

Bericht K 38-09

**Verbundprojekt HAI-TECH
Strömungsgünstige Oberflächen durch Lacksysteme**

**Teilprojekt
Numerische Berechnungen und Modellversuche zur Optimierung und
Überprüfung der Wirksamkeit reibungswiderstands-Reduzierender
Lacksysteme**

**Teilaufgabe
Entwicklung eines Simulationswerkzeugs zur schnellen Ermittlung von optimalem
Auftragsort, Auftragsrichtung und Rillenweiten für verschiedene Rumpfformen**

**Auftraggeber
Bundesministerium für Wirtschaft und Technologie**

Dokumentenkontrollblatt

Auftraggeber	: Bundesministerium für Wirtschaft und Technologie, (BMWi)
Projekt	: HAI-TECH
Projekt Nr.	: 635010
Report Nr.	: K 38-09
Berichtstitel	: Entwicklung eines Simulationswerkzeugs zur schnellen Ermittlung von optimalem Auftragsort, Auftragsrichtung und Rillenweiten für verschiedene Rumpfformen
Datei	: K38_09_AP130_ModJSch.doc

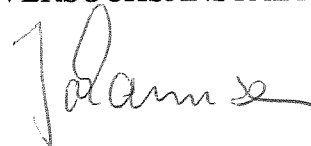
Rev. Nr.	Datum	Grund der Ausgabe	Erstellt von	Geprüft von	Genehmigt von
	22.9.2009	Teilbericht	JSch		Jo
A					
B					
C					
D					
E					

<p>Zusammenfassung:</p> <p>Der vorliegende Bericht präsentiert die von der HSVA im Rahmen des Arbeitspaketes 130, "Entwicklung eines Simulationswerkzeugs zur schnellen Ermittlung von optimalem Auftragsort, Auftragsrichtung und Rillenweiten für verschiedene Rumpfformen", entwickelte Software. Er enthält:</p> <ul style="list-style-type: none"> • einen kurzen theoretischen Hintergrund • Installationsanweisungen • eine Bedienungsanleitung • Beispiele • Ausschnitte aus der internen Quelltextdokumentation • kurze Erläuterungen über die verwendeten Datenformaten <p>Dieser Bericht beschreibt Version 1.2 der Software.</p>

<p>Schlagerworte: Bedienungsanleitung, Software, Flächengitter, Geodäte, Reibungswiderstand</p>
--

Bericht K 38-09**Verbundprojekt HAI-TECH
Strömungsgünstige Oberflächen durch Lacksysteme****Numerische Berechnungen und Modellversuche zur Optimierung und Überprüfung der Wirksamkeit reibungswiderstands-Reduzierender Lacksysteme****Teilaufgabe****Entwicklung eines Simulationswerkzeugs zur schnellen Ermittlung von optimalem Auftragsort, Auftragsrichtung und Rillenweiten für verschiedene Rumpfformen****Auftraggeber****Bundesministerium für Wirtschaft und Technologie**

Hamburg, 22.09.2009

HAMBURGISCHE SCHIFFBAU-
VERSUCHSANSTALT GmbH

ppa. Christian Johannsen

Inhalt

1	Einleitung	3
2	Theorie	3
2.1	Die triangulierte Fläche.....	3
2.1.1	Querverbindungen.....	3
2.2	Geodäten	4
2.2.1	Eigenschaften	4
2.2.2	Erzeugung von Geodäten	4
2.2.3	Der Ausgabedatei beigefügte Daten	6
3	Praktik	6
3.1	Installationsanweisungen	6
3.1.1	Erforderliche Soft- bzw. Hardware.....	6
3.1.2	Auspacken.....	7
3.1.3	Die Installation.....	7
3.2	Die Befehle und ihre Flaggen	8
3.2.1	Symbole und typographische Konventionen	8
3.2.2	geodesics	8
3.2.3	surface_mesh.....	9
3.3	Beispiele.....	10
3.3.1	Eine Sphäre	10
3.3.2	Einfluss des Beschichtungsablaufes, die flagge --sort.....	10
3.3.3	Relative Geodätenrichtungen, dargestellt mit TECPLOT und PARAVIEW.....	10
4	Literaturverzeichnis.....	11
5	Tabellenverzeichnis.....	11
6	Abbildungsverzeichnis	11
7	Abbildungen.....	12
A	Dokumentation aus dem Quelltext.....	17
A.1	surface_mesh.....	17
A.2	geodesics	24
B	Dateiformate.....	31
B.1	Eingabedateien.....	32
B.1.1	TECPLOT	32
B.1.2	STL	32
B.2	Ausgabedateien	33
B.2.1	TECPLOT	33
B.2.2	VTK	33
B.2.3	Asymptote	33
B.2.4	OBJ	33

1 Einleitung

Im Auftrag des Bundesministerium für Wirtschaft und Technologie wurde im Rahmen des Verbundprojektes HAI-TECH / Teilprojekt: “Numerische Berechnungen und Modellversuche zur Optimierung und Überprüfung der Wirksamkeit reibungswiderstandsreduzierender Lacksysteme”— Förderkennzeichen 03SX257D — eine Softwareentwicklung durchgeführt. Der vorliegende Bericht präsentiert diese von der HSVA im Rahmen des Arbeitspaketes 130, “Entwicklung eines Simulationswerkzeugs zur schnellen Ermittlung von optimalem Auftragsort, Auftragsrichtung und Rillenweiten für verschiedene Rumpfformen”, entwickelte Software.

Im Rahmen des vorherigen Arbeitspaketes, AP120 [2], wurde einerseits festgestellt, dass es einerseits Einschränkungen hinsichtlich der am Schiff realisierbaren Rillen-Orientierungen gibt, andererseits, dass die Widerstandsreduktion mit mäßigen Schräganströmungen der Rillen nicht schlagartig verschwindet. Es wurde demonstriert, dass, obwohl die Beschichtungsbahnen wegen der vorhandenen Beschichtungstechnologie nur entlang von Geodäten laufen können, durchaus akzeptable Widerstandsreduktionen erreichbar sind. Gute Ergebnisse setzten trotzdem sorgfältig geplante Beschichtungs-Bahnparameter voraus.

Mit dem hier vorgestellten EDV-Programmen `geodesics` und `surface_mesh` kann der Beschichtungsvorgang grob simuliert werden. Danach können die so entstandenen (virtuellen) Rillenrichtungen mit berechneten Strömungsrichtungen in einem Nachbearbeitungsprogramm, wie z.B. `TECPLOT` oder `PARAVIEW`, verglichen werden.

Nach den Empfehlungen in [2] geeignete Rillenbreiten können mit dem Microsoft Excel Arbeitsblatt `Rillenbreite.xls` berechnet werden. `Rillenbreite.xls` wird zusammen mit `geodesics` geliefert. Die Handhabung dieses Arbeitsblattes ist selbsterklärend und wird in diesem Bericht nicht weiter erläutert.

2 Theorie

Um die vorgestellten EDV-Werkzeuge optimal nutzen zu können, ist es sehr empfehlenswert, einiges über das Innere der Pythonmodule in `geodesics` und `surface_mesh` zu lernen. Die folgenden Abschnitte bieten eine kurze, teilweise mathematische und theoretische Einführung, die auch die wichtigsten Aspekte des Quelltextes erwähnt.

Anlage A enthält Dokumentationen, die direkt aus dem Quelltext extrahiert sind, und bietet dadurch einen tieferen Einblick in die „Mechanik“ der Programme. Ist ein noch tieferer Einblick gewünscht, bleibt noch das Lesen des reinen Quelltextes. Er enthält selbstverständlich noch mehr erläuternde Kommentare.

2.1 Die triangulierte Fläche

Die Pythonmodule `surface_mesh` enthalten die Klassen `Node`, `Edge`, `Element` und `Surface_Mesh`, die die Datenabstraktion von Ecken (Knoten), Kanten, Dreiecken bzw. der ganzen triangulierten, d.h. in Dreiecke zerlegten Fläche durchführen.

Eine triangulierte Fläche (Flächengitter mit nur dreieckigen Elementen) besteht aus Dreiecken, die null bis drei Nachbarn haben. Ein Dreieck und ein Nachbardreieck teilen zwei Eckpunkte und damit auch die Linie/Kante, die diese zwei Punkte verbindet. Eine triangulierte Fläche, die als `Surface_Mesh`-Objekt gespeichert ist, darf keine Kante haben die mehr als zwei Dreiecke verbindet. Die Fläche darf aber aus mehreren, nicht miteinander verbundenen Teilflächen bestehen.

2.1.1 Querverbindungen

Um die Nutzung dieser Bestandteile weitest möglich zu vereinfachen, enthalten alle Knoten, Kanten und Dreiecke Querverweise zu allen Teilen, mit denen sie topologisch verbunden sind (Abbildung 1a). All diese topologische Information ist zwar nicht notwendig, um ein Flächengitter zu definie-

ren, aber es vereinfacht die Entwicklung von Programmen erheblich, die sich mit dem so beschriebenen Gitter beschäftigen.

Im Normalfall existiert eine Gitterdatei, die schon Konnektivitätsinformation enthält, von denen die anderen Referenzen hergeleitet werden können. Das Unterprogramm `read_Tecplot_FEPOINT_zone` im Pythonmodul `surface_mesh` kann Gitterflächen aus Tecplot-Dateien lesen. Eine sogenannte FEPOINT Zone enthält dabei nicht nur eine Liste der Knotenkoordinaten und möglicherweise dazu gehörender Felddatenwerte, sondern auch eine Liste, welche Knoten zu einem bestimmten Element gehören (Abbildung 1b).

Eine Eingabedatei im STL-Format enthält dagegen nur eine Liste über die Eckkoordinaten jedes Dreiecks. D.h. eine STL-Datei, die N Dreieckselemente listet, enthält immer $3N$ Eckkoordinaten (mit je 3 Koordinatenkomponenten), obwohl die Anzahl der Gitterknoten höchst wahrscheinlich viel geringer ist. Das STL-Format ignoriert einfach die Tatsache, dass mehrere Dreieckselemente einen Knoten teilen können.

Das Unterprogramm `read_stl_mesh` im Pythonmodul `surface_mesh` muss deswegen von den Koordinatenwerten allein ableiten, wie viele Knoten die Datei tatsächlich enthält. Der im Augenblick verwendete Algorithmus ist weder beispiellos raffiniert noch atemberaubend schnell. Deshalb sollte die Nutzung des STL-Formates für Eingabedateien auf den Notfall oder aber auf kleine Gitter beschränkt werden.

Da alle denkbaren Querverbindungen in einer als `Surface_Mesh`-Objekt gespeicherten triangulierten Oberflächen schon vorhanden sind, ist die Aufgabe, diese Fläche in ein besonderes Datenformat umzusetzen, erheblich vereinfacht. Teilweise als Folge davon, bietet das Programm `surface_mesh` dem Benutzer viele Ausgabe-Datenformate an: Tecplot, VTK, Asymptote und Wavefront obj-Format (siehe auch Abschnitt B).

2.2 Geodäten

Die Pythonmodule `geodesics` enthalten die Klassen `Geodesic` und `Trace_Point`, die die Abstraktion einer geodätischen Kurve bzw. deren Stützpunkten sind.

