform for all on-site personnel. The systems deployed by [36, 43, 52] allow easy access and sharing of CAD model data while the two latter also provide real-time video footage of a central CCTV camera. Even though this reduces the time spent on communications regarding changes or deviations from the as-planned schedule, the degree of automation has to be considered low, since only the software developed by [43] automatically evaluates the progress state from the manually collected input data. This degree of automation is limited due to the use of a mobile device and the lack of an automatic data collection device or software [8, 34].

The automatic detection of the as-built state using remote sensing technology to replace the manual data collection requires a more complex set of hardware and software on the construction site as well as in the main office. Besides this drawback,

these systems are capable of automatically processing and evaluating the collected data and are thus better suited for a fully automated progress tracking system [11, 22, 34]. The data collection procedure is primarily vision based and is performed with LADAR or traditional video camera systems as remote sensing technology. The approaches made by Bosché [8], the CMS and PF department of the TUM in Braun et al. [11] and the Department of Architectural Engineering of the Chung-Aug University in Seoul in C. Kim et al. [33] involve the detection of the as-built state of a construction site with laser scanned point clouds. The procedure described by Golparvar-Fard et al. [21] is based on images captured with traditional video cameras on construction sites since laser scanning devices are quite expensive and have restricted mobility [1, 8]. The captured video footage or imagery is then used to calculate an as-built point cloud. These point clouds are then matched against the underlying 3D CAD models to confirm the existence of the as-planned objects in the as-built collected data. The details of this procedure will be described further in chapter 3.

2.2.2 Performance regarding efficiency and computation time

Determining the performance of the semi-automatic systems is difficult since it depends on several individual factors. First of all, it depends on the acceptance of the new system by the on-site personnel. The quality of collected data is unlikely to improve if the staff perceives the changes made as adverse [43]. Second and probably most important is the fact that the data collection progress is still performed manual and is, therefore, depending on the skills of the supervisor. The change of platform from time cards or other form sheets to a mobile computing device won't improve the input data from an unskilled project manager. It is difficult to measure the performance of a system that is directly related to a factor that is unknown to a certain degree. In addition, deploying a new framework requires the integration into existing processes, thus leading to additional expenditures and risks. The accessibility of required schedules and CAD models is also often restricted due to legal issues since multiple parties might be involved in the project [43].

The case studies described in [15, 36, 39, 43, 52] all report positive results but lack to provide details like saved costs or a reduction of time spent. The approach made in [1] is to simulate a real-world highway construction site and explore the effects of using RFID and LADAR technology on the time spent by 11 foremen, 3 project engineers and 1 timekeeper with non-value adding activities. The data required for the simulation was collected in interviews and on-site examinations. The experiment states a reduction of 43% and 14.5% using laser scanners or RFID devices respectively.

A performance analysis of fully automated systems includes the performance regarding the recognition and evaluation of progress from collected data and the computa-

tional performance [9]. The detection of progress is based on object detection since the point cloud data is matched against the 3D CAD model to confirm the existence of the CAD model objects in the scan. The object detection performance is commonly described by the recall rate (true positives), the specificity rate (true negatives) and the precision rate [9]. The progress of a construction site in a time interval can then be calculated from the comparison of the recognized as-built state with the as-planned state for the start and the end of the interval [10]. The performance of current state of the art methods for automated progress tracking described in [8, 22, 34, 60, 62, 64] is depicted in table 2.1.

	Method	Points	Images	Objects	Accuracy	Time(min)
[8]	Laser Scan	650,941	-	612	93%	35
[60]	Laser Scan	$\sim 1,200,000$	-	1573	98%	-
[34]	Laser Scan	-	-	-	99%	-
[22]	Photographs	61,638	288	321	91%	"a few hours"
[62]	Laser Scan	$\sim 25,000,000$	80	14	43%	-
[64]	Laser Scan	213,455,636	-	463	78%	148

Table 2.1: Performance of state of the art progress tracking methods.

The accuracy rates are not consistently made up of the recall, specificity and the precision rate but all approaches state an overall good performance regarding progress detection. One problem faced by most automatic systems is the occlusion of objects and the resulting errors when determining the progress of only partly visible CAD objects. The presence of occlusions in photogrammetric point clouds could be reduced by using UAVs or simply taking pictures from the inside of the construction site [22, 63]. Better visibility coverage with LADAR systems can be easily achieved by scanning from different spots and combining the point clouds [38].

The downside of these methods can be seen in the required computation time. Some authors state computation times of over 2 hours while processing scans with sizes in the magnitude of 10^6. Point clouds with multiple millions of points are common when using laser scanners and the point density of current high-tech devices can be even higher[9, 28, 48]. Most time is spent on the registration of the as-planned and the as-built point cloud in a common coordinate system where the computational effort is directly related to the number of scanned points [8]. The size of photogrammetric point clouds depends on the recognition method and the collected data. The structure from motion procedure used in [22] recognized 61,638 points from 288 images while C. Kim et al. [35] recognized 2,891,171 points from 12 images with a proprietary software system. The time required to construct a photogrammetric point cloud is not included

in table 2.1, since it is not directly related to the progress detection procedure, but it can make quite a difference. The results presented in [26] include computation times of about 36 hours for 36 images with a resolution of 136 mega pixels.

2.3 Parallel Acceleration of Automated Progress Tracking

The reduction of occlusion to increase the progress detection performance is always coupled with a significant increase in computational effort. The combination of point clouds from multiple scans requires the registration of the two point sets and results in a larger cloud which has to be registered with the as-planned point cloud afterwards. The registration of photogrammetric point clouds is faster since they are usually not as dense, but the time spent on point cloud recognition is related to the amount of collected data. The recognition of a complete point cloud from more on-site images is, therefore, more time consuming. The amount of data which has to be processed has the most significant impact on the computational effort in both cases. The alignment of two point clouds requires the registration of both into a common coordinate system. The iterative closest point (ICP) algorithm formulated by Besl and McKay in [6] is used in the approaches made by [8, 22, 34, 60, 63] for the fine registration process. The algorithm includes the calculation of the distance between each point $\overrightarrow{p_i}$ from the as-built point set P to a closest point $\overrightarrow{y}$ in the as-planned set X. The computational complexity is therefore $\mathcal{O}(N_p N_x)$ in the worst case where N_p denotes the size of the as-built point cloud and N_x the size of the as-planned point cloud [6].

A possible improvement for the registration of laser scanned point clouds and the matching of the as-planned and the as-built point clouds afterwards is the implementation of the registration and the matching algorithm on a graphics processing unit (GPU) [8, 64]. This is also applicable for the recognition of photogrammetric point clouds as shown in [26] but won't be discussed further in this thesis. The ICP algorithm used in [8, 22, 34, 63] as well as other algorithms for point set registration are well suited for a parallel implementation on the GPU since most computations consist of vector and matrix multiplications and operations performed for each element or element pair [50, 58]. The actual task of finding the optimal transformation between two point sets using Horn's method described in [27] is also based purely on calculating sums of vector-matrix products. The single instruction multiple data (SIMD) architecture of a modern GPU enables data processing with more than $10,000$ parallel working threads with up to 1.5 teraflops in total [37, Chapter 1]. There exist a variety of GPU implementations of ICP and other registration algorithms based on gaussian mixture models (GMM) which provide massive increases in performance without losing accuracy [18, 19, 50, 58]. The actual progress detection procedure is most often based on arithmetic

calculations like the euclidean distance between the scanned points and the model surface or the area covered by each point [8, 13]. Alternative methods include object recognition algorithms or the subdivision of the scene in discrete voxel cells which can then be marked as free or occupied [22, 34, 63]. Support vector machine (SVM) based object recognition algorithms benefit greatly from increased computation power provided by a GPU and recent studies state massive performance increases [40, 59].

The applicability of these parallel accelerated algorithms for automated progress tracking is not yet determined and is subject to research in this thesis. Experimental results are presented in 4.2.

3 Automated Progress Tracking Pipeline

The availability of an as-planned BIM with an integrated work breakdown schedule is a prerequisite for automatic progress detection since the as-built state has to be matched with the schedule [8]. The next step is to generate a geometric representation of the as planned BIM since the level of detail regarding geometric information in a BIM is too low. The stereolithography (STL) file format contains a triangle representation of the building elements contained in the BIM and is used for the calculation of the as-planned point cloud [8, 33].

The automated progress tracking pipeline includes the following stages: (1) Creation of the as-built point cloud; (2) Creation of the as-planned point cloud; (3) Registration of the as-planned and as-built point cloud in a common coordinate system; (4) Recognition of as-built cad model objects; and (5) Progress evaluation.

Stages (2)-(4) include the data processing and are most complex and time-consuming regarding the computation. They are described in detail further in this chapter. Stage (1) and (5) are the input and output stages respectively. The acquisition of as-built point cloud data should be performed on a daily basis to detect mistakes made in the construction progress as soon as possible and to provide a high-quality daily site report (DSR) [46]. It is a complex procedure on its own and will therefore not be addressed in this thesis. An overview of laser scanning technology and photogrammetry, as well as a comparison of both techniques, can be found in [25, 32, 41, 53]. The effective evaluation of actual progress in stage (5) falls within the scope of construction management since the computational effort for the few required arithmetic calculations is not relevant. The progress between two measurements can be calculated with regard to time or budget spent or as building volume or surface gained. A BIM-based approach for earned value (EV) tracking and experimental results are proposed by Turkan et al. [61].

3.1 Creation of an as-planned Model Point Cloud

3.1.1 Sampling

The second stage of the automated progress tracking pipeline is the creation of a model point cloud. The as-planned geometry information contained in the STL file is used to generate the as-planned point cloud. The vertices which define the triangular facets

can be used directly but result in a very sparse model point cloud in comparison to the collected as-built data. Since high differences regarding the resolution of point clouds can lead to errors in the registration process, more points have to be generated [35]. The easiest way is to sample points on the triangle facets with a uniform distance δ between each point. The resolution of the resulting cloud is inversely proportional to δ . This uniform sampling procedure can also be randomized to better simulate real data.

A complex sampling process which reduces the as-built point cloud and creates a point cloud with a uniform grid structure is proposed by C. Kim et al. [33]. The bounding cube containing the as-built point cloud with size $\delta_0 \times \delta_0 \times \delta_0$ is used to build an octree. Every non-leaf node is split up into eight subcubes of size $\delta_k \times \delta_k \times \delta_k$ containing N_p points in each iteration. A cube becomes a leaf node if $N_p \leq N_{min}$ where N_{min} is a predefined threshold or has a predefined minimum size of $\delta_{min} \times \delta_{min} \times \delta_{min}$. The size δ_{k+1} of a node in the next iteration is calculated as: $\delta_{k+1} = \delta_k \div 2$. Once all nodes are leaf nodes, the center points of all leaf nodes are taken as the new point cloud. This octree based method can be used to compute an octree on the as-built data in order to generate a uniform point cloud. An octree with the same size threshold δ_{min} can then be computed on the triangle as-planned model. Since the triangle model contains no points, a node is considered a leaf node once it does not intersect with a triangle.

Uniform point sampling can be performed by a modern CPU with high point density in less than 1 second. The optimization of this procedure is therefore not necessary. The computation of an octree with $5,622,662$ cells on a point cloud with $5,622,787$ points can be done in 1.977 seconds. Both sampling processes have been performed on an Intel ® i5-337U CPU @ 1.8 GHz using the CloudCompare library.

3.1.2 Ray Casting

A different approach is proposed in Bosché et al. [9] which uses the scan angle of each point to calculate the as-planned point cloud. The exact scan angle is only available for laser scanned point clouds. This angle can also be calculated for photogrammetric point clouds if the camera angle of all pictures is recorded. Another possible way of simulating this behaviour is by calculating the inverse surface normal $-n_p$ of each point p_k. The calculation of surface normals is time-consuming and, therefore, not recommended.

