Projekt 1
I^2E - Immersive Interactive Environments

Iwer Petersen

22. Oktober 2014

Inhaltsverzeichnis

1 Einführung

Diese Arbeit beschreibt die Implementation für Projekt 1 im Rahmen der I^{2}E Forschungsgruppe. Das Ziel ist, ein System zur Echtzeit-Erfassung von 3D-Modellen von Benutzern einer verteilten mixed-reality Anwendung zur kollaborativen Konstruktion von Produkten zu realisieren. Neben der Mitarbeit an der Entwicklung der Middleware für die Gesamtanwendung, war zunächst geplant einen selbständigen Prototypen für die User Reconstruction zu entwickeln.
Im folgenden werden die Anforderungen an das zu entwickelnde System. die geplanten Erweiterungen, sowie die Positionierung zur Gesamtanwendung behandelt.

2 Spezifikation

2.1 Kurzbeschreibung

Die Vision des I^{2}E Projektes ist eine Anwendung, welche Benutzern an verschiedenen Standorten erlaubt, gemeinsam in einer Mixed Reality-Umgebung mit und an 3D Modellen konstruieren zu können. Benutzer der Anwendung sollen als virtueller Avatar in der Umgebung erscheinen, um von verteilten Benutzern visuell wahrgenommen werden zu können. Abbildung 1 gibt eine schematische Übersicht über die geplanten Komponenten.

In diesem Bericht wird auf die Entwicklung der User Rekonstruktionskomponente eingegangen. Damit der Benutzer in einer dreidimensionalen Umgebung dargestellt werden kann, benötigt man ein 3D Modell welches möglichst seiner aktuellen, äußeren Erscheinung entspricht. Hierfür wird der Benutzer mittels visuellen Sensoren gescannt, um aus den erhaltenen Scandaten ein 3D Oberflächenmodell des Benutzers, zu möglichst interaktiven Frameraten zu berechnen. Damit ein möglichst vollständiges Modell des Benutzers entstehen kann, werden mehrere Sensoren verwendet.

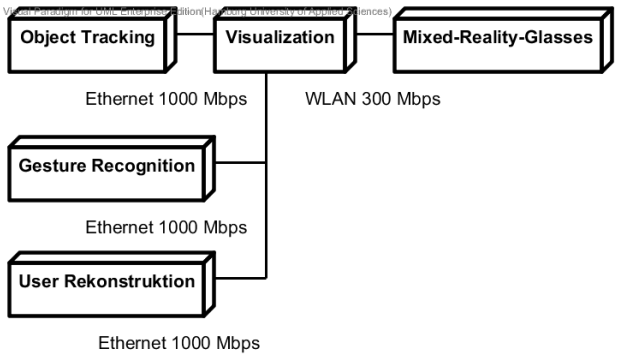

Abbildung 1: Deployment Diagramm des Gesamtsystems

2.2 Anforderungen

1. Das System unterstützt verschiedene Sensoren über ein gemeinsames Interface.
2. Das System erfasst simultan 3D Informationen aus N Sensoren.
3. Das System bietet Methoden um die 3D Informationen in ein gemeinsames Koordinatensystem zu transformieren.
4. Das System extrahiert Personen aus den 3D Informationen.
5. Das System bereinigt und fusioniert die Sensordaten und produziert ein geschlossenes 3D-Polygongitter
6. Die Verarbeitung der Sensordaten soll über definierte, austauschbare Module geschehen, um eine aussagekräftige Auswertung verschiedener Algorithmen zu erlauben.

2.3 Ausschlüsse und Abgrenzungen

1. In Abgrenzung zu Systemen, welche Motion-Capturing Verfahren zum animieren von virtuellen Avataren verwenden soll dieses System explizit Oberflächenscans von Personen zur 3D Modell Generierung verwenden.

2. In Abgrenzung zu Systemen, die ein statisches Modell in Echtzeit anfertigen können, soll dieses System die Rekonstruktionszeit für ein 3D Modell so weit reduzieren, dass ein dynamisches Modell in Echtzeit entsteht.

