

Orbit- und Strahlungstransfersimulation für das Satellitenprojekt CO2Image

Stefan Volz

Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt
Fakultät Angewandte Natur- und Geisteswissenschaften
Bachelorstudiengang Technomathematik
Technischer Bericht zum Praxissemester

Deutsches Zentrum für Luft- und Raumfahrt
Institut für Methodik der Fernerkundung
Abteilung Atmosphärenprozessoren

23. Juli 2022

Zusammenfassung

Bei CO2Image handelt es sich um eine geplante Satellitenmission des DLR, welche anhand von Spektralmessungen der Erdstrahlung hochauflösende Karten der XCO₂-Verteilung¹ auf der Erdoberfläche im Gebiet rund um kritische Erzeuger wie z.B. Kraftwerke, Kohleminen oder große Industrieanlagen erstellen soll (siehe auch [EOC22] für mehr Informationen zur Mission). Diese Messungen sollen in Zukunft z.B. die Überprüfung der von Kraftwerksbetreibern angegebenen CO₂-Ausstoßmengen ermöglichen.

Das Ziel des Praktikums waren Erweiterung bzw. Verbesserung eines in Python und C++ implementierten Orbit simulato rs sowie die Verknüpfung desgleichen mit einem über die IMF-eigene Python Bibliothek Py4CATS² implementierten Strahlungstransfermodell.

Die dadurch entstandene Anwendung stellt eine Messdatensynthese für sowohl aktive als auch passive satellitengestützte Spektrometer dar und soll zur Planung zukünftiger Fernerkundungsmissionen wie zum Beispiel dem beschriebenen CO2Image dienen.

¹XCO₂ steht hierbei für die über eine Atmosphärensäule gemittelte CO₂-Menge

²Python for Computational Atmospheric Spectroscopy

Inhaltsverzeichnis

Inhaltsverzeichnis

1 Einführung	3
1.1 Vorbemerkung	3
1.2 Beschreibung des zu lösenden Problems	3
1.3 Mathematisches Modell	3
1.4 Zustand des Orbitsimulators zu Beginn des Praktikums	5
1.5 Zustand kurz vor Ende des Projektes	7
1.6 Grundlegende physikalische Hintergründe von ATP-OSSE	8
1.6.1 Orbitintegration	8
1.6.2 Strahlungstransfer	9
2 Beschreibung eines Algorithmus zur Überprüfung von Punkt-Polygon Inzidenzen auf der Oberfläche eines Ellipsoids	11
2.1 Problembeschreibung und Rahmenlage	11
2.2 Grundlegende mathematische Definitionen	11
2.3 Vereinfachende Annahmen und Vorüberlegungen	12
2.3.1 Approximation von Geodäten auf dem Erdellipsoid	12
2.3.2 Auflösen einer Mehrdeutigkeit in der Problemstellung	13
2.3.3 Äquivalenz zum konvexen Problem	13
2.4 Entwicklung des Algorithmus	14
2.4.1 Innen-Außen Segmentierung in planarer Geometrie	15
2.4.2 Innen-Außen Segmentierung in sphärischer Geometrie	16
2.5 Implementierung und Zusammenfassung	17
A Kurzzusammenfassung sonstiger Tätigkeiten	20
B Ausblick und zukünftige Erweiterungen	22
C Sonstiges	24
C.1 Lange Listings	24
C.2 Einige Bilder der Benutzeroberfläche	28
C.3 Erzeugte Footprintvisualisierungen	29
D Eidesstattliche Erklärung	32

Vorwort

Dieser Bericht entstand während meines studentischen Praktikums in der Abteilung IMF-ATP³ beim Deutschen Zentrum für Luft- und Raumfahrt am Standort Weßling, Oberpfaffenhofen. Ich möchte mich herzlich bei allen Kolleginnen und Kollegen der Abteilung ATP für die freundliche Aufnahme, gute Zusammenarbeit und interessanten Unterhaltungen bedanken. Insbesondere danke ich Herrn Prof. Dr. Thomas Trautmann und Herrn Dr. Günter Lichtenberg des DLR sowie Herrn Prof. Dr. Holger Walter der FHWS für die gute Betreuung und die Möglichkeit an diesem interessanten Projekt mitwirken zu können. Ihnen danke ich außerdem für die Leihgabe einiger Bücher während des Praktikums.

³Institut für Methodik der Fernerkundung - Atmosphärenprozessoren

Kapitel 1

Einführung

1.1 Vorbemerkung

Aufgrund des Umfangs der im Verlauf des Praktikums ausgeführten Arbeiten muss sich dieser Bericht auf einen kleinen Teilbereich beschränken. Nachdem ein grober Überblick über das Projekt im Ganzen gegeben wurde bzw. die zentralen zu lösenden Probleme grob umrissen sind, wird daher einer der im Zuge des Praktikums entwickelten Algorithmen etwas detaillierter besprochen. Zum Abschluss wird ein kurzer Ausblick auf andere Tätigkeiten und zukünftige Erweiterungen gegeben.

1.2 Beschreibung des zu lösenden Problems

Grundlegend handelt es sich beim Orbitssimulator (im Folgenden auch *ATP-OSSE* oder nur *OSSE*) um ein Programm, welches aus einer Vielzahl an Parametern und unter Berücksichtigung einer Reihe an Störfaktoren¹ zunächst die Umlaufbahn eines Satelliten bestimmt, und anhand dieser sogenannte (*Sensor-)*Footprints berechnet. Sensorfootprints sind als diejenigen Flächenstücke der Erdoberfläche zu verstehen, welche in eine einzelne Messung des am Satelliten befindlichen Sensors eingehen. Anschaulich: stellt man sich den Sensor als Kamera vor, dann ist der zugehörige Footprint das in einem Foto sichtbare Flächenstück. Siehe hierzu auch Abbildung 1.2 in welcher Ω einen einzelnen Footprint darstellt, wie er z. B. von einem räumlich eindimensionalen CCD-Spektrometer aufgenommen wird. Solche Footprints sind auch in Abbildung 1.1 zu sehen. Aus diesen Footprints werden im späteren Verlauf die Radianzen bestimmt welche der Sensor misst.

1.3 Mathematisches Modell

Das mathematische Modell auf dem die Arbeiten des Praktikums aufbauen ist in Abbildung 1.2 dargestellt: der Satellit wird als Punkt des \mathbb{R}^3 aufgefasst. Seine Trajektorie sei eine glatte Kurve $\gamma : T \rightarrow \mathbb{R}^3$, wobei $T \subseteq \mathbb{R}$ die Zeit repräsentiert. Die Erde wird nach WGS84² als die Oberfläche $E \subset \mathbb{R}^3$ eines Ellipsoids bzw. genauer eines abgeplatteten Rotationsellipsoids modelliert. Die beiden grundlegenden Koordinatensysteme sind die durch

$$u : [-\pi, \pi] \times \left[-\frac{\pi}{2}, \frac{\pi}{2}\right] \rightarrow E, \quad (1.1)$$

$$(\phi, \vartheta) \mapsto u(\phi, \vartheta) := \begin{pmatrix} x_1(\phi, \vartheta) \\ x_2(\phi, \vartheta) \\ x_3(\phi, \vartheta) \end{pmatrix} := \begin{pmatrix} a_1 \cos \vartheta \cos \phi \\ a_2 \cos \vartheta \sin \phi \\ a_3 \sin \vartheta \end{pmatrix} \quad (1.2)$$

¹Eine etwas detailliertere Aufschlüsselung dieser wird in Abschnitt 1.6.1 gegeben.

²Das *World Geodetic System 84*(kurz: WGS84) ist ein geodätisches Referenzsystem zur Positionsangabe von Objekten auf der Erde und im erdnahen Weltraum das z. B. auch dem GPS zugrunde liegt [Fou22b]. Eine detaillierte Beschreibung des Systems (inklusive Aussagen bzgl. des Modellfehlers) ist z. B. in [MG00, S. 185] zu finden.



Abbildung 1.1: Visualisierung einiger Footprints einer CCD-basierten Messung über Europa. Die hier dargestellten Footprints sind sehr viel größer als die von CO2Image. Die starken Größen-schwankungen sind eine Folge der Kartendarstellung und eines exzentrischen Orbits.

induzierten, wobei $a_1 = a_2 > a_3 > 0$ die durch WGS84 definierten Hauptachsen des Ellipsoids sind³. Hierbei wird ϑ als Breitengrad und ϕ als Längengrad bezeichnet. Insbesondere drehen sich diese Koordinatensysteme mit der Rotation der Erde mit - jeder Punkt auf der Erdoberfläche bekommt also zu jedem Zeitpunkt die gleichen Koordinaten zugeordnet. Intern ist es mitunter auch nötig auf sich nicht-mitdrehende Koordinaten zurückzugreifen, daher wird außerdem $\omega_E \in \mathbb{R}$ als die Winkelgeschwindigkeit der Erde definiert. Die Kurve $\vartheta \mapsto u(\vartheta, \pi) = u(\vartheta, -\pi)$ wird als Datumslinie oder auch Datumsgrenze bezeichnet.

Der Vektor $\nu(t), \nu \in \Gamma(TM|_{\gamma(T)})$ ⁴ welcher von $\gamma(t)$ in Richtung der Normalenprojektion (normal zu E) von $\gamma(t)$ zeigt, wird als *Nadir-Vektor* oder kurz *Nadir* bezeichnet - er gibt die Richtung "nach unten" vom Satelliten aus an. Nadir ist also gerade der Vektor $\nu(t) = n(p)$ für den

$$\gamma(t) = p + hn(p) \quad (1.3)$$

für ein $p \in E, h \in \mathbb{R}$ wobei $n : E \rightarrow S^2$ das äußere Normaleneinheitsfeld von E bezeichnet. Den zugehörigen Punkt $p \in E$ bezeichnen wir als den Bodenpunkt von $\gamma(t)$ und h als die Höhe des Satelliten. Intuitiv ist p der Punkt auf der Erde, von welchem aus man den Satelliten ansieht wenn man "nach oben" blickt. Das Kreuzprodukt $\dot{\gamma}(t) \times \nu(t)$ wird als *Cross-Track-Vektor* bezeichnet. Das Tripel $\dot{\gamma}, \nu, \dot{\gamma} \times \nu$ liefert eine i. A. nicht-orthogonale Basis des Tangentialraums des \mathbb{R}^3 bei γ . Diese Wahl dieser Basis ist insoweit "natürlich" für unser Modell, als dass $\dot{\gamma}$ als Nebenprodukt der Orbitintegration abfällt und Nadir kovariant im Bezug zur Rotation der Erde ist: wirkt die Rotation der Erde als $R : T \rightarrow \mathbf{SO}(3), R(0) = I_3$ (hierbei bezeichnet $\mathbf{SO}(3)$ die Drehgruppe des \mathbb{R}^3) auf \mathbb{R}^3 und bildet dabei $\gamma(t)$ auf $R(t)\gamma(t)$ ab, so wird $\nu(t)$ auf $R(t)\nu(t)$ abgebildet - dies gilt für allgemeine Vektorfelder nicht. Mitunter wird allerdings aus Effizienzgründen auch auf eine orthonormale Basis gewechselt.

³Also diejenigen Werte a_1, a_2, a_3 sodass für alle $(x_i)_i \in E : \sum_i \left(\frac{x_i}{a_i} \right)^2 = 1$.

⁴Hier wird ein kleiner abuse of notation begangen: $\Gamma(TM|_N)$ bezeichnet die Menge der glatten Vektorfelder entlang von N in M . Es wird also jedem Punkt p in $N \subseteq M$ ein Tangentialvektor in $T_p M$ zugeordnet. Hier wird $p = \gamma(t)$ mit t identifiziert.

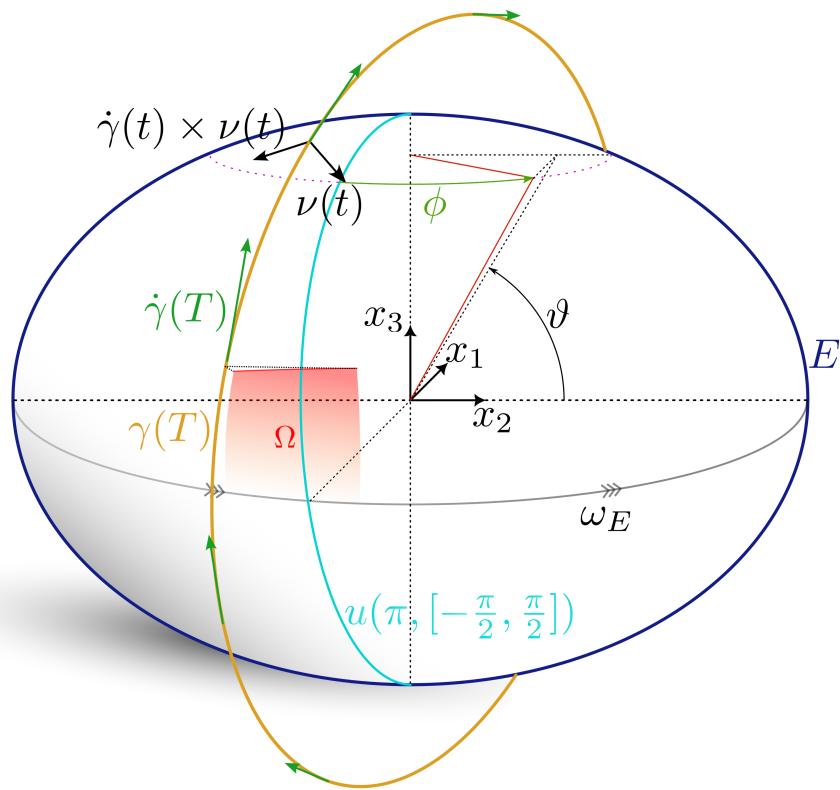


Abbildung 1.2: Mathematisches Modell des Problems

Außerdem wird angenommen, dass es sich bei den Sensorfootprints (in der Abbildung bezeichnet mit Ω) stets um geodätische Polygone handelt - also eine Menge aus Punkten aus E welche mittels einem Weg aus Geodäten auf E verbunden sind. Es ist zwar streng genommen auch nötig mit nicht-polygonalen Footprints zu arbeiten (z. B. bei LIDAR-Messungen - diese arbeiten mit kreisförmigen Laserpulsen. In diesem Fall ist $\partial_E \Omega = K \cap E$, wobei K eine Kegelschale⁵ ist, deren Spitze sich in der Satellitenposition befindet. Hierbei bezeichnet $\partial_E \Omega$ den Rand von Ω in der durch E induzierten Teilraumtopologie. Der beschriebene Rand entspricht also den blauen Kurven aus Abbildung 1.3.), dieses Problem kann jedoch mittels Diskretisierung einfach umgangen werden.