2.2.1 Eigenschaften

Ein bedeutendes Problem der Differentialgeometrie ist die Suche nach dem kürzesten Pfad zwischen zwei Punkten auf einer Fläche [3]. Eine Kurve mit dieser Eigenschaft nennt man eine Geodäte. Diese Eigenschaft kommt (mathematisch gesehen) dreifach äquivalent zum Ausdruck:

- Eine Geodäte ist der kürzeste Weg zwischen zwei Punkten auf einer Fläche.¹
- Die Hauptnormale einer Geodäte ist immer parallel zum Flächennormal.
- Geodäten haben keine geodätische Krümmung.

Die dritte Eigenschaft interessiert uns besonders. Eine Kurve die keine geodätische Krümmung hat, läuft geradeaus auf der Fläche, genau wie die im Vorhaben vorgesehenen Beschichtungsgeräte.

2.2.2 Erzeugung von Geodäten

Auf einer beliebig gekrümmten Fläche ist die Lösung einer partiellen Differentialgleichung gefragt, um eine Geodäte zu berechnen [3]. Um den damit verbundenen praktischen Aufwand zu vermeiden, wurde beschlossen, sich auf triangulierte, d.h. flache Flächen zu beschränken. Dank dieser Einschränkung lassen sich Geodäten mit den folgenden Schritten erstellen:

1. Ausgangspunkt und Initialrichtung auswählen.
2. Verlängern der Geodäte bis zur nächsten Kante der aktuellen Dreiecksfacette.

¹ Diese Aussage ist nur bedingt wahr. Nur innerhalb geodätisch gesehenen konvexen Flächen ist eine frei gewählte Geodät zwischen zwei Punkten automatisch und garantiert die kürzeste [3].

3. Falls die triangulierte Fläche jenseits der im Punkt 2 erreichte Kante weiter geht und kein Grund zum Aufhören vorhanden ist, knickt die Geodäte über die Kante, kehrt zurück zu Schritt 2. und macht weiter.
4. Beginne beim Ausgangspunkt und wiederhole Schritt 2 und 3 in anderer Laufrichtung.

Die Wahl von Ausgangspunkten

Eine Geodäte, die auf einer triangulierten Fläche liegt, wird aus geraden Kurventeilen gebildet. Jeder Kurventeil läuft quer über eines der Dreiecke und an den Kanten hängt er (möglicherweise) zusammen mit anderen Kurventeilen. Da es trivial ist, eine Linie von deren Endpunkten zu bilden, werden von einem `Geodesic`-Objekt nur diese Knickpunkte entlang der Geodäte gespeichert. Die einzige denkbare Ausnahme wären die Endpunkte. Um weitere Komplikationen zu vermeiden, nicht zuletzt um die Benutzeroberfläche möglichst einfach zu halten, wurde entschieden, dass auch die Endpunkte der Geodäte auf den Kanten der triangulierten Fläche liegen müssen.

Der Benutzer wählt Ausgangspunkte indirekt durch die Angaben der Eckdaten einer Ebene, die die triangulierte Fläche durchschneidet. Die Ausgangspunkte sind dann die Schnittpunkte, an denen diese Ebene die Polygonkanten des Gitters durchschneidet (Abbildung 2).

Initialrichtung

Beim ersten Schritt der Geodätenverfolgung muss der Benutzer einen Initialrichtungsvektor angeben. Er wird bei jedem Ausgangspunkt auf die Fläche projiziert (Abbildung 3) und dient dadurch als Initialaufrichtung für jede Geodäte. Normalerweise ist es ziemlich offensichtlich, wo es lang geht, aber für einen ungeschickt gewählten Initialrichtungsvektor oder bei pathologischen Triangulierungen kann die Projizierung scheitern. Diese Art Unfall ist mit den bisher verwendeten Eingabedateien eher selten passiert. Er kann fast immer mit kleinen Justierungen der Eingabeparameter vermieden werden.

Für alle nachfolgenden Schritte der Geodätenverfolgung wird die Laufrichtung über die Kante gefaltet (Abbildung 4).

Das Verlängern der Geodäte bis zur nächsten Kante

Die oben berechnete Laufrichtung ist leider infolge des verwendeten Geometriemoduls `euclid` im 3D Raum definiert und nicht in der Fläche. Dadurch lässt sich die Geodäte nicht ohne Risiko bis zur nächsten Dreiecksseite verlängern, sie würde die gegenüberliegende Kante höchst wahrscheinlich in der dritten Dimension verfehlen.

Um sicher zu stellen, dass die Geodäte wirklich in der Fläche bleibt, wird für jeden Schritt eine Hilfsebene erzeugt, die sowohl den letzten Punkt der Geodäte, die Laufrichtung, als auch das Flächennormal enthält. Der nächste Punkt der Geodäte ist dann der Schnittpunkt zwischen der Hilfsebene und der gegenüberliegenden Kante (Abbildung 5).

Ein Abbruchkriterium

Das Programm `geodesics` erzeugt nicht nur Geodäten, es versucht einen realen Beschichtungsvorgang zu simulieren. Dass die vorgesehenen Beschichtungsgeräte nur entlang von Geodäten fahren können, ist dabei schon berücksichtigt. Außerdem ist ein Abbruchkriterium eingebaut, das der Mehrfach-Beschichtung vorbeugt.

Nach Auswahl der Ausgangspunkte werden die Geodäten aufeinanderfolgend vollständig berechnet. Eine Geodäte darf während des schrittweisen Verlängerungsvorganges keine Flächendreiecke betreten, die bereits von einer der vorherigen Geodäten gekreuzt wurden. Sobald ein bereits „lackiertes“ Dreieck erreicht ist, wird der Verlängerungsvorgang (in dieser Richtung) für die aktuelle Geodäte eingestellt.

Es wurde festgestellt, dass dieses Abbruchkriterium leicht dazu führt, dass die Geodäten sich gegenseitig sperren und nur wenige die ersten Schritte „überleben“. Deswegen hat `geodesics` die Fähigkeit, die Originalauswahl von Ausgangspunkten auszulichten. Oft wird die maximale Beschichtungsquote erreicht, wenn nur etwa jeder dritter Ausgangspunkt benutzt war.

Sortierte Beschichtungsbahnen

Der Autor dieses Berichtes, zugleich der Autor von `geodesics` und `surface_mesh` nimmt an, dass die Beschichtung systematisch abläuft, dass die nächste Bahn als unmittelbarer Nachfolger zum Vorgänger direkt nebenan niedergelegt wird. Aus diesem Grund werden die Ausgangspunkte vor der Geodätenerzeugung sortiert.

Hierbei wird ein mathematischer Ausdruck für jeden Ausgangspunkt berechnet und die Punkte werden dann je nach erreichtem Wert sortiert. Die erste Bahn/Geodäte die berechnet wird, ist die deren Ausgangspunkt am niedrigsten gewertet war.

2.2.3 Der Ausgabedatei beigefügte Daten

Falls der Nutzer ein Datenformat für die Ausgabedatei wählt, dass zusätzliche Datenfelder akzeptiert, werden ein, möglicherweise zwei, extra Datenfelder zusätzlich zu den reinen Geodätenkoordinaten zu der Ausgabedatei hinzugefügt.

Richtungswinkel der Geodäten

Wenn möglich wird der lokale Richtungswinkel „`angle`“ für jeden Geodätenpunkt in der Ausgabedatei gespeichert. Er ist im Prinzip wie im vorherigen Bericht [2] definiert, wird aber nach einer Programmüberarbeitung anders berechnet:

$$\text{angle} = \frac{180}{\pi} \arctan\left(\frac{\hat{t}_\zeta}{\hat{t}_\xi}\right), \quad \hat{\zeta} = \frac{\hat{x} \times \hat{n}}{|\hat{x} \times \hat{n}|}, \quad \hat{\xi} = \hat{\zeta} \times \hat{n} \quad (1)$$

Hier ist \hat{t} der normierte Tangentenvektor der Geodäte, \hat{n} der nach außen gerichtete Normalvektor der Rumpffläche und \hat{x} der erste Basisvektor des Koordinatensystems. Außerdem wird in (1) $\arctan(a/b)$ mit der Standardfunktion `atan2(a, b)` der Programmiersprache C berechnet. Dadurch wird $-180^\circ < \text{angle} \leq 180^\circ$, so dass `angle` Sprünge von 360° aufweisen kann.

Falls solche Sprünge aufgrund der Wahl der Geodätenrichtung, der Definition des Koordinatensystems u.ä. zu häufig oder an ungünstigen Stellen auftreten, kann `geodesics` dazu instruiert werden, dass `atan2(- \hat{t}_ζ , - \hat{t}_ξ)` statt `atan2(\hat{t}_ζ , \hat{t}_ξ)` berechnet wird (siehe Abschnitt 3.2.2).

Richtungswinkel der Strömung

Falls die Datenfelder „U“, „V“ und „W“ in der Eingabedatei vorhanden sind, werden diese wie Komponenten der Strömung behandelt. Dann wird ein auf der Strömung basierender Richtungswinkel berechnet. Dies geschieht wie in (1), um einen Richtungsvergleich zwischen Strömung und Geodäten zu ermöglichen. Hierbei wird $\hat{t} = (U\hat{x} + V\hat{y} + W\hat{z}) / |U\hat{x} + V\hat{y} + W\hat{z}|$ benutzt. In der Ausgabedatei wird das entsprechende Datenfeld mit „`flow_angle`“ bezeichnet.

3 Praktik

3.1 Installationsanweisungen

3.1.1 Erforderliche Soft- bzw. Hardware

Die Anforderungen von `geodesics`, `surface_mesh` und deren Installation

Bisher ist `geodesics` und `surface_mesh` nur auf einer kleinen Auswahl von Rechnern und Betriebssystemen getestet worden². Es handelt sich aber um reine Pythonprogramme ohne exotische Verschachtelungen und daher können wir erwarten, dass sie „überall“ laufen. Was Python betrifft, ist die Entwicklung unter Pythonversion 2.5 und 2.6 durchgeführt werden. Die Programme sollten

² Linux, Windows XP, 32-Bit Intel x86, 64-Bit AMD

— notfalls nach kleinen Anpassungsarbeiten — auch mit älteren Pythonversionen funktionieren. Die Verfügbarkeit von Python im System kann z.B. mit dem Befehl `python --version` getestet werden. Python kann bei der URL <http://www.python.org> heruntergeladen werden.

Das Pythonmodul `euclid` wird benutzt, ist aber mitgeliefert. Für weitere Information und neuere Versionen, siehe <http://partiallydisassembled.net/euclid.html> oder <http://pypi.python.org>.

Die Software wurde mit Hilfe des Pythonwerkzeuges `setuptools` verpackt und ist installiert. Normalerweise muss `setuptools` nicht vorhanden sein, da die gelieferte Software das Selbstladeprogramm `ez_setup.py` enthält. Bei der Installation wird es `setuptools` via Internet holen. Sollte der aktuelle Rechner keinen Kontakt zum Internet haben, muss `setuptools` manuell (mit einem Rechner mit Internetkontakt) entweder von <http://pypi.python.org> oder von <http://peak.telecommunity.com/DevCenter/setuptools> geholt werden.

