Forschungspraktikum

Prototypische Entwicklung zum Export von Straßengeometrien in das IFC-Datenformat

Praktikant:	Stefan Huber
Matrikelnummer:	
Lehrstuhl:	Computergestützte Modellierung und Simulation
Betreuer:	Sebastian Esser
Abgabe:	19.06.2020

_____

Sebastian Esser, Betreuer

Inhaltsverzeichnis

Im Folgenden werde ich die Aufgaben, Ziele und Erfahrungen meines Forschungspraktikums wiedergeben.

Ziel des Praktikums ist es, einen Exporter für eine Straße von einem spezifischen Datenformat einer Applikation zu einem unabhängigen Datenformat (IFC) zu programmieren.

Industry Foundation Classes, kurz IFC, ist ein offenes und softwareunabhängiges Datenformat. IFC konzentriert sich hauptsächlich auf den Bausektor und hat viele Klassen, um Elemente geometrisch und semantisch zu beschreiben. Ziel ist der Austausch von komplexen BIM-Modellen zwischen verschiedenen Softwarelösungen. Deswegen ist es notwendig, herstellerspezifische Formate zu IFC zu konvertieren. In diesem Projekt wird die Planungssoftware Civil 3D verwendet, mithilfe derer Straßenverläufe konstruiert und visualisiert werden können. In der, erst seit kurzen veröffentlichten Version IFC4.3_RC1, wurde neue Klassen zur Beschreibung von Straßen zu dem Standard hinzugefügt. Deshalb müssen nun Programme implementiert werden, um diese Datenmodele zu erstellen.

1. **Meilensteine:**

    - Testprojekt mit beispielhafter Geländeoberfläche und Straßengeometrien erstellen
    - Prozess zum Auslesen der Daten mithilfe der visuellen Programmiersprache Dynamo finden
    - Selektion der Daten und in der Console Ausgaben/CSV exportieren
    - IFC Datenfomat Verständnis entwickeln
    - Daten von Civil 3D in IFC Konvertieren

Aufgrund der Zeitbeschränkung von 80h wird der letzte Meilenstein außerhalb des Forschungspraktikums bearbeitet.

2. **Auswahl der Software:**

Als Software wird aufgrund verfügbarer Studentenlizenz und einer API (Programmierschnittstelle) Civil 3D von Autodesk benutzt. Zusätzlich können im Rahmen einer Forschungskooperation zwischen der TUM und dem Industriepartner Autodesk tiefgehendere Fragen mit Experten aus Forschung und Entwicklung diskutiert werden. Andere Alternativen wie beispielsweise ProVI CAD oder CARD-1 haben keine offene API und können deshalb für die prototypische Entwicklung nur bedingt eingesetzt werden. Da bei Civil 3D eine Vielzahl von Funktionen

angeboten wird, ist der Einstieg schwierig. In diesem Projekt, eher nachrangigen Aufgabengebiete, fallen unter anderem Entwurf, Planung und Vermessungen von Kanalisationen oder Schienennetzwerken. Die Einarbeitung in das Programm ist zeitintensiv, aber durch die gute Dokumentationen und Videos im Internet möglich. Diese Tutorials stammen sowohl von Autodesk selbst als auch von Drittanbietern. Civil 3D bietet eine visuelle Entwicklungsumgebung (IDE) „Dynamo“ an, welche nachträglich installiert werden muss. Damit hat man eine graphische Oberfläche zum Programmieren.

