Technischer Report - Prototypen des TP D2

Letzte Bearbeitung: 22. August 2001
Bearbeiter: Torsten Thurow

Einführung

Ziel der Prototypen

Das Teilprojekt D2 des Sonderforschungsbereiches 524, gefördert von der Deutschen Forschungsgemeinschaft, beschäftigt sich mit der computergestützten Bauaufnahme. Eine Teilaufgabe der Arbeit des TP D2 besteht dabei in der Entwicklung eines hypothetischen, computergestützten Bauaufnahmesystems. Ein Ziel der Prototypen liegt dabei in der implementativen Überprüfung von ausgesuchten Teilaspekten des Systems, seiner Verdeutlichung und der Bereitstellung einer Versuchsplattform zur Entwicklung, zum Test und Vergleich weiterer Prototypen.

Ansatz

Grundidee ist die Bereitstellung einer gemeinsamen Datenbasis, durch welche an ihr angeschlossene Prototypen miteinander verbunden werden. Die angeschlossenen Prototypen stellen somit Werkzeuge dar, die ein gemeinsames, von der Datenbasis gehaltenes digitales Modell bearbeiten. Es lassen sich so verschiedene Werkzeuge kombinieren, in das System einfügen oder aus dem System entfernen, sowie direkt miteinander vergleichen. Da jeder Prototyp ein eigenständiges Programm darstellt, können nach Vereinbarung einheitlicher Schnittstellen unabhängige Implementierungen erfolgen. Somit können beispielsweise mehrere Bearbeiter Code-unabhängig voneinander gleiche Werkzeuge erstellen. Von diesen kann im Anschluß eine beliebige Auswahl in das System übernommen werden. Die Praxis hat dabei allerdings gezeigt, das gerade die Schaffung der Schnittstellen einen wesentlichen Hauptpunkt der Arbeit darstellt.

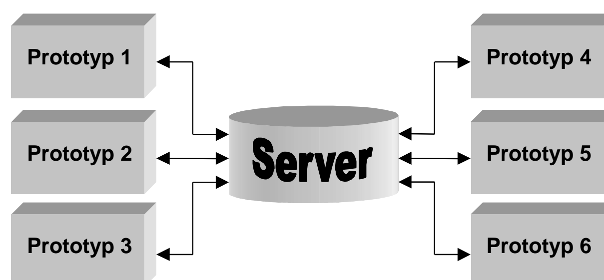

Abbildung 1: System aus Prototypen

Datenbasis

Einführung

Als Basis wurde ein Server gesucht, welcher als gemeinsame Datenbasis dient. Anforderungen an den Server sind:

- Schnelle Umsetzung (der Server ist kein Forschungsgegenstand des Teilprojektes D2, sondern nur Werkzeug für die eigentlichen Aufgaben)
- Schnelle Antwortzeiten und schneller Datentransfer (da Geometriedaten in quasi Echtzeit zu übertragen sind)
- Einfache API der Clients (geringer Einarbeitungsaufwand von Mitarbeitern, Tutoren)
- Speicherung des Inhaltes des Servers als normales File
- Importmöglichkeit des Inhaltes anderer Server
- Unterstützung von Datenstrukturen und Informationen, die zur Zeit der Implementierung des Servers nicht bekannt sind
- Unterstützung von Synchronisierungen / Signalisierungen zwischen den Clients
- Mögliche Portierbarkeit des Servers oder der Clients auf andere Plattformen

Relationale Datenbanken, wie MySQL, wurden nicht gewählt, da aus Sicht des Autoren:

1. Keine größeren Beschaffungskosten entstehen sollten.
2. Ein Netzzugang oder eine Installation und Administration der Datenbank auf einem Demonstrations- oder Arbeitsrechner nötig ist.
3. In Fällen wie MySQL nicht ein File, sondern eine Verzeichnisstruktur als persistente Speicherung dient.
4. Anfragen zunächst auf Stringbasis generiert werden müssen oder spezielle Datenbankcompiler zum Einsatz kommen (Einarbeitung, Kosten, Laufzeit)

Gewählt wurde ein Minimalansatz, bei welchem der Server nur einen minimalen Umfang von Funktionen liefert und die eigentliche Intelligenz in die Clients gelegt wurde. Dabei ist zu beachten, das in den meisten Fällen die Zugriffe zur Datenbasis nicht durch Suchabfragen geprägt sind.

Umsetzung

Der Server kommuniziert mit den Clients ausschließlich über Ports auf TCP-Basis. Dabei wurden zwei Module implementiert:

- Der eigentliche Server
- Die ihn steuernde GUI

Die Implementation des Servers erfolgte unter Visual C++ 6.0 auf Plattform-SDK-Ebene unter NT 4.0 und ist der Socket-Programmierung unter Unix sehr ähnlich. Der Server arbeitet mit den Grunddatentypen Index, String, Binärstring und Size. Index ist ein vorzeichenloser 64-Bit-Integer, welcher zur eindeutigen Adressierung von Binärstrings genutzt wird. Der Server verwaltet dabei zwei Tabellen:

- Eine Tabelle zur Zuordnung von je einem Index zu einem Binärstring.
- Eine Tabelle zur Zuordnung von je einem String zu einem Index.