Notwendige Vorarbeiten

Ohne die entsprechenden Eingabedateien ist die Nutzung von `geodesics` und `surface_mesh` unmöglich. Die triangulierte Fläche kann als eine FEPOINT Zone mit Triangelementen in einer ASCII TECPLOT Datei, oder als ein STL-Datei eingelesen werden. Bezüglich des TECPLOT-Formates wurde `surface_mesh` so geschrieben, dass es die TECPLOT-Datei `sh2tec.dat`, die das Potentialströmungsprogramm `v-Shallo` erzeugt, inklusive der Strömungsfelder lesen kann.

Das Programm TECPLOT wird von Tecplot Incorporated³ vermarktet und `v-Shallo` von der Hamurgischen Schiffbau-Versuchsanstalt GmbH⁴.

Was das STL-Format betrifft, können Dateien in diesem Format mit dem Gitterprogramm ICEM-CFD erzeugt werden. Da dieses Format keine Querverbindungsinformation enthält, muss das Einleseprogramm `surface_mesh` ziemlich aufwendig hergeleitet werden. Daher ist von der Nutzung dieses Formates abzuraten.

Sowohl das STL-Format, als auch das TECPLOT-Format ist relativ einfach und könnte auch von anderen Programmen produziert werden (siehe Abschnitt B).

Notwendige Nachbearbeiten

Um die von `geodesics` berechneten Geodäten bequem studieren zu können, ist ein so genannter Postprozessor notwendig. Sowohl `geodesics` als auch `surface_mesh` können TECPLOT-Dateien liefern. Um den Benutzer aber nicht zum Kauf von TECPLOT zu zwingen, können auch VTK-Dateien erzeugt werden. VTK-Dateien können z.B. mit dem „open source“ Programm PARAVIEW⁵ studiert werden.

Dateien in dem Asymptote-Format und WaveFront OBJ-Format können nur geometrische Information enthalten. Dadurch eignen sie sich nur für bestimmte Zwecke. Beide Formate können von dem Programm Asymptote benutzt werden und mehrere der Abbildungen in diesem Bericht wurden damit erstellt.

3.1.2 Auspacken

Die Software wird als eine ZIP-Datei oder als eine komprimierte TAR-Datei geliefert. Beim Auspacken wird ein Verzeichnis `geodesics-X.Y` erzeugt, welches alles enthält. X und Y ist hierbei die Versions- bzw. Ausgabennummer der Software.

3.1.3 Die Installation

Die Software wurde, wie erwähnt, mit Hilfe des Pythonwerkzeug `setuptools` verpackt. Das, und das mitgelieferte Selbstladeprogramm `ez_install.py` vereinfachen die Installation erheblich. Falls `python` und eine Verbindung mit dem Internet vorhanden sind, kann die Installation aus dem

³ <http://www.tecplot.com>

⁴ <http://www.hsva.de>

⁵ <http://www.paraview.org>

Verzeichnis `geodesics-X.Y` mit dem Befehl „`python setup.py install`“ vollzogen werden. Während der Installation wird folgendes passieren:

- Das Werkzeug `setuptools` wird heruntergeladen und installiert.
- Die Pythonmodule `geodesics`, `surface_mesh` und `euclid` werden in dem Python-Verzeichnis `site-packages` installiert, so dass diese Module zu anderen Pythonprogrammen importiert werden können.
- Die Skripte `geodesics` und `surface_mesh` werden betriebssystemabhängig passend erzeugt und installiert, so dass sie direkt von der Befehlszeile aufrufbar sind.

Der Installateur muss unbedingt die entsprechenden Befugnisse besitzen, um Dateien im System installieren zu können. Der Erfolg der Installation kann z.B. mit dem Befehl „`geodesics --help`“ überprüft werden.

3.2 Die Befehle und ihre Flaggen

Die Dateien `geodesics.py` und `surface_mesh.py` sind sogenannte Pythonmodule, d.h. Sammlungen von Klassen und Funktionen, die man zu anderen Pythonprogrammen importieren und nutzen kann. Beide enthalten dazu je ein Hauptprogramm `main` und eine Benutzeroberfläche `command_line_interface`, die zum Einsatz kommen wenn die Dateien als selbständige Programme aufgerufen werden. Die kommenden Abschnitte beschreiben die Details dieser Oberflächen.

3.2.1 Symbole und typographische Konventionen

In den zwei Abschnitten unten, werden folgende Symbole und typographische Konventionen benutzt:

[...] Was zwischen den eckigen Klammern steht, ist eine Kann-Eingabe.

⟨Beispiel⟩ Weder der Text zwischen den Winkelklammern noch die Klammern selbst, sollen buchstäblich eingetippt werden. Sie sind nur Platzhalter für einen Wert.

n eine Ganzzahl

v Drei Komponenten eines Vektors sind gefragt. Diese sollten mit Leerzeichen voneinander getrennt werden (z.B.: `1 0 0`)

s Eine Zeichenkette ist gefragt. Falls sie Leerzeichen oder „-“ enthält, fordern viele Kommandozeilen-Interpreter, dass sie z.B. in Anführungszeichen gesetzt wird.

w Ein Wort aus einer Liste ist gefragt.

3.2.2 geodesics

Das Programm `geodesics` wird mit folgendem Befehl aufgerufen:

```
geodesics [<Flaggen>] <Eingabedatei> <Ausgabedatei>
```

Die zwei Dateinamen sind normalerweise Pflichtangaben, sind aber nicht gefragt, wenn entweder die Flagge `--help` oder `--version` gegeben ist. Die Flaggen und ihre Argumente werden in Tabelle 1 beschrieben.

Tabelle 1: Flaggen für die `geodesics`-Befehlszeile.

kurz	lang	Wert	Beschreibung
-h	--help		Eine kurze Beschreibung, wie man das Programm betätigt, wird am Bildschirm ausgegeben.
	--version		Das Programm antwortet mit der Versionsnummer der

			Datei geodesic.py. Nicht die offizielle Zahl mit Ganzzahlteil und Zehntel, sondern die im Versionskontrollsystem (Subversion) benutzte Ganzzahl. Sie sollte immer erwähnt werden in den Fehlerberichten an den zuständigen Entwickler.
-i	--input_format	w	Das Datenformat der Gitterdatei: tecplot oder stl (Ausgangswert = tecplot)
-o	--output_format	w	Das Datenformat der Gitterdatei: tecplot, vtk oder asy (Ausgangswert = tecplot)
-p	--point	v	Die Koordinate eines Punktes auf der Schnittebene, siehe Abschnitt 2.2.2 (Ausgangswert = 0 0 0)
-n	--normal	v	Ein Normalvektor der Schnittebene (Ausgangswert = 1 0 0)
-t	--tangent	v	Der Initialrichtungsvektor für die Geodäten. (Ausgangswert = 1 0 0)
-s	--skip	n	Nur jeder <i>n</i> -te Ausgangspunkt wird benutzt. (Ausgangswert = 1)
	--sort	s	Der Ausdruck <i>s</i> sollte ein korrekter Pythonsatz sein. Er wird benutzt um die Ausgangspunkte zu sortieren. Hierbei ist „p“ der Name des Punktes, „x“, „y“ und „z“. Abkürzungen für bzw. „p.x“, „p.y“ und „p.z“, und alle Funktionen aus Standardpython und dem Pythonmodul math dürfen benutzt werden. Ein niedriger Wert sichert eine frühere Berechnung. (Ausgangswert = „-(y + z)“)
-f	--flip_angle		Die Winkelangabe „angle“ in der Ausgabedatei wird durch 180° rotiert (siehe auch Abschnitt 2.2.3).

3.2.3 surface_mesh

Das Programm `surface_mesh` wird mit folgendem Befehl aufgerufen:

```
surface_mesh [<Flaggen>] <Eingabedatei> <Ausgabedatei>
```

Die zwei Dateinamen sind normalerweise Pflichtangaben, sind aber nicht gefragt, wenn entweder die Flagge `--help` oder `--version` gegeben ist. Die Flaggen und ihre Argumente werden in Tabelle 2 beschrieben.

Tabelle 2: Flaggen für die `surface_mesh`-Befehlszeile.

kurz	lang	Wert	Beschreibung
-h	--help		Eine kurze Beschreibung, wie man das Programm betätigt, wird am Bildschirm ausgegeben.
	--version		Das Programm antwortet mit der Versionsnummer der Datei geodesic.py. Nicht die offizielle Zahl mit Ganzzahlteil und Zehntel, sondern die im Versionkontrollsystem (Subversion) benutzte Ganzzahl. Sollte in den Fehlerberichten an den zuständigen Entwickler immer erwähnt werden.
-i	--input_format	w	Das Datenformat der Gitterdatei: tecplot oder stl (Ausgangswert = tecplot)

-o	--output_format	w	Das Datenformat der Gitterdatei: tecplot, vtk oder asy (Ausgangswert = tecplot)
-z	--zone	s	Name der Zone in der TECPLOT-Datei, Eingabedatei und auch Ausgabedatei. (Ausgangswert = cp_tri12)

3.3 Beispiele

3.3.1 Eine Sphäre

Der Einfluss der Flagge `--skip` lässt sich schnell mit der mitgelieferten Eingabedatei `sphere_mesh.dat` demonstrieren. Die Geodäten in Abbildung 6 a) und b) wurden mit dem Befehl

```
geodesics -p 0.1 0 0 -o asy sphere_mesh.dat sphere_A.asy
```

bzw.

```
geodesics -p 0.1 0 0 -s 3 -o asy sphere_mesh.dat sphere_B.asy
```

berechnet.

In dem ersten Fall wurden 78% der Dreiecksfacetten von 169 Geodäten durchquert, in dem letzten Fall 62% von nur 57. Es ist offensichtlich, dass diese 57 Geodäten durchschnittlich viel länger, „effizienter“, waren als die 169 Geodäten in dem ersten Fall. Oft erreicht man sogar einen höheren Deckungsprozentsatz, wenn man nur jeden dritten oder vierten Ausgangspunkt nutzt.

3.3.2 Einfluss des Beschichtungsablaufes, die flagge `--sort`

Die Geodäten werden nacheinander berechnet und können sich nicht kreuzen. Die Reihenfolge, in der die Geodäten berechnet werden, hat daher einen großen Einfluss auf das endgültige Ergebnis. Die Geodäten werden durch ihre Ausgangspunkte sortiert und die Ausgangspunkte mithilfe einer Bewertungsfunktion. Die Geodäte mit einem Ausgangspunkt, der in der Funktion den niedrigsten Wert liefert, wird als erste berechnet. Die Definition dieser Funktion ist frei wählbar mit der Flagge `--sort`.

Die Beispiele in Abbildung 7 wurden mit `--sort "y+z"` und `--sort "-(y+z)"` berechnet. Die Geodäten sind dadurch vom Kiel nach oben, bzw. von oben zum Kiel zustande gekommen.