1.4 Zustand des Orbitintegrators zu Beginn des Praktikums

Die Grundfunktionalität der Orbitintegration und Footprintsberechnung bestand bereits zu Praktikumsbeginn - sie wurde bereits vor einigen Jahren von einem studentischen Praktikanten implementiert. Diese Version hatte jedoch noch einige Probleme und ist daher mehr als Prototyp zu verstehen.

Der Orbitintegrator gliederte sich zu Beginn in einige Module welche mittels File-IO⁶ untereinander kommunizierten. Das Hauptmodul rief demnach also einfach eine Programmkomponente nach der anderen über das Betriebssystem auf, sobald im GUI (Graphical User Interface) der Startknopf geklickt wurde.

Die Projektstruktur ist Listing 2 im Anhang zu entnehmen - im Groben lagen alle Python-Quellcodedateien (im Folgenden auch Sourcefiles) direkt im Projektverzeichnis, die C++ und GUI

⁵ Also das Bild von $\{x \in \mathbb{R}^3 : \|(x_1, x_2)\| = ax_3, x_3 \geq 0\}$ unter einer euklidischen Transformation (also einer orientierungserhaltenden Isometrie auf \mathbb{R}^3) für ein $a \neq 0$.

⁶ Also dem Lesen und Schreiben von Dateien.

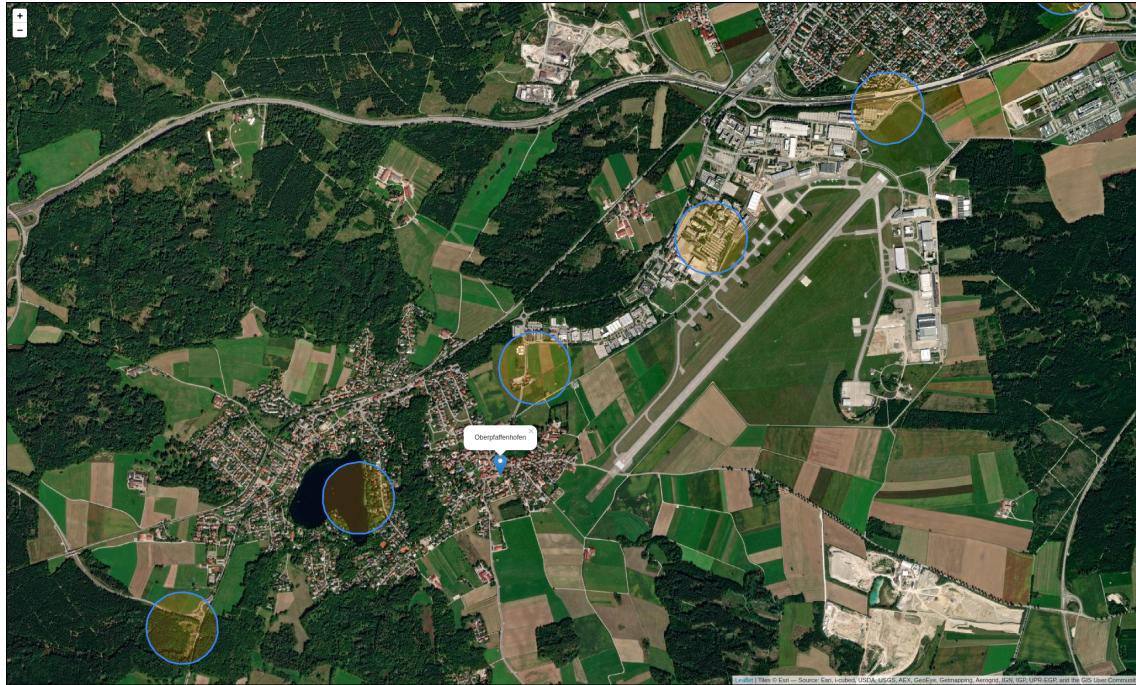


Abbildung 1.3: Footprints einer LIDAR-Messung mit 200m Bodenradius und 5Hz Pulsfrequenz über dem DLR Gelände Oberpfaffenhofen. Visualisiert mittels `folium` über die Geopandas API von OSSE. Es ist außerdem die Diskretisierung der "Kreise" zu sehen.

- Komponenten sowie die Dokumentation waren jeweils in Unterordnern abgelegt.

Um den Orbitimulator auszuführen musste zunächst manuell die C++ Komponente kompiliert und passend in die Struktur platziert, sowie alle Python Abhängigkeiten installiert werden. Anschließend konnte über das Ausführen von `OrbitInt.py` das Programm gestartet werden.

Die bestehenden Probleme waren schlechte Performance, Bugs in der Footprintberechnung sowie ein generell oftmals suboptimaler, nicht idiomatischer Programmierstil, welcher Weiterentwicklung und Wartung erschwerte (z. B. mussten bei Änderungen an der Benutzeroberfläche manuell generierte Codeteile an spezielle Stellen im eigentlichen Quellcode kopiert werden).

Als letzte Statistik zur alten Version des Codes sei noch die Ausgabe von `cloc` (einem Programm zum Zählen von Zeilen von Programmcode) gegeben:

29 text files.				
28 unique files.				
4 files ignored.				
github.com/AlDanial/cloc v 1.82 T=0.03 s (780.3 files/s, 462513.9 lines/s)				
Language	files	blank	comment	code
C++	8	1194	1641	3073
Python	4	482	556	2693
Qt	1	0	0	2500
TeX	3	223	86	557
C/C++ Header	9	313	1022	479
SUM:	25	2212	3305	9302

1.5 Zustand kurz vor Ende des Projektes

Das Projekt wurde im Zuge des Praktikums u. a. grundlegend umstrukturiert, sowie um ein automatisches Build-System ergänzt. Die neue Struktur ist Listing 3 zu entnehmen. Die Anwendungsarchitektur folgt nun dem Prinzip "functional core, imperative shell" - der Großteil des Codes hat also keinerlei Möglichkeiten zur Interaktion mit der Außenwelt und beeinflusst keinen globalen Zustand (besteht also aus sog. *puren* Funktionen). Es ist zu sehen, dass die Dokumentation um einige Beschreibungen und Analysen der implementierten Algorithmen erweitert wurde. Außerdem wurde der Großteil des Codes mit *Type Hints* (vgl. PEP 484 und [Ram22]) entwickelt bzw. i. A. Pythons Typsystem soweit möglich zur Verbesserung der Dokumentation genutzt.

Einen weiteren großen Beitrag liefern außerdem sehr umfassende Docstrings welche die implementierten Algorithmen (falls nötig mit Verweis auf separate Dokumente) erklären, Invarianten, welche Pythons Typsystem nicht erfassen kann, definieren sowie sonstige Informationen über den jeweiligen Code geben. Diese Docstrings sind in einem strukturierten Format⁷ geschrieben, welches die automatische Dokumentationsgeneration mittels Tools wie z. B. pdoc oder Sphinx erlaubt. Weiterhin wurden automatisierte Unit-Tests (sowohl klassische example-based Tests wie auch property-based Tests⁸) ergänzt.

Die Metadaten wie z. B. Versionsnummer, Informationen über Maintainer, Abhängigkeiten etc. des Projekts werden nun alle in einem `pyproject.toml` (siehe hierzu auch PEP 621) File spezifiziert. Dies ermöglicht u. a. eine sehr einfache automatisierte Verwaltung aller Abhängigkeiten mittels pip - kann mit Tools wie poetry aber auch genutzt werden, um eine völlige Kapselung des Projekts inklusive automatischem Management der zu nutzenden Python Version zu erreichen, und ermöglicht weiterhin die einfache Verteilung der Software. Weiterhin kann die Codequalität im Sinne einiger Metriken (z. B. Anzahl an lokalen Variablen innerhalb von Funktionen und Grad der Dokumentation) sowie die Einhaltung von Communitystandards nun über Pylint überprüft werden. Der Pylint-Score ist hierbei 9.95/10 wobei die verbleibenden 3 Warnings noch offene TODOs sind. Die Ausgabe von `cloc --exclude-lang=HTML,SVG .` in der Projektwurzel ist nun:

93 text files.				
92 unique files.				
22 files ignored.				
github.com/AlDanial/cloc v 1.82 T=0.10 s (784.7 files/s, 787805.2 lines/s)				
Language	files	blank	comment	code
Python	33	1103	1210	4616
C++	8	1218	1662	3200
TeX	14	635	258	3017
Qt	1	0	0	2699
C/C++ Header	9	313	1022	479
Jupyter Notebook	5	0	52970	474
make	2	60	37	152
TOML	1	8	8	69
reStructuredText	1	27	1	38
Markdown	1	4	0	17
SUM:	75	3368	57168	14761

Hierbei ist anzumerken, dass die Differenz zwischen altem und neuem Zustand nicht die Gesamtmenge an neuem Code widerspiegelt. Im Zuge der Überarbeitung wurden große Teile des Originalcodes massiv gekürzt (z. B. durch Deduplikation und einen kompakteren Stil - aber auch durch

⁷Konkret orientiert es sich an einem von Google veröffentlichten Format. Dieses ist in Abschnitt 3.8 des Google Python Style Guide beschrieben.

⁸Diese kommen ursprünglich aus der funktionalen Programmierung und die Grundidee ist, dass sich im Gegensatz zu klassischen Tests nicht der Programmierer einige Spezialfälle an Input-Daten für eine Funktion überlegt (welche dann hoffentlich repräsentativ für den kompletten Eingaberaum sind) und anschließend die tatsächlichen Outputs der Funktion anhand dieser überprüft - sondern, dass er den Eingaberaum (oder Teilräume dessen) formal festlegt sowie Invarianten der Funktion ermittelt. Das Testsystem generiert dann automatisch eine Vielzahl an Testcases und überprüft die Invarianten. Für eine detaillierte Beschreibung sei auf die ursprüngliche Haskell library QuickCheck verwiesen.

zeitweise Featureregressionen). Ein Beispiel hierfür ist das im Original 2000 Zeilen umfassende GUI Modul, welches zwischenzeitlich auf ca. 400 Zeilen gekürzt wurde.

1.6 Grundlegende physikalische Hintergründe von ATP-OSSE

1.6.1 Orbitintegration

Unter dem Begriff der Orbitintegration versteht man die Bestimmung des Satellitenorbits aus einem gegebenen Satz an Parametern. Die Parameter umfassen z. B. Satellitenkenndaten wie Masse und effektiven Querschnitt aber auch Orbitdaten wie die sechs Keplerschen Elemente und den exakten Startzeitpunkt der Integration.

Konkret implementiert wird die Orbitintegration mittels einer durch die Abteilung IMF-ATP des DLR erweiterte Version des zu [MG00] gehörigen in C++ implementierten Softwarepakets⁹. Hierbei können je nach Anforderungen des Projektes verschiedene sog. *Perturbations* (engl. Störungen) zugeschaltet werden. Mathematisch wird hierbei ein System gewöhnlicher DGL integriert. Das Grundmodell ist Newton's zweites Gesetz

$$\ddot{\gamma} = \frac{F(t, \gamma)}{m_{\text{sat}}} \quad (1.4)$$

wobei F alle auf den Satelliten wirkenden Kräfte zusammenfasst bzw. diese mittels angemessener Näherungen approximiert.

Symbol	Erklärung
U	Schwerepotential der Erde - sog. <i>Geopotential</i>
G	Gravitationskonstante
m_{lun}	Mondmasse
m_{sat}	Satellitenmasse
m_{sol}	Sonnenmasse
γ_{lun}	Mondposition
γ_{sol}	Sonnenposition
Φ	Lichstrom der Sonne
c	Lichtgeschwindigkeit
A_{sol}	Querschnittsfläche des Satelliten im Bezug zur Sonne
C_D	Reibungskoeffizient des Satelliten
A_{atm}	Querschnittsfläche des Satelliten im Bezug zur Atmosphäre der Erde
$\dot{\gamma}_{\text{atm}}$	Relative Geschwindigkeit des Satelliten im Bezug zur Atmosphäre
ρ	Atmosphärendichte

Tabelle 1.1: Physikalische Größen der die Satelitentrajektorie beschreibenden DGL

Mit allen implementierten Perturbationen und den Symbolbezeichnungen nach Tabelle 1.1 gilt

$$F(\cdot, \gamma) = \underbrace{m_{\text{sat}} \nabla U(\gamma)}_{\text{Anziehung der Erde}} + \underbrace{G m_{\text{sat}} \left(m_{\text{sol}} \left(\frac{\gamma_{\text{sol}} - \gamma}{\|\gamma_{\text{sol}} - \gamma\|^3} \right) + m_{\text{lun}} \left(\frac{\gamma_{\text{lun}} - \gamma}{\|\gamma_{\text{lun}} - \gamma\|^3} \right) \right)}_{\text{Anziehung durch Sonne und Mond}} \quad (1.5)$$

$$+ \underbrace{\frac{\Phi}{c} A_{\text{sol}}(\gamma, \gamma_{\text{sol}})}_{\text{solarer Strahlungsdruck}} - \underbrace{\frac{1}{2} C_D A_{\text{atm}}(\gamma, \dot{\gamma}_{\text{atm}}) \rho(\gamma) \|\dot{\gamma}_{\text{atm}}\| \dot{\gamma}_{\text{atm}}}_{\text{Atmosphärenreibung}}. \quad (1.6)$$

Es sei gesagt, dass die tatsächliche Evaluation der Teilausdrücke hier sehr aufwendig sein kann und ggf. weitere Parameter mit ins Modell bringt (z. B. benötigt man für die Berücksichtigung des

⁹Die Erläuterungen dieses Abschnitts stützen sich auf [MG00].