A corresponding as-planned point is determined for each as-built point by calculating the first point of intersection between the facets defined in the STL file and a ray starting at the as-built point in direction of the points scan angle. The problem of finding the first intersection between a ray and a set of objects is called ray casting. There exist various techniques which optimize the process of finding the first intersection as fast

as possible using octrees or bounding spheres [8]. The pruning technique applied in Bosché [8] is based on back-face culling and frustum culling and includes the calculation of a Bounding Volume Hierarchy (BVH). The BVH contains the frustum (a quadratic cone) of each facet that has not been excluded by back-face culling and the intersection between the ray and a facet is only calculated if the points frustum and the facet's frustum intersect. The computation of each point's frustum and the BVH takes about 1 minute for clouds containing between 500,000 and 800,000 points and 19,478 facets on a 2.41 GHz processor with 2 GB RAM [9]. Since the ray casting problem is very common in computational geometry and computer games, NVIDIA® has developed the OptiX™ Prime C++ API for accelerated parallel ray intersection testing. The API is capable of calculating intersections of about $350 \cdot 10^6$ rays per second on a NVIDIA® Cuda™ GPU.

One downside of this method is the necessity of a good initial guess for the alignment of as-built and as-planned data for the algorithm to converge in a global optimum. A bad initial guess can result in a convergence in a local minimum or no convergence at all [8]. A coarse registration should therefore be applied before this step.

3.2 Point Cloud Registration

Stage (3) is the registration of the as-planned and as-built point cloud in a common coordinate system. This procedure is split up into a coarse and a fine registration process [8, 33] . The algorithms for both procedures can be chosen and combined independently. The coarse registration is required since most point set registration algorithms require a certain overlap of both data sets to provide stable convergence. The ICP algorithm achieves poor convergence results if the rotation angle exceeds 45 degrees along a certain axis [19].

3.2.1 Coarse Registration methods

Horns Method

The procedure described by Horn [27] finds the optimal transformation between two point sets M(model) and D(data) based on n pairs of corresponding points. A transformation in $\mathbb{R}^3$ is made up from a translation t, a rotation R and a scale s. This transformation has 7 (3 + 3 + 1) degrees of freedom. Since one corresponding pair of points provides 3 equations (one in each dimension), the choice of $n = 3$ results in 9 constraints which is enough to calculate seven unknowns. The point sets r_m (model) and r_d (data) containing the three corresponding pairs are mostly detected manually and make up the only manual part of the registration process [8]. The algorithm

determines the s, R and t which solve

$$r_m = sR(r_d) + t. \tag{3.1}$$

The scale s does not depend on R or t since it is based on the point's offsets $r'_{m,i}$ and $r'_{d,i}$ from their centroids $\bar{r_m}$ and $\bar{r_d}$ respectively. The scale can then be calculated by

$$s = (\sum_{i=1}^{n} \|r'_{m,i}\|^2 \,/\, \sum_{i=1}^{n} \|r'_{d,i}\|^2)^{1/2}. \tag{3.2}$$

The best aligning rotation R for r'_m and r'_d is the unit quaternion $\mathring{q}$ that maximizes

$$\sum_{i=1}^{n} (\mathring{q}\mathring{r}'_{d,i}\mathring{q}^*) \cdot \mathring{r}'_{m,i} \tag{3.3}$$

A unit quaternion $\mathring{q} = q_0 + iq_x + jq_y + kq_z$ can be interpreted as a 4 component vector $\mathring{q} \in \mathbb{H}$ with $\|\mathring{q}\| = 1$ which is often used to represent rotations in $\mathbb{R}^3$. The components i,j and k of q denote complex variables. The complex conjugate $\mathring{q}^*$ of $\mathring{q}$ is denoted by $\mathring{q}^* = q_0 - iq_x - jq_y - kq_z$. Since a vector $r \in \mathbb{R}^3$ can not be multiplied with a quaternion $q \in \mathbb{H}$, the vector $r = (x, y, z)^T$ has to be converted to a purely imaginary quaternion $\mathring{r} = 0 + ix + jy + kz$.

A vector r can be rotated about an axis $\omega = (\omega_x, \omega_y, \omega_z)^T$ by an angle θ to a vector r' by the composite product with the unit quaternion $\mathring{q} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(i\omega_x + j\omega_y + k\omega_z)$ by solving

$$\mathring{r}' = \mathring{q}\mathring{r}\mathring{q}^*. \tag{3.4}$$

Finding the $\mathring{q}$ which maximizes (3.3) as a solution for the best fitting rotation matrix R is valid since the composite product of a unit quaternion and a purely imaginary quaternion (a vector) is dot product preserving. So given

$$\vec{a} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}, \vec{b} = \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} \in \mathbb{R}^3 \text{ and } \mathring{a} = \begin{pmatrix} 0 \\ ia_x \\ ja_y \\ ka_z \end{pmatrix}, \mathring{b} = \begin{pmatrix} 0 \\ ib_x \\ jb_y \\ kb_z \end{pmatrix} \in \mathbb{H} \text{ then } \vec{a} \cdot \vec{b} = \mathring{a} \cdot \mathring{b} \,. \tag{3.5}$$

In addition

$$\vec{b}' = R\vec{b} \text{ and } \mathring{b}' = \mathring{q}\mathring{b}\mathring{q}^* \text{ then } \vec{a} \cdot \vec{b}' = \mathring{a} \cdot \mathring{b}' = \mathring{a} \cdot \mathring{q}\mathring{b}\mathring{q}^* \tag{3.6}$$

if R has been constructed from $\mathring{q}$ as described in Besl et al. [6]. Equation (3.3) can be

transformed to

$$\mathring{q}^T N \mathring{q} = \sum_{i=1}^{n} (\mathring{q} \mathring{r}'_{d,i} \mathring{q}^*) \cdot \mathring{r}'_{m,i}$$

$$\text{where } N = \begin{pmatrix} (S_{xx}+S_{yy}+S_{zz}) & S_{yz}-S_{zy} & S_{zx}-S_{xz} & S_{xy}-S_{yx} \\ S_{yz}-S_{zy} & (S_{xx}-S_{yy}-S_{zz}) & S_{xy}+S_{yx} & S_{zx}+S_{xz} \\ S_{zx}-S_{xz} & S_{xy}+S_{yx} & (-S_{xx}+S_{yy}-S_{zz}) & S_{yz}+S_{zy} \\ S_{xy}-S_{yx} & S_{zx}+S_{xz} & S_{yz}+S_{zy} & (-S_{xx}-S_{yy}+S_{zz}) \end{pmatrix}$$

$$\text{and } S_{xx} = \sum_{i=1}^{n} x'_{d,i} x'_{m,i}.$$

The unit quaternion $\mathring{q}$ turns out to be the normalized eigenvector $\vec{e_m}$ to the maximal eigenvalue λ_m of N. The eigenvalues of a quadratic matrix are the roots of the characteristic polynomial

$$\chi_N(\lambda) = det(\lambda I - N)$$

and the eigenvector $\vec{e_m}$ can be calculated by solving

$$(N - \lambda_m I)\vec{e_m} = 0.$$

Finding the translation

$$t = \bar{r_m} - sR(\bar{r_d}) \tag{3.7}$$

between two point sets is trivial once the rotation has been found since it is the difference between the scaled and rotated centroids.

So the solution for (3.1) can be calculated with the following steps:

1 Calculate the scale s.
   - 1.1 Calculate the centroids $\bar{r_m}$ and $\bar{r_d}$.
   - 1.2 Calculate $\forall i \in 0, \ldots, n : r'_{m,i} = r_{m,i} - \bar{r_m}$ and $r'_{d,i}$ analog.
   - 1.3 Solve (3.2).

2 Calculate the rotation R as unit quaternion $\mathring{q}$.
   - 2.1 Calculate $S_{xx} = \sum_{i=1}^{n} x'_{d,i} x'_{m,i}$, $S_{xy} = \sum_{i=1}^{n} x'_{d,i} y'_{m,i}$, $S_{xz}, \ldots$ to determine N.
   - 2.2 Find the biggest eigenvalue λ_m of N and the corresponding eigenvector $\mathring{e}_m$.
   - 2.3 Extract R as orthonormal rotation matrix from $\mathring{q} = \mathring{e}_m \;/\; \|\mathring{e}_m\|$.

3 Calculate the translation t by solving (3.7).

Principal Component Analysis

The coarse alignment process can be fully automated by principal component analysis (PCA) [33]. The two point clouds are aligned by the main axes and the centroids of their volumes. Since the main axis describes the whole volume, an overlap of at least 50% is required to achieve good results [55]. This can be problematic in a construction process since the main axis of the complete as-planned model might not match the main axis of the as-built data. This can be the case if the difference between the as-planned and the as-built model is very big, so many building elements do not exist in the collected data. It should be noted that principal component analysis is only applicable if the as-planned point cloud contains solely points of the building elements which should have been built according to the schedule. If the as-planned point cloud would contain all points, no progress detection would be possible in the early stages of the construction site since a building won't be built a little bit at all places the same time.

The principal axes of a point cloud correspond to the eigenvectors of the covariance matrix [16]. Let $r_{m,i}$ (model) and $r_{d,i}$ (data) denote i^{th} point of M or D and N_m or N_d respectively the point cloud size, then

$$K_m = \frac{1}{N_m} \sum_{i=0}^{N_m-1} (r_{m,i} - \bar{r_m})(r_{m,i} - \bar{r_m})^T \qquad K_d = \frac{1}{N_d} \sum_{i=0}^{N_d-1} (r_{d,i} - \bar{r_d})(r_{d,i} - \bar{r_d})^T \tag{3.8}$$

denote the respective covariance matrices while $\bar{r_m}$ and $\bar{r_d}$ are the centroids of the corresponding point sets. These covariance matrices are real symmetric matrices since

$$\forall \vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3 : xx^T = \begin{pmatrix} x^2 & xy & xz \\ yx & y^2 & yz \\ zx & zy & z^2 \end{pmatrix}. \tag{3.9}$$

For all symmetric matrices A, a diagonal matrix Λ can be found so that

$$A = Z\Lambda Z^* = \sum_{i=1}^{n} \lambda_i z_i z_i^* \tag{3.10}$$

$$I \ = ZZ^* \quad = \sum_{i=1}^{n} z_i z_i^* \tag{3.11}$$

where Z is made up from the normalized eigenvectors of A [49]. From (3.11) follows that Z is orthogonal since all entries of Z are real and therefore $Z^* = Z^T$. A matrix is orthogonal if the inverse of the matrix is equal to the transposed matrix. So (3.10) and (3.11) can be reformulated as

$$A = Z\Lambda Z^* = Z\Lambda Z^T = Z\Lambda Z^{-1} \tag{3.12}$$

$$I \ = ZZ^* \quad = ZZ^T \quad = ZZ^{-1}. \tag{3.13}$$

This procedure is called eigendecomposition and is a special case of the singular value decomposition (SVD). A method to compute the SVD of a matrix is described in Golub et al. [23] and open-source implementations of SVD as well as eigendecomposition are provided by the Eigen library. The algorithm described by Golub et al. [23] is numerically stable and uses Householder transformations as well as a QR algorithm.

The rotation between the two point sets can be computed as the product of the orthogonal eigenvector matrices of the covariance matrices [16, 55].

$$\begin{aligned} K_d &= Z_d \Lambda_d Z_d^T \\ K_m &= Z_m \Lambda_m Z_m^T = Z_m \Lambda_m Z_m^{-1} \\ R &= Z_d Z_m^{-1} \end{aligned} \tag{3.14}$$

The translation is calculated nearly the same way as in Horn [27], but the scale is neglected. The translation

$$t = \bar{r_m} - R(\bar{r_d})$$

is therefore the offset of the rotation-wise aligned centroids.

The efficiency of the PCA algorithm largely depends on the similarity of the two point sets and on the unambiguity of the eigenvectors. The presence of symmetric shapes results in similar eigenvalues along different axes. Since the eigenvectors in Z are ordered according to their eigenvalues, two similar eigenvalues can lead to different matrices

$$\begin{aligned} &\lambda_a \simeq \lambda_b \\ &\lambda_1 > \ldots > \lambda_a > \lambda_b > \ldots > \lambda_n \rightarrow Z = \begin{pmatrix} z_1, & \ldots, & z_a, & z_b, & \ldots, & z_n \end{pmatrix} \\ &\lambda_1 > \ldots > \lambda_b > \lambda_a > \ldots > \lambda_n \rightarrow Z = \begin{pmatrix} z_1, & \ldots, & z_b, & z_a, & \ldots, & z_n \end{pmatrix} \end{aligned}$$

which in return result in a different rotation matrix R [55].