Weiter unterstützt der Server das Versenden von Messages zwischen den Clients, die sich auf diese Weise synchronisieren bzw. benachrichtigen können.

Der Server kann seinen gesamten Dateninhalt als File speichern und laden. Er ist in seinen Datenstrukturen bereits so angelegt, später auch Daten eines anderen Servers importieren zu können. Auf diese Weise lassen sich später Daten eines externen Rechners netzwerk- oder filebasiert übernehmen.

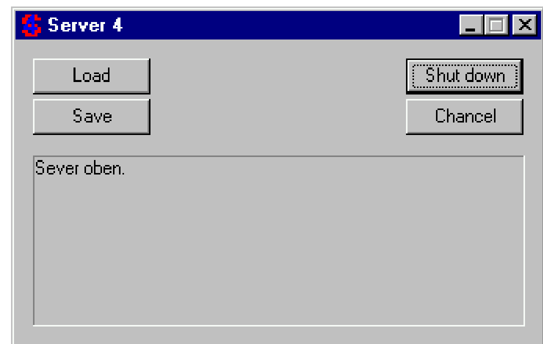

Abbildung 2: Server unter NT

Client-API für C++

Zur Anbindung der Prototypen an den Server wurde eine entsprechende Basis-API geschrieben. Diese API unterstützt das direkte Versenden und Empfangen von Binärstrings und Grunddatentypen, sowie der mit ihnen gefüllten STL-Container vector, map und multmap. Weiter unterstützt die API das Senden und Empfangen von Messages. Eine Message wird dabei als Index, also als vorzeichenloser 64-Bit Integer mit einem beliebigen, angehängten Binärstring versendet. Jeder Client kann für beliebige Messages Callback-Funktionen angeben, die im Falle einer Benachrichtigung durch einen eigenen Thread der API aufgerufen werden. Eine Übersicht zur API findet sich im Anhang.

Die unteren Diagramme zeigen Zeiten bei Testbeispielen mit Hilfe des Servers und der Client-API auf einem

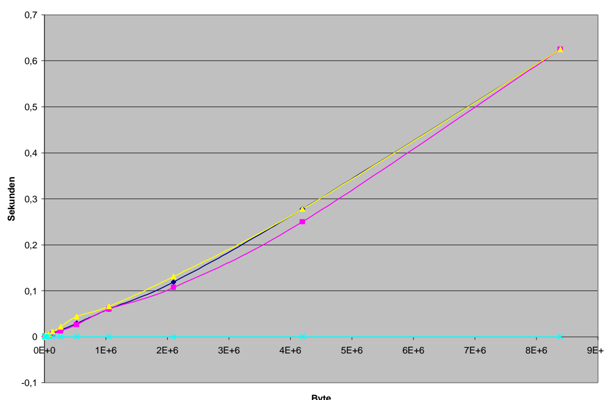

Abbildung 3: Server und Client auf dem selben Rechner

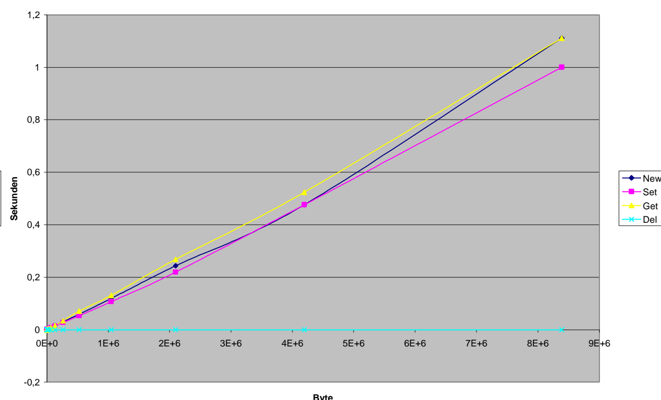

Abbildung 4: Server und Client auf verschiedenen Rechnern

und zwei Rechnern unter NT 4.0 und einem 100Mbit-Netzwerk.