3. In Abgrenzung zu Phase-Shift Structured-Light Scanning Systemen welche mit sichtbarem Licht sehr hoch aufgelöste Modelle von zum Beispiel Gesichtern erstellen, soll hier zum einen nicht aktiv mit sichtbarem Licht gearbeitet werden sondern vorwiegend im infrarotem Lichtbereich. Weiterhin ist das Ziel nicht eine möglichst hohe Auflösung zu erreichen, sondern ein möglichst performantes und vollständiges 3D Modell von Menschen zu erstellen.

Folgende Ausschlüsse und Abgrenzungen besitzen nur für Projekt 1 Gültigkeit:

1. Das System unterstützt zunächst ausschließlich OpenNI2-kompatible Sensoren

2. Das System ist nicht verteilt und verwendet zunächst eine lokale Visualisierung.

3. Das System arbeitet zunächst mit einem einzelnen Sensor.

2.4 Zukünftige Erweiterungen

1. Einbindung von Sensoren über Microsoft Kinect SDK und KinectSDKv2

2. Parallele Verarbeitung der Sensordaten multipler Sensoren

3. Streaming des Meshes an eine entfernte Visualisierungskomponente

2.5 Glossar

1. Anwendung: Die von der Forschungsgruppe I^{2}E entwickelte Mixed-Reality Anwendung

2. System: Das User-Reconstruction System

3. PointCloud: Ein Array von Punkten. Kann organisiert sein, das heißt die Punktwolke kann in eine Matrixstruktur aus Zeilen und Spalten zerlegt werden.

4. Punkt: Im wesentlichen kartesische 3D Koordinaten. Können durch zusätzliche Datentupel (Farbe, Flächennormale, etc.) angereichert werden.

5. Sensor: Im Fokus liegen in dieser Arbeit vor allem Tiefenbildkameras, prinzipiell kann dies aber jedes Punktwolken generierende Verfahren darstellen.

6. Polygonmesh: Im Sinne der 3D Modellierung eine Sammlung von Punkten, Kanten und Flächen, die ein polyhedrales 3D Objekt darstellen.

3 3D Rekonstruktion

3.1 Tool Ökosystem

3.1.1 3D Datenverarbeitung

Als zentrales Werkzeug zur Verarbeitung dreidimensionaler Informationen wurde die Pointcloud Library (PCL) [1] aufgrund von vorhandenen Vorkenntnissen aus vorigen Arbeiten ausgewählt. PCL ist ein umfangreiches OpenSource C++ Framework zur Verarbeitung von Punktwolken. In mehreren Bibliotheksmodulen bietet PCL Implementierungen von gängigen Algorithmen zur Verarbeitung von räumlichen Informationen.
PCL selber verwendet ein Anzahl gängiger Bibliotheken zur Datenrepräsentation und -verarbeitung:

- Boost 1.55 [2] - für Shared Pointer
- Eigen3 [3] - Matrizen, Vektoren, Lineare Algebra
- flann [4] - Fast Library for Approximate Nearest Neighbours

[1] `http://www.poindclouds.org`
[2] `http://www.boost.org`
[3] `http://eigen.tuxfamily.org`
[4] `http://www.cs.ubc.ca/research/flann/`

- Qt [5]- GUI Framework
- vtk 6.1 [6] - Visualization ToolKit
- qHull [7] - Triangulation (Convex/Concave Hull, Delaunay, Voronoi usw.)
- OpenNI2 [8] - zur Einbindung OpenNI-kompatibler Sensoren
- CUDA [9] - für GPU Implementierungen verschiedener Algorithmen

Diese Vielzahl von Abhängigkeiten sowie der Entwicklungszyklus von PCL haben einen großen Anteil am Aufwand zum Aufsetzen der Produktionsumgebung. Da vor allem die GPU-beschleunigten Verfahren sowie die Einbindung neuerer OpenNI2-kompatibler Sensoren noch nicht im neuesten Release eingebunden sind, ist die eigene Kompilierung der Bibliothek im aktuellen Stand der Entwicklung erforderlich. Im Laufe der Projekt 1 Veranstaltung wurde dies teilweise am tagesaktuellen Entwicklungsstand von PCL durchgeführt.