3.2.2 Fine Registration Methods

The fine registration algorithms used for automatic progress tracking are predominantly iterative procedures which aim to minimize the error according to a pre-defined metric and to maximize a correspondence between the data sets. These correspondences can be *hard* (one-to-one) or *soft* (weighted one-to-all) [29, 58]. The registration process repeats the following steps: (1) Establish correspondences; (2) Find optimal transformation parameters; (3) Calculate the remaining error; and (4) Finish if remaining error is below a certain threshold, otherwise repeat.

Iterative Closest Point

The iterative closest point algorithm originally developed by Besl et al. [6] calculates a hard correspondence between the points or shapes of two data sets. The algorithm supports various kinds of explicit or implicit geometries, but point-to-point, point-to-triangle or point-to-projection based approaches are the most common ones [8]. Let $d(a, b)$ denote the closest distance operator between two arbitrary shapes and $M = \{r_{m,1}, \ldots, r_{m,N_m}\}$ and $D_0 = \{r_{d,1}, \ldots, r_{d,N_d}\}$ the model or data shape sets respectively. The optimal transformation is computed by iterating the following steps:

(1) Compute the closest point set $Y_k = \{r_{y,1}, r_{y,2}, \ldots, r_{y,N_d}\} \subseteq M$ where $r_{y,i} = \{\forall i, j \in \mathbb{N}; r_{m,j} \in M; r_{d,i} \in D_k | d(r_{y,i}, r_{d,i}) < d(r_{m,j}, r_{d,i})\}$.

(2) Calculate the optimal transformation parameters s, R and t so that $Y_k = sR(D) + t$.

(3) Apply the transformation $D_{k+1} = sR(D_0) + t$.

(4) Finish if $d_k - d_{k+1} < \tau$ where d_k is the mean square error after the current iteration, otherwise repeat the process.

The iterative closest point algorithm establishes a hard correspondence since each element in the data shape set is assigned to the closest element of the model shape set. Finding the closest element is a nearest neighbour search (NNS) problem and has a complexity of $\mathcal{O}(N_d N_m)$ in the worst case and $\mathcal{O}(N_d \log(N_m))$ in the average case. The complexity reduction for the average case can be achieved with advantageous search structures like octrees and k-d trees [5, 54]. The detailed procedure of distance calculation between any geometric shapes during NNS is described in Schneider et al. [56].

Once the correspondences are established, the optimal transformation can be calculated using Horn's method as described in chapter 3.2.1 but using all corresponding point pairs instead of three manually selected ones. Arun et al. [4] propose another variant using singular value decomposition. It is similar to PCA as described in chapter 3.2.1, but the common covariance matrix

$$H = \sum_{i=1}^{N_d} r_{d,i} r_{y,i}^T = U \Lambda V^T \tag{3.15}$$

$$R = V U^T \tag{3.16}$$

is decomposed to calculate the rotation R.

The error d_k is the remaining mean squared distance between the corresponding pairs and can be calculated as

$$d_k = \frac{1}{N_d} \sum_{i=1}^{N_d} \| r_{y,i}^k - r_{d,i}^{k+1} \|^2. \tag{3.17}$$

Proof for the convergence of the ICP algorithm with this error metric is given in Besl et al. [6]. The ICP algorithm converges in $\mathcal{O}(N_d \, log \, (N_x))$, the average case complexity of step (1), since the complexity of steps (2) - (4) is linear in $\mathcal{O}(N_d)$ and $\mathcal{O}(N_d) \subset \mathcal{O}(N_d \, log \, (N_x))$. Optimizing the NNS, therefore, has the biggest impact on the time performance of the algorithm when processing larger amounts of data [50]. Finding the corresponding elements is indeed the most time-consuming process in the ICP algorithm. Measurements made in Qiu et al. [50] are presented in table 3.1.

Procedure	Time (ms)
NNS	164.265
Alignment	0.534
Covariance matrix	0.754
SVD	0.0069
Transformation	11.678

Table 3.1: Time spent on the different stages of a sequential point-to-point matching based ICP implementation on a point cloud containing 68,229 points. The experiments have been conducted with a Intel® Core™ 2 Duo 6600 CPU.

Traversing the k-d tree to find the nearest neighbour can be done in parallel for all query points since the k-d tree is constructed once for the model shape set and remains constant for the whole process [50, 65]. The nearest neighbour to a query node is found by starting at the root node and then recursively expanding the subtree on the same side of the splitting hyperplane as the query node until a leaf node is found. The distance to the leaf node is then stored as the current minimal distance. Figure 3.1 shows that this solution might not be optimal.

To find the optimal solution, all expanded nodes are stored in a search stack. Once the first leaf node is found, the last element of the stack is removed and the current minimal distance is compared to the distance to the splitting hyperplane of the removed node. If the distance is larger than the current minimal distance, the node is discarded and the next element is removed from the stack. If the distance is smaller, the current minimal distance is updated and the algorithm repeats the search process for the nearest neighbour in the expanded node. This results in a lot of time spent with backtracking to find the optimal solution. Improvements by using priority search queues were made by

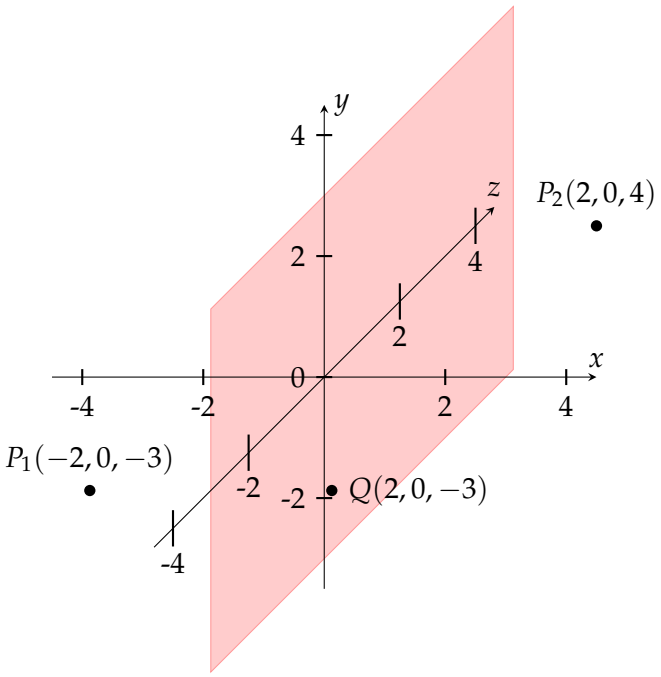

Figure 3.1: Node P_2 is the first visited leaf node but $\|P_2 - Q\| = 7$ where $\|P_1 - Q\| = 4$.

Arya et al. [5]. The priority queue is sorted in ascending order according to the distance of the splitting hyperplane to the query node so that nodes which are more likely to contain a closer node are expanded earlier. The additional computations required to sort the priority queue are less expensive than traversing nodes which are a lot less likely to contain the nearest neighbour.

The use of Voronoi cells is proposed by Zabatani et al. [67] and Kats et al. [31] but is not applied. This priority queue based k-d tree search structure is used in the parallel CUDA implementations of the ICP algorithm presented in Wu et al. [65] and Qiu et al. [50]. Both algorithms distribute the work between GPU and CPU in a similar fashion, as depicted in figure 3.2 and implement a point-to-point matching procedure. The parallel NNS is performed in a similar way as described by [5] and implemented by Brown et al. [14]. Since NNS on a GPU is performed by many threads in parallel, dynamic memory allocations for each thread in order to store a long priority queue in fast shared memory should be avoided. The length of the priority queue is therefore limited to a size of one in [50]. The loss of accuracy is acceptable regarding the much better computation times. The k-d tree is left balanced and the array elements contain the index of the point at which the splitting hyperplane is constructed. This parallel NNS approach provides a speedup between 7 and 29 times compared to a CPU

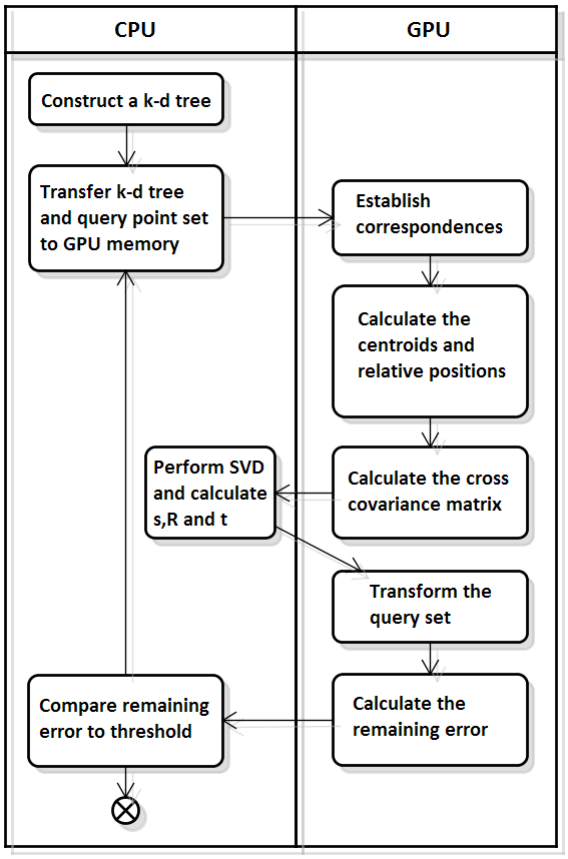

Figure 3.2: Work distribution between GPU and CPU as presented in [50, 65].

implementation [14]. The experimental results are presented in table 3.2. The optimal transformation parameters are calculated using the cross-covariance matrix as described in [4]. Parallel ICP variants prove to be at least 7 times faster than CPU implementations on comparable hardware [31, 50, 65]. A detailed performance comparison of various CPU and GPU implementations of ICP and other point set registration algorithms is listed in table 3.3.

Robust Registration Algorithms

Establishing a hard correspondence brings two different problems: (1) NNS is computation intensive; (2) A good initial alignment is required to ensure convergence in a global minimum. The problem of unstable convergence is still present after parallel acceleration of the NNS. Therefore there exist robust point set registration algorithms which

Algorithm	Points	Time (GPU/CPU)
QNN	1,000,000	62 / 2,357
QNN	10,000,000	723 / 30,008
kNN	1,000,000	608 / 10,889
All-kNN	1,000,000	657 / 11,858

Table 3.2: Performance comparison of CPU-NNS and GPU-NNS algorithm results for one iteration presented in [14]. Experiments have been conducted with an Intel®Core™ i7-920 CPU and a NVIDIA® GeForce GTX285™ GPU. Times in milliseconds.

are based on soft correspondences. These algorithms prove to be more stable and less reliant on a good initial alignment. The improved stability and lesser remaining errors after convergence have the cost of being often more computationally expensive [18–20, 45, 58]. The expectation maximization based ICP (EM-ICP) variant and the softassign algorithm proposed and implemented by Tamaki et al. [58] are soft correspondence establishing point set registration algorithms which are more robust than the traditional ICP algorithm. They compute a full cross-covariance matrix to assign correspondences between each pair of points. Measurements regarding the time performance of these algorithms with real world data can be found in section 4.2 as well as table 3.3.