3. Dynamo:

Dynamo besitzt den Vorteil, bei einfachen Programmen keine Programmiersprache oder Syntax lernen zu müssen (vgl. Preidel). Außerdem beinhaltet es einzelne Blöcke/Codeschnipsel (Unterfunktionen) in einer Bibliothek, welche per Klick verbunden werden können. Diese Codeschnipsel enthalten Basisoperatoren, wie z.B. der Bildung von Summen, Differenzen, Produkten und Quotienten aber auch komplexere Funktionen, wie ein Sortieralgorithmus. Hilfreich für den Anfang ist ebenfalls, dass Blöcke explizite Objekte, wie Corridors, Baselines oder auch Subassemblies verlangen. Die einzelnen Knoten sind selbsterklärend und dokumentiert. Des Weiteren gibt es die Möglichkeit, selbst Blöcke zu erstellen oder einzelne Codezeilen in Python zu schreiben. Zudem bestehen es die Option zusätzliche Bibliotheken zu importieren. Da das Programm aufgrund der vielen Blöcke schnell unübersichtlich wirkt, muss die Struktur in regelmäßigen Abständen aktualisiert werden. Auch ähnlich zu einer IDE lassen sich Breakpoints definieren und aktuelle Werte auslesen um zu debuggen.

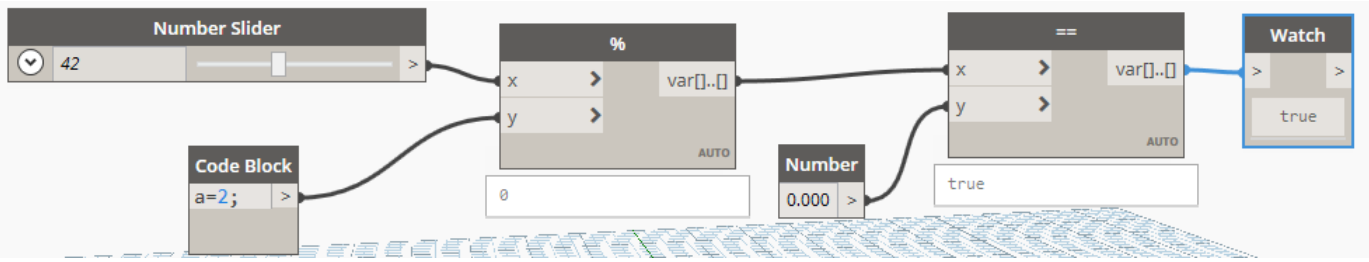

Abb.1 Beispielprogramm zur Bestimmung gerader und ungerader Zahlen.

Aufgetretene Probleme und Lösungsmöglichkeiten:

- Nach dem ersten Starten von Dynamo ist Civil 3D eingefroren. Die Lösung dafür ist Civil 3D und die Dynamo Exe auf der GPU auszuführen.

- Wenn nach dem Öffnen eines Skriptes Skript- und Codeblöcke nicht mehr angezeigt werden, so soll das Skript nicht gespeichert, aber geschlossen werden. Anschließend muss man das Programm neu starten. Danach sollten die Blöcke wieder sichtbar sein.

4. Testprojekt mit beispielhafter Geländeoberfläche und Straßengeometrien:

Die Teststraße ist die Beispieldatei, mit der der Exporter getestet wird. Damit verschiedene Szenarien abdecket sind, muss die Teststraße unterschiedlichen Eigenschaften besitzen. Diese sind verschiedene Straßenquerschnitte (Variation in Fahrbahnlänge und Art der Konstruktion), sowie Kurven mit unterschiedlichen Radien. Damit sind auch unterschiedliche Querschnittsneigungen abgedeckt. Nicht berücksichtigt werden Verschiebungen vom Alignments in horizontaler und vertikaler Richtung.