3.1.2 Visualisierung

Um zunächst losgelöst von der Unity-basierten [10] Visualisierungskomponente des I^{2}E-Projekts eine Visualisierungslösung nutzen zu können, wurde ein lokales Userinterface entwickelt, welches auf openFrameworks [11] basiert. Dieses wird über das Signal-Slot System von Boost an die 3D Verarbeitung angebunden, um Parameter von Punkt- und Meshprozessoren zur Laufzeit zu verändern und Zwischenergebnisse und das Resultat der Rekonstruktion zu visualisieren. Für Steuerelemente im Userinterface werden die openFrameworks Erweiterungen ofxUI [12] und ofxXmlSettings [13] verwendet. Weiterhin werden die Erweiterungen ofxMSATimer [14] und ofxTimeMeasurements [15] verwendet um dem Zeitbedarf einzelner Schritte messen zu können.

[5] http://qt-project.org/
[6] http://www.vtk.org/
[7] http://www.qhull.org/
[8] http://structure.io/openni
[9] https://developer.nvidia.com/about-cuda
[10] http://unity3d.com/
[11] http://openframeworks.cc
[12] https://github.com/rezaali/ofxUI
[13] http://www.openframeworks.cc/documentation/ofxXmlSettings/ofxXmlSettings.html

3.2 Architektur

Die Kernfunktionalität der User-Rekonstruktion wird in der Bibliothek *libdynrecon* implementiert, welche nur Abhängigkeiten zur PCL Biliothek und deren Abhängigkeiten besitzt.

3.2.1 Sensoranbindung

Die Daten verschiedener Sensoren werden über ein gemeinsames Interface entgegengenommen. Hierfür wurde die abstrakte Klasse *AbstractPointCloudGenerator* (siehe Abbildung 2) entworfen, die eine PointCloud bereitstellt. Da damit gerechnet werden muss, dass eine angebundene Sensorkomponente in einem eigenen Thread laufen kann, wird der Zugriff auf die produzierte PointCloud mittels Mutex geschützt. Die letzte PointCloud wird in einer separaten Variable gespeichert, damit im Falle einer Lock-Kollision von Generator und Konsumententhread trotzdem nicht-blockierend eine PointCloud geliefert werden kann.

Im Moment ist mit *PclOpenNI2Grabber* (siehe Abbildung 2) eine Implementation vorhanden, welche den in PCL implementierten OpenNI2Grabber verwendet. Mit dieser Implementierung können OpenNI2-kompatible Sensoren wie die Microsoft XBOX Kinect, Primesense Carmine oder Asus Xtion verwendet werden.

Als Erweiterung ist hier zunächst ein PointCloudGenerator geplant, welcher allgemein Tiefenbilder in Punktwolken umwandeln kann. Dieser Schritt vereinfacht die zukünftige Einbindung verschiedener Sensoren die solch ein Tiefenbild liefern können. Unter anderem soll damit die Anbindung von Microsoft Kinect2SDK kompatibler Sensoren realisiert werden.

[14] https://github.com/memo/ofxMSATimer
[15] https://github.com/armadillu/ofxTimeMeasurements

3.2.2 3D Datenverarbeitung

Die Verarbeitung der Sensordaten zu Polygonmeshes wird generell in einer Pipeline realisiert. Dafür werden verschiedene Punkt- und Meshprozessoren definiert, die in unterschiedlicher Weise kombiniert werden können. Punktprozessoren nehmen eine Punktwolke entgegen, und liefern eine modifizierte Punktwolke zurück. Meshprozessoren nehmen ebenso eine Punktwolke entgegen, produzieren aber ein PolygonMesh. Um dies abzubilden wurden die abstrakten Klassen *AbstractProcessingPipeline*, *AbstractPointProcessor* und *AbstractMeshProzessor* entworfen (siehe Abbildung 2).

Im Moment existiert eine implementierte Pipeline (*Pipeline01*) welche aus einem *DepthThreshold* Punktprozessor und einem *OrganizedFastMesh-Processor* Meshprozessor besteht. Mit Hilfe des Punktprozessors wird der weiterzuverarbeitende Tiefenbereich eingegrenzt. Der Tiefenbereich wird durch einen Minimal- und Maximalwert definiert, und Punkte außerhalb dieses Intervalls werden verworfen. Hierdurch wird eine ausreichend gute Segmentierung des Benutzers vom Hinter- und Vordergrund erreicht. Die so gefilterte Pointcloud wird an den Meshprozessor weitergegeben. Der *OrganizedFastMeshProcessor* nutzt nun die Tatsache, dass die Punktwolke aus einem Tiefenbild entstanden ist, und somit organisiert ist. Das bedeutet, dass zu jedem Punkt die Nachbarpunkte wohlbekannt sind. Mit einer definierbaren Schrittweite werden aus in X-Y Ebene benachbarte Punkte zu Dreiecken verbunden. Diese Dreiecke werden dann als Vektoren aus Punktindizes geliefert, welche im Moment direkt zu Erzeugung eines openFrameworks Polygonmeshes genutzt werden. Diese Umwandlung ist momentan verantwortlich für die letzte verbliebene Abhängigkeit der Rekonstruktionsbibliothek von openFrameworks, welche in der weiteren Entwicklung ausgelagert werden wird.