Another possibility for robust point cloud alignment is the approximation of both point sets with density functions like Thin Plate Splines (TPS) or Gaussian Mixture Models (GMM) while the both are closely related to each other. This GMM based approach is described in Jian et al. [29]. The criterion for the goodness of fit of the two Gaussians is the L_2E distance or the integrated squared error (ISE) and each point is considered one component of the GMM with the position of the point as the mean value μ_i and a spherical covariance matrix Σ_i. The probability of a GMM M with n components can be calculated as

$$f_M(x) = \sum_{i=1}^{n} w_i f_{x_i}(x|\mu_i, \Sigma_i) \tag{3.18}$$

where $f_{x_i}(x|\mu_i, \Sigma_i)$ denotes the density function of the multivariate gaussian distributed continuous random variable $X_i \sim \mathcal{N}_3(\mu_i, \Sigma_i)$ as the i^{th} component. So if M and D denote the GMM representations of the model and data point sets respectively, the correlation or cost function $F(T(D, \theta)) = DR^T + t$ with the transformation parameter θ

and the transformation function $T(D,\theta)$ or $D(\theta)$ is given as

$$\begin{aligned} F(T(D,\theta)) &= \int f_D^2 dx - 2\int f_D f_M dx + \int f_M^2 dx \\ &\equiv \frac{\int f_D f_M dx}{\sqrt{\int f_D^2 dx \int f_M^2 dx}} \\ &\equiv \sum_{i=1}^{n_M}\sum_{k=1}^{n_D} w_{M_i} f_{M_i}(X_{D_k}). \end{aligned} \tag{3.19}$$

These equivalences are valid since the first and last term of the initial cost function are invariant under transformation, since the goodness of fit of a model with itself won't change if the model is transformed, and maximizing kernel correlation and minimizing the L_2E is similar for rigid point set registration [29, 57]. The complexity of evaluating the cost function F therefore grows in $\mathcal{O}(n_D n_M)$. The gradient of the cost function can then be evaluated as

$$\frac{\partial F}{\partial \theta} = \frac{\partial F}{\partial T(D,\theta)} \cdot \frac{\partial T(D,\theta)}{\partial \theta} \tag{3.20}$$

$$\frac{\partial F}{\partial t} = \left(\frac{\partial F}{\partial T(D,\theta)}\right)^T 1_{n_D} \tag{3.21}$$

$$\frac{\partial F}{\partial r_i} = 1_d^T \left(\left(\left(\frac{\partial F}{\partial T(D,\theta)}\right)^T D\right) \circ \left(\frac{\partial R}{\partial r_i}\right)\right) 1_d \tag{3.22}$$

Once the gradient is known, F can be optimized with numerical methods to find the best parameter θ for the optimal fit of both point sets. This proves to be more robust towards outliers and differing point densities since the functions are continuous and the ISE is more resistant to bad data [57]. In addition, the GMMs or TPSs can be clustered and their parameters can be optimized to reduce the number of components and therefore the number of required calculations.

3.3 Progress Detection

Once the as-built point cloud and the BIM are successfully aligned in a common coordinate system, the actual progress measurement can be made. The existence of each object contained in the BIM and marked as built according to the schedule should be decisively confirmed or disproved. The recognition metric has to be chosen with regard to multiple factors. Depending on the data available for each point, like color information or normals and the percentage of building elements visible in the scan, the more aspects can be included in the object recognition process. The most basic

Software	Hardware	Points	Time (s)	Reference
C++	Xeon® E5-1603	40,256	0.945	[65]
C++ (OpenMP)	Core™ i7-965EE	68,229	13.140 (2.800)	[50]
CUDA	GTX295®	68,229	0.260	[50]
CUDA	GTX660®	40,256	0.050	[65]

Table 3.3: Performance comparison of CPU-ICP and GPU-ICP algorithms presented in [31, 50, 65]. All CPUs are manufactured by Intel® and all GPUs are part of the NVIDIA® GeForce series.

recognition technique, the point-to-surface or point-to-point distance can be extended to also compare the surface normals and the color of the matched data. The use of precedence relationship graphs as proposed in [13] can further increase the amount of distinctive statements which can be made about the construction progress.

3.3.1 Voxel Grid Occupation

The progress of a construction site can be measured by segmenting the scene into discrete voxel cells and by marking them as *free* or *occupied*, independent from the data collection technique or the data available for each point since only the position is required. The first step is to create a uniform three-dimensional grid structure in the bounding box of the aligned data sets and assigning the *free* state to all cells. Each cell which intersects a triangle of the BIM can then be labelled as *occupied*. Afterwards, all cells which are labelled as occupied can be labeled as *built* if the number of points from the as-built cloud inside the cell is above a predefined threshold, otherwise they are marked as *not built*. If a certain percentage of the cells intersecting a triangle are marked as *built*, then the triangle can be marked as *built*. The same process can then be repeated for the triangles belonging to a building element. This method can be applied if the amount of occlusion in the scene is low or no information about the scanning positions and directions is available. It has to be stated that this case is rare in real-world data sets [22, 63].

If the scanning directions of the as-built points are available, this voxel cell-based approach can be extended to handle occlusions. How a voxel cell is marked as *visible* or *occupied* depends on the scanning technology. A naive approach for visibility checking can be implemented with ray casting. All voxel cells in the field of view of a scanning position are assigned the state *visible* if the ray from the scanning position passes the center of the voxel cell without intersecting a surface triangle and *unknown* otherwise. More sophisticated methods are described in [12, 22, 63].

3.3.2 Surface Coverage

Another way to confirm the existence of a triangle in the 3D representation of a BIM is to measure the surface covered by as-built points which are matched with the triangle. The matching parameters can include the point-to-surface distance, the point and surface color as well as normals. If an as-planned point cloud has been used for alignment and if it has been constructed as described in 3.1.2, the euclidean distance between each point pair is sufficient. If a point is recognized as belonging to a surface, the surface covered by this point has to be calculated. A tangible function for laser scanned point clouds is given in [9]. This surface coverage estimation is more difficult for photogrammetric point clouds since they have a less uniform density. To decide whether the recognized points are sufficient to confirm the existence of the building element, the number of recognized points and the mean distance between them is compared with the same parameters of a point set sampled uniformly on the triangle surface. If the deviations are above a certain threshold, the recognized points are not dense enough or close together to represent a continuous surface. If the number of points is high enough and the mean distance sufficiently small, the area covered by each point is calculated as the area under the circle with a radius of $\frac{\overline{E}}{2}$ where $\overline{E}$ denotes the mean edge length.

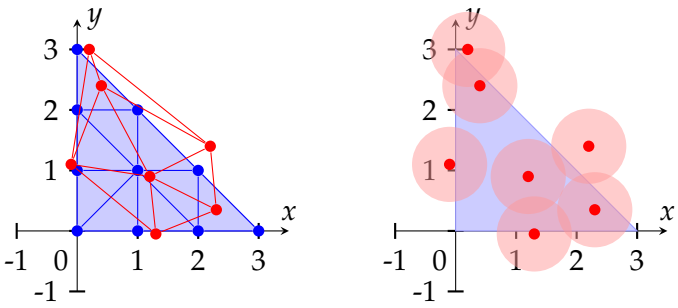

Figure 3.3: Sampled points in blue and scanned points in red. If the number of scanned points is above a certain threshold and the mean edge lengths are close to each other, the covered surface can be calculated as the area under the circles. Overlapping regions have to be considered.

The circle is projected onto the triangle and the circle is tested for intersection with the edges of the triangle since only the surface of the overlap should be measured. In addition, intersections between the circles of multiple points have to be subtracted according to the inclusion-exclusion principle. The area of a circle segment or an

intersection between two circles can be calculated as

$$A_t = \frac{r^2}{2}(\theta - \sin(\theta)) \tag{3.23}$$

$$A_c = r_2^2 \cos^{-1}\left(\frac{d^2 + r_2^2 - r_1^2}{2dr_2}\right) + r_1^2 \cos^{-1}\left(\frac{d^2 + r_1^2 - r_2^2}{2dr_1}\right) - \frac{1}{2}\sqrt{(-d + r_2 + r_1)(d + r_2 - r_1)(d - r_2 + r_1)(d + r_1 + r_2)}. \tag{3.24}$$

See figure 3.4 for more details.

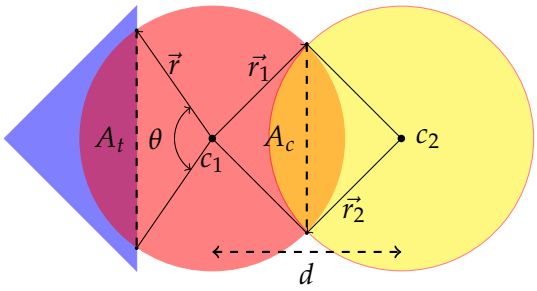

Figure 3.4: A_t defines the purple area while the orange area is denoted by A_c. The angle θ is given in radians. $r = |\vec{r}|$.

To detect intersections, all points have to be checked against each other in $\mathcal{O}(n^2)$. Afterwards, all overlaps have to be checked against all other intersections again to find overlaps of three circles. This process is then repeated for all overlaps until none are detected. If no intersection including i circles is present, then no intersection of $i+1$ circles is possible. To calculate the overlap area of n circles all collision points between any circles have to be calculated. Then the polygon which is bounded by the n collision points which lie inside all circles can be constructed. The area of a n-polygon can be calculated as the sum of triangles

$$\sum_{i=0}^{n-1} A(p_i, p_{i+_n 1}, p') \text{ where } p' = \frac{1}{n} \cdot \sum_{i=0}^{n-1} p_i \tag{3.25}$$

denotes the center of the shape. The circle segments of each circle are calculated with equation (3.23). The angle θ is the angle between the inner overlap points with the highest euclidean distance from the center point of the circle and the center point itself.

If the surface covered by the points is above the predefined threshold, the triangle can be confirmed as built. The presence of occlusions can also be handled in a similar way as in chapter 3.3.1. Instead of the voxel cells, the triangles themselves can be marked as visible or not. If for example all visible triangles of a building element are confirmed,

one can argue that the building element should be marked as fully built even if the number of recognized triangles is below the required minimum value.

4 Parallel automatic progress tracking

4.1 Proposed Methods

4.1.1 Data Alignment

The procedure of aligning two data sets in a common coordinate system is conducted with the registration algorithms described in 3. All algorithms are used without previous coarse alignment to test their robustness. The classic ICP as described in [6] is implemented in a CPU variant by the CloudCompare (CC) open-source software. This hard correspondence establishing algorithm is measured against CPU and GPU based robust point set registration algorithms. The first soft correspondence establishing algorithm which is part of the test suite is the GMM based registration algorithm proposed in [29] which will be called GMMReg in the following. The GMMReg algorithm can be used with three different kernel functions and is implemented without parallel acceleration. The software provides thin plate splines (TPS), gaussian radial basis functions (GRBF) and rigid functions (RF). The transformation which has to be computed is a rigid transformation since it is made up of a translation and a rotation component. Therefore the use of a rigid function as kernel is sufficient, as it can handle more data points in less time and requires less memory. The downside of not being able to compute non-rigid transformations is not relevant for this case. These sequential algorithms are compared with a NVIDIA Cuda based implementation of the Softassign and EM-ICP algorithm presented and implemented by Tamaki et al. [58]. The model point cloud used for the registration process is created by sampling the triangle mesh representation of the BIM since this allows to adjust the number of model points and it is very fast. The data point clouds collected on real world construction sites can contain multiple millions of points and are therefore reduced by random subsampling if a point cloud of a smaller size is required.

4.1.2 Progress Detection

The algorithm which classifies the building components through comparison with the collected point cloud was implemented in C++ AMP, in the following called amp, to be able to process the data in a parallel nature. The core of an amp based program is

a parallel loop function which launches one thread for each element. The dimension and number of threads to spawn in each can be specified with an *extent* object that is passed to the loop. The actual data is accessed via an `array_view` object which is templated with the data type and the dimension of the array, the so-called *rank*. The array view obtains a pointer to the actual data resident in the random access memory on construction. The dimension of the data which is being processed can be supplied by passing the actual *extent* object which belongs to the `array_view` to the loop or can be explicitly specified. It is mandatory that the rank of the *extent* and the *index* are identical.

```
#include "amp.h"
using namespace concurrency;

int data[10][10];
array_view<int,2> data_view(10, 10, data);

// First option
parallel_for_each(data_view.extent,[=](index<2>idx) restrict(amp)
{ data[idx] = 0; /* do stuff in parallel ...*/ });

// Second option
parallel_for_each(extent<2>(10,10),[=](index<2>idx) restrict(amp)
{ data[idx] = 0; /* do stuff in 2 dimensions parallel ...*/ });

// Also allowed
parallel_for_each(extent<1>(10),[=](index<1>idx) restrict(amp)
{ for(int i = 0; i < 10; ++i) { data[index<2>(idx[0],i)] = 0; }
  /*do stuff in 1 dimension parallel...*/ });
```

Figure 4.1: Example usage of C++ AMP to process data in parallel. The data residing in an `array_view` can be accessed via the `index`.