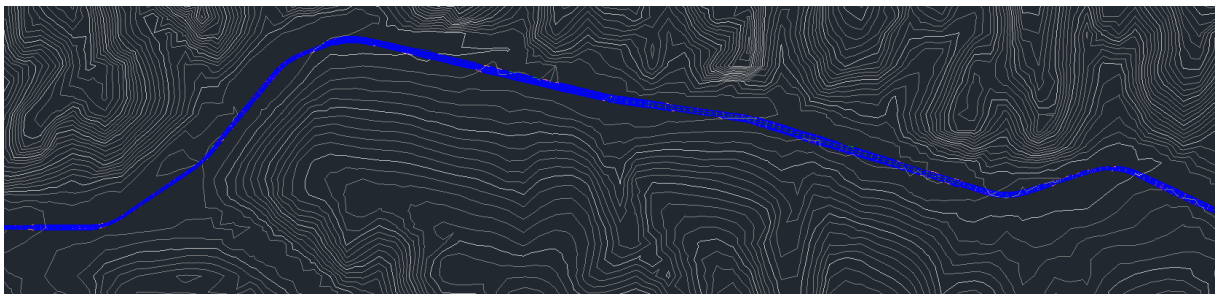

Abb. 2: Ausschnitt der Teststraße im Objekt-Viewer von Civil 3D

5. Prozess zum Auslesen der Daten in Dynamo:

Das Ziel ist es folgende Daten auszulesen:

- Station
- Streifenbreite links
- Streifenbreite rechts
- Neigung links
- Neigung rechts

Station beschreibt die Stützstellen an der, die Daten ausgelesen wurden. Die Fahrbahnbreite wird als Streifenbreite bezeichnet. Neigung steht für die Neigung der Fahrbahn von der Mitte nach außen. Die Breite ist nicht in dem Profilkörper, sondern in den einzelnen Querabschnitten

gespeichert. Auf diese kann nur durch die Baselines des Corridors zugegriffen werden. Zu beachten ist auch, dass durch die Modularität des Querschnittbaus einige Sonderfälle für die Export zu beachten sind. Es ist möglich, gleichaussehende Straßen verschieden zu konstruieren (vgl. Abb. 2 und Abb. 3). Problematisch ist, dass es kein Flag oder einen einheitlichen Namen für Fahrbahnen gibt. Damit muss auf den Namen der Eigenschaft gefiltert werden. Da allerdings jedes Fahrbahnbauteil einen anderen Namen besitzt, muss dies hardcoded für jede Fahrbahn hinzugefügt werden. Beispielsweise ist der Name des Fahrbahnstreifens bei einem selbst erstellten Querschnitt „LaneSuperelevationAOR“ mit einem Flag, welches die Seite festlegt. Bei den „Standard“ Querschnitten werden diese aber „LaneSuperelevationAOR - (Right)“ und analog dazu auch „LaneSuperelevationAOR - (Left)" für die Richtung benannt. Daraus entsteht das bereits beschriebene Problem, dass jeder neue Querschnitt in den Code implementiert werden muss. Außerdem ist es notwendig die Breite, ebenso wie die Fahrbahn, mit dem Namen der Eigenschaft zu filtern. Diese ist ebenfalls nicht einheitlich benannt, weshalb für jedes neue Subassembly ein neuer Filter implementiert werden muss. Es muss aber berücksichtigt werden, dass die Reihenfolge nicht verschoben werden darf. Deshalb muss besonders darauf geachtet werden, dass bei unterschiedlichen Querschnitten die Datensätze in identischer Reihenfolge und ein „NULL“ Wert, falls kein passender Wert gefunden wird, zurückgegeben werden. Das Programm wurde so implementiert, dass Erweiterungen mit einer möglichst geringen Änderung an der Struktur und Code einhergehen. Ebenso müssen die Breiten verschiedener Querschnitte einer Seite addiert werden, um die korrekte Breite zu erhalten. Um die Neigung auszulesen, muss das Add-On „Civil 3D Toolkit“ hinzugefügt werden. In diesem Add-On ist ein Block enthalten, welcher die Neigung selektieren kann.