3.3.3 Relative Geodätenrichtungen, dargestellt mit TECPLOT und PARAVIEW

Als Eingabedatei für dieses Beispiel wurde eine Ausgabedatei von v-Shallo benutzt⁶. Der Befehl, der die Geodäten in Abbildung 8 produzierte, war:

```
geodesics -p 90.2 0 0 -t -1 0 0 -s 5 sh2tec.dat \
        ship_geodesics.dat
```

Der Initialrichtungsvektor für die Geodäten wurde mit Flagge `-t` zu $-\hat{x}$ festgelegt, um einen reibungslosen Vergleich zwischen den Strömungs- und Geodätenrichtungen zu sichern. Der Differenzwinkel müsste innerhalb TECPLOT mit dem Ausdruck „{angle difference} = {angle} - {flow_angle}“ in „Data → Alter → Specify Equations... → Equation(s)“ berechnet werden. In dieser Differenzwinkelberechnung ist eine „Korrektur“ von $+90^\circ$ oder -90° notwendig, falls die Rillen quer über den Beschichtungsbahnen (d.h. den Geodäten) liegen.

Die gleiche Art Datendarstellung kann auch mit PARAVIEW erstellt werden. VTK-Dateien für PARAVIEW können z.B. mit den Befehlen

```
geodesics -p 90.2 0 0 -t -1 0 0 -s 5 -o vtk sh2tec.dat \
        geodesics.vtk
```

und

⁶ Die Datei `sh2tec.dat`, nicht erhältlich als Beispielsdatei. Sie ist eine von den Standarddateien die v-Shallo erzeugt.

```
surface_mesh -o vtk sh2tec.dat nu-Shallo_data.vtk
```

erzeugt werden. Hier wurde `surface_mesh` nur als ein Übersetzungsprogramm benutzt. Der Differenzwinkel kann dann in PARAVIEW mit dem Taschenrechner-Werkzeug berechnet werden (siehe Abbildung 9).

4 Literaturverzeichnis

- 1 (anon.), *File Formats for VTK 4.2*, Auszug aus *The VTK User's Guide*, zugänglich an der URL <http://www.vtk.org/VTK/img/file-formats.pdf>
- 2 Schön, J., *Ermittlung optimaler Riblet-Geometrien hinsichtlich der Strömungsverhältnisse*, HSVA Bericht K 27-09.
- 3 Stoker, J. J., *Differential Geometry*, John Wiley & Sons, New York, 1989, ISBN 0-471-50403-3

5 Tabellenverzeichnis

Flaggen für die <code>geodesics</code> -Befehlszeile.	8
Flaggen für die <code>surface_mesh</code> -Befehlszeile.	9

6 Abbildungsverzeichnis

Querverbindungen zwischen Knoten, Kanten und Elementen, a) innerhalb eines <code>Surface_Mesh</code> -Objektes, b) in einer <code>TECPLOT FEPOINT</code> -Zone.....	12
Die Ausgangspunkte sind die Schnittpunkte, an denen eine Ebene die Polygonkanten trifft.	12
Projektion des Initialrichtungsvektors.....	13
Die Laufrichtung wird über die Kante gefaltet.	13
Eine Ebene wird benutzt, um den nächsten Punkt zu finden.	14
Der Einfluss der Flagge <code>--skip</code> : a) <code>--skip 1</code> , b) <code>--skip 3</code>	14
Einfluss von Geodätensortierung, von unten nach oben (a) und von oben nach unten (b).	15
Winkeldifferenzen zwischen Geodäten und Strömung, berechnet mit <code>v-Shallo</code>	15
Die gleichen Geodäten wie in der Abbildung 8, aber diesmal mit <code>PARAVIEW</code>	16

7 Abbildungen

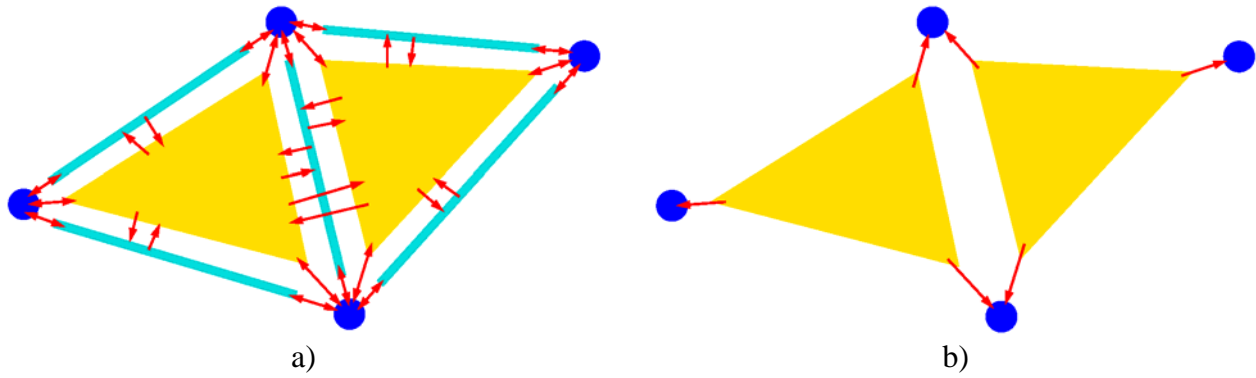


Abbildung 1: Querverbindungen zwischen Knoten, Kanten und Elementen, a) innerhalb eines `Surface_Mesh`-Objektes, b) in einer `TECPLOT FEPOINT`-Zone

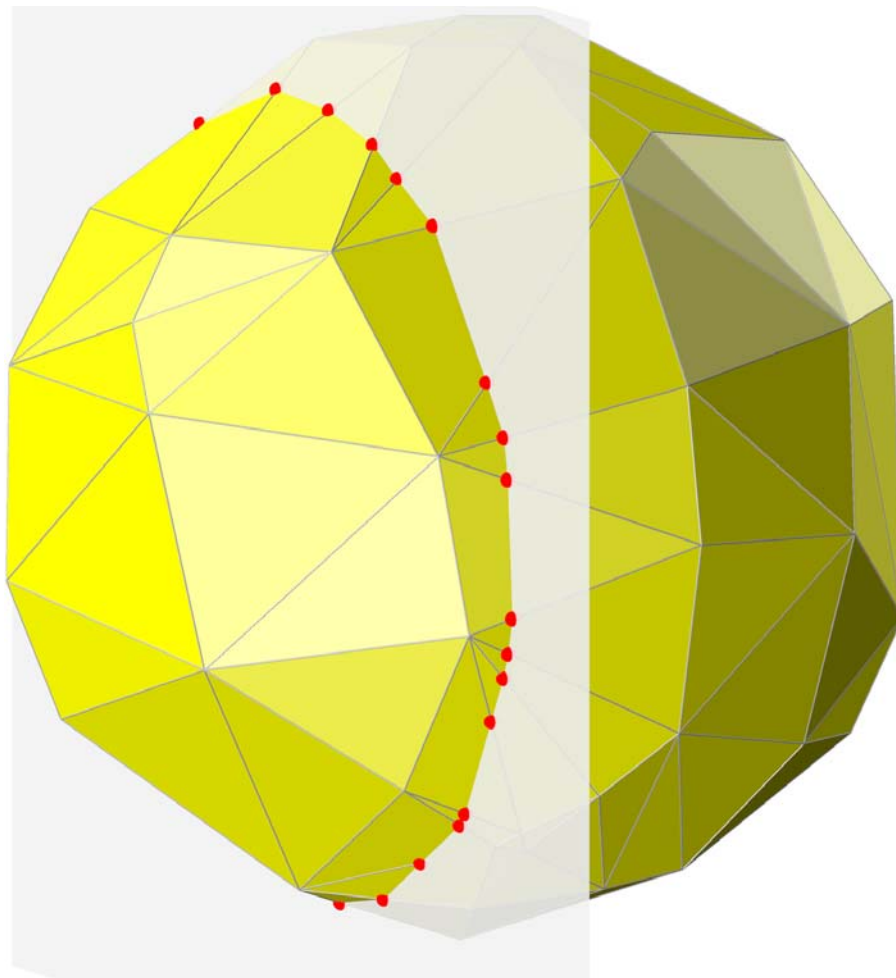


Abbildung 2: Die Ausgangspunkte sind die Schnittpunkte, an denen eine Ebene die Polygonkanten trifft.

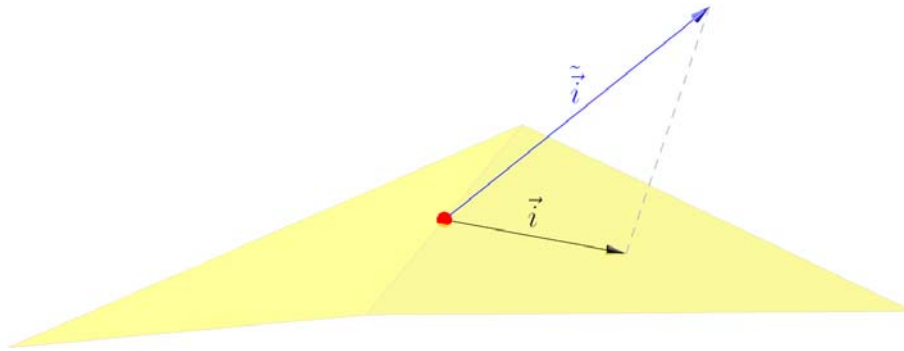


Abbildung 3: Projektion des Initalrichtungsvektors.

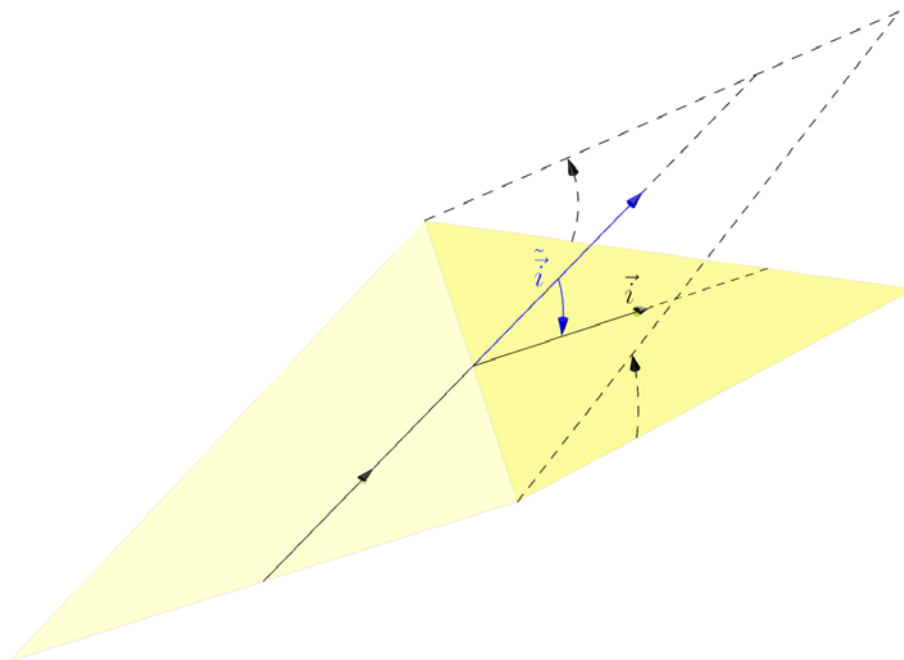


Abbildung 4: Die Laufrichtung wird über die Kante gefaltet.