Each thread has read/write access to all data captured by the lambda expression or the passed method, but the access is not synchronized. Functions which are passed to a parallel for each loop have to be declared with the *restrict(amp)* keyword and must not use functions which have not been declared with this keyword. The amp_math header provides all mathematical functions defined in the standard math library for amp programs. The lambda expression constructed for the `parallel_for_each` loop can call functions which are declared and implemented with both *amp* and *cpu* restriction

specifiers. A function declaration can contain both restriction keywords but is then only allowed to call functions declared in the same way. The second option is to overload the functions, while one declaration features the *restrict(amp)* keyword and one the *restrict(cpu)* keyword. Another condition is that all amp functions have to be inlineable.

```
#include "amp_math.h"
using namespace concurrency;

// base type operations are always allowed
float dot(float3 a, float3 b) restrict(cpu, amp)
{ return a.x * b.x + a.y * b.y + a.z * b.z; }

// amp version using the fast_math namespace sqrtf function
float length(float3 vector) restrict(amp)
{ return fast_math::sqrtf(dot(vector, vector)); }

// cpu version using the std namespace sqrtf function
float length(float3 vector) restrict(cpu)
{ return std::sqrtf(dot(vector, vector)); }
```

Figure 4.2: The vector length function is overloaded for it to be allowed to be used in parallel for each loops. A function can be declared with one or both restrict keywords. It then has to apply to the union of rules for both types.

The libraries used for point cloud management each feature their own math library with neither of them supporting the amp framework. The first step in the algorithm is to convert the triangle mesh and the point cloud to data formats which support parallelism with amp. Since the C++ AMP basic linear algebra subprograms (BLAS) library is still under construction, all basic vector operations have been implemented for CPU and GPU usage. All triangles present in the mesh are converted to a format containing the corner points, the edges, the normals on all edges pointing inwards and the area. The vertices, edges and normals are implemented as three component float vectors. All point cloud data points are simply converted to a three component float vector which contains the point position. After initializing the data, all points are matched with the triangles. The criterion applied to decide whether a point belongs to a triangle or not is not the usual point to triangle distance, but a pair of two checks. The real point-to-triangle distance includes 7 different cases and was therefore avoided [56]. The distance measurement is split up into checking whether the point lies inside the triangle if projected onto the triangle plane and then calculating the point-to-plane

distance. The point p lies inside the triangle t if

$$\forall i \in \{0,1,2\}; n_i \cdot (p - v_i) \geq 0 \tag{4.1}$$

where n_i is the inwards pointing normal on edge e_i, the edge connecting v_i with the next point in a clockwise ordering [56]. Then the point-to-plane distance is the point's distance to the triangle. If this value is below a certain threshold d_{p2p}, then a copy of the point shifted onto the triangle plane is added to the container which holds all points matched with the triangle. Each triangle can be matched with $5,000$ points at maximum. The access to the variable holding the number of points which have already been assigned to the triangle is synchronized between all threads to avoid undefined behaviour. The next step is the sequential processing of all triangles to calculate their built state. If a triangle is not matched with any points it is immediately recognized as not built. The next step otherwise is to calculate the mean distance between all matched points in parallel.

The intention of using the area covered by the points matched with the triangle as single confirmation criterion had to be discarded due to the high complexity of the operations and the lack of a fast general solution for an unknown number of points. Intersecting one circle with one triangle already leads to more than 6 different cases for which different areas and shapes have to be computed. Calculating the area covered by a circle inside a triangle which intersects each edge twice splits the shape up into three triangles and three circle segments. The computational effort to find this area includes the evaluation of 18 square root functions and dot products and 3 trigonometric functions. Considering that this is one of the less computation expensive cases, it is obvious that the performance of this approach is insufficient. As a result, the proposed method to measure the uniformity of the points associated with a triangle was changed from calculating the area coverage to comparing the mean distance between the points to the uniform sampling distance which is calculated as

$$d_{us} = \sqrt{\frac{c^2}{\sqrt{n_p}} + \frac{h_c^2}{\sqrt{n_p}}} \tag{4.2}$$

where c denotes one edge of the triangle, h_c the height on this edge and n_p the number of matched points.

The threshold parameters which determine whether a triangle is marked as built or not are the point-to-plane distance between point and triangle and the difference between the uniform sampling distance between points on the triangle surface and the actual mean distance between all points associated with the triangle.

4.2 Experimental Results

The testing procedure of the proposed alignment and triangle recognition methods was conducted using data from two real-world construction sites in Munich. This data includes the building information models in IFC file format and the STL files containing the triangulated geometric representation of the buildings. Five photogrammetric point clouds acquired at different time steps throughout the construction process are available for each engineering project. The number of points in each cloud varies from about 900,000 to about 6,500,000. The gtest library was used for the overall test suite. It offers the possibility of writing parameterized test cases and to output the test results as well as intermediate states to an XML file. The test cases only measure the time required for the actual alignment operation since additional operations made while setting the test case up are not measured by the gtest library. Data which is shared by all test cases is provided by a global test environment which is set up and deleted in the test's main function. This environment is used to avoid repetition of time-consuming IO operations. The downside of this method is the reduction of RAM available for the computation since the test data has to be kept in the RAM for the whole duration of the test case.

4.2.1 Data Alignment

Each algorithm is tested with varying point numbers and different point clouds. The triangle mesh was always sampled randomly to be able to adjust the number of points. Each algorithm is tested with the same random rotations which are generated using the Eigen library. These rotations are loaded from a text file at the beginning of the test session and are then passed to the individual test cases. All algorithms are measured regarding multiple factors:

1. The difference between the remaining cloud to cloud distance after alignment and the expected cloud to cloud distance if perfectly aligned.
2. The scalability regarding the sizes of the data sets.
3. The robustness towards bad data.
4. The time required for computation.
5. The overall stability regarding program failure.

The first test suite is run using four different point clouds and two different triangle meshes. The real world data point clouds are randomly sampled to a size of 2,000 points to test the base time performance of all algorithms. The second measurement

factor is the deviation from the expected root mean square (RMS) cloud-to-cloud distance after alignment. The deviation instead of the result is chosen because of the dissimilarity between the model and data point sets. Points which are present in the data cloud belonging to objects not existing in the triangle mesh are included in the alignment process, leading to arbitrary transformations which minimize the cloud to cloud distance but do not align the data sets onto each other. All results are presented in figure 4.3. Each algorithm is launched ten times per point cloud, resulting in 40 tests each.

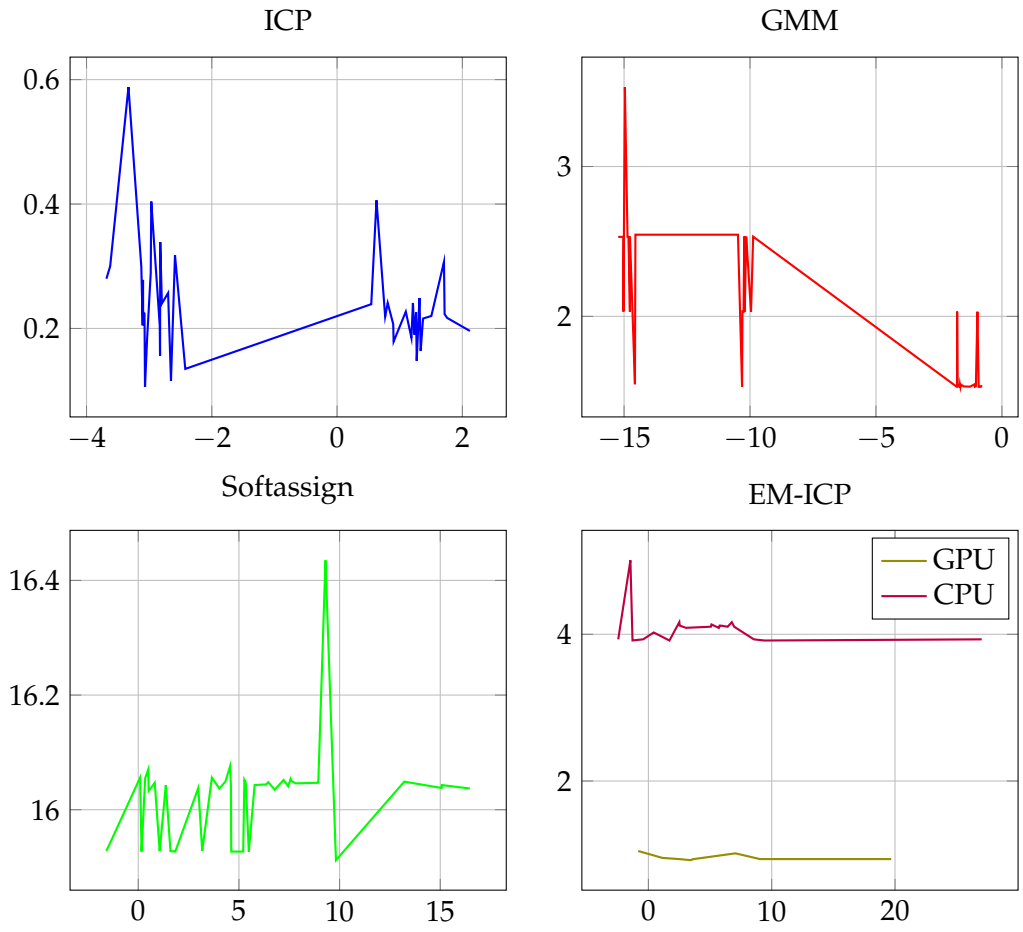

Figure 4.3: The deviation from the expected cloud to cloud distance RMS error is displayed on the x-axis and the processing time on the y-axis. Each algorithm was supplied with 2,000 points from the sampled model point cloud and 2,00 points sampled from the photogrammetric data point cloud.

A comparison towards the different processing times between the GPU and the CPU based algorithms reveals no distinct categories. The GPU based EM-ICP implementation and the classic ICP algorithm both prove excellent computation times. The acceleration gained by parallel data processing with Cuda does not improve the accuracy of the soft correspondence ICP version but reduces the time required for computation to about $1/8^{th}$ of the OpenMP-based implementation. The alignment results are often far from the expected values and especially the GPU variant turns out to execute very unstably. The NVIDIA Cuda version crashes in 29, the CPU-based implementation in half the cases due to errors which are likely related to struct member alignment. They are therefore discarded from the test suite.

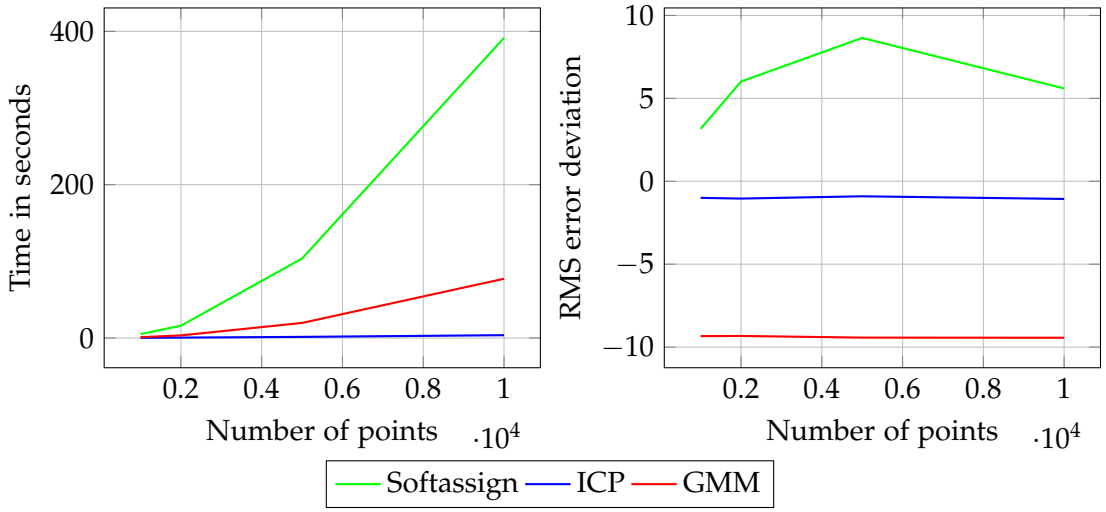

Figure 4.4: The average processing time for the alignment procedure and the deviation from the expected RMS cloud-to-cloud distance after alignment.

