OptoInspect3D alg3Dlib

Schnittstellenbeschreibung

Version: 1.2

Kontakt: Dr.-Ing Christian Teutsch
Fraunhofer Institut für Fabrikbetrieb und -automatisierung
Geschäftsfeld Mess- und Prüftechnik
Sandtorstr. 22, 39106 Magdeburg
Tel: +49 391 4090-239
christian.teutsch@iff.fraunhofer.de

Datum der Dokumentation: 17. Februar 2011

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Voraussetzungen . . . 3
1.2 Optimierungen . . . 3

2 Schnittstellen **3**
2.1 Fehlercodes . . . 3
2.2 Datentypen . . . 4
2.3 Funktionen . . . 4
2.3.1 Allgemeine Funktionen . . . 4
2.3.2 I/O-Funktionen . . . 5
2.3.3 Registrierung . . . 6
2.3.4 Ausdünnung/Homogenisierung . . . 7
2.3.5 Abstandsbestimmung . . . 7
2.3.6 Regionenerkennung . . . 7
2.3.7 Geometrieapproximation . . . 8

3 Beispiele **9**

1 Einleitung

Die Funktionen der Bibliothek `OptoInspect3D alg3Dlib` implementieren praktische und hocheffiziente Algorithmen zur Verarbeitung und Analyse von 3D-Punktewolken.

Die Schnittstelle zu den internen Funktionen ist C-konform ausgelegt.

1.1 Voraussetzungen

- SSE2-kompatibler Prozessor (>=Pentium 4)
- Installation der mitgelieferten `vc_redist.exe`

1.2 Optimierungen

Eine Vielzahl der internen funktionen startet mehrere Threads, welche effizient auf mehrere CPU-Kernen verteilt werden. Die hierfür genutzte API des MS Visual Studio hat den Nachteil, dass diese internen Threads(Handles) nicht wieder gelöscht werden, wenn sie aus einem neuen Windows-Thread gestartet werden. Daher empfielt es sich, die Algorithmen in dieser Bibliothek stets aus dem selben Thread zu starten und das stete neue Anlegen von Threads zu vermeiden.

2 Schnittstellen

Die Schittstellen zu den Funktionen der Bibliothek als auch die möglichen Fehlercodes bzw. Rückgabewerte sind im Folgenden dokumentiert.

2.1 Fehlercodes

```
#define ERR_NONE          1 ///< Funktion wurde normal beendet
#define ERR_CONDITION     0 ///< Funktion wurde normal beendet aber Bedingung nicht erfüllt
#define ERR_UNKNOWN      -1 ///< Bei der Ausführung ist ein unbekannter Fehler aufgetreten
#define ERR_FCT_LOCKED   -2 ///< Funktion ist nicht freigeschaltet
#define ERR_NUM_POINTS   -3 ///< zu wenig Punkte übergeben
```

Listing 1: Definition der Fehlercodes.

2.2 Datentypen

Die Schnittstelle bietet den Datentyp **MPTpoint3d** für Vektoren mit drei Koordinaten. Zusätzliche Werte können an den Daten-Zeiger des freien Typs `MPT_USER_TYPE` gebunden werden.

```
//-----------------------------------------------------------------------------
/** Datentyp für 3D-Punkte mit anwendungsspezifischem Pointer
*/
//-----------------------------------------------------------------------------
typedef void* MPT_USER_TYPE;  ///< Zeiger auf anwenderspezifische Daten

struct MPTpoint3d
{
#ifdef __cplusplus
  typedef double value_type;                                          ///< Datentypexport
  double&       operator[](int index)       {return vec[index];} ///< Random-Access
  const double& operator[](int index) const {return vec[index];} ///< Random-Access
#endif

  double vec[3];          //z.B. 0=x,1=y,2=z
  MPT_USER_TYPE  data;    //Zeiger auf zusätzliche, anwenderspezifische Daten
};
```

Listing 2: MPTpoint3d

2.3 Funktionen

Die Schnittstelle bietet Allgemeine Funktionen, I/O-Funktionen und Algorithmen-Funktionen. Funktionen, welche Daten einlesen fordern internen Speicher an, der dann mit der Funktion `MPT_freeMemory()` wieder freigegeben werden sollte.