Bibliothek IOBuffer

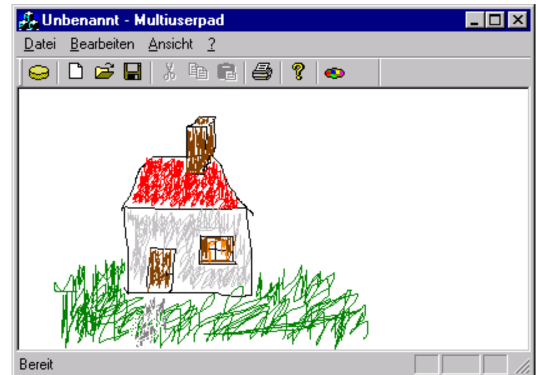

Abbildung 5: MultiUserBoard

Die Client-API unterstützt das direkte Versenden von Speicherbereichen. Es ist jedoch beispielsweise problembehaftet, eigene structs als Specherbild zu versenden, da diese durch compilertechnische Optimierungen speichermäßig größer als die Summe ihrer Grundtypen ausfallen können. Weiter lassen sich verschachtelte Containerklassen oder variable Speicherbereiche so nur sehr schwer nutzen. Darum wurde eine Serialisierungsbibliothek geschrieben, welche an die MFC-typische Serialisierung angelegt das Überführen von Datenstrukturen zu einem Binärstring und umgekehrt unterstützt. Die Bibliothek arbeitet in weiten Teilen mit Hilfe von Templates und unterstützt verschiedene Container der STL. Diese können auch ineinander verschachtelt behandelt werden. Ein sehr einfacher Typcheck dient der Erkennung von Fehlern bei Differenzen zwischen Lese- und Schreibvorgang. Die entstandenen Binärstrings können vom Nutzer zur Nutzung mittels der Client-API genutzt werden, könnten aber ebenso der Fileerstellung oder zum Auslesen eines Files dienen. Als Prototyp zur Verdeutlichung der Funktionen dienen das Programm MultiUserBoard und Chat. Sie ermöglichen mehreren Nutzern, gleichzeitig auf einem Board mit Stiften unterschiedlicher Farbe zu malen und Textnachrichten abzusetzen.

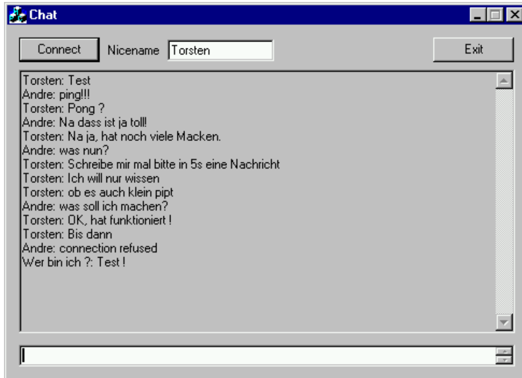

Abbildung 6: Chat

Ausgleichungskern

Der Ausgleichungskern dient der Verbindung zwischen Meßwerten, Attributen und Punktpositionen. Als Basis wurde eine C++Bibliothek für dünn besetzte Matrizen und Vektoren und den später benötigten Grundoperationen geschrieben. Darauf aufsetzend wurde eine Bibliothek für das Ausgleichung vermittelnder Messungen gesetzt. Diese arbeitet mittels einer erweiterbaren Kollektion von Objekten, welche für unterschiedliche Messungen stehen bzw. Beziehungen zwischen Punkten näher beschreiben. Der Ausgleichungskern verfügt über einen Fastsolver und verschiedene Mechanismen zu Eliminierung von Datums- und Konfiguartionsdefekten. Folgende Messungsarten und Bedingungen werden unterstützt:

Datumsdefekte:

- Zwangsfreie Netzausgleichung
- Ausgleichung unter Zwang
- Freie Netzausgleichung
  - Gesamtspurminimierung
  - Teilspurminimierung

Messungsarten:

- Distanzen
- Winkel zwischen drei Punkten
- Horizontalwinkel zwischen zwei Punkten (mit Hilfe eines Winkeloffsets für Orientierung der Bezugsachse)
- Vertikalwinkel zwischen zwei Punkten
- Tachymeter
- Winkel zwischen zwei Punktpaaren
- Photogrammetrie (Projektion von Punkten in einer Kameraaufnahme)

Bedingungen:

- Punkt liegt auf Sehne zweier Punkte
- Punkte liegen in einer Ebene
- Punkte liegen übereinander (Lot)
- Punkte liegen in einer Höhe

Prototyp Freak

Dieser Prototyp dient dem Test und der Weiterentwicklung des Ausgleichungskerns. Der Prototyp nutzt eine eigene Grammatik, in welcher Eingabefiles mit gewöhnlichen Texteditoren erstellt werden können. Mit Hilfe des Prototypes wurden und werden Tests mit reiner Tachymetrie, Tachymetrie und Photogrammetrie, reiner Photogrammetrie und zur inneren Geometriebeschreibung von vordefinierten Grundgeometrien getestet. Für das Programm „Elco-Vision" besitzt der Prototyp einen einfachen Exportmechanismus für Punkte auf ASCII-Basis.

Da der Prototyp vor der Erstellung des Servers erstellt wurde, besitzt er bisher keine Anbindung an die gemeinsame Datenbasis.