The GMMReg algorithm has worse time performance than ICP but always finds an alignment which reduces the remaining cloud to cloud error below the expected value. This behaviour is caused by the dissimilarity of the model and the data point cloud. Each point is considered as a multivariate gaussian with a spherical covariance matrix. A model point cloud which has few large spaces containing no points establishes a continuous space without locations corresponding to no point at all. If the data point cloud contains areas without any points, as it is most often the case, the data point cloud will be transformed so that the probability for all points to be generated at their location is maximized. The algorithm prefers a higher number of points with a lower probability over a medium number of points with very high probability and a medium number of points with low or zero probability. The Softassign algorithm exhibits the

worst time performance despite being implemented for parallel processing. The long computation times are compensated by the algorithm's stability and scalability, as well as the acceptable alignment accuracy.

The second test suite is executed with the same metrics as the first one but with varying point counts to determine the scalability of the algorithms and whether the alignment results improve with the number of available points. The test data again includes at least four different point clouds from two real-world construction sites. The results shown in figure 4.4 reveal that the alignment behaviour is independent from the number of processed points. Both the ICP and GMM registration algorithm provide the same results, no matter if 1,000 or 10,000 randomly sampled points are processed. Only the Softassign algorithm performs slightly worse at around 5,000 points. The processing time required by the Softassign algorithm increases significantly from 5.13 seconds average at 1,000 points to 391.43 seconds at 10,000 points. The CPU-based algorithms required 77.3 and 3.71 seconds respectively to process the same amounts of data while also providing better alignment results.

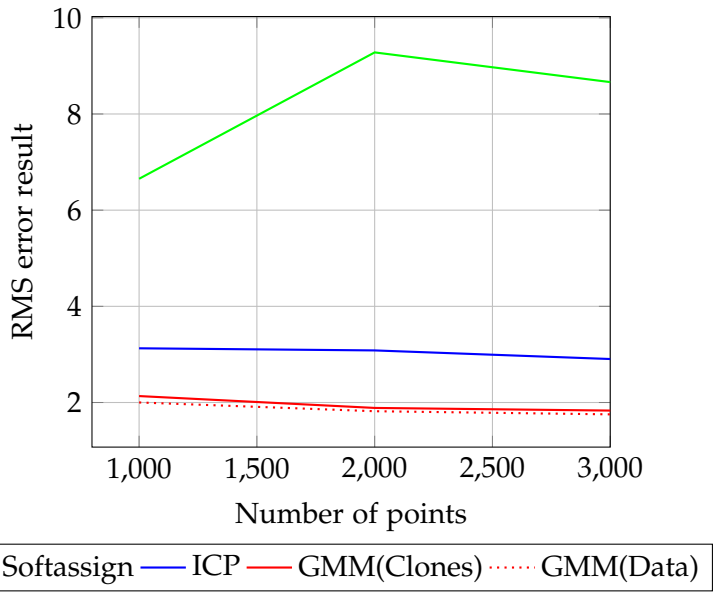

Figure 4.5: The data point cloud is a clone of the model point cloud. The remaining cloud to cloud distance after alignment is measured. The alignment results with sampled data from the second test suite achieved with the GMM registration algorithm are also displayed with the dotted line.

The third test suite aims to analyse the impact of the similarity or dissimilarity of both point clouds. Two identical point clouds are aligned after transforming the data

point cloud. All point clouds registered in previous tests are created by sampling and therefore contain elements without a corresponding counterpart. This makes the alignment results less expressive since the optimal solution is not the one with the least remaining error after alignment. The remaining cloud to cloud distance after the alignment of two identical point clouds is expected to be zero. This constitutes a better metric for a qualitative analysis. The classic ICP and the GMM registration based algorithm consecutively perform better than the Softassign algorithm. They additionally indicate a tendency towards slightly better performance while processing bigger point clouds.

The initially proposed method of using a robust GPU based point set registration algorithm with large point clouds which are used for construction engineering is not supported by the experimental results collected in three different test suites. Accelerating an algorithm through parallel data processing is possible and provides the expected results but choosing the correct alignment algorithm and a more sensitive approach towards which data to use results in a more precise registration. The Softassign algorithm doesn't scale well with bigger data sets and fails to consistently align identical point clouds. The traditional ICP algorithm as proposed by Besl et al. [6] provides the best fitting alignments for data containing points without a corresponding element in the model set but doesn't profit much from high-quality data. The scalability and stability traits are overall very good. This makes the algorithm suitable for larger scenes and applicable for automated progress tracking if coupled with an initial coarse alignment. These empirical findings correspond to the methods proposed in current state of the art automatic progress detection procedures [8, 22, 34, 63]. The second CPU based algorithm originally proposed by Jian et al. [29] using gaussian mixture models achieved the best alignment results while still being able to process an acceptable number of points in reasonable time. How well the data point cloud fits the model cloud after alignment significantly depends on the existence of a solution which doesn't only provide the best result for the sum of all points but also a high probability for each individual point.

4.2.2 Progress Detection

The actual progress detection algorithm is tested independently from the alignment procedure. The results from section 4.2.1 show that the remaining error after alignment is very unlikely to be close to zero. The recognition of triangles present in the geometric representation of a building depends on a good alignment and high-quality data. The alignment procedure was therefore not included in the test session. The test session itself is split up to test the algorithm in an ideal environment and to configure the best parameter set for the procedure. The resulting test suites are

1. Progress detection with artificial point clouds sampled on the triangle mesh.
2. Progress detection with manually aligned real world data sets.

The first test suite was used to verify the suitability of the algorithm and to find the optimal parameters for triangle recognition. In order to find the best parameter set, one parameter is kept constant and the other one is changed. The parameters are separated to avoid disambiguation regarding the effects of changing one of the parameters. The results are shown in figure 4.6.

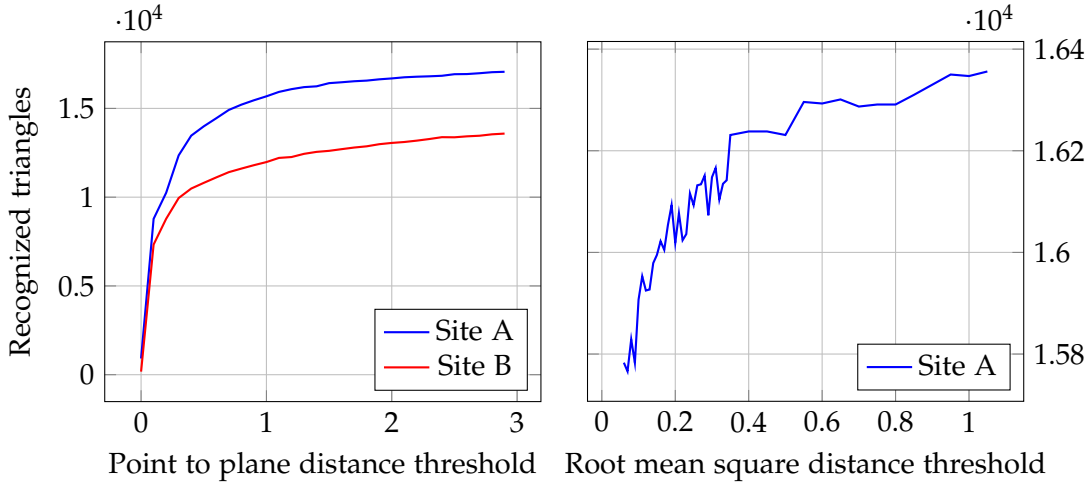

Figure 4.6: Number of triangles recognized as built from a point cloud created by sampling the triangle mesh. The sampled cloud has a density of $\approx 262,400$ points. Site A contains $19,184$ triangles and Site B $15,620$.

To find the best point-to-plane distance threshold, the root mean square distance threshold was fixed at 0.05 and the free parameter tested with values from 0 to 3. The optimal value was chosen to 0.85 since the rate of change is not too small and it provides compensation for errors in the alignment phase. The probably best value for high-quality data would be the root of the second derivative since it is the point up to which the rate of change increases. The graph in figure 4.6 on the right hand side shows only minimal change with an increasing threshold value. This is due to the fact that the point cloud was created by uniformly sampling the triangle mesh. The mean distance between the sampled points matched with a triangle is therefore very close to the uniform sampling distance. The parameter was selected at 0.1 since a quite uniform distribution of the points confirming a triangle in a point cloud is also desired.

The progress detection algorithm was then tested with artificial data and the determined parameter set. The model point cloud was sampled again with about $242,000$

points. The results are presented in figure 4.7.

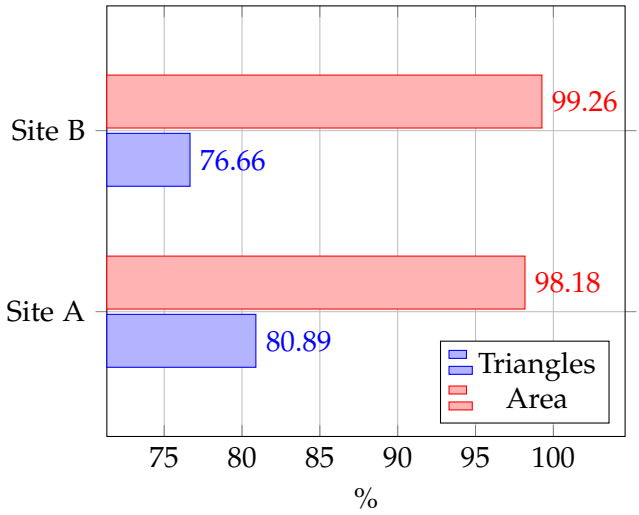

Figure 4.7: Percentage of building element triangles and area recognized in an ideal environment.

The recognition rate of 81% in the best case may be considered bad, but it should be noted that this 80% which have been recognized make up about 99% of the buildings area. These really small triangles are part of curvatures, which are approximated by a high number of small triangles in the triangulated mesh representation. Nearly all triangles were recognized with a sampled point cloud containing more than $1,000,000$ points.

The second test suite is conducted with the data acquired at the construction sites. The data point cloud had previously been aligned manually with the triangle mesh. The point clouds were reconstructed from imagery at five different time steps for each construction site. The data contains very few points on the inside of the building since all pictures have been taken from the outside. The progress detection procedure was run once per point cloud. The effectiveness of the algorithm is measured regarding the time performance and the object recognition performance as described in [10]. In addition to the recall and precision rates presented in figure 4.9, the overall correctly recognized area is also included in the evaluation. These results are shown in figure 4.8. The recognized area of construction site A remains at a rate about 70% while the precision rate constantly increases from 40% up to about 90%. This indicates that the rate of false positives is diminishing. This is most likely related to the fact that more building elements which are falsely recognized as built from early on are actually built in the later observations. The presence of objects like scaffolding or covering blankets

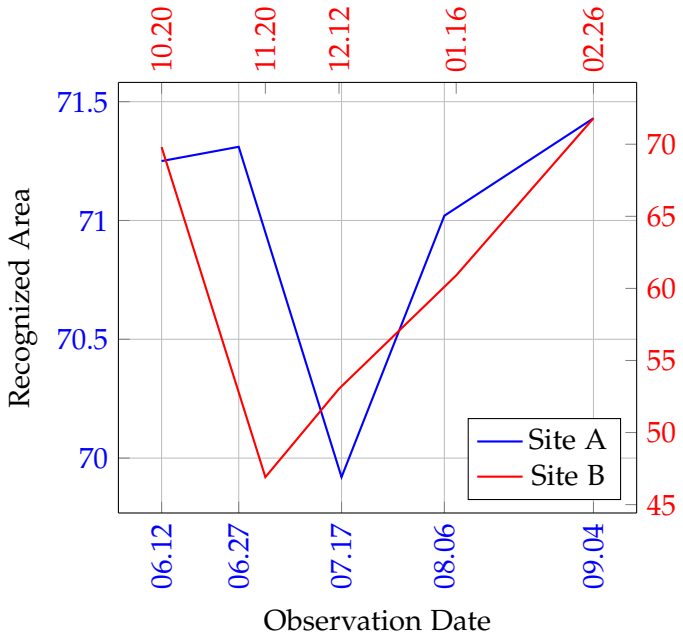

Figure 4.8: Area recognition with manually aligned photogrammetric point clouds. The distance threshold was set to 0.85 and the root mean square distance threshold to 0.1.

close to the actual building elements combined with the rather high point-to-plane distance threshold of 0.85 is likely to result in false point-to-triangle matchings. The second threshold parameter which measures the uniform point distribution fails to deny the existence of the triangle if the whole surface of the element is covered with points not belonging to the real building component. Distinguishing between a blanket or banner fixed at the scaffolding in front of a wall is not possible with a loose distance threshold or without additional per point and triangle information like colour values. The same test suite was repeated with a point to plane distance threshold of 0.15 which corresponds to a distance of 15cm from the triangle surface. The result showed a slightly better recall rate but an overall decrease regarding the correctly classified area. No other significant changes could be identified.