2.3.1 Allgemeine Funktionen

```
//-----------------------------------------------------------------------------
/** Funktion zur Ausgabe der Bibliotheksversionsinformation.
    @return  C-String mit Name, Version und Erstelldatum der Bibliothek
*/
//-----------------------------------------------------------------------------
MPT_DLL const char* __cdecl MPT_getLibraryInfo();

//-----------------------------------------------------------------------------
/** Ausgabe interner Statusmeldungen anschalten.
    @note  ist nur für Debug-Zwecke sinnvoll
*/
//-----------------------------------------------------------------------------
MPT_DLL void __cdecl MPT_enableLogOutput();
```

Listing 3: Allgemeine Funktionen

2.3.2 I/O-Funktionen

```
//-----------------------------------------------------------------------------
/** Einlesen von XYZ-Daten aus einer Datei.
    @param fname   Dateiname
    @param data    Zeiger auf Speicherbereich in den die Daten geschrieben werden
    @param numPts  Rückgabe: Anzahl der eingelesen XYZ-Daten
    @return        Fehlercode
    @note          Die Funktion legt intern Speicher an, welcher durch den Aufruf
                   von MPT_freeMemory() wieder freigegeben werden sollte.
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_readXYZ (const char* fname, struct MPTpoint3d** data,
                                 size_t* numPts);

//-----------------------------------------------------------------------------
/** Schreiben von XYZ-Daten in eine Datei.
    @param fname   Dateiname
    @param data    Zeiger auf die Daten
    @param numPts  Anzahl Daten auf die der Zeiger verweist
    @return        Fehlercode
    @note          Es werden lediglich die xyz-werte der Daten geschrieben.
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_writeXYZ(const char* fname, const struct MPTpoint3d* data,
                                 size_t numPts);

//-----------------------------------------------------------------------------
/** Funktion um Speicher freizugeben, der durch diese Bibliothek angelegt wurde.
    @param vec  Zeiger auf den freizugebenden Speicherbereich
*/
//-----------------------------------------------------------------------------
MPT_DLL void __cdecl MPT_freeMemory (struct MPTpoint3d* vec);
```

Listing 4: I/O-Funktionen

2.3.3 Registrierung

```
/** ICP Transformationen */
enum icpTransformFlags
{ tpNone      =1L<<0, ///< alle Transformationen ausschalten
  tpTranslate =1L<<1, ///< anschalten der Translation
  tpRotate    =1L<<2, ///< anschalten der Rotation
  tpScale     =1L<<3, ///< anschalten der Skalierung
  tpMatchCoG  =1L<<4  ///< anschalten der initialen Schwerpunktverschiebung
};

/** ICP Eingabe-Parameter*/
struct icpParam
{ double radius;      ///< Suchradius für die erste Iteration (0=automatisch)
  size_t iterations;  ///< maximale Anzahl Iterationen
  double tolerance;   ///< Erlaubter minimaler Abstand
  size_t samples;     ///< Punktanzahl mit der gerechnet werden soll (0=alle)
  int    trfFlags;    ///< Setzen der icpTransformFlags
};

/** ICP Ergebnis-Parameter*/
struct icpResult
{ double minDist;     ///< minimaler Punktabstand
  double maxDist;     ///< maximaler Punktabstand
  double meanDev;     ///< mittlerer Abstand zwischen den Datensätzen
  double stdDev;      ///< Standardabweichung der Punktabstände
  size_t pairs;       ///< Anzahl der korrespondierenden Datenpaare
  size_t iterations;  ///< Anzahl benötigter Iterationen
};
```

Listing 5: Ein- und Ausgabe-Parameter der ICP-Registrierung

```
//-----------------------------------------------------------------------------
/** Registrierung von zwei Punktewolken zueinander. Dabei gibt es eine statische,
    die als Referenz dient und unverändert bleibt und eine dynamische, für die
    solange Translation, Skalierung und Rotation iterativ bestimmt wird, bis ein
    (lokales) Abstandsminimum zwischen statischer und dynamischer Punktewolke
    erreicht ist oder durch Abbruchkriterien wie maximale Iterationen.
    @param ptsStat  Pointer auf den Beginn der statischen Daten
    @param nStat    Anzahl statischer Daten
    @param ptDyn    Pointer auf den Beginn der dynamischen Daten
    @param nDyn     Anzahl dynamischer Daten
    @param prm      Parameterset zum Konfigurieren des Algorithmus
    @param res      Rückgabe: Ergebnisdatenset
    @param R        Rückgabe: 3x3 Rotationsmatrix
    @param T        Rückgabe: 3x1 Translationsmatrix
    @param S        Rückgabe: 1x1 Skalierungswert
    @return         ERR_NONE bei Erfolg (eine der Bedingungen erfüllt)
                    ERR_CONDITION bei vorzeitigem Abbruch
                    Um herauszufinden welches Status der Algorithus am Ende hatte,
                    können die Ergebnisparameter befragt werden
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_calcRegistration(const struct MPTpoint3d* ptsStat, size_t nStat,
                                         struct MPTpoint3d* ptDyn  , size_t nDyn,
                                   const struct icpParam prm, struct icpResult* res,
                                   double R[9], double T[3], double S[1]);
```