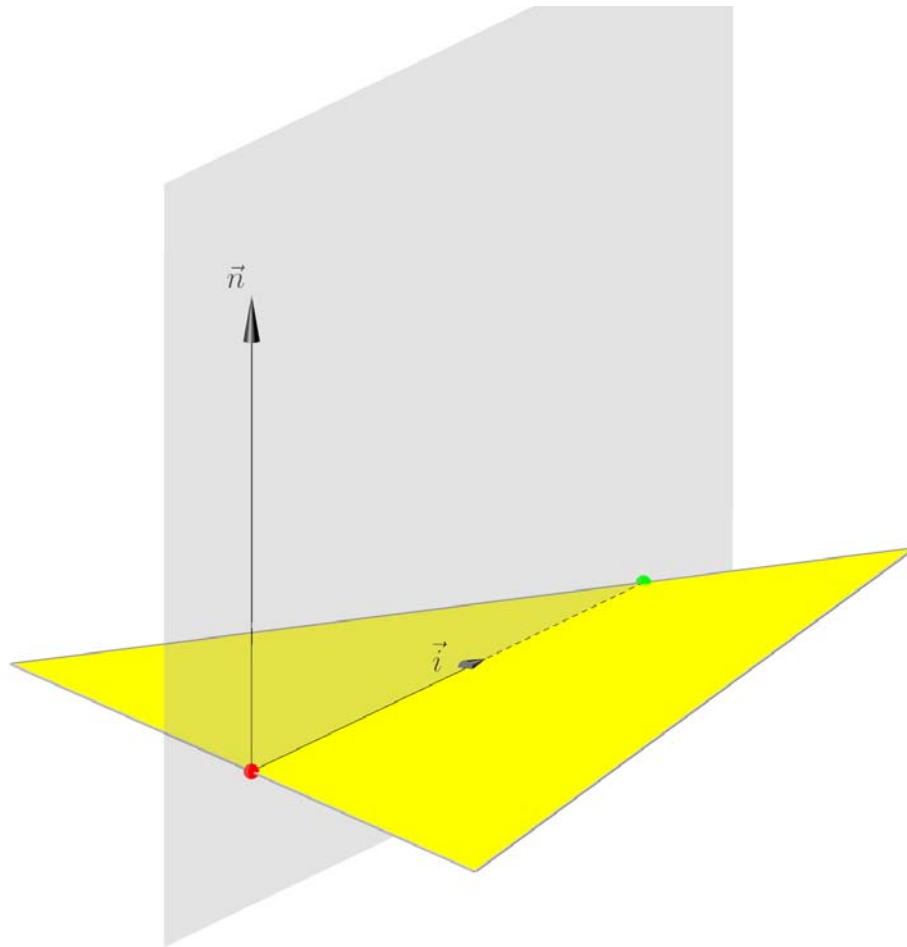


Abbildung 5: Eine Ebene wird benutzt, um den nächsten Punkt zu finden.

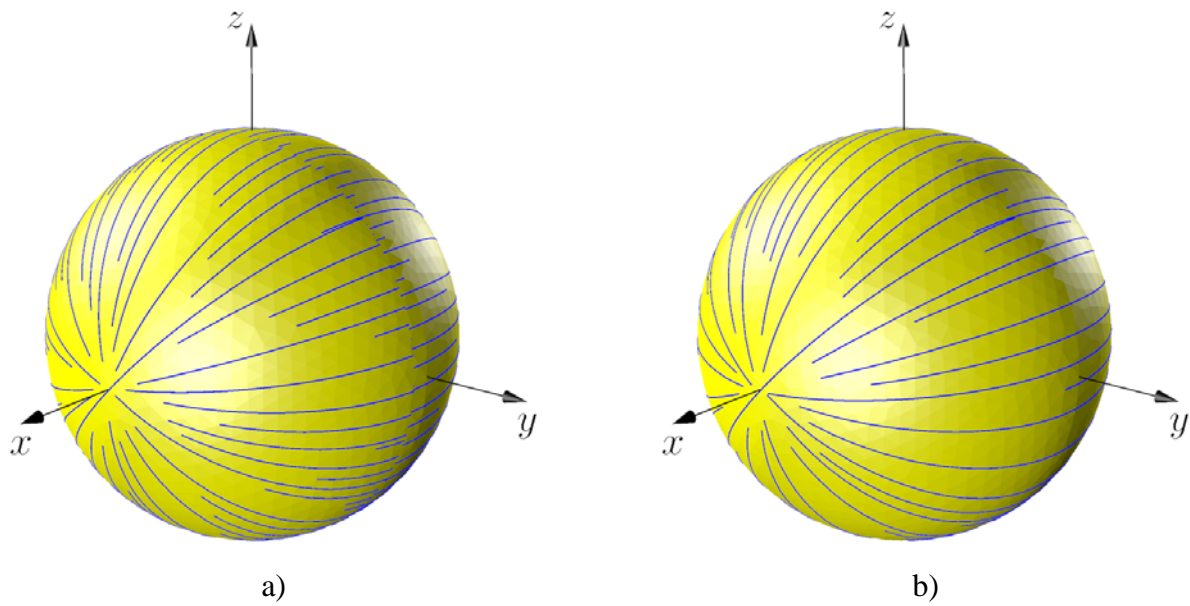


Abbildung 6: Der Einfluss der Flagge --skip: a) --skip 1, b) --skip 3.

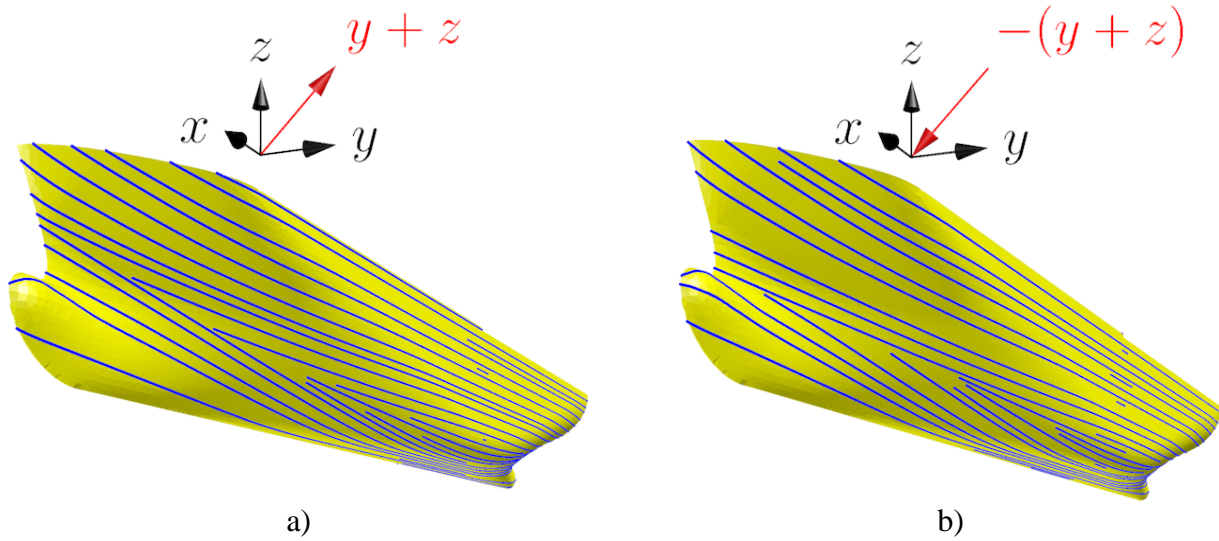


Abbildung 7: Einfluss von Geodätensortierung, von unten nach oben (a) und von oben nach unten (b).

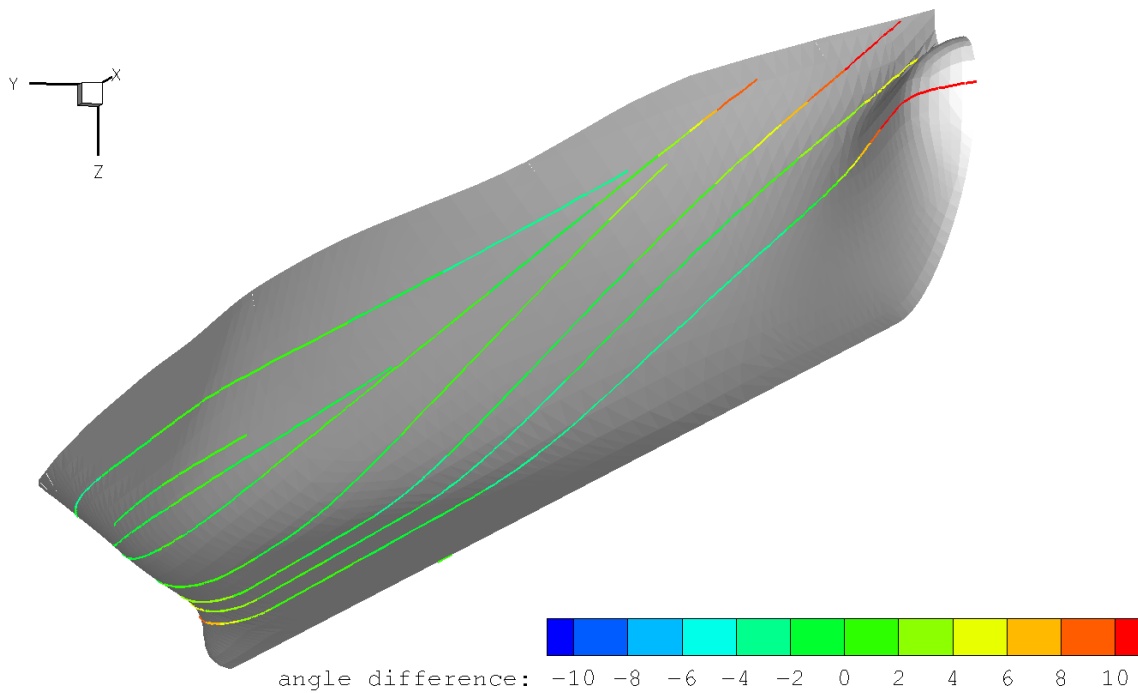


Abbildung 8: Winkeldifferenzen zwischen Geodäten und Strömung, berechnet mit v-Shallo.