The low recall rate reveals a persistently high number of false negatives. The main reason for this behaviour is the presence of occlusions in the scene and the fact that all pictures were taken from outside the building. The interior of the construction site is therefore not visible at all. These problems can only be solved by reconstructing the point cloud from more imagery or by including the dependency relationships between building components as described by Braun et al. [13].

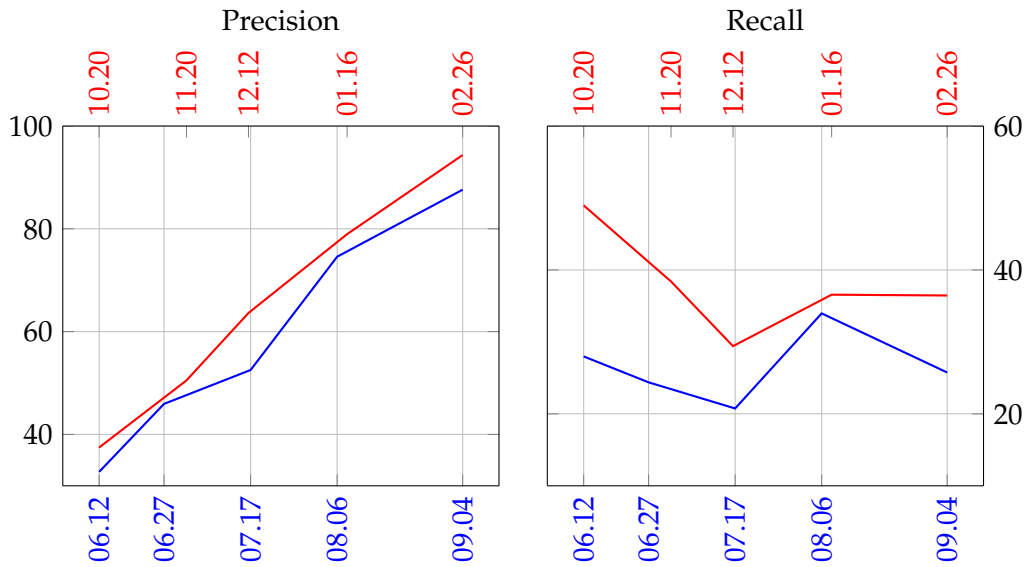

Figure 4.9: The precision and recall rates for both construction sites in %.

The tests shown in 4.7 took 36.758 seconds in average for Site A and 47.198 seconds for Site B. Each triangle was matched with 25,000 points in parallel. A speedup from higher point counts could not be achieved since the bottleneck of the registration procedure is the synchronized access of resources. This synchronization could be split up on more variables to provide further acceleration through parallelism. It has to be kept in mind that the algorithm should also be performing on other than high-end hardware. Even though the maximum number of threads which are allowed for parallel processing in amp is not reached, the software results in fatal failure with the advice towards the less aggressive use of the computing device when processing more than the mentioned 25,000 points on an Intel HD 4000 integrated graphics chip. The time performance of the algorithm with real world data can be seen in 4.10. The overall complexity is linear and scales from 54.473 seconds at 889,686 points to 318.629 seconds at 6,431,022 points. These processing times proved to be very stable despite different alignment results. The average computation time in a test suite including the previous alignment from 10 different rotations was 44.19 seconds for the smallest and 305.41 seconds for the largest point cloud. These tests were executed on a NVIDIA Quadro 600 GPU but quite similar processing times were achieved with an Intel HD 4000 mobile graphics chip. The NVIDIA GPU has a processing capability of 245 GFlops while the integrated Intel chip is measured at 332.8 GFlops.

All operations which process a previously unknown number of points are executed

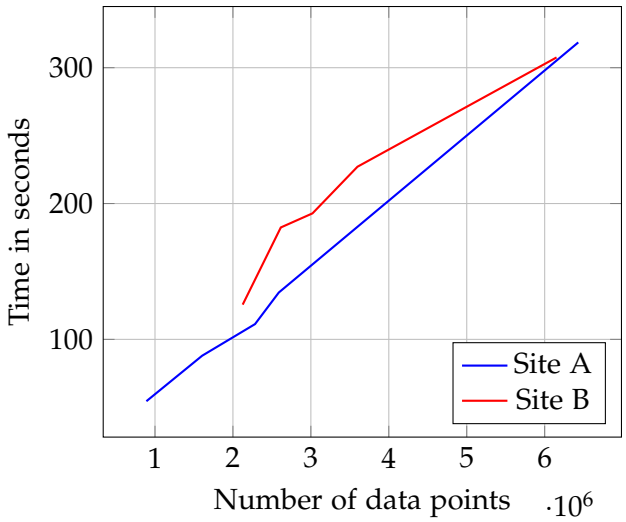

Figure 4.10: Time required to confirm or deny the existence of building component triangles. The triangle meshes contain 19,184 and 15,620 triangles respectively.

in parallel. The performed computations are the mean distance calculation between all points matched with a triangle, which can be quite computation expensive if one triangle is matched with multiple thousands of points and the matching of all points and triangles. Since the number of matched points is not known in advance, even triangles matched with very few points have their coverage computed in parallel. All triangles are therefore processed after another since the number of triangles is likely to be much smaller than the number of points. One additional step which would improve this behaviour would be the fusion of very small triangles. The smooth edges of curvatures which consist of many small triangles are not important for the correct classification of building components and only result in additional computations and misleading results.

The overall duration of approximately 5 minutes while processing about 6,000,000 points and a linear scaling of 1 minute per 1 million points at about 20,000 triangles is acceptable.

5 Conclusion and prospects

The proposed approach for the application of parallel hardware acceleration of robust point set registration algorithms and automated progress detection in civil engineering projects does not meet the requirements for real world construction sites. The intention of using the GPU to be able to process more data in less time and therefore achieve better alignment results turned out not to be viable using the parallel point set registration algorithms originally proposed by Tamaki et al. [58]. The EM-ICP implementations were not operating properly in conjunction with the overall application. The framework includes multiple libraries for point cloud processing like the Point Cloud Library (PCL) and Cloud Compare. Each library has its own data formats and mathematical libraries. The conversion of complex data structures to raw data pointers which are then passed to functions is susceptible towards errors and software bugs. The code containing the Cuda implementations was modified to fit into the main application which was primarily based on the Cloud Compare library. Some memory management methods had to be moved out of the Cuda files to avoid compilation errors which are caused by the Eigen library which is used as maths library in the PCL. These errors arise from the incompatibility between different versions of libraries and their dependencies in return. The very stable ICP variant is implemented inside the CC library and uses the internal data formats while the GMM registration algorithm is available via a plugin. The experience that implementations running in a cohesive framework often work better than more sophisticated algorithms if they are ported to a different environment is shared by Kats et al. [31].

Transferring the computation intense progress detection procedure to the graphics processing unit worked out well on the other hand. The C++ AMP-based software provides a parallel and easy to use implementation. It operates completely independent from the hardware and is, therefore, suitable for real world scenarios. No additional libraries or other dependencies are required. The time performance of the proposed progress detection procedure can still be significantly improved by reducing or rearranging the bottlenecks created by synchronized data access. Procedures which are now executed in a single loop containing atomic operations can be split into multiple loops where each thread has to share the atomic resource with fewer threads. These operations do not require intermediate synchronization with the host side memory and can be executed independently from the CPU side program. The amp language

extension even provides the functionality required to run an amp based loop on the graphics hardware and an OpenMP-based loop on the central processing unit in parallel. Arranging the program structure and the processed data in a way that allows parallel CPU and GPU processing can avoid a lot of overhead coming from transferring small amounts of data to the GPU and back for a single operation. These small implementation optimizations can result in a significant acceleration if combined all together. The actual progress detection performance is determined by the factors which are included into the triangle recognition algorithm. The choice for the uniformity of the point-to-point distance proves to be non-optimal regarding the required computations and experimental results. The combination with a simplified area coverage metric could not significantly improve the performance. Other measurement criteria could include the average nearest neighbour distance, which is quite similar to the uniform sampling distance, or evolve around circle packing. The triangle recognition metric proposed by Bosché [8] could still be applied if the constraint of being applicable for non-uniform point clouds is removed.

The experimental results show that processing more points does not necessarily lead to better alignments. The possibilities which are available to reduce the amount of occlusion in data acquired on construction sites are not connected with the ability to process larger point clouds. It is proposed to use more reliable and stable algorithms with point clouds containing fewer points, but better correspondences to solve the alignment problem instead of processing the complete point cloud. The problem of classifying building components which are not visible in the data set is better addressed by applying a more sophisticated strategy like the one described by Braun et al. [13] using the dependencies between individual components.

List of Figures

List of Tables

Bibliography

[1] B. Akinci, S. Kiziltas, E. Ergen, I. Z. Karaesmen, and F. Keceli. "Modeling and Analyzing the Impact of Technology on Data Capture and Transfer Processes at Construction Sites: A Case Study." In: *Journal of Construction Engineering and Management* 132.11 (Nov. 2006), pp. 1148–1157.

[2] O. E. Akinsiku and A. Akinsulire. "Stakeholders' Perception of the Causes and Effects of Construction Delays on Project Delivery." In: *Journal of Construction Engineering and Project Management* 2.4 (Nov. 2012), pp. 25–31.

[3] Andi and T. Minato. "Design document quality in the japanese construction industry: factors influencing and impacts on construction process." In: *International Journal of Project Management* 21.7 (Oct. 2002), pp. 537–546.

[4] K. S. Arun, T. S. Huang, and S. Blostein. "Least-Squares Fitting of Two 3-D Point Sets." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 9.5 (Sept. 1987), pp. 698–700.

[5] S. Arya and D. M. Mount. "Algorithms for fast vector quantization." In: *Data Compression Conference*. IEEE, 1993, pp. 381–390.

[6] P. J. Besl and N. D. McKay. "A Method for Registration of 3-D Shapes." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (Feb. 1992), pp. 239–256.

[7] A. Borrmann, M. Koenig, C. Koch, and J. Beetz. *Building Information Modeling*. Springer Vieweg, 2015.

[8] F. Bosché. "Automated recognition of 3D CAD model objects in laser scans and calculation of as-built dimensions for dimensional compliance control in construction." In: *Advanced Engineering Informatics* 24.1 (Jan. 2010), pp. 107–118.

[9] F. Bosché, C. T. Haas, and B. Akinci. "Automated Recognition of 3D CAD Objects in Site Laser Scans for Project 3D Status Visualization and Performance Control." In: *Journal of Computing in Civil Engineering* 23.6 (Nov. 2009), pp. 311–318.

[10] F. Bosché, C. T. Haas, and P. Murray. "Performance of automated project progress tracking with 3d data fusion." In: *Congrès annuel de la SCGC*. June 2008.

[11] A. Braun, A. Borrmann, S. Tuttas, and U. Stilla. "Towards automated construction progress monitoring using BIM-based point cloud processing." In: *Proceedings of the 10th European Conference on Product and Process Modelling*. *eWork and eBusiness in Architecture, Engineering and Construction*. Sept. 2014, pp. 101–107.

[12] A. Braun, S. Tuttas, A. Borrmann, and U. Stilla. "A concept for automated construction progress monitoring using BIM-based geometric constraints and photogrammetric point clouds." In: *Journal of Information Technology in Construction* 20.8 (Jan. 2015), pp. 68–79. ISSN: 1874-4753.