Strahlungsdrucks im vergleichsweise einfachen implementierten Modell einen Reflexionskoeffizienten für den Satelliten) - auf eine detaillierte Ausführung wird an dieser Stelle allerdings verzichtet. Exemplarisch seien die Grundzüge des Verfahrens für den Fall des Geopotentials dargestellt: das Geopotential U bzw. sein Gradient modellieren den Einfluss der inneren (i. A. inhomogenen) Masseverteilung der Erde auf den Orbit. Mathematisch werden sie zunächst mithilfe von Legendre-Polynomen und Kugelflächenfunktionen entwickelt, welche dann anhand empirisch bestimmter Koeffizienten (siehe hierzu z. B. das Geopotentialmodell JGM-3) und bestimmter Rekursionsschemata numerisch evaluiert werden können. Im Allgemeinen stehen für viele Perturbationen bereits wohltabulierte Modelle zur Verfügung.

Es sind eine Vielzahl weiterer Perturbationen denkbar (z. B. Gezeiteeffekte durch Sonne und Mond, ein Satellitenantrieb, der Strahlungsdruck der Erde oder auch relativistische Effekte), diese haben jedoch einen so geringen Einfluss, dass sie in der vorliegenden Anwendung vernachlässigt werden können. Abseits der relativistischen Effekte bleibt das zu lösende Grundproblem beim hinzufügen dieser Perturbationen gleich - im relativistischen Fall würde es zur Lösung der Geodätengleichung übergehen. Insbesondere handelt es sich aber in jedem Fall um gewöhnliche DGL für welche eine Vielzahl numerischer Löser zur Verfügung stehen. Der Text [MG00, Kapitel 4] enthält eine Übersicht über einige dieser in der Astrodynamik verbreiteten Methoden, einschließlich einer Analyse ihrer Vor- und Nachteile für verschiedene Anwendungen, Implementierungsdetails etc. Es sei hier nur gesagt, dass der implementierte Code sowohl ein Runge-Kutta Verfahren vierter Ordnung sowie ein Mehrschrittverfahren mit Ordnungs- und Schrittweitensteuerung (sog. DE/DEABM von Shampine & Gordon) umfasst.

1.6.2 Strahlungstransfer

Bei der Strahlungstransferrechnung wird grundlegend die Radianz bestimmt welche ein Instrument aufnimmt. Dies erfolgt abseits der Bestimmung der nötigen Eingabeparameter gänzlich mittels Py4CAtS. Die genauen Hintergründe sind in der Py4CAtS Dokumentation bzw. den zugehörigen Artikeln (z. B. [Sch+19]) sowie entsprechender Literatur wie [ZTB07] nachzulesen. Hierbei wird im Grunde die Schwarzschild Strahlungstransfergleichung - ein Spezialfall der allgemeinen Strahlungstransfergleichung unter vereinfachenden Annahmen - integriert.

Symbol	Erklärung
I	Vom Instrumente gemessene Radianz
\tilde{I}	Am Instrument eintreffende Radianz
I_b	Hintergrundstrahlung
ν	Strahlungsfrequenz
T	Temperatur
c	Lichtgeschwindigkeit
h	Plancksches Wirkungsquantum
k	Boltzmann-Konstante
\mathcal{T}	Monochromatische Transmission
τ	optische Tiefe
srf	Instrumentenfunktion / "Spectral response function"
α	Volumenabsorptionskoeffizient
s	Bogenlänge entlang des Strahlungspfads vom Instrument aus

Tabelle 1.2: Physikalische Größen des Strahlungstransports

Mit den Symbolen aus Tabelle 1.2 und mit Unterschlagung einiger Abhängigkeiten (z. B. Höhe, Druck etc.) lautet diese

$$\tilde{I}(\nu) = \lim_{\sigma \rightarrow \infty} I_b(\nu) \mathcal{T}(\nu, \sigma) - \int_0^{s_b} B(\nu, T(\sigma)) \partial_2 \mathcal{T}(\nu, \sigma) d\sigma \quad (1.7)$$

$$= \lim_{\sigma \rightarrow \infty} I_b(\nu) \mathcal{T}(\nu, \sigma) - \int_0^{s_b} B(\nu, T(\sigma)) d\mathcal{T}(\nu, \sigma) \quad (1.8)$$

wobei

$$B(\nu, T) = \frac{2h\nu^3}{c^2 \exp(\frac{h\nu}{kT} - 1)} \quad (1.9)$$

die sog. Planck Funktion bzw. das Plancksche Strahlungsgesetz bezeichnet und

$$\mathcal{T}(\nu, s) = e^{-\tau(\nu, s)} = \exp\left(-\int_0^s \alpha(\nu, \sigma) d\sigma\right) \quad (1.10)$$

nach dem Lambert-Beerschen Gesetz die monochromatische Transmissionsfunktion angibt. Durch Kettenregel und Substitution erhält man

$$\tilde{I}(\nu) = \lim_{\sigma \rightarrow \infty} I_b(\nu) \mathcal{T}(\nu, \sigma) + \int_0^{\tau_b} B(\nu, T(\tau)) \exp(-\tau) d\tau, \quad (1.11)$$

dies ist auch die aktuell implementierte Gleichung welche die Radianz mittels der optischen Tiefen bestimmt. Nachdem \tilde{I} bestimmt ist, ergibt sich die tatsächlich gemessene Radianz durch Faltung (über \mathbb{R}) mit der Instrumentenfunktion srf:

$$I = \tilde{I} * \text{srf}. \quad (1.12)$$

Die Instrumentenfunktion gibt hierbei an wie sensitiv das Instrument für eine jeweilige Frequenz ist und modelliert außerdem "bleedover"-Effekte bei denen das Instrument bei einer Frequenz anspricht, der es eigentlich nicht ausgesetzt ist: die Werte von I sind für $\nu, \varepsilon > 0$ nicht unabhängig auf $B_\varepsilon(\nu)$ ¹⁰; Strahlung der Frequenz ν verursacht potentiell eine Instrumentenantwort bei allen $\tilde{\nu} \in B_\varepsilon(\nu)$. Drei typische Beispiele für srf sind

$$\text{srf}(\nu) \propto \Theta(a - |\nu + \tilde{\nu}|) \quad (\text{Box response}) \quad (1.13)$$

$$\text{srf}(\nu) \propto \max\{0, 1 - |a\nu + \tilde{\nu}| \} \quad (\text{Triangle response}) \quad (1.14)$$

$$\text{srf}(\nu) \propto \exp\left(-\frac{(\nu - \tilde{\nu})^2}{a}\right) \quad (\text{Gaussian response}) \quad (1.15)$$

für geeignete $a, \tilde{\nu} \in \mathbb{R}$ wobei Θ die Heaviside-Stufenfunktion repräsentiert und \propto Proportionalität symbolisiert (also $f(x) \propto g(x) \iff \exists A : f(x) = Ag(x)$). Details zur numerischen Implementierung sind der Py4CAtS Dokumentation sowie [Sch+14] zu entnehmen.

¹⁰ $B_r(x)$ bezeichnet die offene Kugel mit Radius $r > 0$ um $x \in \mathbb{R}^n$ - also $B_r(x) = \{y \in \mathbb{R}^n : \|x - y\| < r\}$.

Kapitel 2

Beschreibung eines Algorithmus zur Überprüfung von Punkt-Polygon Inzidenzen auf der Oberfläche eines Ellipsoids

2.1 Problembeschreibung und Rahmenlage

Um einem Sensorfootprint eine Radianz zuordnen zu können, muss zunächst ermittelt werden, wie die Beschaffenheit der Erde im Bereich des Footprints ist, da diese das Reflexionsverhalten maßgeblich beeinflusst (im Besprochenen Strahlungstransfermodell geht sie als Höhe der Fläche ein). Diese Oberflächenbeschaffenheit wird durch ein topografisches Modell der Erde gegeben¹: dieses liefert zu jedem Punkt der Erdoberfläche die Höhe an diesem Punkt. Formal ist also (in der Notation von Abschnitt 1.3) ein Skalarfeld $\mathcal{T} : E \rightarrow H$ gegeben, aus welchem die Einschränkung $\mathcal{T}|_{\Omega}$ von \mathcal{T} auf Ω zu bestimmen ist.

Rein formal ist dieses Problem trivial, in der Implementierung ist \mathcal{T} jedoch durch ein (equidistantes) Netz der Erdoberfläche² gegeben, während von Ω nur die Eckpunkte bekannt sind. Die Eingabedaten des zu entwerfenden Algorithmus sind also eine diskrete Menge aus Punkten auf E (das Netz) sowie die Ecken des zu behandelnden Polygons. Aus diesen ist zu bestimmen, welche Gitterpunkte "innerhalb" des Polygons liegen; also ob für $p \in E$ gilt, dass $p \in \Omega$ oder $p \notin \Omega$.

Zur Lösung dieses Problems existieren bereits Algorithmen (alle untersuchten Python-Implementierungen stützen sich hierbei auf `pygeos` - einem Python-Wrapper der C Library GEOS), diese sind jedoch (vermutlich aufgrund einer hier unzweckmäßig hohen Genauigkeit) für die vorliegende Anwendung zu langsam. Ein sehr einfacher Test auf dem Bürorechner des DLR mit `geopandas` bei 100,000 Punkten und 200 Polygonen mit jeweils 8 Ecken benötigte ~ 96 s während der hier beschriebene Algorithmus für die selben Eingabedaten ~ 2 s benötigt.

2.2 Grundlegende mathematische Definitionen

Im Folgenden gelten folgende Definitionen:

Definition 2.1 (Disjunkte Vereinigung disjunkter Mengen).

Sei $(A_i)_{i \in I}$ eine Familie disjunkter Mengen (es gilt also $i \neq j \implies A_i \cap A_j = \emptyset$) über einer Index-

¹Solche Modelle sind auch als DEMs(engl. *digital elevation models*) bekannt[Env22].

²Eines der in ATP-OSSE verfügbaren Modelle ist z. B. dasETOPO1 Global Relief Model welches Paare aus Längen- und Breitengraden mit einer Auflösung von einer Winkelminute zu Höhen relativ zum Meeresspiegel auflöst.

menge I . Dann bezeichnen wir die Vereinigung aller A_i als disjunkte Vereinigung und schreiben

$$\bigsqcup_{i \in I} A_i \quad (2.1)$$

für $\bigcup_{i \in I} A_i$. Insbesondere impliziert diese Schreibweise die Disjunkttheit aller A_i , auch wenn diese nicht explizit spezifiziert wurde.

2.3 Vereinfachende Annahmen und Vorüberlegungen

2.3.1 Approximation von Geodäten auf dem Erdellipsoid

An vielen Stellen im Projekt sind Berechnungen mit Geodäten durchzuführen. Das Problem hierbei ist, dass Geodäten auf einem Ellipsoid sehr viel komplizierter als auf einer Kugel sind: diese sind i. A. keine geschlossenen Kurven und können nicht in elementarer Form beschrieben werden. Daher werden diese wie folgt angenähert:

Seien $p, q \in E, p \neq \pm q$ zwei Punkte, zwischen denen die Geodäte $\gamma : S \rightarrow E$ zu bestimmen ist³. Weiterhin sei $D = \text{diag}(a_1, a_2, a_3) \in \mathbb{R}^{3,3}$ eine Diagonalmatrix mit den Hauptachsen des Ellipsoids. Dann definieren wir

$$\tilde{\gamma}(\theta) := \tilde{\gamma}_{p,q}(\theta) := \frac{R(\theta, p \times q)p}{\|D^{-1}R(\theta, p \times q)p\|} \quad (2.2)$$

als Näherung für die Geodäte welche p und q verbindet. Hierbei ist $R(\theta, p \times q) \in \mathbf{SO}(3)$ eine Drehung um die Achse $p \times q$ und den Winkel θ . Diese Kurve ist die Projektion eines Großkreises (also einer Geodäten auf der Sphäre) auf das Ellipsoid - da die Erde im WGS84 Modell nahezu eine perfekte Sphäre ist, liegt diese Approximation nahe. Sie hat folgende Eigenschaften⁴:

- $\tilde{\gamma}(0) = p$ und $\tilde{\gamma}(\angle(p, q)) = q$
- $\tilde{\gamma}$ liegt in einem zweidimensionalen Unterraum des \mathbb{R}^3 (der Ebene durch p, q und 0) (dies sieht man leicht ein, da $p \times q$ ein Eigenvektor von $R := R(\theta, p \times q)$ und somit von $R^T = R^{-1}$ zum Eigenwert 1 ist; $p \in (p \times q)^\perp$ und daher $\langle Rp, p \times q \rangle = \langle p, R^T(p \times q) \rangle = \langle p, p \times q \rangle = 0$).
- Für $a_1 = a_2 = a_3$ (also E eine Sphäre) ist $\tilde{\gamma}$ eine Geodäte (folgt aus Unterraumseigenschaft und $\partial_\theta \|\tilde{\gamma}_{p,q}(\theta)\| = 0$).

In der Dokumentation des Projektes findet sich auch eine Fehleranalyse dieser Annäherung sowie eine Berechnung der skalaren Krümmungen (geodätische sowie Normalkrümmung) von $\tilde{\gamma}$. Diese Analyse bzw. die zugehörige numerische Simulation, ermöglicht für eine gegebene Geodätenlänge $\ell \geq 0$ sowie eine Bogenlänge $s \in [0, \ell]$ entlang der Geodäten die Evaluation der Funktion

$$\delta(\ell, s) := \max_{\substack{p, q \in E \\ d_E(p, q) = \ell}} \left(\inf_{\tilde{s} \in [0, \angle(p, q)]} \|\gamma_{p,q}(s) - \tilde{\gamma}_{p,q}(\tilde{s})\| \right), \quad (2.3)$$

also einer oberen Schranke der (euklidischen) Abweichung zwischen (nach Kurvenlänge parametrisierter) tatsächlicher Geodäte $\gamma_{p,q} : S \rightarrow E$ und approximativer Kurve $\tilde{\gamma}_{p,q}$ im zu erwartenden Längenbereich der Geodäte⁵. Hierbei bezeichnet d_E die geodätische Entfernung auf E , wobei aufgrund der Dreiecksungleichung stets $\|u - v\| \leq d_E(u, v)$ gilt.