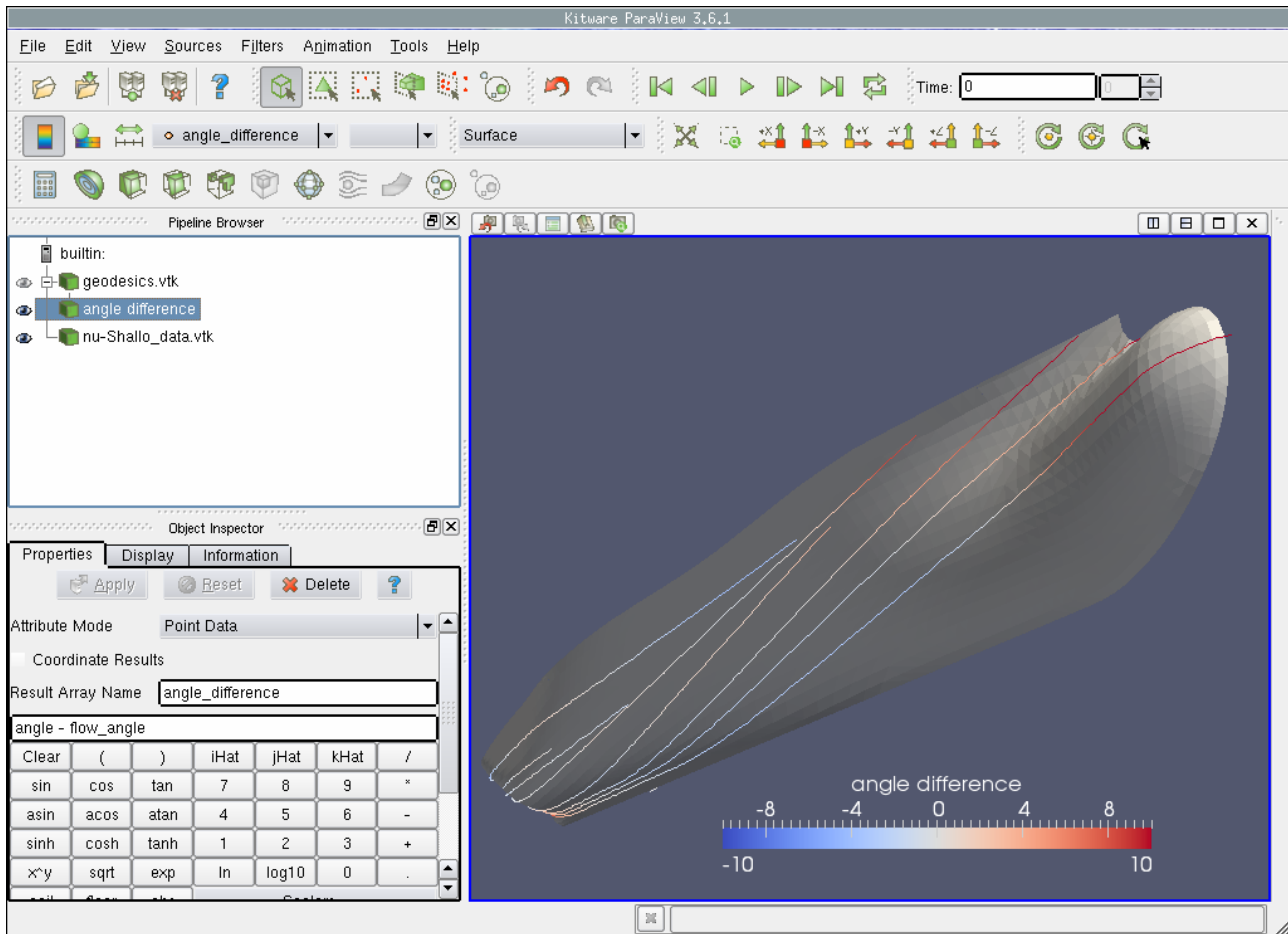


Abbildung 9: Die gleichen Geodäten wie in der Abbildung 8, aber diesmal mit PARAVIEW.

A Dokumentation aus dem Quelltext

Die folgenden zwei Abschnitte enthalten Dokumentation die man mit Hilfe des Befehls „pydoc surface“ bzw. „pydoc geodesics“ aus dem Quelltext extrahieren kann. Das Programm pydoc ist ein Teil jeder vollständigen Installation von Python.

Zahlreiche Funktionen und Methoden aus den Pythonmodulen `math` und `euclid` sind hierbei manuell ausgeblendet, um die Lesbarkeit zu fördern.

A.1 surface_mesh

Help on module surface_mesh:

NAME

surface_mesh

FILE

/home/jsch/haitech/surface_mesh/surface_mesh.py

DESCRIPTION

The python module surface_mesh contains a small set of simple tools for managing polygonal surface meshes:

- * The class Surface_Mesh for storing such meshes.
- * A number of input and output subprograms for reading and storing surface meshes.
- * A command line interface to the io subprograms, turning this python module into a primitive file format translator for surface meshes.

In general, it is not assumed that all surface elements are triangles, but some subprograms do (e.g. `Element.area` and `Element.normal`)

Copyright HSVA 2009

CLASSES

```
__builtin__.object
  Edge(euclid.LineSegment3, __builtin__.object)
  Element
  Node(euclid.Point3, __builtin__.object)
  Surface_Mesh
euclid.LineSegment3(euclid.Line3, __builtin__.object)
  Edge(euclid.LineSegment3, __builtin__.object)
euclid.Point3(euclid.Vector3, euclid.Geometry, __builtin__.object)
  Node(euclid.Point3, __builtin__.object)
```

```
class Edge(euclid.LineSegment3, __builtin__.object)
| Mesh edges have the following attributes:
|
| node_[AB]   the two nodes at the ends
| elements    list of all elements (0?, 1 or 2) that are
|              attached to this edge
|
| Method resolution order:
|   Edge
|   euclid.LineSegment3
|   euclid.Line3
|   __builtin__.object
|
| Methods defined here:
```

```

__init__(self, node_a, node_b)

get_other_node(self, node)
    This method returns the other node connected to this edge.

    Input:

    node    the node that should not be returned

    N.B. This method will bail out (stop your program) if the
    input node isn't one of the end points of this edge

same_endpoints(self, other)
    Returns true if self and other have the same nodes as
    end-points.

```

Methods inherited from euclid.LineSegment3:

```

__abs__(self)

__repr__(self)

magnitude_squared(self)

```

Data descriptors inherited from euclid.LineSegment3:

```

__dict__
    dictionary for instance variables (if defined)

__weakref__
    list of weak references to the object (if defined)

length

```

Methods inherited from euclid.Line3:

```

__copy__(self)

__getstate__(self)

__setstate__(self, state)

connect(self, other)

copy = __copy__(self)

intersect(self, other)

```

Data descriptors inherited from euclid.Line3:

```

p

p1

p2

v

```

```

class Element(__builtin__.object)
    Mesh elements have the following attributes:

    nodes          list of nodes at the vertices of the element
    edges          list of edges forming the perimeter
    elements       list of neighbour elements
    data           a dictionary, possibly containing element data

    Methods defined here:

    __init__(self, list_of_nodes)

    align_normal_with_reference(self, reference_element)
        This subprogram makes sure that the surface normal of self is
        pointing in the same direction as that of the reference
        element. Here "same direction" means that the surface normals of
        the two elements are on the same side of the surface formed by the
        two elements.

    area(self)
        Returns the area of the element.

        Note: This method can only cope with triangular elements.

    normal(self)
        Returns the surface normal of the element. It will point in the
        right handed direction, as defined by the node order in the list
        self.nodes.

        Note: This method can only cope with triangular elements.

    -----
    Data descriptors defined here:

    __dict__
        dictionary for instance variables (if defined)

    __weakref__
        list of weak references to the object (if defined)

class Node(euclid.Point3, __builtin__.object)
    Mesh nodes have the following attributes:

    x, y, z       location (taken care of by Point3)
    edges         list of edges to which the node is connected
    elements      list of all elements that have the node as
                  one of its vertices
    data          a dictionary, possibly containing node data

    Method resolution order:
        Node
        euclid.Point3
        euclid.Vector3
        euclid.Geometry
        __builtin__.object

    Methods defined here:

    __init__(self, x, y, z, data=[])

    __str__(self)

    -----

```

```

Methods inherited from euclid.Point3:

__repr__(self)

connect(self, other)

intersect(self, other)

-----
Data descriptors inherited from euclid.Point3:

__dict__
    dictionary for instance variables (if defined)

__weakref__
    list of weak references to the object (if defined)

-----
Methods inherited from euclid.Vector3:

__abs__(self)

...
normalized(self)

reflect(self, normal)

-----
Data descriptors inherited from euclid.Vector3:

x

y

z

-----
Methods inherited from euclid.Geometry:

distance(self, other)

```

```

class Surface_Mesh(__builtin__.object)
    This class provides a data abstraction of polygonal surface meshes.

    Instance attributes:

    nodes            list of nodes
    edges            list of edges
    elements          list of elements
    data_names        list of the node data field names

    Methods defined here:

    __init__(self)

    build_topology(self)
        This method assumes that the mesh is in a partially unconnected
        state and will make sure that nodes, edges and elements are set
        up with all the convenient cross references that their attribute
        lists make possible.

        To be precise, the assumed initial state is as follows:

```

- * nodes
 - know their own positions
 - possibly contain data
 - ...but have no references to elements or edges
- * edges
 - None created.
- * elements
 - They have all their references to nodes...
 - ...but have no references to neighbours and edges.

The tasks at hand are thus:

- * create edges (...which, obviously, creates ref's from edges -> nodes)
- * add references:
 - edges -> elements
 - nodes -> elements
 - nodes -> edges
 - elements -> elements
 - elements -> edges

N.B. The algorithm used for pruning away superfluous edges will bail out if there are more than two edges between two nodes.

make_consensus_normals(self)

This subprogram makes sure that all element normals are pointing in the same direction, that is, that all normals "stick out" on the same side of the surface.

The algorithm used, expanding front, will silently fail in two cases:

1. The mesh has only one side, i.e. a Moebius strip.
2. The mesh consist of several isolated patches.

Data descriptors defined here:

`__dict__`
dictionary for instance variables (if defined)

`__weakref__`
list of weak references to the object (if defined)

FUNCTIONS

command_line_interface()

Reads and interprets the command line arguments and launches the main program.

Output from the command "surface_mesh --help":

Usage:

The file surface_mesh.py is actually a module containing an abstraction for unstructured triangle surface meshes, but can also be used directly from the command line (when properly installed):

surface_mesh [<options>] <input file> <output file>

It then converts a mesh stored as an STL or Tecplot file into either Tecplot, VTK, Asymptote or Wavefront obj format. (A Tecplot to Tecplot "conversion" merely produces a file where only the selected zone remains, take or leave some, hopefully, unimportant changes.)

Note: An awful lot of info can be packed into the Tecplot file and zone headers. This program only accepts a small fraction of that, namely the type of headers that are produced by nu-Shallo. If your headers don't work, read the source...

Options:

```
--version          show program's version number and exit
-h, --help        show this help message and exit
-z <name>, --zone=<name>
                  zone name to be used in the tecplot file(s)
                  (default: cp_tril2)
-o <format>, --output_format=<format>
                  Output file format, either tecplot, vtk, asy
                  or obj. (default: tecplot)
-i <format>, --input_format=<format>
                  Input file format, either tecplot or stl.
                  (default: tecplot)
```

```
main(input_filename, output_filename, tecplot_zone, input_type, output_type)
This program reads a surface mesh from a file and writes it
back to another. A small number of file formats are available.
```

Input:

```
input_filename    string containing the name of the input file
output_filename   ----- ' ----- output file
tecplot_zone      The zone name that will be used when a) looking
                  for the mesh in the input file, and b) when
                  writing the new file. (whatever applies)
input_type        input file type, "tecplot" or "stl"
output_type       output file type, "tecplot", "vtk", "asy"
                  (Asymptote), or "obj" (Wavefront)
```

Output:

A file with the same mesh (and data) as was found in the input file.

```
read_Tecplot_FEPOINT_zone(file, zone_name)
```

Extracts a FEPOINT zone with triangle elements from a Tecplot file

and returns a Surface_Mesh object. The data at each point in the input file will be stored in the data attribute of the Node objects of the mesh.