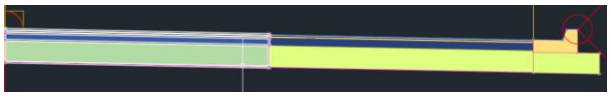

Abb. 3: Straßen durch zwei Einzelteile erstellt

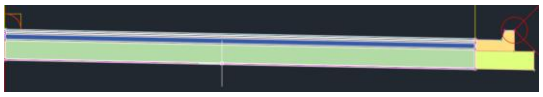

Abb. 4: Straße durch einen einzelnen Streifen erstellt

6. Selektion der Daten und in der Konsoleausgaben/CSV exportieren

Die bereits ausgelesen Daten werden nun in eine Liste geschrieben und auf drei Nachkommastellen gerundet, damit numerische Ungenauigkeiten abgedämpft werden. Die Selektion und Konsolenausgabe werden in einem Skript abgearbeitet. Darin wird iterativ verglichen, ob

sich ein Querschnitt im Vergleich zu dem vorhergehenden verändert hat. Dabei wird natürlich die Änderung der Stützstelle ignoriert. Haben sich relevante Daten verändert, wird der Datensatz ausgeben und in eine neue Liste geschrieben.

```
for i in range(len(data)):
    editor.WriteMessage("\n")
    if (i == 0):  # First crosssection
        for j in range(len(data[i])):#Crosssection output
            editor.WriteMessage("\t" + str(data[i][j]))

        editor.WriteMessage("\t" + str(0.000))  # Here add the Offset
        editor.WriteMessage("\t" + str(0.000))
        continue
    #Check if the crosssection has changed
    for n in range(len(data[i])-2):
        if data[i][n+2]!=data[i-1][n+2]:
            for j in range(len(data[i])):#Crosssection output
                editor.WriteMessage("\t" + str(data[i][j]))

            editor.WriteMessage("\t"+ str(0.0))
            editor.WriteMessage("\t"+ str(0.0))
            ret.append(data[i]);
            break
```

Abb. 5: Implementierung der Selektion und Ausgabe

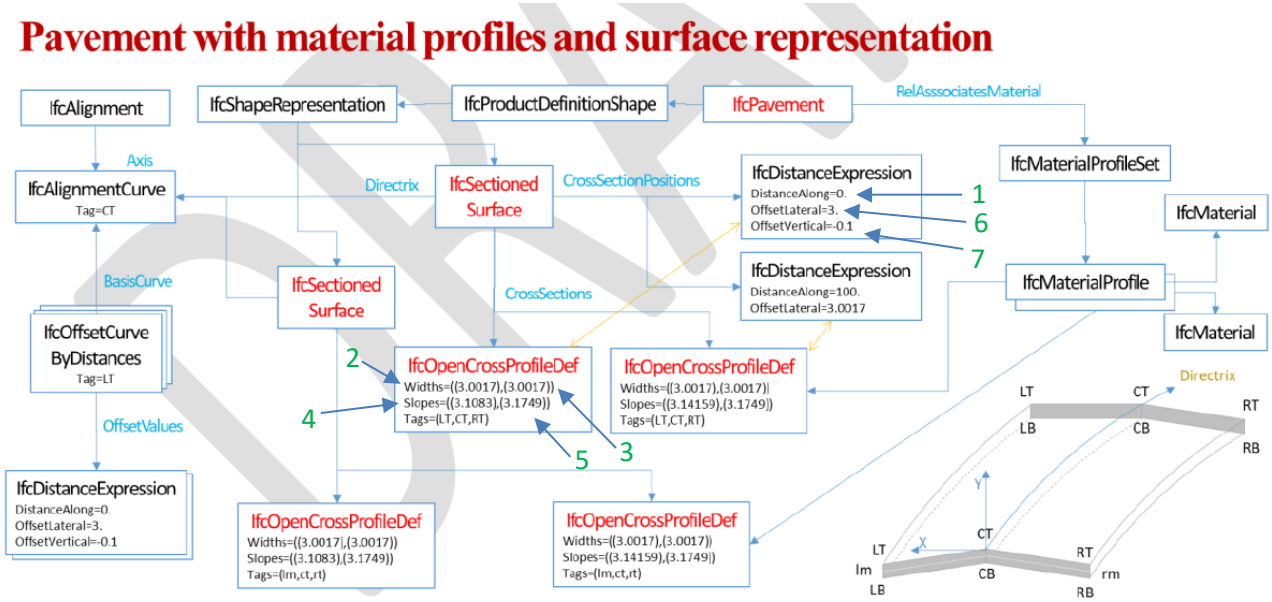

Abb. 6: Verknüpfungen der IFC Komponenten in einem Diagramm.