³Hierbei ist diejenige Geodäte gemeint, welche gleichzeitig die kürzeste Verbindungscurve von p, q auf E ist. Intuitiv ist die Existenz dieser Geodäten klar, formal folgt sie nach [GQ20, S. 7] daraus, dass es sich bei E um eine kompakte riemannsche Mannigfaltigkeit handelt.

⁴Hierbei bezeichnet $\angle(p, q) := \arccos \frac{\langle p, q \rangle}{\|p\| \|q\|}$ den "inneren" Winkel zwischen p und q .

⁵Das äußere Maximum ist hierbei aufgrund der Kompaktheit der Familie $M_q := \{p \in E : d_E(p, q) = \ell\} = d_E(\cdot, q)^{-1}\{\ell\}$ sowie der Stetigkeit des inneren Ausdrucks wohldefiniert. Dabei ist M_q kompakt da es als Urbild einer kompakten Menge unter einer stetigen Funktion abgeschlossen und als Teilmenge eines Kompaktums somit kompakt ist.

Die Simulation zeigt, dass $\delta(3 \cdot 10^6 \text{ m}, s)$ näherungsweise der Funktion $\tilde{f}(s) = \alpha_0 + \alpha_1 s + \alpha_2 s^2$ mit $\alpha_0 = 1.08720361, \alpha_1 = -1.09662020 \cdot 10^{-5}, \alpha_2 = 1.08543551 \cdot 10^{-11}$ folgt (hierbei sind sowohl s als auch $\tilde{f}(s)$ Längen in Metern.). Insbesondere gilt auf dem Intervall $[0, 3 \cdot 10^6]$

$$\|\delta(3 \cdot 10^6 \text{ m}, \cdot)\|_\infty < 70 \text{ m}; \quad (2.4)$$

entlang einer Strecke von $3 \cdot 10^6 \text{ m}$ ist der durch die Näherung verursachte Fehler also kleiner 70 m - auf kürzeren Intervallen ist er entsprechend sehr viel kleiner, so ergibt sich bei einer Länge von 1000 km beispielsweise eine Fehlerschranke von 2.4 m. Der Grenzwert von $3 \cdot 10^6 \text{ m}$ wurde gewählt, da selbst die größten Footprints kleinere Seitenlängen haben⁶.

Somit liegt der Fehler in einem für das Projekt vertretbaren Bereich und es wird in den meisten Fällen mit der Näherung statt einer tatsächlichen Geodäten gearbeitet.

2.3.2 Auflösen einer Mehrdeutigkeit in der Problemstellung



(a) Polygon A welches den gegebenen Rand "über die Vorderseite" der Erde interpoliert.

(b) Polygon welches den gegebenen Rand "über die Rückseite" der Erde interpoliert. Die Polygonfläche ist also unter Vernachlässigung des Randes $E \setminus A$.

Abbildung 2.1: Zwei Flächenstücke mit gleichem Rand

Ein weiteres Problem ist, dass die Eckpunkte eines Footprints diesen zunächst nicht eindeutig festlegen (vgl. Abbildung 2.1). Wir definieren daher den Footprint, welcher durch eine Folge von Eckpunkten interpolierenden geodätischen Weg ω gegeben ist, als diejenige Minimalfläche Ω welche

$$\min_{\substack{\Omega \subseteq E \\ \partial_E \bar{\Omega} = \omega}} \text{vol}_2(\Omega) \quad (2.5)$$

löst, sofern diese existiert und eindeutig ist. Offensichtlich existieren Fälle, in denen kein solches eindeutiges Ω existiert (z. B. falls ω entlang des Äquators verläuft - in diesem Fall existieren zwei Flächen mit identischem, minimalen Flächeninhalt). Diese degenerierten Fälle können jedoch aufgrund der Beobachtungsgeometrie des Satelliten in der Praxis nicht auftreten und daher vernachlässigt werden.

2.3.3 Äquivalenz zum konvexen Problem

Weiterhin sind Polygone denkbar, die nicht "konvex auf E " sind. Diese treten bei festen Beobachtungen nicht auf, können allerdings beim sog. *Targeting* auftreten: hier wird der Sensor während der Bewegung des Satelliten kontinuierlich nachjustiert, um ein gegebenes *Ziel* auf der Erdoberfläche (z. B. ein Kraftwerk) zu beobachten. Bei einem räumlich eindimensionalen Sensor mit räumlicher Ausdehnung orthogonal zur Bewegungsrichtung tritt hierdurch aufgrund der Krümmung der Erde ein Effekt ein, der zu einem grob "fliegenförmigen" Footprint führt. Ein solcher Footprint ist insbesondere nicht konvex im Sinne der folgenden Definition:

⁶Der Erdbeobachtungssatellit Sentinel-5P hat beispielsweise eine Footprintgröße (einen sog. *Swath*) von ca. $2600 \text{ km} \times 7 \text{ km}$ (Wert aus persönlichem Gespräch mit Herrn Dr. Günter Lichtenberg aus der Abteilung ATP des DLR - kann jedoch auch im Instrumentenhandbuch unter [KNM] gefunden werden).

Definition 2.2 (Geodätisch konvexe Menge).

Sei M eine riemannsche Mannigfaltigkeit und $N \subseteq M$. Wir nennen N *geodätisch konvex auf M* , wenn für alle Punkte $p, q \in N$ eine minimale Geodäte $\gamma : [a, b] \rightarrow M$ für $a, b \in \mathbb{R}$ auf M existiert, sodass $\gamma(a) = p, \gamma(b) = q$ und $\gamma([a, b]) \subseteq N$. Den Präfix "geodätisch" lassen wir weg sofern dieser aus dem Kontext klar ist. Im Spezialfall $M = E$, sprechen wir auch von *Geokonvexität*.

Rein formal arbeitet der Algorithmus mit einer leicht modifizierten Version dieser Definition, welche als Verbindungskurven die zuvor besprochene Approximation $\tilde{\gamma}$ nutzt. Wir nennen die entsprechenden Mengen *fast geodätisch konvex*, verzichten jedoch i.d.R. auf den Präfix "fast".

Da Footprints stets eine Triangulation besitzen [Tho92, Theorem 4.1] und wir jedem Dreieck dieser Triangulation homöomorph ein geodätisches Dreieck zuordnen können, lässt sich eine Familie $\Omega_1, \dots, \Omega_n \subseteq E$ aus disjunkten geodätischen Dreiecken finden, sodass $\Omega = \bigsqcup_i \Omega_i$. Da diese Ω_i insbesondere konvex auf E sind, ist das Ursprungsproblem äquivalent zu dem für geodätisch konvexe Mengen: können wir für einen Punkt bestimmen, ob er innerhalb einer der geodätisch konvexen Mengen liegt, so können wir auch bestimmen ob, er innerhalb der i. A. nicht konvexen Vereinigung der Mengen liegt.

Somit ergibt sich die Forderung, dass Ω geodätisch konvex auf E ist.

Eine mögliche Formalisierung der bisherigen Überlegungen, welche auch dem Beweis des Algorithmus (welcher hier aus Platzgründen nicht weiter ausgeführt wird) zugrunde liegt, lautet wie folgt:

Definition 2.3 (Fast-geokonvexer Footprint).

Sei $v_1, \dots, v_n \in E$ eine endliche E -wertige Folge. Weiterhin sei $\Delta(\Omega)$ ein orientierter Simplizialkomplex^a über

$$\bigcup_i \{v_i\} \cup \bigcup_i \{v_i, v_{i+1}\} \cup \{v_n, v_1\} = \{\{v_1\}, \dots, \{v_n\}, \{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}, \{v_1, v_n\}\}. \quad (2.6)$$

Dann nennen wir Ω den *fast-geokonvexen Footprint* zur simplizialen 1-Kette

$$\sigma(\Omega) := v_1 v_2 + \dots + v_{n-1} v_n + v_n v_1 \in C^1(\Delta(\Omega)) \quad (2.7)$$

mit Ecken $V(\Omega) = \{v_1, \dots, v_n\}$ genau dann, wenn

$$(F0) \text{ conv}_E(V) = \Omega$$

$$(F1) \partial_E \Omega = \bigcup_{i=1}^n \text{conv}_E(\{v_i, v_{i+1}\}), \text{ wobei die Vereinigung überall außer an den Punkten } v_i \text{ disjunkt ist und } v_{n+1} := v_1$$

$$(F2) \nexists t \in [0, 1], i \in \{1, \dots, n\}, i \neq k \neq i+1 : \alpha_{v_i, v_{i+1}}(t) = v_k \text{ (Keines der } v_i \text{ liegt auf der zwei anderen Punkte verbindenden Geodäte).}$$

Wir bezeichnen $\sigma(\Omega)$ als *Rand-Kette* von Ω .

^aDie genutzte Formulierung über Komplexe / Ketten wird hier nur eingesetzt, um eine einfachere und weniger mehrdeutige Definition geben zu können - man kann sich die Kette als orientierten Rand des Polygons vorstellen. Details können z. B. in [Hir03] und [Sav16] nachgelesen werden.

Hierbei ist conv_E die entsprechende Verallgemeinerung der konvexen Hülle auf E analog zur (fast) geodätischen Konvexität und $\alpha_{p,q} : [0, 1] \rightarrow E, t \mapsto \tilde{\gamma}_{p,q}(t\angle(p, q))$ die approximierte minimale Geodäte von p und q . Weiterhin ist die Abbildung $\sigma(\Omega) \mapsto \Omega$ i. A. nicht invertierbar, dies ist für die Überlegungen des Algorithmus jedoch nicht relevant. Im Folgenden werden diese Formalitäten und die exakte Definition jedoch unterschlagen - es wird stattdessen mit einem eher intuitiven Footprintbegriff gearbeitet.

2.4 Entwicklung des Algorithmus

Das planare Analogon des Problems ist ein Standardproblem der algorithmischen Geometrie. Diese liefert verschiedene gängige Lösungsansätze; einige sind z. B. in [Fou22a] aufgeführt. Eine Klasse

an Algorithmen basiert hierbei auf der sog. Umlaufzahl des Randes des Polygons: diese gibt an, wie oft (und in welche Richtung) eine Kurve bzw. ein Weg einen Punkt "umkreist". Auch weitaus komplexere Segmentierungsalgorithmen, wie z. B. der in [JKS13] beschriebene, basieren auf der Umlaufzahl.

Im Folgenden bezeichnen wir sowohl den Footprint Ω wie auch seinen Rand $\partial_E \Omega$ sowie die Eckpunkte, welche diesen Rand generieren, als das zu betrachtende Polygon.

2.4.1 Innen-Außen Segmentierung in planarer Geometrie

Ein einfacher Algorithmus für den planaren Fall basiert auf der Erkenntnis, dass jede Kante des Polygons den \mathbb{R}^2 in zwei offene Halbräume (sowie die Kante selbst) zerlegt. Man findet nun eine Möglichkeit, "gleichmäßig" einen dieser Halbräume auszuwählen (gerade so, dass die gewählten Halbräume $(H_i)_i$ die Eigenschaft $\Omega = \bigcap_i H_i$ erfüllen) und zu evaluieren, ob p in den Raum fällt oder nicht. Der Algorithmus lautet wie folgt:

Sei $V = (v_1, \dots, v_n)$, $v_i \in \mathbb{R}^2$, $v_{n+1} := v_1$ ein konvexes Polygon und $p \in \mathbb{R}^2$ ein Punkt, für welchen zu überprüfen ist, ob er innerhalb oder außerhalb von V liegt.

Zunächst wird den Liniensegmenten (v_i, v_{i+1}) nun eine einheitliche Orientierung zugewiesen. Hierdurch lässt sich das Gesamtproblem darauf zurückführen, festzustellen ob sich p relativ zu dieser Orientierung auf der gleichen Seite aller Liniensegmente befindet (intuitiv fährt man also die Kanten des Polygons nacheinander ab und ermittelt jeweils ob p sich "links" oder "rechts" befindet).

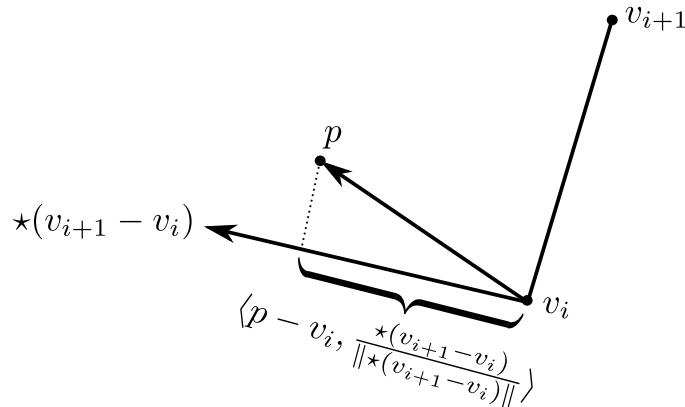


Abbildung 2.2: Geometrischer Zusammenhang im planaren Fall

Sei daher \star der Hodge-Stern-Operator (siehe hierzu z. B. [Win10] oder [Cra+13]), und sgn die Signumsfunktion, dann gilt

$$\text{sgn}\langle p - v_i, \star(v_{i+1} - v_i) \rangle \in \{-1, 0, 1\} \quad (2.8)$$

an auf welcher Seite des Liniensegments (v_i, v_{i+1}) sich p befindet (oder ob es koinzident mit der Linie ist). Ist dieser Ausdruck für alle i gleich, dann befindet sich p innerhalb des Polygons. Es gilt also

$$p \text{ innerhalb von } V \iff \left| \sum_{i=1}^n \text{sgn}\langle p - v_i, \star(v_{i+1} - v_i) \rangle \right| = n. \quad (2.9)$$

Insbesondere ist bei diesem Vorgehen die Orientierung des Randes unseres Polygons egal: das Polygon (v_1, \dots, v_n) und das Polygon (v_n, \dots, v_1) liefern das gleiche Resultat für v (Zum Beweis: Aussage folgt aus Orthogonalität von \star durch Ausnutzen der Bilinearität von $\langle \cdot, \cdot \rangle$ und direktem Nachrechnen für beide Orientierungen.).