[13] A. Braun, S. Tuttas, A. Borrmann, and U. Stilla. "Automated progress monitoring based on photogrammetric point clouds and precedence relationship graphs." In: *Proceedings of the 32nd International Symposium on Automation and Robotics in Construction*. ISARC, 2015.

[14] S. Brown and J. Snoeyink. "GPU Nearest Neighbor Searches using a Minimal kd-tree." Second Workshop on Massive Data Algorithmics (MASSIVE 2010). June 2010.

[15] M.-Y. Cheng and J.-C. Chen. "Integrating barcode and GIS for monitoring construction progress." In: *Automation in Construction* 11.1 (Jan. 2002), pp. 23–33.

[16] D. H. Chung, I. D. Yun, and S. U. Lee. "Registration of multiple-range views using the reverse-calibration technique." In: *Pattern Recognition* 31.4 (June 1998), pp. 457–464.

[17] M. Delius. *Bericht zur Aufklärung der Ursachen, Konsequenzen und Verantwortung für die Kosten- und Terminüberschreitung des im Bau befindlichen Flughafens Berlin Brandenburg Willy Brandt (BER)*. Bericht des 1. Untersuchungsausschusses des Abgeordnetenhauses von Berlin - 17. Wahlperiode. Abgeordnetenhaus Berlin, 2016.

[18] B. Eckart and A. Kelly. "REM-Seg: A Robust EM Algorithm for Parallel Segmentation and Registration of Point Clouds." In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Nov. 2013, pp. 4355–4362.

[19] B. Eckart, K. Kim, A. Troccoli, A. Kelly, and J. Kautz. "MLMD: Maximum Likelihood Mixture Decoupling for Fast and Accurate Point Cloud Registration." In: *International Conference on 3D Vision*. IEEE, Oct. 2015, pp. 241–249.

[20] A. W. Fitzgibbon. "Robust registration of 2D and 3D point sets." In: *Image and Vision Computing* 21.13–14 (Dec. 2003), pp. 1145–1153.

[21] M. Golparvar-Fard, F. Peña-Mora, C. A. Arboleda, and S. Lee. "Visualization of Construction Progress Monitoring with 4D Simulation of Model Overlaid on Time-Lapsed Photographs." In: *Journal of Computing in Civil Engineering* 23.6 (Nov. 2009), pp. 391–404.

[22] M. Golparvar-Fard, F. Peña-Mora, and S. Savarese. "Automated progress monitoring using unordered daily construction photographs and IFC based Building Information Models." In: *Journal of Computing in Civil Engineering* 29.1 (Jan. 2015).

[23] G. H. Golub and C. Reinsch. "Singular value decomposition and least squares solutions." In: *Numerische Mathematik* 14.5 (Apr. 1970), pp. 403–420.

[24] P. González, V. González, K. Molenaar, and F. Orozco. "Analysis of Causes of Delay and Time Performance in Construction Projects." In: *Journal of Construction Engineering and Management* 140.1 (Jan. 2014), p. 04013027.

[25] P. Grussenmeyer, T. Landes, T. Voegtle, and K. Ringle. "Comparison Methods of Terrestrial Laser Scanning, Photogrammetry and Tachometry Data for Recording of Cultural Heritage Buildings." In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Vol. 37. ISPRS, July 2008, pp. 213–218.

[26] N. Haala. "The Landscape of Dense Image Matching Algorithms." In: *Photogrammetric Week* 13 (2013), pp. 271–284.

[27] B. K. P. Horn. "Closed-form solution of absolute orientation using unit quaternions." In: *Journal of the Optical Society of America* 4.4 (Apr. 1987), pp. 629–643.

[28] D. Huber, P. Tang, B. Akinci, R. Lipman, and A. Lytle. "Automatic reconstruction of as-built building information models from laser-scanned point clouds: A reviews of related techniques." In: *Automation in Construction* 19.7 (Nov. 2010), pp. 829–843.

[29] B. Jian and B. C. Vemuri. "Robust Point Set Registration Using Gaussian Mixture Models." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.8 (Aug. 2011), pp. 1633–1645.

[30] S. H. Al-Jibouri. "Monitoring Systems and their effectiveness for project cost control in construction." In: *International Journal of Project Management* 21.2 (Feb. 2003), pp. 145–154.

[31] A. Kats and M. McCartin-Lim. "Point Cloud Alignment Accelerated with a GPU." Apr. 2009.

[32] T. P. Kersten and M. Lindstaedt. "Generierung von 3D-Punktwolken durch kamera-basierte low-cost Systeme. Workflow und praktische Beispiele." In: *Terrestrisches Laserscanning 2012*. DVW, 2012, pp. 25–46.

[33] C. Kim, J. Lee, M. Cho, and C. Kim. "Fully automated registration of 3D CAD model with point cloud from construction site." In: *Proc. 28th International Symposium on Automation and Robotics in Construction*. July 2011, pp. 917–922.

[34] C. Kim, H. Son, and C. Kim. "Automated construction progress measurement using a 4D building information model and 3D data." In: *Automation in Construction* 31 (May 2013), pp. 75–82.

[35] C. Kim, H. Son, and C. Kim. "Fully automated registration of 3D data to a 3D CAD modelfor project progress monitoring." In: *Automation in Construction* 35 (Nov. 2013), pp. 587–594.

[36] H. Kim, C. Kim, T. Park, and H. Lim. "On-site construction management using mobile computing technology." In: *Automation in Construction* 35 (Nov. 2013), pp. 415–423.

[37] D. B. Kirk and W.-m. W. Hwu. *Programming Massively Parallel Processors*. 2nd ed. Elsevier, 2013. 496 pp.

[38] S.-W. Kwon, F. Bosché, C. Kim, C. T. Haas, and K. A. Liapi. "Fitting range data to primitives for rapid local 3D modeling using sparse range point clouds." In: *Automation in Construction* 13.1 (Jan. 2004), pp. 67–81.

[39] S.-w. Leung, S. Mak, and B. L. Lee. "Using a real-time integrated communication system to monitor the progress and quality of construction works." In: *Automation in Construction* 17.6 (Aug. 2008), pp. 749–757.

[40] Q. Li, R. Salman, E. Test, R. Strack, and V. Kecman. "GPUSVM: a comprehensive CUDA based support vector machine package." In: *Central European Journal of Computer Science* 1.4 (Dec. 2011), pp. 387–405.

[41] D. D. Lichti, S. Gordon, M. Stewart, J. Franke, and M. Tsakiri. "Comparison of Digital Photogrammetry and Laser Scanning." In: *Proc. of the CIPA WG6 Int. Workshop on scanning for cultural heritage recording*. Jan. 2002, pp. 39–44.

[42] T. Y. Lo, I. W. H. Fung, and K. C. F. Tung. "Construction Delays in Hong Kong Civil Engineering Projects." In: *Journal of Construction Engineering and Management* 132.6 (June 2006), pp. 636–649.

[43] P. E. Love, J. Matthews, S. Heinemann, R. Chandler, C. Rumsey, and O. Olatunj. "Real time progress management: Re-engineering processes for cloud-based BIM in construction." In: *Automation in Construction* 59 (July 2015), pp. 38–47.

[44] S. F. Mohamed and C. J. Anumba. "Potential for improving site management practices through knowledge management." In: *Construction Innovation* 6.4 (2006), pp. 232–249.

[45] C. Mourning, S. Nykl, H. Xu, D. Chelberg, and J. Liu. "GPU Acceleration of Robust Point Matching." In: *Advances in Visual Computing*. Lecture Notes in Computer Science 6455. Springer-Verlag, 2010, pp. 417–426.

[46] R. Navon. "Research in automated measurement of project performance indicators." In: *Automation in Construction* 16.2 (Mar. 2007), pp. 176–188.

[47] N. O. Nawari and M. Kuenstle. *Building Information Modeling. Framework for Structural Design*. CRC Press, 2015.

[48] S. El-Omari and O. Moselhi. "Integrating automated data acquisition technologies for progress reporting of construction sites." In: *Automation in Construction* 20.6 (Oct. 2011), pp. 699–705.

[49] B. N. Parlett. *The Symmetric Eigenvalue Problem*. Classics in Applied Methematics. Society for Industrial and Applied Mathematics, 1998.

[50] D. Qiu, S. May, and A. Nüchter. "GPU-Accelerated Nearest Neighbor Search for 3D Registration. 7th International Conference on Computer Vision Systems, ICVS 2009 Liège, Belgium, October 13-15, 2009. Proceedings." In: *Computer Vision Systems*. Lecture Notes in Computer Science 5815. Springer-Verlag, 2009, pp. 194–203.

[51] M. A. El-Reedy. *Construction Management for Industrial Projects. A Modular Guide for Project Managers*. 2011. Chap. Quality - From Theory to Reality, pp. 219–296.

[52] J. Reinhardt, B. Akinci, and J. H. Garrett Jr. "Navigational Models for Computer Supported Project Management Tasks on Construction Sites." In: *Journal of Computing in Civil Engineering* 18.4 (Oct. 2004), pp. 281–290.

[53] F. Remondino and T. P. Kersten. "Low-cost und open-source-Lösungen für die automatisierte Generierung von 3D-Punktwolken. Ein kritischer Überblick." In: *Terrestrisches Laserscanning 2012*. DVW, 2012, pp. 63–82.

[54] S. Rusinkiewicz and M. Levoy. "Efficient Variants of the ICP Algorithm." In: *3D Digital Imaging and Modeling*. IEEE, 2001, pp. 145–152.

[55] J. Salvi, C. Matabosch, D. Fofi, and J. Forest. "A review of recent range image registration methods with accuracy evaluation." In: *Image and Vision Computing* 25.5 (May 2007), pp. 578–596.

[56] P. J. Schneider and D. H. Eberly. *Geometric Tools for Computer Graphics*. Elsevier, 2003.

[57] D. W. Scott. *Multivariate density estimation. Theory, practice and visualization*. John Wiley & Sons, 1992.

[58] T. Tamaki, M. Abe, B. Raytchev, and K. Kaneda. "Softassign and EM-ICP on GPU." In: *First International Conference on Networking and Computing*. IEEE, Nov. 2010, pp. 179–183.

[59] K. Tan, J. Zhang, Q. Du, and X. Wang. "GPU Parallel Implementation of Support Vector Machines for Hyperspectral Image Classification." In: *Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 8.10 (Oct. 2015), pp. 4647–4656.

[60] Y. Turkan, F. Bosché, C. T. Haas, and R. Haas. "Automated progress tracking using 4D schedule and 3D sensing technologies." In: *Automation in Construction* 22 (Mar. 2012), pp. 414–421.

[61] Y. Turkan, F. Bosché, C. T. Haas, and R. Haas. "Toward Automated Earned Value Tracking Using 3D Imaging Tools." In: *Journal of Construction Engineering and Management* 139.4 (Apr. 2013), pp. 423–433.

[62] S. Tuttas, A. Braun, A. Borrmann, and U. Stilla. "Konzept zur automatischen Baufortschrittskontrolle durch Integration eines Building Information Models und photogrammetrisch erzeugten Punktwolken." In: *DGPF Tagungsband 23*. DGPF, 2014.

[63] S. Tuttas, A. Braun, A. Borrmann, and U. Stilla. "Validation of BIM components by photogrammetric point clouds for construction site monitoring." In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Vol. 2. ISPRS, Mar. 2015, pp. 231–237.

[64] J. Váncza, G. Erdős, T. Nakano, G. Horváth, and Y. Nonaka. "Recognition of complex engineering objects from large-scale point clouds." In: *CIRP Annals - Manufacturing Technology* 64.1 (2015), pp. 165–168.

[65] F. Wu, F. Wang, P. Jiang, C. Zhao, and J. Cheng. "A nearest neighbor searches algorithm for fast registration of 3D point clouds based on GPGPU." In: *International Conference on Intelligent Systems Research and Mechatronics Engineering*. Atlantis Press, 2015, pp. 2153–2158.

[66] J. K. Yates. *Global Engineering and Construction*. 2007. Chap. Global Planning and Construction Delays, pp. 214–234.

[67] A. Zabatani and A. M. Bronstein. "Parallelized algorithms for rigid surface alignment on GPU." In: *Eurographics Workshop on 3D Object Retrieval*. 2012, pp. 1–7.