Input:

```
file              a python file object
zone_name         string containing the preferred name of the zone in which
                  the mesh will be stored
```

Output:

a Surface_Mesh object

Notes:

This subprogram has only been tested with the sh2tec.dat files delivered by nu-Shallo. Your mileage may vary.

An attempt is made at double-checking the element normals, making sure that all "stick out" from the same side of the surface.

`read_stl_mesh(file)`

This subprogram returns a mesh, just like `read_Tecplot_FEPOINT_zone` above, but from a stereolithography (stl) file. Since stl-files, at least those produced by ICEM-CFD appear to lack connectivity information, this must be deduced.

Input:

`file` the file containing the stl data

Output: a `Surface_Mesh` object

Note: The algorithm used for inferring the connectivities from node coordinates is very expensive. This subprogram should only be used for small meshes, or in utter desperation.

`skip_lines_until_finding_regexp(file, pattern_string)`

Subprogram for scrolling down to a pattern in file.

Input:

`file` a python file object
`pattern_string` a pattern string ready to be compiled by `re.compile`

Output: The first line in the file that matched the pattern.

Side effect: The file pointer is advanced to the line that was found.

`write_Tecplot_file(file, mesh, zone_name)`

Writes a surface mesh and its (node) data to a FEPOINT zone with triangle elements in an ASCII Tecplot file.

Input:

`file` the output file
`mesh` a `Surface_Mesh`
`zone_name` string containing the preferred name of the zone in which the mesh will be stored

Note: This subprogram should create a file that can be read back with the subprogram `read_Tecplot_FEPOINT_zone`.

`write_asymptote_file(file, mesh)`

Produces a file containing an Asymptote plot showing the mesh. The mesh facets are stored as Asymptote surface objects, as defined by the Asymptote module "three".

Input:

`file` an opened file where the output will be written
`mesh` a `Surface_Mesh` object

Notes:

The file will not contain the element and node data of the mesh.

Use the subprogram `write_obj_file` and the Asymptote `obj` module, if

the files produced by this subprogram runs to slowly in Asymptote.

The produced file is fairly human readable and can be edited if the default size, colour (whatever) does not fit your taste.

```
write_obj_file(file, mesh)
```

Writes a Surface_Mesh to a Wavefront obj file.

Input:

```
file          the output file
mesh         an instance of Surface_Mesh
```

```
write_vtk_file(file, mesh)
```

Dumps a mesh and its (node) data onto an serial ASCII VTK file in "legacy" format, i.e. *not* XML. The dataset structure used will be "unstructured grid" and all the data in the mesh nodes will be saved as "field data".

See also "File Formats for VTK Version 4.2", available as pdf at www.kitware.com.

Input:

```
file          the output file (object)
mesh         an instance of Surface_Mesh storing your mesh
```

Output: a Surface_Mesh object

DATA

```
ARGPARSE = <optparse.OptionParser instance at 0x7a45a8>
CLI_doc_help_IO = <StringIO.StringIO instance at 0x7a4758>
CLI_doc_help_contribution = '  Usage: \n  \n  The file surface_mesh...
CLI_doc_help_raw = 'Usage: \n\nThe file surface_mesh.py is actually a ...
USAGE = "\n\nThe file surface_mesh.py is actually a module .... If you...
__version__ = '$Revision: 489 $'
```

VERSION

```
489
```

A.2 geodesics

Help on module geodesics:

NAME

geodesics - A tool for tracing geodesic curves on a triangulated surface.

FILE

```
/home/jsch/haitech/surface_mesh/geodesics.py
```

DESCRIPTION

```
Copyright HSVA 2009
```

CLASSES

```
__builtin__.object
  Geodesic
  Sort_Function
  Trace_Point(euclid.Point3, __builtin__.object)
euclid.Point3(euclid.Vector3, euclid.Geometry, __builtin__.object)
  Trace_Point(euclid.Point3, __builtin__.object)
```

```

class Geodesic(__builtin__.object)
    A collection of Trace_Point objects, forming a geodesic curve.

    Instance attributes:

    head          the downstream (or ultimate) Trace_Point object of the
                  geodesic curve
    tail          the upstream, or first, Trace_Point object of the geodesic
    mesh          a Surface_Mesh object, the "universe" of the geodesic

    Methods defined here:

    __init__(self, trace_point, mesh)

    __iter__(self)

    __len__(self)

    add_paint_angle_to_data(self, atan2_flip)
        This subprogram adds "angle", and, if possible, "flow angle" to
        the geodesic data.

        The angle is the difference in direction between the geodesic and
        a reference direction "xi", normal to a section curve, on the
        mesh surface.

        The flow angle is the difference in direction between the velocity
        vector (U, V, W), if present in the original mesh file, and the
        reference direction xi.

        The reference direction of "angle" will be flipped 180 degrees if
        atan2_flip is True.

    extend(self, direction=None)
        This subprogram will try to add a new point to our geodesic curve.

        Input:

        direction  An euclid.Vector3 object pointing in the direction we
                  intend to go on tracing the geodesic curve.

                  If supplied, it will be assumed that self is the
                  first point in the geodesic curve, and suitable
                  projections will be carried out. If not, self.head -
                  self.head.backward will be used.

        PAINT_ONCE_ONLY (global constant)
            False -> Geodesics only stop at outer edges or when
                    the number of steps reaches the limit
                    MAX_STEPS.
            True -> They also stop when they reach an element
                   that has already been crossed by a geodesic

        Output:

        False      if self.head already was at the outer edge of the mesh
                  surface
        True       if the geodesic could be extended

    next(self)

    reverse(self)
        Reverses the orientation of the geodesic.

```

`trace(self, initial_direction)`

Traces the entire geodesic in both directions until it reaches an outer edge of the mesh or it has taken `max_steps` steps, whichever comes first.

Input:

`initial_direction` This vector will be projected onto the mesh surface. The tracing will then commence in the projected direction.

`MAX_STEPS` limit on the number of elements a geodesic may cross (global variable!)

Static methods defined here:

`trace_coverage(mesh)`

Calculates how much of the surface was covered by the geodesics. This fraction of coverage is defined as the number of elements crossed by geodesics, divided by the total number of elements in the mesh.

Input:

`mesh` a `Surface_Mesh` instance with elements having the additional attribute "painted" set to `True` or `False` depending on whether it has been crossed by a geodesic

Output: The fraction of mesh elements that were crossed

Data descriptors defined here:

`__dict__`

dictionary for instance variables (if defined)

`__weakref__`

list of weak references to the object (if defined)

`class Sort_Function(__builtin__.object)`

This class is supposed to simplify the dynamic creation of functions suitable as argument to the `list.sort` method.

Object attribute:

`function` a compiled "eval-able" expression

Methods defined here:

`__call__(self, A, B)`

This is what will be called when the `Sort_Function` object is used as comparison function in the `list.sort` method.

`__init__(self, expression_string)`

Initialises the `Sort_Function` object. The argument should be a string containing a legal Python expression that produces a real valued answer upon evaluation. This value decides the sort order of a list, when the `Sort_Function` object is used as the comparison function in the `list.sort` method.

Note 1: The name of the unknown list object will be 'p' where the

expression will be evaluated, but 'x', 'y' and 'z' will be made available as convenient synonyms for 'p.x', 'p.y' and 'p.z' respectively.

Note 2: All of the python math module has been imported to the scope where the function expression string will be evaluated.

Examples of (conditionally) legal sort expressions:

```
"p**2 + 1"           if p happens to be a plain number
"p.magnitude()"      if p has this method
"sqrt(x**2 + y**2)"  x, y, and z are other names for p.[xyz]
"log(p.x)"           if p.x > 0
```

Examples of illegal sort expressions:

```
"abs(p + q)"         Heh? No q is known where self will be
                     executed.
"abs(p) + 'a'"        Does not seem to return a real number.
```

Data descriptors defined here:

```
__dict__
    dictionary for instance variables (if defined)
__weakref__
    list of weak references to the object (if defined)
```

```
class Trace_Point(euclid.Point3, __builtin__.object)
```

These are the type of points that are used when constructing a geodesic.

Object attributes:

```
forward    the trace point downstream of self
backward   ----- " ----- upstream   -- " --
edge       a reference to the mesh edge that the trace point "sits on"
data       a dict that can be loaded with whatever
```

Method resolution order:

```
Trace_Point
euclid.Point3
euclid.Vector3
euclid.Geometry
__builtin__.object
```

Methods defined here:

```
__init__(self, x, y, z, edge=None, backward=None, forward=None)
```

```
__str__(self)
```

```
sort_attached_elements(self, direction)
```

This subprogram is used when figuring out where to go next.

Input:

direction A vector that should only be supplied prior to the first step taken. After the first step, the trace direction is deduced from the tangent of the previous

step.

Output:

A tuple with references to the mesh elements that surrounds the edge associated with the Trace_Point self:
(ahead, abaft)

ahead refers to the element that will be -- should be! -- straddled the next time we extend our geodesic curve in the direction of "direction"

abaft is the element that lies behind the point "self"

One of the two could be "None", which signals that "self" is at an outer edge of the mesh.

Note 1: It is assumed that self is positioned on the edge to which it refers. That is, this method must be called by properly set-up Trace_Point objects.

Note 2: Due to some stupid approximations, this subprogram may fail for some pathological cases.

Methods inherited from euclid.Point3:

__repr__(self)

connect(self, other)

intersect(self, other)

Data descriptors inherited from euclid.Point3:

__dict__
 dictionary for instance variables (if defined)

__weakref__
 list of weak references to the object (if defined)

Methods inherited from euclid.Vector3:

__abs__(self)

...
normalize(self)

normalized(self)

reflect(self, normal)

Data descriptors inherited from euclid.Vector3:

x

y

z

```
| Methods inherited from euclid.Geometry:
|
| distance(self, other)
```

FUNCTIONS

`command_line_interface()`

Reads and interprets command line arguments and launches the main program.

The command "geodesics --help" will tell you this:

Usage:

The script `geodesics.py` traces geodesic lines along a surface defined by a triangle mesh. The seed points for the line tracing are selected with a cutting plane; They are the intersections of the plane and all the edges in the mesh. The plane is specified with a point lying on the plane, and a normal vector. The initial trace direction is specified with a vector (which will be projected onto the surface).

If the output file format supports field data, the geodesic tangent direction, "angle", will be added to the output file. The direction is given in degrees and is measured relative to a reference direction "xi". Xi is orthogonal to the intersection between a local x-plane and the surface.

If the input file contained the flow velocity components U, V and W, the flow-field direction "flow direction", measured just like "angle", will be calculated and added to the output file.

Command:

```
geodesics [<options>] <input file> <output file>
```

Notes:

1. Due to assumptions made in `Trace_Point.sort_attached_edges`, it will be highly questionable to start tracing geodesics at sharp edges.
2. Having an initial direction that is very close to parallel to the edge upon which the first `Trace_Point` sits, is another way of finding the trolls in `Trace_Point.sort_attached_edges`.
3. Pathological cases can often be circumnavigated by nudging the input parameters.