In Zukunft sollen weitere Punkt- und Meshprozessoren implementiert werden, um mit ihnen verschiedene Pipelines aufzubauen und gegeneinander evaluieren zu können. Dazu wird die oben beschriebene Architektur vermutlich noch einige Anpassungen erfahren. Es zeichnet sich zum Beispiel die Notwendigkeit einer weiteren abstrakten Prozessor-Klasse zum Mesh Postprocessing ab. Über diese Klasse könnten dann Nachbearbeitungsschritte wie zum Beispiel Mesh-Glättung realisiert werden.

3.3 Zukünftige Entwicklung

3.3.1 Einbindung mehrerer Sensoren

Damit die Daten mehrerer Sensoren parallel erfasst werden können ist die Realisierung einer Sensor Management Komponente geplant. Diese soll sich um die Synchronisierung der eingehenden Daten kümmern sowie die Transformation der Daten in ein gemeinsames Koordinatensystem übernehmen.

3.3.2 Kalibrierung

Für die Überführung der Daten verschiedener Sensoren in ein gemeinsames Koordinatensystem ist die Implementierung eines Kamerakalibrierungsverfahrens erforderlich. Geplant ist hier ein möglichst automatisches Verfahren, welches aus Korrespondenzen in den Punktwolken der einzelnen Sensoren Transformationsparameter für die Daten jedes Sensors berechnet. Diese Korrespondenzen könnten zum Beispiel über die Erkennung dreidimensionaler Objekte in den Punktwolken gefunden werden.

3.3.3 Mesh-Streaming

Damit ein rekonstruiertes Mesh auch auf der zentralen Visualisierungskomponente der Projektanwendung dargestellt werden können, muss das Mesh über die Netzwerkkomponente serialisiert und gestreamt werden. Hierfür wird zunächst die C++ Implementierung des Netzwerkadapters fertiggestellt, auf den noch in Kapitel 4 eingegangen wird. Danach muss zunächst getestet werden ob sich die geplante Netzwerk Architektur der Gesamtanwendung auch für das Streaming von Polygonmeshes eignet, oder ob hierfür eine separate Lösung gefunden werden muss.