Listing 6: Funktion zur ICP-Registrierung

2.3.4 Ausdünnung/Homogenisierung

```
//-----------------------------------------------------------------------------
/** Funktion zum Ausdünnen von XYZ-Daten.
    @param ptsIn          Input-Vector
    @param numIn          Anzahl Eingangsdaten
    @param radius         Suchradius
    @param numNeighbours  Mindestanzahl an Nachbarpunkten
    @param useAverage     Mittelwertbildung an/ausschalten (an=1, aus=0)
    @param ptsOut         Zeiger auf Ergebnisdaten
    @param numOut         Anzahl Ergebnisdaten
    @see                  Diese Funktion sucht für jeden Punkt aus 'ptsIn' im
                          Umkreis 'radius' Nachbarpunkte. Durch Entfernen der
                          Nachbarpunkte entsteht eine homogene Ausdünnung.
                          Durch Angabe einer Mindestanzahl an Nachbarbarn (>0)
                          können Ausreißer eliminiert werden, d.h. Punkte,
                          die keine/wenige Nachbarn werden zusätzlich entfernt.
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_calcThinning (const struct MPTpoint3d* ptsIn, size_t numIn,
                                      double radius, size_t numNeighbours, int useAverage,
                                      struct MPTpoint3d** ptsOut, size_t* numOut);
```

Listing 7: Ausdünnungs-Funktion

2.3.5 Abstandsbestimmung

```
//-----------------------------------------------------------------------------
/** Abstandsberechnung zwischen zwei Punktewolken.
    @param ptsSrc     Daten, für die Abstände bestimmt werden sollen
    @param numSrc     Anzahl Daten
    @param ptsDes     Daten, zu denen der Abstand bestimmt wird
    @param numDes     Anzahl Daten
    @param radius     maximaler Suchradius (0=auto)
    @param vNeighbors Vordimensionierter Vektor der Größe numSrc für nächste Nachbarn
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_calcDistances(const struct MPTpoint3d* ptsSrc, size_t numSrc,
                                      const struct MPTpoint3d* ptsDes, size_t numDes,
                                      double radius, struct MPTpoint3d* vNeighbors);
```

Listing 8: Abstandsfunktion

2.3.6 Regionenerkennung

```
//-----------------------------------------------------------------------------
/** Funktion zur Berechnung räumlicher Regionen.
    @param ptsIn      Eingangspunkte
    @param numIn      Anzahl Eingangspunkte
    @param radius     Abstand den ein Punkt zun einem anderen haben darf, um zur selben
                      Region zu gehören
    @param regionIdx  Indexvektor der vordimensionierten Größe numIn. Er enthält für
                      jeden Punkt aus ptsIn die Nummer der Region zu der er gehört.
    @param numIdx     Anzahl gefundener Regionen. Die erste Region hat ID 0.
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_calcRegionGrow(const struct MPTpoint3d* ptsIn, size_t numIn,
                                       double radius, size_t* regionIdx, size_t* numIdx);
```