Quelle: buildingSMART International (2019): IR-IfcRoad Project: ExpertPanel Conceptual Model Report

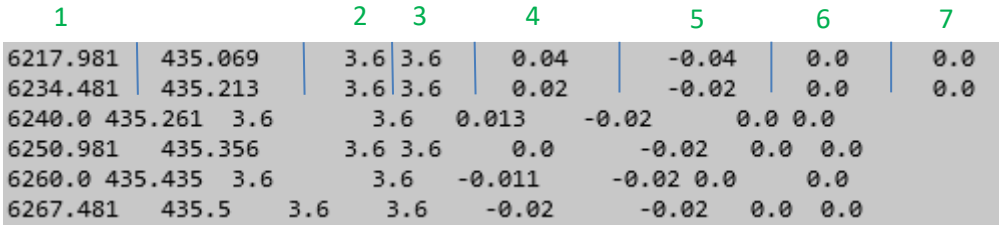

Abb. 7: Ausgabe in der Konsole. Die Ausgabewerte sind mit Nummern bezeichnet. In Abb. 6 sind diese Nummern den jeweiligen Eigenschaften zugeordnet. Die zweite Spalte enthält eine Debug Ausgabe der Höhe.

7. IFC Datenfomat Verständnis entwickeln

Dieses Verständnis stammt hauptsächlich aus Experimenten und reverse engineering von bestehenden Beispieldateien. Diese Dateien wurden unter Zuhilfenahme der Programmiersprache Python und das dort erhältliche IFC-Framework "IfcOpenShell" nachgebaut (vgl. Ifcopenshell). Hilfreich war für diese Untersuchungen die öffentlich verfügbare Schemadokumentation des IFC-Datenmodells, welche wiederum dem Umgang mit Funktionalitäten von IfcOpenShell erleichtert. Der Prozess war sehr zeitaufwendig für die Recherche und Fehlersuche. Die Diskussionen von Probleme mit wissenschaftlichen Mitarbeitern fördert das Verständnis für IFC und dessen Struktur.

8. Fazit

Ich wollte meine bereits erworbenen Kenntnisse der Informatik in einem neuen Gebiet einsetzen und Erfahrungen in der Forschung sammeln, um zu testen, ob dies ein Bereich ist in dem ich später einmal arbeiten möchte. Das für mich nur oberflächlich bekannte Konzept IFC ist sehr spannend, aber auch ein komplexes Thema. Die anfänglich überwältigende Struktur ist für Neulinge schwer nachzuvollziehen. Durch längeres arbeiten damit entwickelt man ein Verständnis für die Logik und kann nachvollziehen, warum die Struktur demensprechend komplex ist. Gerade die Komplexität und Modularität macht IFC zu so einem mächtigen Tool um Strukturen zu beschreiben. Persönlich war es ein gutes Gefühl, ein Problem, an dem man mehrere Stunden gearbeitet hat, selbst zu lösen. Ich bin froh, diese Gelegenheit bekommen zu haben, denn dadurch wurde mir gezeigt, dass ich zukünftig gerne in einem Forschungs- oder einem Entwicklungsprozess arbeiten möchte.

9. Quellen:

Preilel C. (2016) https://www.researchgate.net/profile/Cornelius_Preidel/publication/309804310_Towards_Code_Compliance_Checking_on_the_basis_of_a_Visual_Programming_Language/links/5829b1d708ae509544734fb7.pdf (zuletzt besucht am 18.06.2020)

Ifcopenshell http://ifcopenshell.org/python (zuletzt besucht am 19.06.2020)