2.4.2 Innen-Außen Segmentierung in sphärischer Geometrie

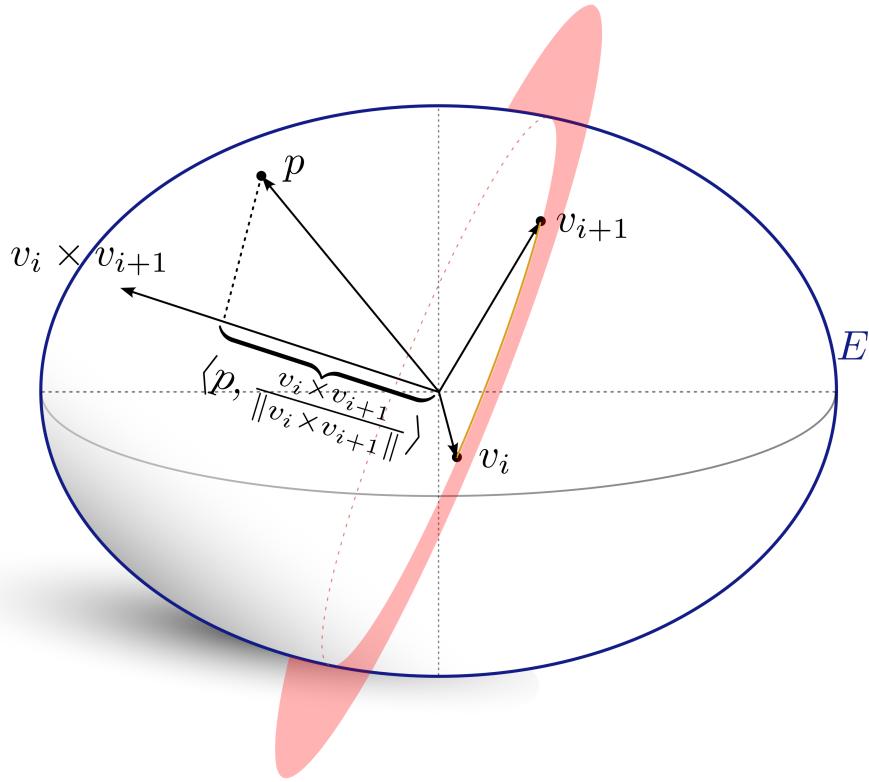


Abbildung 2.3: Halbraum-Test im \mathbb{R}^3 : der rote Ring symbolisiert hier die durch 0 und $v_i, v_{i+1} \in E$ definierte Ebene, welche zusammen mit der Randnormalen $v_i \times v_{i+1}$ einen Halbraum $H = \{\alpha_1 v_i + \alpha_2 v_{i+1} + \beta v_i \times v_{i+1} : \alpha_j \in \mathbb{R}, \beta > 0\}$ aufspannt. Es wird überprüft, ob $p \in E$ in H liegt.

Das Grundverfahren im sphärischen Fall ist analog: es sei $V = (v_1, \dots, v_n)$, $v_i \in E$, $v_{n+1} := v_1$ ein geodätisch konvexes Polygon auf E und $p \in E$ der Punkt, für welchen die Lage im Bezug auf das Polygon zu ermitteln ist. Die durch $v_{i+1} - v_i$ definierte Verbindungsgerade zwischen zwei Ecken des Polygons geht nun über zur durch $v_i \wedge v_{i+1}$ definierten Ebene (da der getroffenen Annäherung nach die geodätische Verbindung der beiden Punkte der Schnitt dieser Ebene mit E ist) und der zu betrachtende Ausdruck wird somit

$$\chi(p, V) := \sum_i \operatorname{sgn} \langle p - v_i, \star(v_i \wedge v_{i+1}) \rangle \quad (2.10)$$

$$= \sum_i \operatorname{sgn} \langle p, v_i \times v_{i+1} \rangle \quad (2.11)$$

$$= \sum_i \operatorname{sgn} \det(p, v_i, v_{i+1}), \quad (2.12)$$

$$\text{mit } v_{n+1} := v_1. \quad (2.13)$$

Definiert man $-V := (v_n, \dots, v_1)$, so ist aufgrund der vollen Antisymmetrie der Determinante (und der Ungeradheit von sgn) sofort ersichtlich, dass

$$\chi(p, -V) = -\chi(p, V) \quad (2.14)$$

und aufgrund der Multilinearität von \det

$$\chi(-p, V) = -\chi(p, V). \quad (2.15)$$

χ ist also ungerade in beiden Argumenten. Insbesondere ist $\chi(-p, -V) = \chi(p, V)$: das Drehen der Orientierung des Polygons entspricht also einer Spiegelung des Punktes am Ursprung. Dieses Verhalten ist in Abbildung 2.4 dargestellt und der Grund, wieso hier im Gegensatz zum planaren Fall, kein betragsmäßiger Vergleich durchzuführen ist und stattdessen eine Orientierung des Polygons festgelegt werden muss.

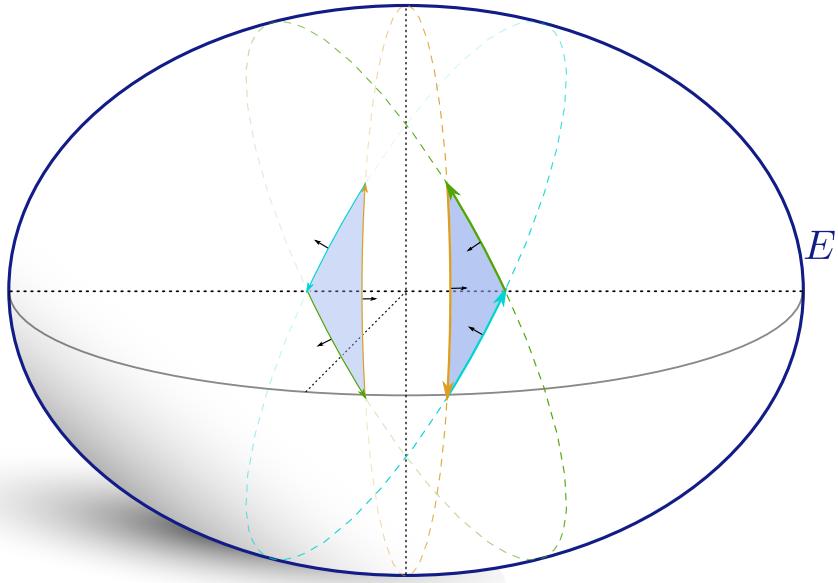


Abbildung 2.4: Auftretendes Problem beim sphärischen Fall: Es existieren zwei unterschiedliche Polygone bzgl. derer Punkte "gleichmäßig" orientiert sind.

Je nach Orientierung ist der durchzuführende Test also

$$\chi(p, V) = n \quad (2.16)$$

$$\text{oder } \chi(p, V) = -n, \quad (2.17)$$

bzw. Orientierungsunabhängig

$$\chi(p, V) = no \text{ mit } o := \operatorname{sgn} \det(v_1, v_2, v_3). \quad (2.18)$$

2.5 Implementierung und Zusammenfassung

Wie im vorhergehenden Abschnitt erläutert, lässt sich also für einen durch die Eckpunkte $v_1, \dots, v_n \in E$ definierten Footprint und einen Punkt $p \in E$ auf der Erdoberfläche durch die Überprüfung der Gültigkeit der Gleichung

$$\sum_i \operatorname{sgn} \langle p, v_i \times v_{i+1} \rangle = n \quad (2.19)$$

ermitteln, ob p "innerhalb" oder "außerhalb" des Footprints liegt. Hiermit lässt sich ein diskretes Netz mit dem Footprint schneiden, indem man für jeden Punkt des Netzes den "innen oder außen"-Test durchführt. Dies ermöglicht letztendlich die Evaluation von diskreten Skalarfeldern (bzw. generellen Funktionen) auf den Satellitenfootprints. Dies wird u.a. dazu genutzt, um in Strahlungstransportrechnungen die Oberflächenbeschaffenheit der Erde korrekt mit einzubeziehen.

Beispiel 2.4.

Das oben erwähnte *ETOPO1 Global Relief Model* besteht im Grunde genommen aus 3-Tupeln

$$(\vartheta_i, \phi_j, h_{ij})_{i=0, \dots, 10800-1, j=0, \dots, 21600-1} \quad (2.20)$$

mit

$$\vartheta_i = \pi \frac{i}{10800 - 1} - \frac{\pi}{2} \in [-\frac{\pi}{2}, \frac{\pi}{2}] \quad (\text{Breitengrad}) \quad (2.21)$$

$$\phi_j = 2\pi \frac{j}{21600 - 1} - \pi \in [-\pi, \pi] \quad (\text{Längengrad}). \quad (2.22)$$

Durch Anwenden der Parametrisierung u von E auf die ersten beiden Komponenten erhält man Paare (p_{ij}, h_{ij}) aus Punkten auf E und den entsprechenden Höhen. Man kann nun für jedes p_{ij} überprüfen, ob es in einem Footprint liegt oder nicht, und somit jedem Footprint eine Teilmenge all dieser Paare zuordnen welche die Topographie des Footprints modelliert. Diese Topografie fließt dann als mittlere Oberflächenhöhe in die mittels `py4cats.od2ri` bestimmte Lösung der Schwarzschild-Strahlungstransfergleichung ein ^a.

^aEs sei angemerkt, dass die aktuell online verfügbare Version von py4cats den entsprechenden Parameter für Höhendaten noch nicht besitzt. ATP-OSSE nutzt eine noch nicht öffentliche Entwicklungsversion mit mehr Parametern.

Listing 1 zeigt, wie die Gültigkeitsüberprüfung der entwickelten Gleichung tatsächlich in ATP-OSSE implementiert ist (abzüglich eines Teils des Docstrings, welcher die Implementierung beschreibt, und weiterer Notizen welche hier bereits durch den Haupttext abgedeckt sind). Diese führt den Test für alle Paare des kartesischen Produktes aus einer Menge an Punkten und einer aus Polygonen durch; was im vorgesehenen Einsatzzweck die Effizienz erhöht - weiterhin erlaubt dies auch sehr bequem die Überprüfung für nicht-konvexe Polygone. Um die Effizienz weiter zu erhöhen, wird außerdem vorab noch sog. ein Bounding-Box-Check (siehe z. B. [PJH16]) durchgeführt, um eine grobe Vorauswahl der Punkte zu treffen.

Bild 2.5 zeigt mittels dieser Implementierung erstellte Punktwolken. Hierbei ist zu sehen, dass tatsächlich das ursprünglich festgelegte Polygon als "innerhalb" erkannt wird. Weiterhin ist hier das Polygon dargestellt, welches man für die analoge Gleichung mit $-n$ als rechte Seite erhält.

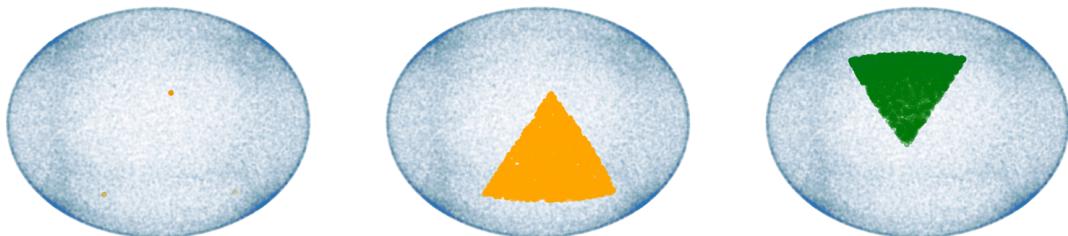


Abbildung 2.5: Mithilfe von Listing 1 berechnetes Beispiel. Links: Punktwolke aus Punkten eines Ellipsoids sowie drei Eckpunkte eines Polygons. Mitte: innerhalb des Polygons liegende Punkte nach "korrekter" Orientierung. Rechts: innerhalb des Polygons liegende Punkte nach "falscher" Orientierung. Der dreieckige Footprint entspricht hier keinem tatsächlichen Sensor (zumindest nicht direkt - der weiter oben beschriebene "fliegenförmige" Footprint ließe sich z. B. in zwei Dreiecke zerlegen), sondern stellt einen der implementierten Testfälle dar.

```

def point_inside_geodetic_polygon(vertices: np.ndarray, points: np.ndarray) → np.ndarray:
    """Check if a point is inside a polygon (if many points are inside many polygons
       actually) on the surface of an ellipsoid.
    This does not require the polygons to be wrapped on the dateline.

    Returns: T×P bool array where entry (t,p) is `True` iff point p is inside
            polygon t.

    Args:
        vertices: T×V×3 float array. First dimension is number of polygons, second
                  the number of vertices of the polygon and third cartesian coordinates
                  of the points *in space*. Units are arbitrary as long as they fit
                  the points.
        points: P×3 float array of points in cartesian coordinates that are to be
                checked against the polygons spanned by the `vertices`.

    Impl: [ ... ]

    Note: [ ... ]
          This function always uses the orientation induced by considering the
          side to which  $v_1, v_2$  points as inside. So it assumes that  $\text{sgn} \langle v_1, v_2 \rangle = 1$ .
    """
    line_starts = vertices
    # associates to each vertex  $v_i$  the vertex  $v_{i+1}$  where possible and  $v_0$  to the last one.
    line_stops = np.hstack((vertices[ ..., 1:, :], vertices[ ..., [0], :]))
    side = np.int64(np.sign(
        np.einsum("ps,tvs→tvp", points, np.cross(line_starts, line_stops)))
    ))
    # note that we compare with vertices.shape[1] - 1 because every vertex is the start of one
    # edge, but the starting vertex is contained *twice* (so that the polygon's boundary is a
    # closed curve)
    return side.sum(axis=-2) = vertices.shape[1] - 1

```

Listing 1: Implementierung des Punkt-Polygon-Inzidenz Tests

Anhang A

Kurzzusammenfassung sonstiger Tätigkeiten

Das Praktikum umfasste eine Vielzahl weiterer Tätigkeiten. Mathematisch interessant waren hierbei z. B.