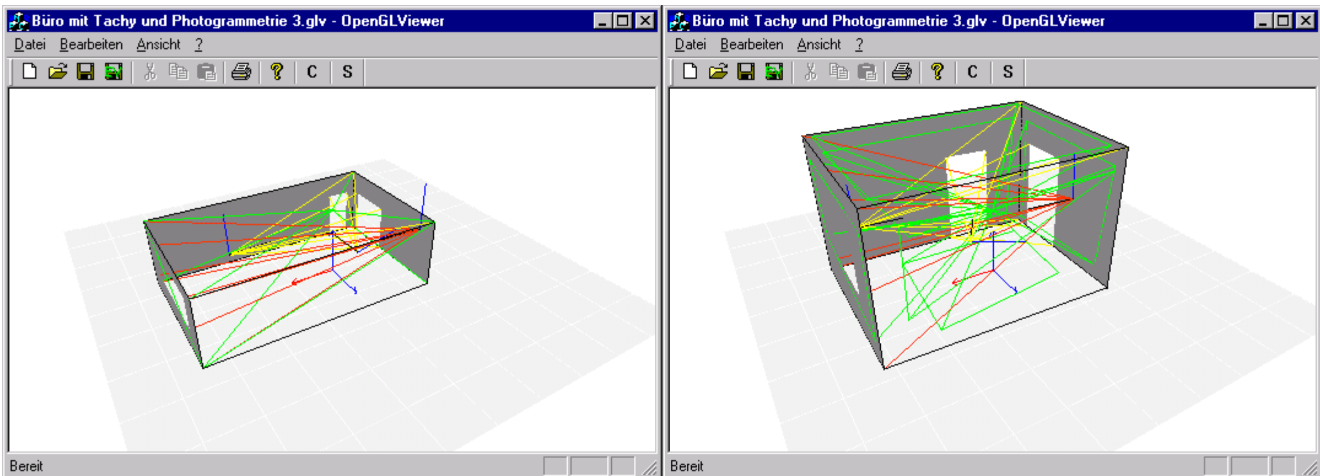

Abbildung 7: Büro, tachymetrisch und photogrammetrisch aufgenommen, vor und nach Ausgleichung

Geometrie-API

Die Geometrie-API dient der Visualisierung und der Bereitstellung von Schnittstellen für Interaktionsmechanismen ihrer internen, BRep-orientierten Datenstrukuren. Sie soll das verteilte, aber nicht gleichzeitige Arbeiten von Prototypen an einem Geometriemodell unterstützen.

Basis der API ist eine an Facettenmodelle orientierte Datenstruktur. Diese unterstützt die Geometrieelemente

- Punkt
- Linie
- Polygon

Die Elemente sind in eine topologische Datenstruktur integriert, welche aus den Elementen

- Assembly und
- Body

besteht.

Die Polygone müssen dabei keinen rendertechnischen Restriktionen, wie konkave Form, erfüllen. Sie werden als Vektor von Vektoren von Punktverweisen beschrieben, welche das Beschreiben eines beliebig begrenzten Polygons mit beliebigen Innenöffnungen erlaubt.

Weiter nutzt die API OpenGL als 3D-Render-Engine. Zur Unterstützung dienen dabei weitere Datenstrukturen für optische Oberflächeneigenschaften, Texturen und Beleuchtungen. Texturen lassen sich dabei auf die oben beschriebenen Polygone aufbringen, sie werden bei der Triangulierung entsprechend berücksichtigt.

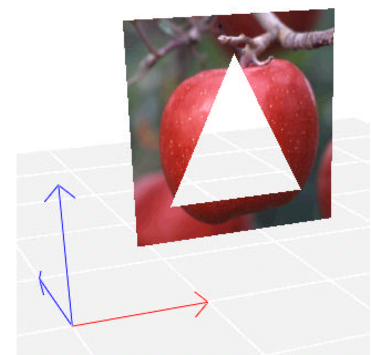

Abbildung 4: allgemeines Polygon

Eine besondere Rolle spielt die Kamera. Es wurde versucht, diese dem menschlichen Sehen unter normalen Bedingungen nachzuempfinden. Die Kamera kann in ihrer Position beliebig verändert werden. Ihre Blickrichtung läßt sich über Horizontal- und Vertikalwinkel steuern, das heißt, ihre Drehbewegungen sind der eines Theodoliten nachempfunden. Dieser Ansatz wurde gewählt, da auch der menschliche Körper analog aufgebaut ist. Ein Drehen auf dem Standpunkt hat keine Änderung der Nackenstellung zur Folge und umgekehrt.

Weiter wurde eine 2D-Visualisierung mittels MFC-2D-Funktionen begonnen, welche eine grundrißartige Repräsentation des 3D-Geometriemodells ermöglichen soll.