Options:

```
--version          show program's version number and exit
-h, --help         show this help message and exit
-f, --flip_angle   rotate the reference direction of "angle" 180
                  degrees (default: False)
-i <format>, --input_format=<format>
                  Input file format, either tecplot or stl.
                  (default: tecplot)
-n <nx ny nz>, --normal=<nx ny nz>
                  normal vector of the cutting plane (default:
                  (1, 0, 0))
-o <format>, --output_format=<format>
                  Output file format, either tecplot, vtk or
                  asy. (default: tecplot)
-p <px py pz>, --point=<px py pz>
```

```

the coordinates of a point lying on the
cutting plane (default: (0, 0, 0))
-s <n>, --skip=<n>      pruning of geodesics: only trace from every
                        n'th seed point (default: 1)
--sort=<expression>    python expression defining the sort order of
                        the seed points (default: -(y + z))
-t <tx ty tz>, --tangent=<tx ty tz>
                        initial tracing direction (default: same as
                        the cutting plane normal)
-z <name>, --zone=<name>
                        zone name to read from the tecplot file
                        (default: cp_tril2)

```

```

dump_geodesics_on_Asymptote_file(file, geodesics, title='This file was
                                written by the Hai-Tech Geodesics script.')
Writes an Asymptote file with all the traced geodesics as paths.
The data stored in the Trace_Point objects is ignored.

```

```

dump_geodesics_on_Tecplot_file(file, geodesics, title='This file was written
                                by the Hai-Tech Geodesics script.')
Writes a Tecplot file with all the traced geodesics as I-ordered
line zones. The data stored in the Trace_Points is also dumped
to the file.

```

```

dump_geodesics_on_VTK_file(file, geodesics, title='This file was written by
                                the Hai-Tech Geodesics script.')
Writes a VTK file with all the traced geodesics. The data stored in
the Trace_Point objects is also dumped to the file.

```

```

find_seed_points(mesh, plane, sort_expression='-(y + z)')
Finds a list of suitable seed points for the tracing of geodesics.
The points are the intersection of a plane and all the edges in the
input mesh.

```

Input:

```

mesh          a Surface_Mesh on which we would like to trace geodesics.
plane         an euclid.Plane defining where the mesh should be cut.
sort_expression
              Since the mesh edges could be stored nilly-willy in the
              mesh object, we need to specify how we would like to have
              the list of seed points, the returned list of Geodesics,
              sorted. The string sort_expression should contain a python
              expression that can be applied when sorting a list of
              intersection points.
              (See also the documentation of Sort_Function.)

```

Output: a list of newly created Geodesics objects, sorted according to the sort expression

```

main(input_filename, output_filename, input_filetype, output_filetype,
      cut_plane, initial_direction, tecplot_zone, skip, sort_expression,
      atan2_flip)
This program reads a surface mesh from a file, cuts it with a plane
in order to obtain seed points for geodesics, calculates geodesics,
and writes them to a file.

```

The directions of the geodesics with respect to x-planes (-90 degrees) will be calculated and added to the output file (if possible). The field name "angle" will be used.

If the input file contains the flow velocity components U, V, and W, a "flow direction" with respect to the same reference direction as "angle" will be calculated and added to the output file (if possible).

Input:

```

input_filename      string containing the name of the input file
output_filename     ----- ' ----- output file
input_filetype      either "stl" or 'tecplot', selects which mesh file
                    reader will be used
output_filetype     either "tecplot", "vtk", or "asy"
cut_plane           An euclid.Plane that will be used when cutting the
                    mesh. The points where this plane intersects mesh
                    edges, will be used as seed points for the
                    geodesic lines.
initial_direction   The direction in which the tracing of each line
                    will begin.
tecplot_zone        The name of the zone in the tecplot file where the
                    mesh is stored. (Obviously ignored when
                    input_filetype = "stl".)

skip                Skip values > 1 signals pruning of seed
                    points. Only every skip'th seed point will be
                    used.
sort_expression     A string containing a python expression that will
                    be evaluated as the sort order of the seed points
                    i decided.
atan2_flip          The reference direction of the output variable
                    "angle" will be flipped 180 degrees if atan2_flip
                    is True.

```

Output: A Tecplot file with I-ordered pointdata line zones.

DATA

```

ARGPARSE = <optparse.OptionParser instance at 0x8bf0e0>
CLI_doc_help_IO = <StringIO.StringIO instance at 0x8bf5a8>
CLI_doc_help_contribution = ' Usage: \n \n The script geodesics...
CLI_doc_help_raw = 'Usage: \n\nThe script geodesics.py traces geodesic...
MAX_STEPS = 10000
PAINT_ONCE_ONLY = True
USAGE = '\n\nThe script geodesics.py traces geodesic lines ...cumnavig...
__version__ = '$Revision: 487 $'

```

VERSION
487

B Dateiformate

In diesem Abschnitt wird kurz Auskunft über die verschiedenen Dateiformate gegeben, mit denen geodesics und surface_mesh umgehen können. Falls die eigenen Vorverarbeitungs- oder Nachbearbeitungs-Werkzeuge nicht genau die passende Dateiformate schreiben bzw. lesen können, wird dem Benutzer empfohlen, den Quelltext von geodesics und surface_mesh zu erweitern oder zu modifizieren. Dank modularer Bauweise und guter Quelltext-Dokumentation sollte das eine weitaus einfachere und elegantere Lösung bieten als mit externem Filter zu arbeiten.

B.1 Eingabedateien

Bezüglich der zur Verfügung stehenden Eingabedatei-Formate „Tecplot“ und „STL“ soll der Benutzer nicht glauben, dass `geodesics` und `surface_mesh`⁷ alle vorstellbaren Verschachtelungen versteht, die man innerhalb der Dateiformatdefinitionen inszenieren kann. Das genaueste Verständnis für die Beschränkungen erhält man zwar vom Quelltext in `surface_mesh.py`, aber hier kann man hoffentlich mit weniger Aufwand das Wesentliche lernen,.

Genau wie in den Abschnitten 3.2.2 und 3.2.3 werden in den folgenden Abschnitten Platzhalter mit Winkelklammern bezeichnet.

B.1.1 TECPLOT

ASCII TECPLOT-Dateien werden mithilfe der Funktion `read_Tecplot_FEPOINT_zone` in `surface_mesh.py` gelesen. Sie ist sehr einfach und arbeitet sich in der Eingabedatei vom Anfang bis zum Ende durch:

1. Zeilen werden übersprungen bis eine Zeile mit „Variables = “ oder „VARIABLES = “ anfängt.
2. In dieser Zeile, nach „Variables = “, wird eine mit Komma getrennte Liste von Feldnamen erwartet.
3. Zeilen werden dann übersprungen bis eine Zeile, die mit „ZONE T=<Name>“ anfängt, gefunden ist. Hier ist <Name> der Zonenname, der mit der Flagge `--zone` gegeben war.
4. Diese Zeile wird dann weiter interpretiert und soll⁸ so aussehen:
“ZONE T=<Name> F=FEPOINT N=<k> E=<e> ET=TRIANGLE“
Hierbei wird die Anzahl von Knoten, k , und Elementen, e , in der Zone eingelesen.
5. Danach wird k Zeile gelesen. Jede diese Zeilen soll folgendes enthalten:
 - a. die x -, y - und z -Koordinate des k -te Gitterknotens
 - b. weitere Datenwerte, die mit dem k -te Gitterknoten assoziiert sind
6. Zuletzt wird e Zeile mit Elementendefinitionen gelesen. Jede Zeile enthält drei Ganzzahlen, die ein Gitterelement definieren. Es sind Knotenzahlen⁹, Hinweise auf die bereits eingelesene Knotenkoordinaten bzw. -daten.

Im Grunde ist `read_Tecplot_FEPOINT_zone` einfach angepasst, um die Ausgabedatei `sh2tec.dat` die von `v-Shallo` zu erzeugen.

B.1.2 STL

Notfalls können `geodesics` und `surface_mesh` auch ASCII STL-Dateien mit der Funktion `read_stl_mesh` in `surface_mesh.py` lesen. Da STL-Dateien die Knoten nicht eindeutig definieren, muss diese Information hinterher hergeleitet werden. Der Algorithmus der im Augenblick für diese Arbeit zuständig ist, ist leider sehr langsam.

Die Funktion `read_stl_mesh` arbeitet sich durch die Eingabedatei in folgender Art und Weise:

1. Zeilen werden übersprungen bis eine Zeile gefunden ist, die „outer loop“ enthält.
2. Die nächstfolgenden drei Zeilen werden dann gelesen. Dort sollten sich die Koordinaten für die Eckpunkte eines Dreiecks befinden, als Wort 2, 3, und 4 in jeder Zeile.
3. Schritt 1 und 2 wird wiederholt, bis eine Zeile gefunden ist, die „endsolid“ enthält.

⁷ Da `geodesics` nur die Einleseprogramme in `surface_mesh` benutzt, sind die Fähigkeiten und Begrenzungen identisch.

⁸ Wie schon gesagt, befindet sich die genaue Definition von " was erlaubt ist, oder nicht", im Quelltext.

⁹ Die Knoten werden intelligent nummeriert, d.h. die Erste ist Nummer 1.

B.2 Ausgabedateien

B.2.1 TECPLOT

Das Programm `surface_mesh` erzeugt TECPLOT-Dateien, die wieder von `geodesics` oder `surface_mesh` eingelesen werden können. Die Geodäten aus dem Programm `geodesics` müssen dagegen wegen ihres Inhalts zwangsläufig eine andere Art TECPLOT-Datei erzeugen. In den Geodäten-Dateien wird jede Geodäte als eine „I-ordered“ Zone gespeichert.

B.2.2 VTK

VTK, *Visualization Toolkit*, ist ein Softwaresystem für die Herstellung von 3D-Diagrammen (siehe <http://www.vtk.org>). Dieses Kürzel ist auch die bevorzugte Endsilbe für Dateien die 3D-Daten auf einem bestimmten, für VTK-Werkzeuge angepassten Datenformat enthalten. Dieses ASCII Datenformat ist das einfachste, was VTK bietet, nämlich das so genannte „legacy“ format [1]. Dateien in diesem Datenformat können mit PARAVIEW gelesen und studiert werden.

B.2.3 Asymptote

Asymptote ist sowohl der Name einer Vektorgraphik-Sprache, mit der man technische Diagramme beschreiben/definieren kann, als auch der Name eines Programms, das aus solchen Beschreibungen wirkliche Diagramme erzeugen kann. Sowohl `geodesics` als auch `surface_mesh` können Asymptote-Dateien schreiben.

Viele der Darstellungen in diesem Bericht wurden mit Hilfe von Asymptote erzeugt (Abbildungen 2 bis 7). Asymptote-Dateien haben normalerweise die Endsilbe `asy`.

B.2.4 OBJ

Das Datenformat „OBJ“ wurde zuerst von der Firma Wavefront Technologies entwickelt. Es eignet sich für die Beschreibung von dreidimensionalen Objekten. Das Programm `surface_mesh` kann OBJ-Dateien als Alternative zu Asymptote-Dateien nutzen. Das Programm Asymptote kann mit seinem OBJ-Modul solche Dateien schneller als reine Asymptote-Dateien lesen.