- Vektorfeldtransport über einer rotierenden Erde: gegeben ist ein Vektorfeld entlang von γ in sog. "earth-fixed" (also mit der Erde rotierenden) Koordinaten und zu ermitteln ist das korrespondierende Vektorfeld in "space-fixed" (also nicht rotierenden) Koordinaten. Durch die Drehung der Erde bzw. induzierte Corioliskraft ändert sich z. B. die Geschwindigkeit des Satelliten relativ zu einem (nicht) inertialen Beobachter im Erdkern, was zu einer Scherung des Tangentialraums führt. Die Lösung umfasste im Grunde genommen die Konstruktion einer linearen Abbildung in einer "natürlichen" Basis, die Entwicklung einer passenden Orthonormalbasis und Transfer der Abbildung in diese.
- Kartenprojektion von Vektorfeldern entlang des Satellitenorbits: gegeben ist wiederum ein Vektorfeld V entlang der Satellitentrajektorie im Raum (z. B. Beobachtungsrichtungen) und gesucht ist ein entsprechendes Vektorfeld im Kartengebiet, sodass diese Vektorfelder einander "entsprechen". Gelöst wurde dies über das Bilden der Pseudoinverse der Jacobimatrix der Kartenabbildung - das Vektorfeld im Kartengebiet ist also $(J_u)^\dagger V^\top$ wobei \cdot^\top Projektion auf den Tangentialraum von E bezeichnet.
- Ermitteln einer sinnvollen Definition des zunächst nicht wohldefinierten *Bodenradius* und Lösen verbundener inverser Probleme: LIDAR Messungen haben wie oben beschrieben "runde" Footprints. In der Planung einer Satellitenmission plant man hierbei auf Basis eines sog. Bodenradius: also des Radius des Footprints. Da die Footprints i. A. jedoch keine geodätischen Kreise sind, ist dieser Radius zunächst nicht wohldefiniert. Es wurde also eine Formalisierung des intuitiven Bodenradius gesucht. Diese ist für eine LIDAR Messung bei einem Optiköffnungswinkel¹ α mit Kegel $K(\alpha)$ beim Punkt $p \in E$ durch

$$r(\alpha) := \frac{\int_{\partial_E(K(\alpha) \cap E)} d_E(p, x) \, dx}{\text{vol}_1(\partial_E(K(\alpha) \cap E))} \quad (\text{A.1})$$

gegeben. Da für die Rechnungen der Öffnungswinkel α der Optik relevant ist, muss die Inverse dieser Abbildung bestimmt werden. Dies geschieht für einen Bodenradius $\rho > 0$ durch Lösen des Optimierungsproblems

$$\min_{\alpha \in (0, \frac{\pi}{2})} |\rho - r(\alpha)|, \quad (\text{A.2})$$

mittels einiger vereinfachender Annahmen (z. B. ist α bzw. ρ i.d.R. sehr klein [$\rho < 500 \text{ m}$], weshalb die Erde lokal in äußerst guter Näherung als Sphäre angenommen werden kann).

¹Aufgrund der hohen Entferungen können die Strahlen des beim LIDAR genutzten Lasers nicht mehr als parallel angenommen werden. Den Winkel zwischen diesen bezeichnen wir als Optiköffnungswinkel α .

- Generation orthogonaler Vektorfelder: an mehreren Stellen (z. B. bei Tests oder auch als Teils eines alternativen Punkt-in-Polygon Algorithmus) ist es nötig aus einem gegebenen, nirgends verschwindenden Vektorfeld V im \mathbb{R}^3 weitere Vektorfelder E_2, E_3 zu generieren, sodass $\frac{V}{\|V\|} =: E_1, E_2, E_3$ an jedem Punkt eine Orthonormalbasis des \mathbb{R}^3 ist. Dies ist dazu äquivalent ein Vektorfeld $U : S^2 \rightarrow \mathbb{R}^3$ zu finden, sodass für alle x die Vektoren $x, U(x)$ linear unabhängig sind. Dann erfüllt nämlich die Basis $E_1, E_2 := \text{proj}_{E_1^\perp}(U(E_1)), E_1 \times E_2$ die Forderung (und umgedreht lässt sich aus einem Satz Basisvektoren stets ein solches U konstruieren). Im Idealfall sollte U stetig und "einfach" bzw. effizient zu implementieren sein (also z. B. jede Komponente von $U(x)$ ein Polynom in den Komponenten von x). Es stellt sich jedoch heraus, dass dies mathematisch nicht möglich ist: der *Satz vom Igel* (teils auch *Satz von Poincaré-Brouwer* oder engl. *hairy ball theorem*) [Hir76, S. 125] sagt aus, dass jedes Vektorfeld auf geradzahligen Dimensionalen Kugeln eine Singularität besitzt. Im vorliegenden Fall ist die Dimension der Kugel 2 und somit muss eine Singularität existieren. Bei jeder solchen Singularität ergibt sich aber in unserem Fall lineare Abhängigkeit zwischen x und $U(x)$ und es muss daher ein unstetiges Vektorfeld gewählt werden. Konkret wird die Abbildung

$$x \mapsto \begin{cases} \begin{pmatrix} -x_2 & x_1 & x_3 \end{pmatrix}^T, & |x_3| \neq 1 \\ \begin{pmatrix} \sqrt{2} & \sqrt{2} & 0 \end{pmatrix}^T, & |x_3| = 1 \end{cases} \quad (\text{A.3})$$

gewählt.

Aber auch die vorgestellten Tätigkeiten hatten weitere mathematisch interessante Aspekte. Außerdem gab es durchaus auch rein programmiertechnisch interessante Teile des Praktikums. Nennenswert sind hier z. B. die Entwicklung eines einfachen nebenläufigen Systems zur Nutzereinstellungsverwaltung (mit Persistenz), welches auch für parametrisierte Simulationen genutzt werden kann, ein Cachingsystem welches automatisiert berechnete Größen zwischenspeichert bzw. falls nötig eine Neuberechnung dieser durchführt, oder auch die Generation großer Teile des UI-Codes mittels Metaprogrammierung.

Anhang B

Ausblick und zukünftige Erweiterungen

Für die Zukunft gibt es noch einige mögliche Verbesserungen, welche im GitLab Repository des Projektes vermerkt sind. Einige Beispiele sind:

- Die Visualisierung erfolgt aktuell komplett zweidimensional mittels Karten (statisch über Cartopy / Matplotlib bzw. interaktiv mittels Geopandas / Folium). Dies kommt mit einigen Problemen einher, ist jedoch aufgrund der speziellen Anforderungen des Orbitssimulators (Visualisierung von Datensätzen, welche sich über die ganze Erde sowie den erdnahen Weltraum erstrecken, sowie Einbindung von Kartendaten bzw. Texturierung - bevorzugt in Nativer Anwendung) die einzige Lösung, welche sich direkt mit frei zugänglichen Bibliotheken umsetzen lässt. Sollte eine neue Visualisierungsbibliothek auftreten oder eine bestehende um die nötigen Features erweitert werden (sehr vielversprechend ist hier z. B. mayavi), wäre es sicherlich interessant diese einzubauen. Es wäre auch denkbar, eine einfache Visualisierung mittels z. B. OpenGL selbst zu implementieren. Siehe hierzu auch [Nis+19].
- Die in C++ geschriebene Komponente verursacht bei Rechnungen, welche gleichzeitig lange Zeiträume abdecken und mit hoher Präzision arbeiten, Speicherzugriffsfehler. Diese sollten behoben werden; siehe hierzu z. B. den memory error detector von valgrind. Da einer der Autoren von [MG00] (Herr Dr. rer. nat. Oliver Montenbruck) in einer anderen Abteilung des DLR angestellt ist, ist es eventuell auch möglich eine neuere Version des Codes zu erhalten in welcher dieser Fehler bereits behoben werden konnte.
- Zur Validierung der Messungen eines Satelliten bzw. darauf aufbauender Retrievals¹ werden mitunter Vergleiche mit Messstationen bzw. -netzwerken auf der Erdoberfläche durchgeführt. Hierzu ist es notwendig, die Schnittmenge aus dem Messgebiet der Bodenstationen und der Satellitenfootprints zu ermitteln - was mittels des in diesem Bericht beschriebenen Algorithmus bzw. darauf aufbauender weiter abstrahierter Funktionen relativ unkompliziert implementiert werden kann.
- Aktuell werden Berechnungen und UI im gleichen Thread gemanaged. Für die direkt in ATP-OSSE implementierten Berechnungen ist das ein eher kleines Problem, da diese größtenteils "sofort" abgeschlossen sind - jedoch sind einige externe Funktionen wie z. B. die Strahlungstransportrechnung sehr aufwendig / langsam, was darin resultiert, dass das Benutzerinterface eine gewisse Zeit einfriert, sobald eine längere Berechnung angestoßen wurde. Eine bessere Herangehensweise wäre daher die Parallelisierung / Verteilung auf mehrere Prozesse (z. B. mittels Python's multiprocessing Modul)² bzw. der Umstieg auf ein asynchrones Modell wie

¹Unter einem *Retrieval* versteht man die Berechnung von Atmosphärenparametern (also z. B. die Verteilung einzelner Spurengase in der Atmosphäre) aus Messwerten wie z. B. der Radianz. Siehe hierzu auch die Informationsseite über Modellierung und Retrieval des Earth Observation Centers am DLR.

²Eine detaillierte Erläuterung wie sich hiermit PyQt-basierende Nutzerinterfaces implementieren lassen ist z. B. in [KV19] zu finden. Siehe außerdem [GO20].

es beispielsweise mit Python's `asyncio` (siehe hierzu auch [Hat20]) implementiert werden kann. Da das Management der hierbei auftretenden temporalen Abhangigkeiten zwischen den einzelnen Berechnungsschritten (welche zusatzlich noch optional deaktiviert werden konnen) potenziell nichttrivial wird, ist die formale Spezifikation bzw. das Testen des Modells mit einem Tool wie z. B. TLA^+ oder Alloy dabei sicherlich angebracht. Siehe hierzu auch [Sch13, Kapitel 7].

Anhang C

Sonstiges

C.1 Lange Listings

```
.
├── CoordinateTransformations.py
├── Documentation
│   ├── atposse.bib
│   ├── DeveloperManual.tex
│   └── DPM
│       └── DPM-atp-osse.org
├── Images
│   ├── AppCoordinates.jpg
│   ├── Bug1.png
│   ...
├── Main.tex
└── UserGuide
    └── OrbIntUserGuide.org
└── UserManual.tex
GUI
└── OrbitIntGUI.ui
OrbitCore
├── colors.h
├── OrbitCore-3.0.cpp
├── README
├── SAT_Const.h
├── SAT_DE.cpp
├── SAT_DE.h
├── SAT_Filter.cpp
├── SAT_Filter.h
├── SAT_Force.cpp
├── SAT_Force.h
├── SAT_Kepler.cpp
├── SAT_Kepler.h
├── SAT_RefSys.cpp
├── SAT_RefSys.h
├── SAT_Refsys.oclear
├── SAT_Time.cpp
├── SAT_Time.h
├── SAT_VecMat.cpp
└── SAT_VecMat.h
└── OrbitInt.py
└── SelectiveExtractor.py
```

```
└── ShapelyPolygons.py
```

```
  6 directories, 46 files
```

Listing 2: Ordnerstruktur zu Beginn des Projektes (Output des Bash-Kommandos `tree` in der Projektwurzel; einige Elemente des `Images` Ordners wurden durch `...` ersetzt)

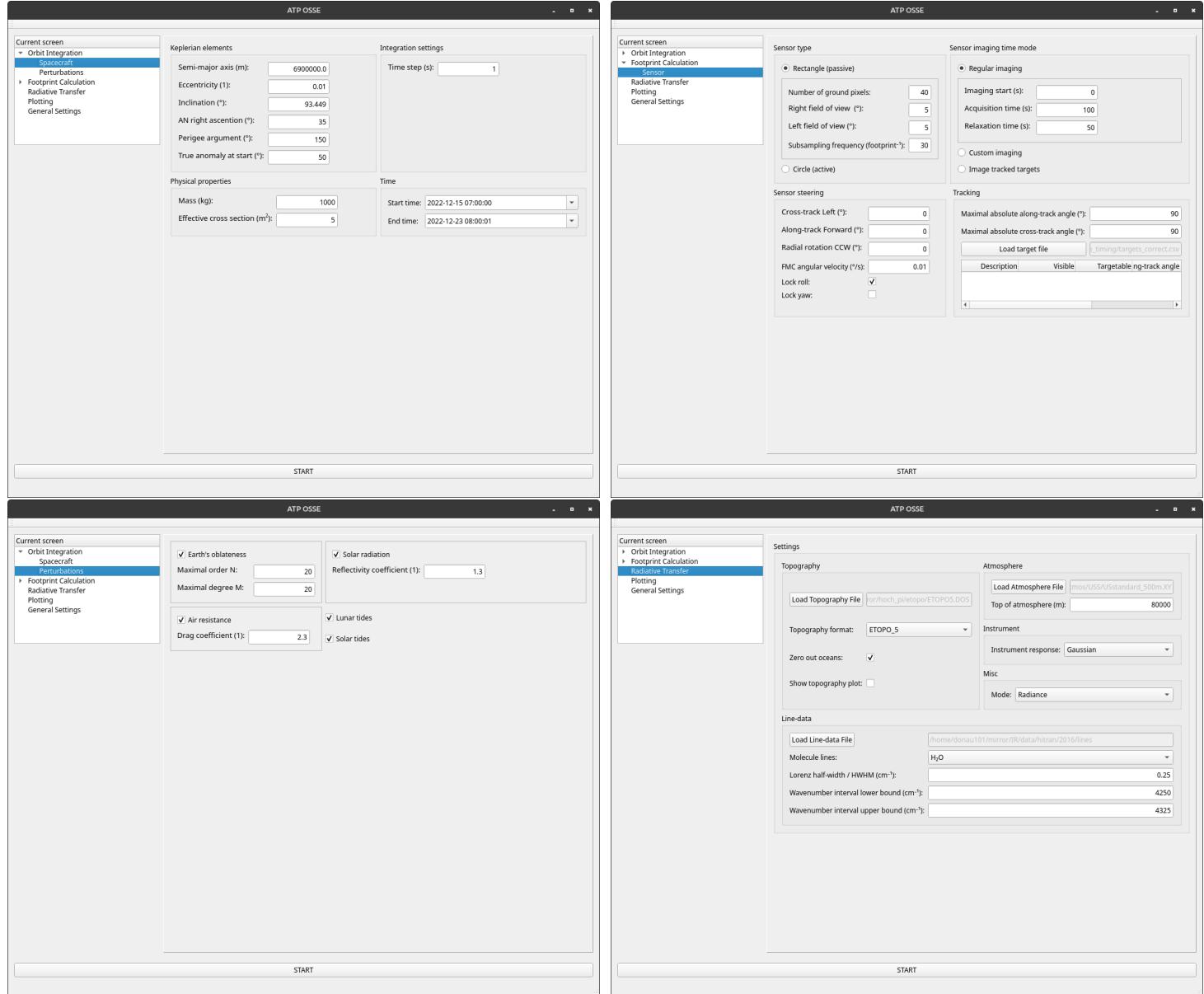
```
.
├── Documentation
│   ├── Algorithms
│   │   ├── error_great_circle_approx
│   │   │   ├── approx_geodesics.html
│   │   │   ├── approx_geodesics.ipynb
│   │   │   ├── Deviation1000km.pdf
│   │   │   ├── Deviation100km.pdf
│   │   │   ├── Deviation2600km.pdf
│   │   │   ├── Deviation3000km.pdf
│   │   │   ├── error_great_circle_approx_copy.pdf
│   │   │   ├── error_great_circle_approx_old.pdf
│   │   │   ├── error_great_circle_approx.pdf
│   │   │   └── error_great_circle_approx.tex
│   │   ├── geodetic.py
│   │   ├── MIT License.md
│   │   ├── odes.py
│   │   ├── prelude.tex
│   │   └── visualize.py
│   ├── point_inside_geoconvex_polygon
│   │   ├── curve.ipynb
│   │   ├── img
│   │   │   ├── p1.pdf
│   │   │   └── p1.svg
│   │   ├── p1.svg
│   │   ├── Polygon_Innen_Außen.svg
│   │   ├── poly.pdf
│   │   ├── poly.png
│   │   ├── poly.tex
│   │   ├── prelude.tex
│   │   └── references.bib
│   └── Tracking
│       ├── img
│       │   ├── Tracking.pdf
│       │   └── Tracking.svg
│       ├── prelude.tex
│       ├── tracking.pdf
│       └── tracking.tex
└── atposse.bib
└── DataDescription
    ├── datadescription.pdf
    ├── datadescription.tex
    └── prelude.tex
└── DeveloperManual.tex
└── DPM
    └── DPM-atp-osse.org
└── Images
```