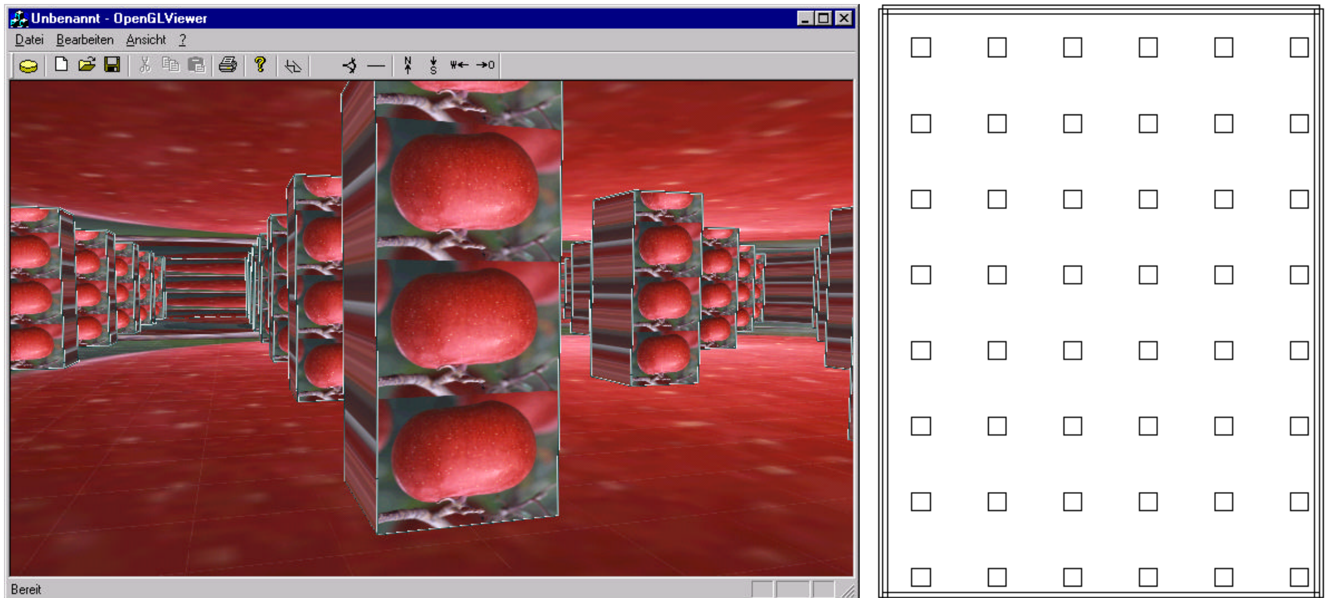

Abbildung 8: Testszene über 3D- und 2D-Viewer

Prototyp 1

Erste Umsetzung war eine API mit Hilfe des Cosmo3D-Scenengraphen. Da dieser oft in seiner API von der zugehörigen Dokumentation abwich und seine Grundphilosophie zu sehr von der gewünschten API abwich, was Adapterprogrammierungen im größeren Maße zur Folge hatte, wurde dieser Ansatz eingestellt.

Mit Hilfe des ersten Prototypen waren bereits erste Skizzierungsfunktionen im 3D möglich, wie das setzten von Punkten, verbinden mit Linien oder Polygonen. Eine Anbindung zum erst später entstandenen Minimalserver besteht nicht.

Prototyp 2

Eine neue Umsetzung der API arbeitet nur noch auf OpenGL. Sie arbeitet mit einzeln zwischen Server und Client ausgetauschten Elementen. Da bei diesem Ansatz noch nicht die Bibliothek IOBuffer existierte, wird mit direkten Speicherbildern von Grunddatenstrukturen gearbeitet, was sich als sehr störanfällig und API-seitig nutzerunfreundlich herausstellte.

Prototyp 3 und 4

Eine weitere Umsetzung nutzte bereits die Bibliothek IOBuffer, arbeitete jedoch mit einem gekapselten Geometriemodell, welches als ganzes zu übertragen ist. Dieser Ansatz ist jedoch nur dann vielversprechend, wenn Änderungen nur mittels Teilmengen des Gesamtmodells übertragen werden. Daher wurde ein neuer Ansatz gestartet, welcher die Vorteile von Ansatz 2 und 3 vereinigt.

GSI_Ansteuerung

Es wurde eine Bibliothek geschrieben, welche zur Steuerung von Leica-Tachymetern dient, welche über eine serielle Schnittstelle mit dem Rechner verbunden sind. Die Kommunikation erfolgt mittels GSI. Im Gegensatz zur GeoCOM lassen sich auch „kleinere“ und ältere Tachymeter ohne GeoCOM-Schnittstelle ansprechen sowie mehrere Tachymeter gleichzeitig an mehreren seriellen Schnittstellen steuern. Die Bibliothek ist objektorientiert gehalten, eine Instanz steht für einen Tachymeter, die Aufrufe für Messungen etc. werden über die Methoden durchgeführt. Die Bibliothek verfügt auch über einen AutoScan-Mechanismus, welcher selbstständig nach einem angeschlossenen Tachymeter und seinen Übertragungsparametern sucht.

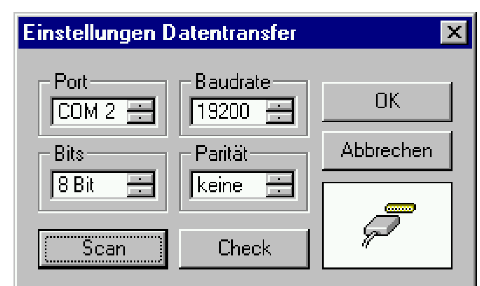

Abbildung 9: Schnittstellendialog

Anhang

Grammatik des Prototypes „Freak“

Grundaufbau ist die Syntax `Objekttyp Objektname (Paramter 1, Parameter2, ...)`. Beispielsweise wird ein Punkt vereinbart:

```
Point pt  (  0.0,  0.0,  1.5) // Für Totalstation
```

Mittels der Zeichen „//“ wird ein Kommentar bis zum Zeilenende eröffnet. Das Kommando Fixpoint besagt, das die Koordinaten des angegebenen Punktes bei der Ausgleichung nicht verändert werden.

```
Fixpoint (pt)
```

Im folgenden werden mehrere Punkte vereinbart und mittels Polygonen verbunden:

```
// Die Raumpunkte

Point p0  ( -1.5, -1.5,  0.0 )
Point p1  (  3.5, -1.5,  0.0 )
Point p2  (  3.5,  1.5,  0.0 )
...

Polygon ((p0,p4,p7,p3))
Polygon ((p2,p6,p5,p1),(p23,p22,p21,p20))
```

Es können sowohl einfach geschlossene Polygone (erstes Polygon), wie auch Polygone mit Öffnungen (zweites Polygon) generiert werden.