Listing 9: Funktion zur Regionenerkennung

2.3.7 Geometrieapproximation

```
//-----------------------------------------------------------------------------
/** Funktion zur Bestimmung der Best-Fit-Ebene.
    @param ptsIn       Punkte für den best-fit
    @param numIn       Anzahl Eingabepunkte
    @param point       Rückgabe: ein Punkt auf der Ebene
    @param direction   Rückgabe: normierter Richtungsvektor
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_calcPlane(const struct MPTpoint3d* ptsIn, size_t numIn,
                                  struct MPTpoint3d* point,
                                  struct MPTpoint3d* direction);

//-----------------------------------------------------------------------------
/** Funktion zur Bestimmung des Best-fit-Zylinders.
    @param ptsIn       Punkte für den best-fit
    @param numIn       Anzahl Eingabepunkte
    @param point       Rückgabe: Ein Punkt auf der Zylinderachse
    @param direction   Rückgabe: normierter Richtungsvektor der Zylinderachse
    @param radius      Rückgabe: Radius des Zylinders
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_calcCylinder(const struct MPTpoint3d* ptsIn, size_t numIn,
                                     struct MPTpoint3d* point,
                                     struct MPTpoint3d* direction,
                                     double* radius);

//-----------------------------------------------------------------------------
/** Funktion zur Bestimmung der Best-fit-Kugel.
    @param ptsIn       Punkte für den best-fit
    @param numIn       Anzahl Eingabepunkte
    @param point       Rückgabe: Mittelpunkt
    @param radius      Rückgabe: Radius
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_calcSphere(const struct MPTpoint3d* ptsIn, size_t numIn,
                                   struct MPTpoint3d* point, double* radius);

//-----------------------------------------------------------------------------
/** Funktion zur Bestimmung des Best-Fit-Kreises.
    @param ptsIn       Punkte für den best-fit
    @param numIn       Anzahl Eingabepunkte
    @param point       Rückgabe: Mittelpunkt
    @param direction   Rückgabe: normierter Richtungsvektor für die Ebene in der
                       der Kreis liegt.
    @param radius      Rückgabe: Radius
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_calcCircle(const struct MPTpoint3d* ptsIn, size_t numIn,
                                   struct MPTpoint3d* point,
                                   struct MPTpoint3d* direction,
                                   double* radius);

//-----------------------------------------------------------------------------
/** Funktion zur Bestimmung der Best-fit-Geraden.
    @param ptsIn       Punkte für den best-fit
    @param numIn       Anzahl Eingabepunkte
    @param point       Rückgabe: Punkt auf der Geraden
    @param direction   Rückgabe: normierter Richtungsvektor
*/
//-----------------------------------------------------------------------------
MPT_DLL int __cdecl MPT_calcLine(const struct MPTpoint3d*  ptsIn, size_t numIn,
                                 struct MPTpoint3d* point,
                                 struct MPTpoint3d* direction);
```

Listing 10: Funktionen zur Geometrieapproximation

3 Beispiele

```
//-------------------------------------------------------------------------
/** Beispielfunktion für den Registrierungs-Algorithmus.
*/
//-------------------------------------------------------------------------
void test_Registration()
{
  const char* fnameStat ="daten/icpSet1.xyz";
  const char* fnameDyn  ="daten/icpSet2.xyz";
  const char* fnameOut  ="daten/icpOut.xyz";

  struct MPTpoint3d *vecStat=0,*vecDyn=0;
  size_t ptsStat=0, ptsDyn=0;
  int i=0;
  double R[9]={0,0,0,0,0,0,0,0,0},T[3]={0,0,0},S[1]={0};
  struct icpResult res;
  struct icpParam prm;
    prm.radius=100;
    prm.iterations=500;
    prm.tolerance=0.001;
    prm.samples=2000;
    prm.trfFlags=tpTranslate | tpRotate; //nur Translation und Rotation anschalten

  printf("\n***test_Registration***\n");

  if(!checkResult(MPT_readXYZ(fnameStat,&vecStat,&ptsStat))) return;
  if(!checkResult(MPT_readXYZ(fnameDyn ,&vecDyn ,&ptsDyn))) return;

  if(!checkResult(MPT_calcRegistration(vecStat,ptsStat,vecDyn,ptsDyn,prm,&res,R,T,S)))
    return;

  if(!checkResult(MPT_writeXYZ(fnameOut,vecDyn,ptsDyn))) return;

  MPT_freeMemory(vecStat);
  MPT_freeMemory(vecDyn);

  printf("minDist:%lf, maxDist:%lf, meanDev:%lf, stdDev:%lf, pairs:%d, iterations:%d\n",
         res.minDist,res.maxDist,res.meanDev,res.stdDev,res.pairs,res.iterations);

  printf("** R **");
  for(i=0;i<9;++i)
  {if(i%3==0)printf("\n");
   printf("%lf\t",R[i]);
  }printf("\n");

  printf("** T **\n");
  for(i=0;i<3;++i)
  {printf("%lf\t",T[i]);
  }printf("\n");

  printf("** S **\n");
  printf("%lf\n",S[0]);
}
```

Listing 11: Beispielfunktion zur Registrierung