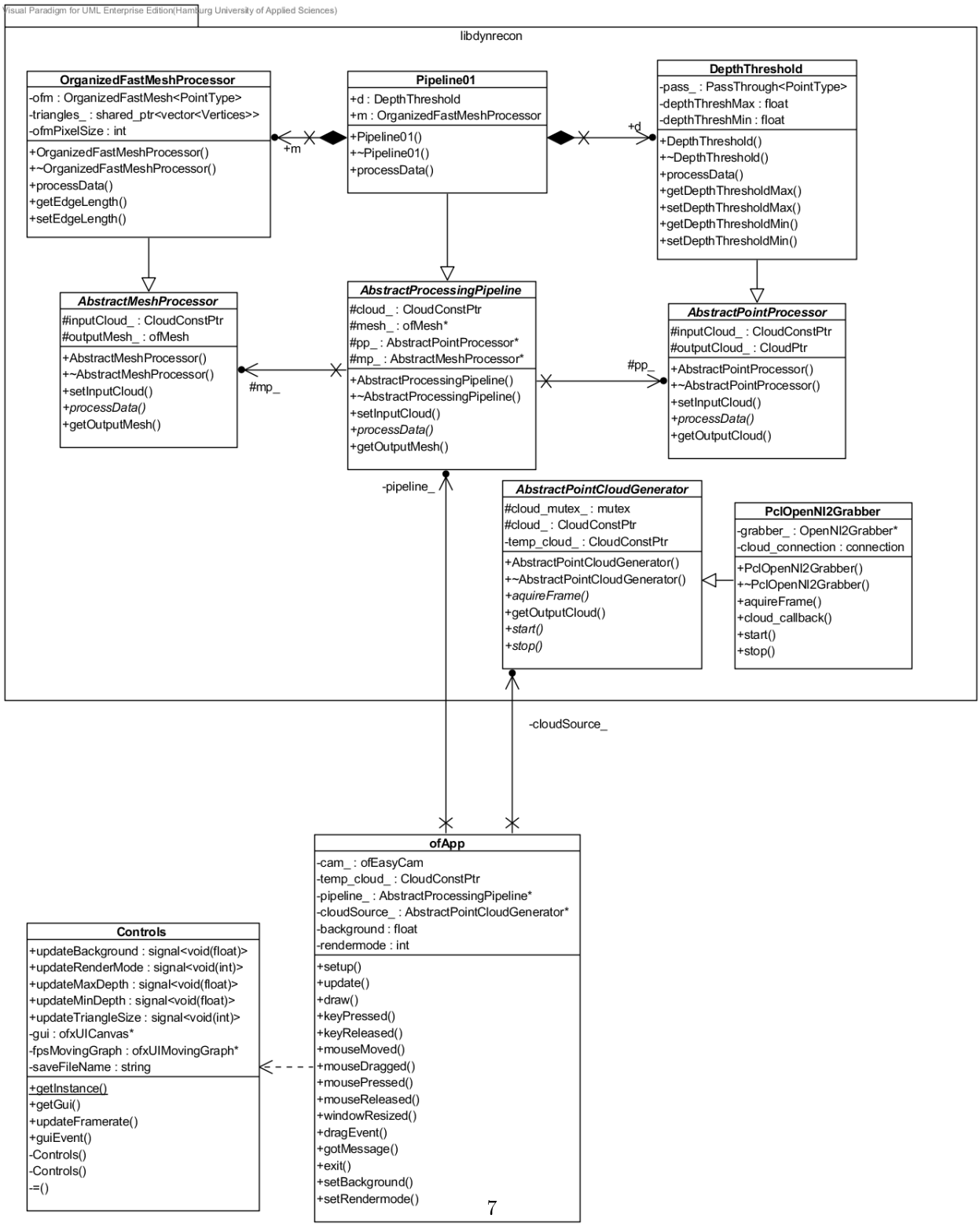

Abbildung 2: Klassen Diagramm der User-Reconstruction Komponente

4 Netzwerk Middleware

Damit einzelne Komponenten der Mixed-Reality Anwendung verteilt kommunizieren können, wird eine gemeinsame Netzwerk-Middleware entwickelt, welche auf die Serialisierungslösung Google protobuf [16] setzt. Da Implementierungen für verschiedenste Programmiersprachen existieren können wir ein geringe Kopplung zwischen einzelnen Komponenten erreichen. Hierfür muss jedoch ein Netzwerk-Adapter für jede Programmiersprache entwickelt werden, welcher den Netzwerktransport der mit protobuf serialisierten Daten übernimmt. Malte Eckhoff hat eine Referenzimplementierung in C# entwickelt, an welcher sich Implementierungen in anderen Programmiersprachen orientieren sollten.

In einer protobuf eigenen Definitionssprache werden Nachrichtentypen definiert. Aus diesen Definitionen wird für die genutzte Programmiersprache Quellcode generiert, welcher die Nachricht in ein Byte-Array serialisiert und umgekehrt ein Byte-Array in eine Nachricht de-serialisiert.
Das Byte-Array wird schließlich in eine standardisierte protobuf Nachricht verpackt und versendet. Alle weiteren Meta-Informationen, die zum deserialisieren benötigt werden, wie zum Beispiel der Nachrichtentyp, werden in diesen "Briefumschlag"geschrieben.
Es sollen prinzipiell Nachrichten mit und ohne Antwort unterstützt werden. Um bei Nachrichten die eine Antwort erfordern eine Zuordnung zur Frage zu ermöglichen, wird für solche Anfragen eine global eindeutige GUID generiert, welche eine sehr geringe Wahrscheinlichkeit hat dupliziert zu werden.
Tritt bei der Verarbeitung einer Anfrage ein Fehler auf, wird im Falle einer zu beantwortenden Frage eine Fehlernachricht zurückgeschickt. Bei Nachrichten die keiner Antwort bedürfen, wird die GUID leer gelassen, und es erfolgt auch keine Benachrichtigung im Fehlerfall.
Im weiteren wird auf die Entwicklung der C++ Implementierung des Netzwerk-Adapters eingegangen.