Ähnlich der Polygone lassen sich Linien generieren:

```
// Kanten des Raumes

Line ( p0, p1)
Line ( p1, p2)
Line ( p2, p3)
```

Bei Polygonen und Linien ist auch die Verwendung von Farben möglich:

```
// Linien von Totalstation zu ihren Meßpunkten

RGB colorTachy (0,1,0)

Line ( pt, lp1,colorTachy)
Line ( pt, lp2,colorTachy)
Line ( pt, lp3,colorTachy)
```

Für tachymetrische Messungen wird ein Tachymeter mit seinem Standort und seinen Meßgenauigkeiten horizontal, vertikal und in der Distanz vereinbart. Dieser dient als Referenz der Messungen:

```
// Tachy und Messungen

Tachy t1     ( pt, 0.01, 0.01, 0.01)

TMessure m1  ( t1, lp1,  341.7910,  70.2395, 2.654 )
TMessure m2  ( t1, lp2,   48.5475,  77.9435, 3.369 )
TMessure m3  ( t1, lp3,   48.5460, 111.8455, 3.243 )
```

Es lassen sich Punkte mit einer gewissen Genauigkeit in einer Ebene fangen:

```
// Meßpunkte den Ebenen zuordnen

Plane    pl1 ((p3,p7,p6,p2,lp1,lp2,lp3),0.1)
Plane    pl2 ((p2,p1,p5,p6,lp4,lp5,lp6,lp7,p20,p21,p22,p23),0.1)
Plane    pl3 ((p1,p0,p5,p4,lp8,lp9,lp10,lp11,p30,p31,p32,p33),0.1)
```

Ebenso sind photogrammetrische Messungen möglich. Dazu wird eine Kamera vereinbart, welche für eine Aufnahme steht. Der Aufnahme werden Messungen zugeordnet, welche besagen, welcher Punkt an welcher Position auf dem Foto abgebildet wurde.

```
// Photogrammetrie Tür

Point  pc1 (2,-1,0.5)

RGB colorCamera1 (1,1,0)

Camera cam1 (pc1,-1.55,0,1.55,1136, 1280,1024)
Pixel  px1  (cam1,p0,  271, 922)
Pixel  px2  (cam1,p4,  262,  94)
Pixel  px3  (cam1,p7, 1131, 131)
```

Auch Winkel, wie Parallelitäten zwischen zwei Strecken, lassen sich mit einer wählbaren Genauigkeit zufügen:

```
Parallel par1 ((p30,p33,p31,p32),0.1)
```

Dokumentation MyClient

Philosophie

MyClient ist eine API, welche zur Verbindung von Applikationen zu einem Server4 dient. Der Server arbeitet mit den Grunddatentypen Index, String, Binärstring und Size. Index ist ein vorzeichenloser 64bit-Integer, welcher zur eindeutigen Adressierung von Binärstrings genutzt wird. Der Server verwaltet dabei zwei Tabellen:

- Eine Tabelle zur Zuordnung von je einem Index zu einem Binärstring.
- Eine Tabelle zur Zuordnung von je einem String zu einem Index.

Definitionen

Headerdatei:

```
#define CLIENT_h
```

INDEX als vorzeichenloser 64bit-Integer:

```
typedef unsigned __int32 INDEX;
```

Funktionen

Verbindungsauf- und –abbau

`bool InitSocket  (const char * name, unsigned int port)`

Es wird über das Netzwerk (TCP-Protokoll) der Rechner name gesucht. Wurde der Rechner nicht gefunden, so wird false zurückgegeben. Ansonsten wird unter dem Port port ein Server4 gesucht und eine Verbindung aufgebaut. Konnte die Verbindung aufgebaut werden, so wird true zurückgegeben, ansonsten false. Standartwert des Portes für Server4 ist 4011.

`void CloseSocket ()`

Die Verbindung mit dem Server wird beendet.

Basisfunktionen für Binärstrings

`void New (       INDEX & i, const void * source,   const size_t & size)`

Der Binärstring source mit einer Länge size wird zum Server übertragen. Dieser vergibt für den Binärstring den neuen Index i.

`bool Set (const INDEX & i, const void * source,   const size_t & size)`

Ein alter im Server gespeicherter Binärstring mit dem Index i wird mit dem neuen Binärstring sourse, welcher eine Länge von size hat, überschrieben. Rückgabewert ist true. Wurde jedoch kein Binärstring mit dem Index i gefunden, so erfolgt kein Eintrag auf dem Server und es wird false zurückgegeben.