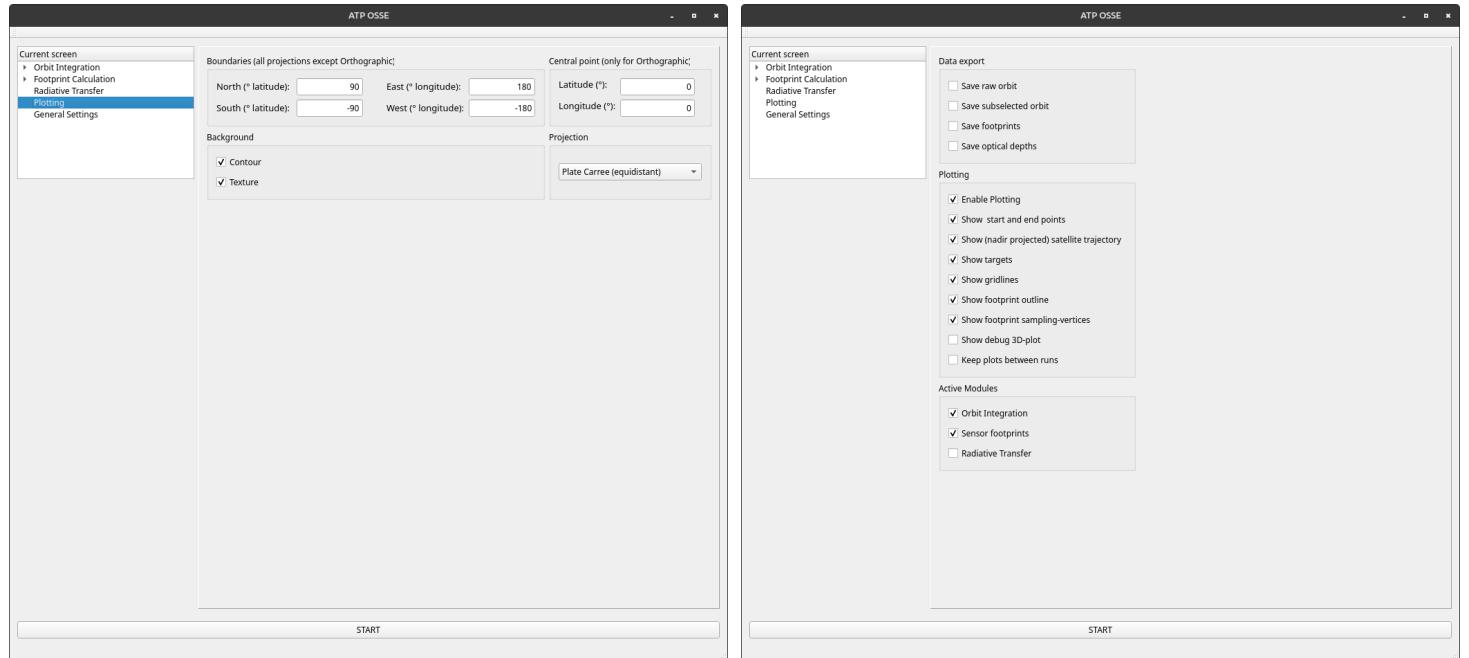
```
    └── AppCoordinates.jpg
      ...
    └── Main.tex
    └── UserGuide
        └── OrbIntUserGuide.org
    └── UserManual.tex
    └── Vektorfeldtransport
        ├── prelude.tex
        └── vektorfelder.pdf
        └── vektorfelder.tex
    └── examples
        ├── api_example.html
        ├── api_example.ipynb
        ├── circ_footprints.csv
        └── rect_footprints.csv
    └── GUI
        └── OrbitIntGUI.ui
    └── Makefile
    └── pyproject.toml
    └── radiation.pickled_csv
    └── README.rst
    └── src
        ├── atp_osse
        │   ├── accessors.py
        │   ├── api.py
        │   ├── base.py
        │   ├── constants.py
        │   ├── footprints.py
        │   ├── geodata.py
        │   ├── __init__.py
        │   ├── OrbitCore-3.0
        │   ├── radiation.py
        │   ├── reference_ellipsoids.py
        │   ├── selective_extractor.py
        │   ├── target.py
        │   ├── topography.py
        │   ├── util.py
        │   └── visualize.py
        ├── cache.py
        ├── custom_widgets.py
        ├── gui.py
        └── IO
            └── main.py
        └── monkey.py
        └── orbit_core
            ├── __init__.py
            └── libOrbitCore.so
        └── OrbitCore
            ├── colors.h
            ├── Makefile
            ├── OrbitCore-3.0.cpp
            ├── README
            ├── SAT_Const.h
            ├── SAT_DE.cpp
            ├── SAT_DE.h
            └── SAT_Filter.cpp
```

```
    └── SAT_Filter.h
    ├── SAT_Force.cpp
    ├── SAT_Force.h
    ├── SAT_Kepler.cpp
    ├── SAT_Kepler.h
    ├── SAT_RefSys.cpp
    ├── SAT_RefSys.h
    ├── SAT_Time.cpp
    ├── SAT_Time.h
    ├── SAT_VecMat.cpp
    └── SAT_VecMat.h
    ├── settings.py
    ├── settings_schema.py
    └── util.py
└── tests
    ├── conftest.py
    ├── doc.py
    ├── doc_test.ipynb
    ├── __init__.py
    ├── strategies.py
    ├── test_accessors.py
    ├── test_api.py
    ├── test_base.py
    ├── test_cache.py
    ├── test_geodata.py
    ├── test_reference_ellipsoids.py
    └── test_util.py
22 directories, 129 files
```

Listing 3: Ordnerstruktur gegen Ende des Projektes (Output des Bash-Kommandos `tree` in der Projektwurzel; einige Elemente des `Images` Ordners wurden durch `...` ersetzt)

C.2 Einige Bilder der Benutzeroberfläche





C.3 Erzeugte Footprintvisualisierungen

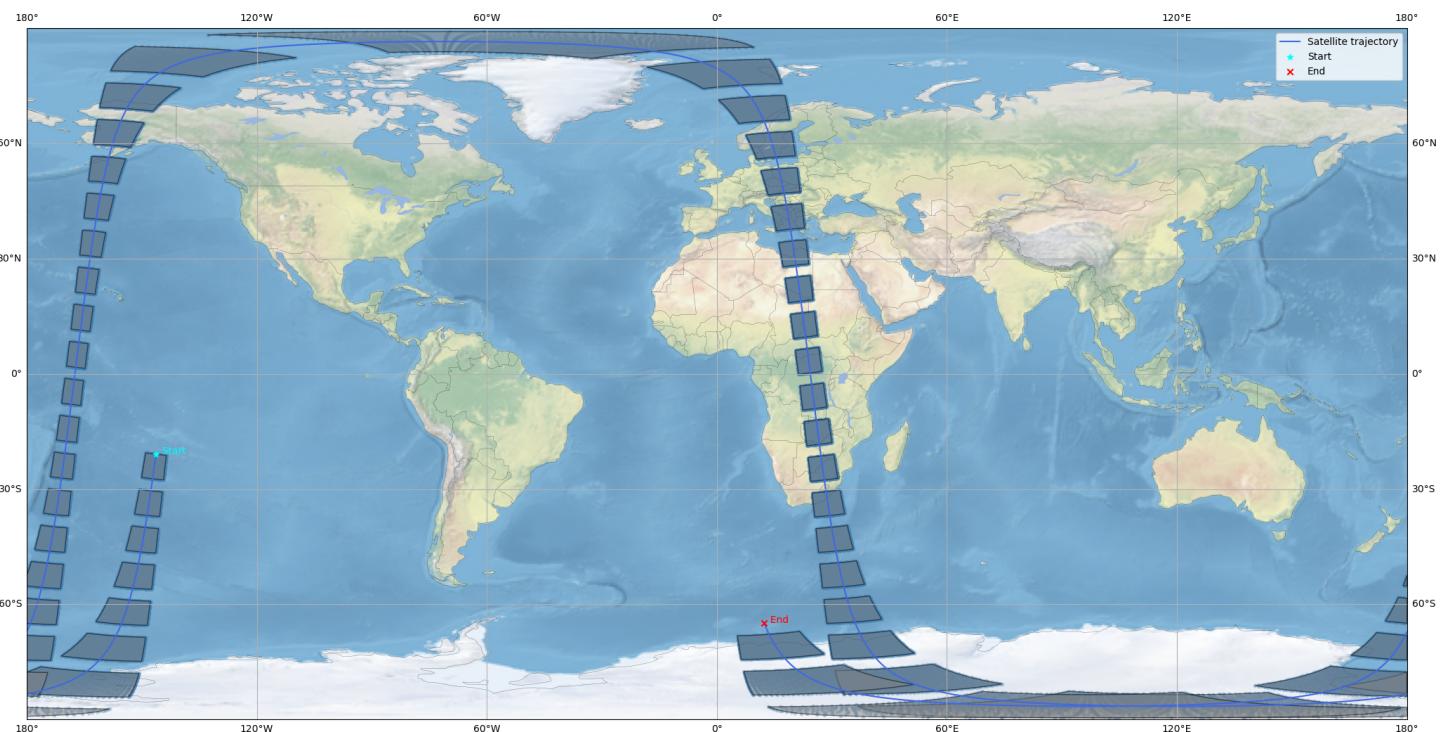


Abbildung C.1: Visualisierung einiger "rechteckiger" Footprints. Die Streckung an den Polen resultiert hierbei aus der Kartendarstellung.

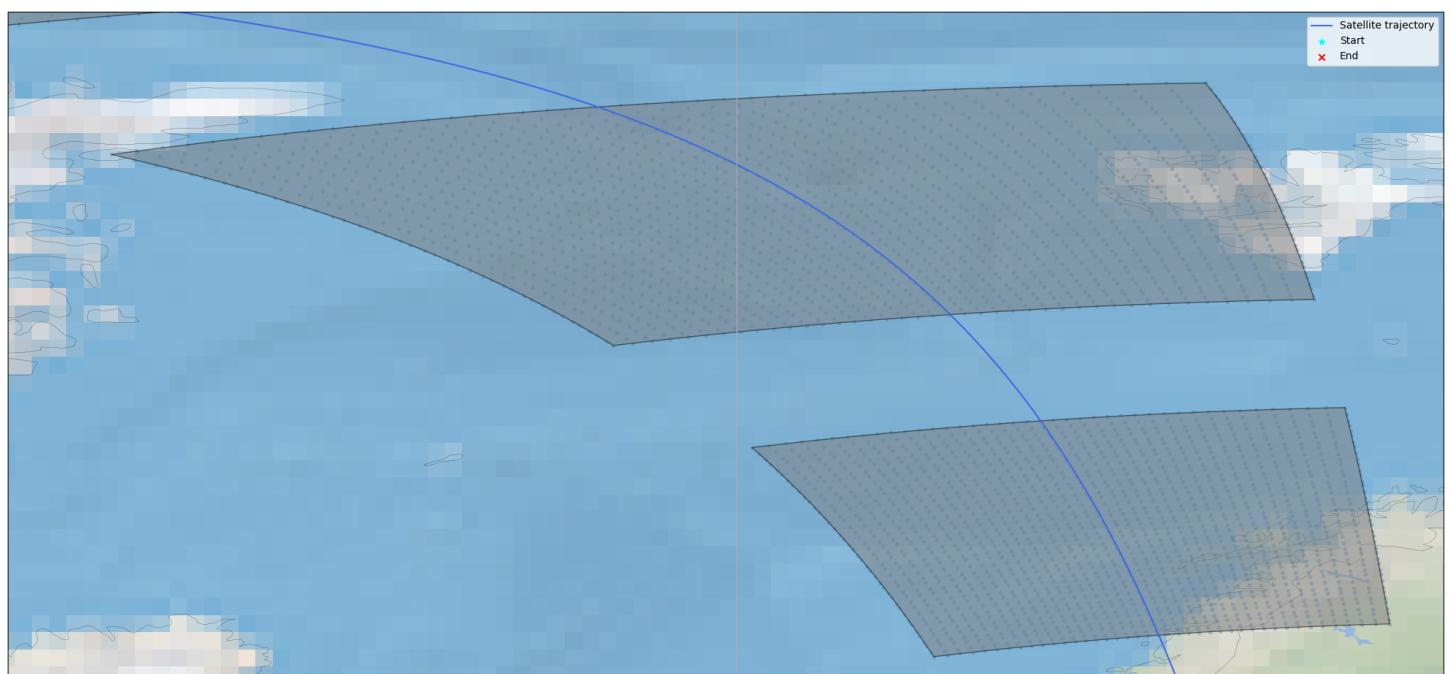


Abbildung C.2: Detailaufnahme einiger Footprints in Polnähe. Die Punkte stellen hierbei die Ecken der einzelnen Sensorpixel dar aus denen ein Footprint besteht.