[16] `https://code.google.com/p/protobuf/`

4.1 Netzwerk-Adapter (C++)

Der erste Versuch den Adapter in C++, sehr nahe am Referenzdesign zu implementieren scheiterte an der Schwierigkeit Techniken, die in C# Bestandteil des Sprachstandards sind, in C++ abzubilden. Im speziellen stellten die C# Delegates eine große Schwierigkeit dar. Diese arbeiten ähnlich wie C Funktionszeiger, sind aber im Gegensatz dazu objektorientiert und typsicher. Aufgrund dieser Schwierigkeiten wurde beschlossen, dass vor allem das Interface des Netzwerkadapters der Referenzimplementierung entsprechen sollte. Für die interne Realisierung der gemeinsamen Konventionen besteht so eine größere Freiheit.

Teil von Projekt 2 wird diese erneute Implementierung des C++ Netzwerkadapters sein. Geplant ist eine Realisierung des Transports mit Hilfe der Boost::Asio Bibliothek. Für die Realisierung der Anbindung des Netzwerkadapters an eine Applikation wird die Boost::Signals Bibliothek verwendet werden. Die Applikation muss dann einen Callback-Receiver pro Nachrichtentyp beim Netzwerkadapter registrieren.

5 Fazit und Ausblick

Trotz anfänglicher Schwierigkeiten beim Einrichten der Entwicklungsumgebung ist die Umsetzung des Prototypen für die User-Reconstruction Komponente aus der Theorie in die Praxis soweit gelungen. Es ist eine aus heutiger Sicht flexible, gut verstandene Experimentierumgebung entstanden, welche sich für die weitere Arbeit am Thema Echtzeitrekonstruktion gut eignet. Der vorerst fehlgeschlagenen Versuch den Netzwerkadapter zu implementieren hat noch einmal dazu geführt, das Konzept des Netzwerkadapters zu überdenken. Dadurch wurde ein gemeinsames Verständnis des Netzwerkkommunikationsmodells der Gesamtanwendung vertieft.

Für Projekt 2 steht zunächst die Fertigstellung des C++ Netzwerkadapters auf dem Plan. Dies legt die

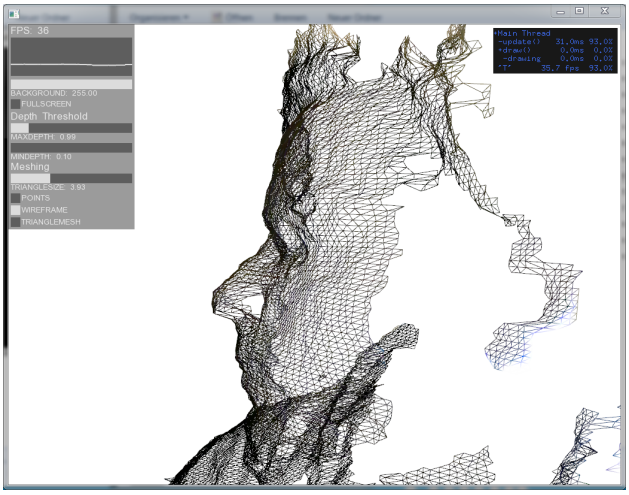

Abbildung 3: Screenshot des lokalen User Interface des entwickelten Systems mit Rekonstruktionsergebnis von *Pipeline01* als Wireframedarstellung orthogonal zur Kamerablickrichtung

Grundlage für die ersten Tests der Netzwerkkommunikation sowie für die eigentliche Verteilung der Anwendungskomponenten. Der nächste Schritt bei der Entwicklung des User Reconstruction Systems ist die Entwicklung einer Sensor Management Komponente zur Verwaltung mehrerer Sensoren sowie einer Kalibrierungskomponente. Parallel dazu wird eine zweite Entwicklungsumgebung auf der Basis von Windows 8 aufgebaut, um die Einbindung des moderneren Kinect2 Sensors zu ermöglichen.