`bool Get (const INDEX & i,       void * & buffer,       size_t & size)`

Ein auf dem Server gespeicherter Binärstring mit dem Index i wird auf einen mit new allokierten Puffer kopiert, buffer auf dessen Adresse und size auf seine Länge gesetzt. Rückgabewert ist true. Wurde jedoch

kein Binärstring mit dem Index i gefunden, so werden buffer und size nicht verändert und es wird false zurückgegeben.

```
bool GetF (const INDEX & i,       void * buffer,   const size_t & size)
```

Auf dem Server wird nach einem Binärstring mit dem Index i gesucht. Ist dort kein Binärstring unter dem Index i abgelegt, so wird false zurückgegeben. Ansonsten wird die Länge des gefundenen Binärstrings mit size verglichen. Bei Übereinstimmung wird der Binärstring in den von buffer gezeigten Puffer kopiert und true zurückgegeben, ansonsten wird false zurückgegeben.

Schwachpunkt: Keine Unterscheidung im Fehlerfall, ob der Fehler ausgelöst wurde, weil kein Eintrag zu i vorliegt oder sich die Größen der Binärstrings unterscheiden !

```
bool GetBufferSize (const INDEX & i, size_t & s)
```

Auf dem Server wird nach einem Binärstring mit dem Index i gesucht. Ist dort kein Binärstring unter dem Index i abgelegt, so wird false zurückgegeben. Ansonsten wird die Länge des gefundenen Binärstrings in s geschrieben.

```
bool GetNextIndex (const INDEX & i, INDEX & j )
```

Auf dem Server wird nach dem ersten größeren Index als i gesucht. Ist kein höherer Index zu finden, so wird false zurückgegeben. Ansonsten wird der gefundene Index in j geschrieben. Es ist nicht relevant, ob der Index i selber auf dem Server existiert.

```
bool Del (const INDEX & i                                              )
```

Befindet sich auf dem Server ein unter dem Index i gespeicherter Binärstring, so wird dieser gelöscht, der Index i für New wieder freigegeben und true zurückgegeben. Ansonsten wird false zurückgegeben.

Basisfunktionen für Namenszuordnung

```
void Set (const char * name, const INDEX & i)
```

Auf dem Server wird unter dem Namen name der Index i gespeichert.

```
bool Get (const char * name,      INDEX & i)
```

Wurde auf dem Server unter name ein Index gespeichert, so wird dieser i zugewiesen und true zurückgegeben. Ansonsten wird i nicht verändert und false zurückgegeben.

```
bool Del (const char * name                 )
```

Wurde auf dem Server unter name ein Index gespeichert, so wird diese Zuordnung gelöscht und true zurückgegeben. Ansonsten wird false zurückgegeben.

Aufsätze für beliebige Datentypen I

```
template <class T> void New (INDEX & i, const T & value)
```

Der Speicherbereich der Variablen value wird zum Server übertragen und als Binärstring abgespeichert. Dem Binärstring wird ein neuer Index zugeordnet und in i geschrieben.

ACHTUNG ! Die Speicherung der Variablen value erfolgt so, das ihr Speicherbereich, beginnend ab ihrer Anfangsposition (this) mit der Länge sizeof(value) als Binärstring übertragen wird.

```
template <class T> bool Set (const INDEX & i, const T & value)
```

Der Speicherbereich der Variablen value wird zum Server übertragen und als Binärstring abgespeichert. Dem Binärstring wird ein neuer Index zugeordnet und in i geschrieben.

ACHTUNG ! Die Speicherung der Variablen value erfolgt so, das ihr Speicherbereich, beginnend ab ihrer Anfangsposition (this) mit der Länge sizeof(value) als Binärstring übertragen wird.

template <class T> bool Get (const INDEX & i, T & value)

Wurde auf dem Server unter dem Index i ein Binärstring gespeichert und besitzt dieser die selbe Speichergröße wie die Variable value, so wird der Speicherbereich von value mit dem Binärstring überschrieben und true zurückgegeben, ansonsten wird der Speicherbereich von value nicht verändert und false zurückgegeben.

ACHTUNG ! Das Überschreiben der Variablen value erfolgt so, das ihr Speicherbereich, beginnend ab ihrer Anfangsposition (this) mit der Länge sizeof(value) mit einem Binärstring überschrieben wird.

Aufsätze für beliebige Datentypen II

template <class T> void Set (const * name, const T & value)

Befindet sich auf dem Server eine Zuordnung von name zu einem Index, so wird unter diesem Index der Speicherbereich von value abgelegt. Ansonsten wird ein neuer Index vergeben, unter diesem der Speicherbereich von value abgelegt und der neue Index unter name gespeichert.

ACHTUNG ! Die Speicherung der Variablen value erfolgt so, das ihr Speicherbereich, beginnend ab ihrer Anfangsposition (this) mit der Länge sizeof(value) als Binärstring übertragen wird.

template <class T> bool Get (const * name, T & value)

Wurde auf dem Server unter dem Namen name ein Index abgelegt, und wurde unter dem Index ein Binärstring mit der Größe des Speicherbereiches von value eingetragen, so wird der Speicherbereich von value mit dem gefundenen Binärstring überschrieben und true zurückgegeben. Ansonsten wird der Speicherbereich von value nicht verändert und false zurückgegeben.