Anhang D

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich den vorliegenden Bericht selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Schweinfurt, 23. Juli 2022



Stefan Volz

Literatur

- [Abe+11] I. Aben u. a. *SCIAMACHY. Exploring the Changing Earth's Atmosphere*. Hrsg. von Manfred Gottwald und Heinrich Bovensmann. 1. Aufl. Springer, 2011. ISBN: 978-90-481-9895-5. DOI: [10.1007/978-90-481-9896-2](https://doi.org/10.1007/978-90-481-9896-2).
- [BMW71] Roger R. Bate, Donald D. Mueller und Jerry E. White. *Fundamentals of Astrodynamics*. Vielen Dank an Herrn Prof. Dr. Holger Walter für die Leihgabe dieses Buches. Dover, 1971. ISBN: 978-0-486-60061-1.
- [Boh66] Alfred Bohrmann. *Bahnen künstlicher Satelliten*. 2. Aufl. Vielen Dank an Herrn Prof. Dr. Holger Walter für die Leihgabe dieses Buches. Bibliographisches Institut, 1966.
- [Cra+13] Keenan Crane u. a. „Digital Geometry Processing with Discrete Exterior Calculus“. In: *ACM SIGGRAPH 2013 courses*. SIGGRAPH '13. Anaheim, California: ACM, 2013. URL: <https://www.cs.cmu.edu/~kmcrane/Projects/DDG/>.
- [EK17] Johannes Ernesti und Peter Kaiser. *Python 3. Das umfassende Handbuch*. Hrsg. von Anne Scheibe. 5. Aufl. Rheinwerk Computing, 2017. ISBN: 978-3-8362-5864-7.
- [Env22] National Centers for Environmental Information. *Registration of Structured Square-Cell Grids*. www.ngdc.noaa.gov. 2022. URL: <http://www.ngdc.noaa.gov/mgg/global/gridregistration.html>.
- [EOC22] EOC. *DLR-Mission CO2Image – das Klima im Fokus*. dlr.de. 21. Juni 2022. URL: https://www.dlr.de/eoc/desktopdefault.aspx/tabcid-18220/29005_read-77522 (besucht am 06.07.2022).
- [For18] Jon Pierre Fortney. *A Visual Introduction to Differential Forms and Calculus on Manifolds*. 1. Aufl. Birkhäuser, 2018. ISBN: 978-3-319-96991-6. DOI: [10.1007/978-3-319-96992-3](https://doi.org/10.1007/978-3-319-96992-3).
- [Fou22a] Wikimedia Foundation. *Point in polygon*. Wikipedia. Abgerufen am 11.06.2022. 2022. URL: https://en.wikipedia.org/wiki/Point_in_polygon.
- [Fou22b] Wikimedia Foundation. *World Geodetic System 1984*. Wikipedia. Abgerufen am 11.06.2022. 2022. URL: https://de.wikipedia.org/wiki/World_Geodetic_System_1984.
- [Gad10] Kenneth Gade. „A Non-singular Horizontal Position Representation“. In: *The Journal of Navigation* 63 (3 2010), S. 395–417. DOI: [10.1017/S0373463309990415](https://doi.org/10.1017/S0373463309990415). URL: https://www.navlab.net/Publications/A_Nonsingular_Horizontal_Position_Representation.pdf.
- [GO20] Micha Gorelick und Ian Ozsvárd. *High Performance Python. Practical Performant Programming for Humans*. Hrsg. von Amanda Quinn u. a. 2. Aufl. O'Reilly, 2020. ISBN: 978-1-492-05502-0.
- [GQ20] Jean Gallier und Jocelyn Quaintance. *Differential Geometry and Lie Groups. A Computational Perspective*. Hrsg. von Herbert Edelsbrunner, Leif Kobbelt und Konrad Polthier. Springer, 2020. ISBN: 978-3-030-46039-6. DOI: [10.1007/978-3-030-46040-2](https://doi.org/10.1007/978-3-030-46040-2).
- [Har+20] Charles R. Harris u. a. „Array programming with NumPy“. In: *Nature* 585.7825 (Sep. 2020), S. 357–362. DOI: [10.1038/s41586-020-2649-2](https://doi.org/10.1038/s41586-020-2649-2). URL: <https://doi.org/10.1038/s41586-020-2649-2>.
- [Hat20] Caleb Hattingh. *Using Asyncio in Python. Understanding Python's Asynchronous Programming Features*. Hrsg. von Jessica Haberman u. a. 1. Aufl. O'Reilly, 2020. ISBN: 978-1-492-07533-2.

- [Hir03] Anil N. Hirani. „Discrete Exterior Calculus“. Pasadena, California: California Institute of Technology, Mai 2003. URL: <https://www.cs.jhu.edu/~misha/Fall09/Hirani03.pdf>.
- [Hir76] Morris W. Hirsch. *Differential Topology*. Hrsg. von F.W. Gehring, P.R. Halmos und C. C. Moore. Springer, 1976.
- [Hun07] J. D. Hunter. „Matplotlib: A 2D graphics environment“. In: *Computing in Science & Engineering* 9.3 (2007), S. 90–95. DOI: [10.1109/MCSE.2007.55](https://doi.org/10.1109/MCSE.2007.55).
- [JKS13] Alec Jacobson, Ladislav Kavan und Olga Sorkine. „Robust Inside-Outside Segmentation using Generalized Winding Numbers“. In: *ACM Trans. Graph.* 32.4 (2013). DOI: [10.1145/2461912.2461916](https://doi.org/10.1145/2461912.2461916). URL: <https://igl.ethz.ch/projects/winding-number/robust-inside-outside-segmentation-using-generalized-winding-numbers-siggraph-2013-jacobson-et-al.pdf>.
- [Jor+20] Kelsey Jordahl u.a. *geopandas/geopandas: v0.8.1*. Version v0.8.1. Juli 2020. DOI: [10.5281/zenodo.3946761](https://doi.org/10.5281/zenodo.3946761). URL: <https://doi.org/10.5281/zenodo.3946761>.
- [KNM] Climate Observations division of KNMI. *TROPOMI. TROPOspheric Monitoring Instrument*. tropomi.eu. Abgerufen am 18.06.2022 über die Wayback Machine des Internet Archive. Instrumentenhandbuch zu S5P / TROPOMI. URL: <https://web.archive.org/web/20110524013421/http://www.tropomi.eu/TROPOMI/Instrument.html>.
- [KV19] Yannis Köhler und Stefan Volz. „TUI - Test- und Integrationssystem für SCADA-Software“. In: *Technikerarbeit bei Franz Oberthür Schule Würzburg* (2019). URL: <https://github.com/SV-97/TUInventory/blob/master/TUInventory/Documentation/Documentation.pdf>.
- [LM14] Jörg Liesen und Volker Mehrmann. *Lineare Algebra. Ein Lehrbuch über die Theorie mit Blick auf die Praxis*. 2. Aufl. Springer Spektrum, 2014. ISBN: 978-3-658-06609-3. DOI: [10.1007/978-3-658-06610-9](https://doi.org/10.1007/978-3-658-06610-9).
- [Mec04] Robert Mecklenburg. *Managing Projects with GNU Make*. Hrsg. von Andy Oram und Matt Hutchinson. 3. Aufl. O'Reilly, 2004. ISBN: 978-0-596-00610-5.
- [Met15] Met Office. *Cartopy: a cartographic python library with a Matplotlib interface*. Exeter, Devon, 2010 - 2015. URL: <https://scitools.org.uk/cartopy>.
- [MG00] Oliver Montebruck und Eberhard Gill. *Satellite Orbits : Models, Methods and Applications*. Springer, 2000. ISBN: 978-3540672807. DOI: [10.1007/978-3-642-58351-3](https://doi.org/10.1007/978-3-642-58351-3).
- [Nis+] Mash Nishihama u.a. *MODIS Level 1A Earth Location: Algorithm Theoretical Basis Document Version 3.0*. URL: https://modis.gsfc.nasa.gov/data/atbd/atbd_mod28_v3.pdf (besucht am 06.07.2022).
- [Nis+19] Alfred Nischwitz u.a. *Computergrafik. Band 1 des Standardwerks Computergrafik und Bildverarbeitung*. 4. Aufl. Springer, 2019. ISBN: 978-3-658-25383-7. DOI: [10.1007/978-3-658-25384-4](https://doi.org/10.1007/978-3-658-25384-4).
- [PJH16] Matt Pharr, Wenzel Jakob und Greg Humphreys. *Physically Based Rendering. From Theory to Implementation*. 3. Aufl. Morgan Kaufmann, 2016. ISBN: 978-0128006450. URL: <https://www.pbr-book.org/>.
- [Ram22] Luciano Ramalho. *Fluent Python. Clear, Concise, and Effective Programming*. Hrsg. von Amanda Quinn u.a. 2. Aufl. O'Reilly, 2022. ISBN: 978-1-492-05635-5.
- [Rom08] Steven Roman. *Advanced Linear Algebra*. 3. Aufl. Springer, 2008. ISBN: 978-0-387-72828-5. DOI: [10.1007/978-0-387-72831-5](https://doi.org/10.1007/978-0-387-72831-5).
- [Sav16] Peter Saveliev. *Topology illustrated*. 2016. ISBN: 978-1-0878-0346-3.
- [Sch+14] F. Schreier u.a. „GARLIC – A General Purpose Atmospheric Radiative Transfer Line-by-Line Infrared-Microwave Code: Implementation and Evaluation“. In: *J. Quant. Spectrosc. & Radiat. Transfer* 137 (2014), S. 29–50. DOI: [10.1016/j.jqsrt.2013.11.018](https://doi.org/10.1016/j.jqsrt.2013.11.018).
- [Sch+19] Franz Schreier u.a. „Py4CAAtS - PYthon for Computational ATmospheric Spectroscopy“. In: *Atmosphere* 10.5 (2019), S. 262. DOI: [10.3390/atmos10050262](https://doi.org/10.3390/atmos10050262). URL: <https://www.mdpi.com/2073-4433/10/5/26>.

- [Sch13] Michael Schenke. *Logikkalküle in der Informatik. Wie wird Logik vom Rechner genutzt?* 1. Aufl. Springer, 2013. ISBN: 978-3-8348-1887-4. DOI: [10.1007/978-3-8348-2295-6](https://doi.org/10.1007/978-3-8348-2295-6).
- [Sti08] John Stillwell. *Naive Lie Theory*. Hrsg. von S. Axler und K.A. Ribet. 1. Aufl. Springer, 2008. ISBN: 978-0-387-78214-0. DOI: [10.1007/978-0-387-78214-0](https://doi.org/10.1007/978-0-387-78214-0).
- [Tho92] Carsten Thomassen. „The Jordan-Schonflies Theorem and the Classification of Surfaces“. In: *The American Mathematical Monthly* 99.2 (Feb. 1992). DOI: [10.2307/2324180](https://doi.org/10.2307/2324180). URL: <http://www.jstor.org/stable/2324180> (besucht am 20.06.2022).
- [Tu10] Loring W. Tu. *An Introduction to Manifolds*. Hrsg. von S. Axler und K.A. Ribet. 2. Aufl. Springer, 2010. ISBN: 978-1-4419-7399-3. DOI: [10.1007/978-1-4419-7400-6](https://doi.org/10.1007/978-1-4419-7400-6).
- [Tu17] Loring W. Tu. *Differential Geometry. Connections, Curvature, and Characteristic Classes*. Hrsg. von S. Axler und K.A. Ribet. 1. Aufl. Springer, 2017. ISBN: 978-3-319-55082-4. DOI: [10.1007/978-3-319-55084-8](https://doi.org/10.1007/978-3-319-55084-8).
- [Win10] Sergei Winitzki. *Linear Algebra via Exterior Products*. Lulu, 2010. ISBN: 978-1-4092-9496-2. URL: <https://sites.google.com/site/winitzki/linalg>.
- [Zhu94] J. Zhu. „Conversion of Earth-centered Earth-fixed coordinates to geodetic coordinates.“ In: *IEEE Transactions on Aerospace and Electronic Systems* 30 (3 1994), S. 957–961. DOI: [10.1109/7.303772](https://doi.org/10.1109/7.303772). URL: <https://ieeexplore.ieee.org/document/303772>.
- [ZTB07] Wilford Zdunkowski, Thomas Trautmann und Andreas Bott. *Radiation in the Atmosphere. A Course in Theoretical Meteorology*. 1. Aufl. Vielen Dank an Herrn Prof. Dr. Thomas Trautmann sowie Herrn Prof. Dr. Holger Walter für die Leihgabe dieses Buches. Cambridge University Press, 2007. ISBN: 978-0521871075. DOI: [10.1017/CBO9780511535796](https://doi.org/10.1017/CBO9780511535796). URL: <https://www.cambridge.org/core/books/radiation-in-the-atmosphere/699E7D2AE832A34874D9E13AD5F6D4D6>.