ACHTUNG ! Das Überschreiben der Variablen value erfolgt so, das ihr Speicherbereich, beginnend ab ihrer Anfangsposition (this) mit der Länge sizeof(value) mit einem Binärstring überschrieben wird.

Exceptions

Sämtliche Funktionen reagieren auf Probleme bei der Netzwerkkommunikation mittels der Exception NetworkError. Diese ist wie folgt aufgebaut und enthält in errorCode einen Fehlercode analog der Windows Base Service SDK-Funktion GetLastError ().

```
struct NetworkError
{
   int errorCode;

   NetworkError(const int e): errorCode(e) {};
};
```

Messagedienst

Der Messagedienst dient der Benachrichtigung der Clients untereinander. Ein Client kann Nachrichten senden und empfangen. Jede Nachricht besteht aus einem Index, der die Nachricht selber symbolisiert, und einem beliebig langen, angehangenen Binärstring.

Um eine Nachricht empfangen zu können, muß diese im Vorfeld mit einer Funktion des Typs MessageCall verknüpft werden. Wird nun die verknüpfte Nachricht empfangen, so wird die Funktion von einem eigenen Thread der Client-API aufgerufen.

typedef void MessageCall (const INDEX & i, void * buffer, size_t size)

Form für Funktionen zum Empfang von Nachrichten. Die Funktionen werden mit der Nachrichten-ID i und einem Binärstring buffer aufgerufen, der die Länge size hat. Der Binärstring muß von der Funktion selber mittels delete freigegeben werden.

void SetMessage (const INDEX & i, MessageCall * call)

Verknüpft eine Funktion call mit der Nachricht i. Empfängt die Client-API die Nachricht i, so wird call mit der Nachricht i und einem Binärsting aufgerufen.

`void DelMessage (const INDEX & i)`

Die Verknüpfung zwischen der Nachricht i und einer Funktion wird aufgehoben. Praktisch bedeutet dies, das die Nachricht i von der Client-API nicht mehr berücksichtigt wird.

`void Send1Message (const INDEX & i, const void * buffer, const size_t size)`

Sendet die Nachricht i mit dem Binärstring buffer zu allen Clients, einschließlich dem eigenen Client (Echo).

`void Send2Message (const INDEX & i, const void * buffer, const size_t size)`

Sendet die Nachricht i mit dem Binärstring buffer zu allen anderen Clients, aber nicht an den eigenen Client.

Aufsatz für Vektoren

Dieser Aufsatz unterstützt die Arbeit mit Vektoren. Dabei wird ausgenutzt, das der Standartvektor der STL alle Elemente in einem zusammenhängendem Speicherbereich gleich einem Array ablegt. Der Aufsatz behandelt diesen großen Speicherbereich als einen Binärstring.

Definitionen

Headerdatei:

```
#define VECTOR_TOOL_h


#include "Client.h"
#include <vector>
```

Funktionen

`template <class T> void NewVector (INDEX & index, const std::vector<T> & vec)`

Analog New (INDEX & index, const T & value).

`template <class T> bool SetVector (const INDEX index, const std::vector<T> & vec)`

Analog Set (const INDEX index, const T & value).

`template <class T> char GetVector (const INDEX index, std::vector<T> & vec)`

Analog Get (const INDEX index, T & value).

`inline bool DelVector (const INDEX index)`

Ruft direkt Del (index) auf. Diese Funktion dient lediglich zur Namensgleichheit der Aufrufe und kann durch Del (index) ersetzt werden.

`template <class T> char SetVector (const char * name, const std::vector<T> & vec)`

Analog Set (const char * name, const T & value).

`template <class T> char GetVector (const char * name, std::vector<T> & vec)`

Analog Get (const char * name, T & value).

Aufsatz für Sets

Dieser Aufsatz unterstützt die Arbeit mit Sets. Ein Set wird dabei in einen temporären Vektor umgewandelt, welcher analog des Aufsatzes für Vektoren behandelt wird.

Definitionen

Headerdatei:

```
#define SET_TOOL_h


#include "Vector.h"
#include <set>
```

`template <class T> void NewSet (INDEX & index, const std::set<T> & set)`

Analog New (INDEX & index, const T & value).

`template <class T> bool SetSet (const INDEX index, const std::set<T> & set)`

Analog Set (const INDEX index, const T & value).

`template <class T> char GetSet (const INDEX index, std::set<T> & set)`

Analog Get (const INDEX index, T & value).

`inline bool DelSet (const INDEX index)`

Ruft direkt Del (index) auf. Diese Funktion dient lediglich zur Namensgleichheit der Aufrufe und kann durch Del (index) ersetzt werden.

`template <class T> char SetSet (const char * name, const std::set<T> & set)`

Analog Set (const char * name, const T & value).

`template <class T> char GetSet (const char * name, std::set<T> & set)`

Analog Get (const char * name, T & value).