Untersuchung verschiedener Open Source Geometric Constraint Solver für die Anwendung von Knowledge based Engineering im Infrastrukturbau

Lehrstuhl
Computergestützte Modellierung und Simulation

Arbeitszeit
01.07.2015 bis 31.10.2015 (5h/Woche)

Abgabe
22.01.2016

Dominik Singer, Betreuer

Benedikt Kohl, Praktikant

Inhalt

Erklärung der grundlegenden Software

In diesem Praktikum wurden anfangs zwei Arten von Software betrachtet:

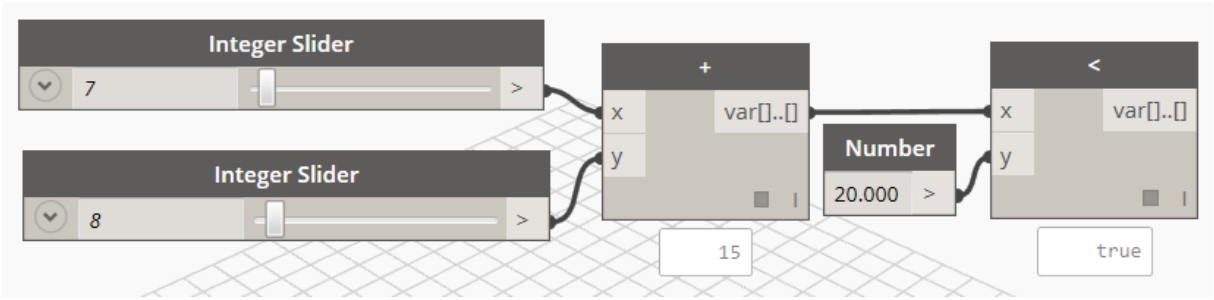
1. Geometric Constraint Solver
2. Visuelle Programmiersprachen

Als Constraint Solver bezeichnet man ein Programm, welches im Großen und Ganzen aus vorgegebenen Bedingungen (Constraints) eine Lösungsmenge bestimmt. Ein Geometric Constraint Solver zeichnet sich dadurch aus, dass Constraints über geometrische Parameter bestimmt werden. Ein einfaches Beispiel ist folgendes:

1. In einem Zeichenprogramm wird eine Linie mit Anfangspunkt P1 und Endpunkt P2 gezeichnet
2. Dieser Linie gibt man die Bedingung vor, dass sie horizontal sein muss
3. Die Lösungsmenge, die durch diese Bedingung entsteht sind also alle Linien, bei denen gilt: P1.Y = P2.Y, wobei P1.Y den Y-Wert des Punktes P1 in einem 2D-Koordinatensystem entspricht

Das Ausgeben der in Schritt drei beschrieben Lösungsmenge ist dabei Aufgabe des Geometric Constraint Solvers.

Visuelle Programmiersprachen sind auf Knoten und deren Verbindungen untereinander basierende Programmiersprachen. Die Knoten sind fertige Code-Blöcke, welche Inputs (z.B. eingehende Verbindungen) Verarbeiten und daraus Outputs (ausgehende Verbindungen) erzeugen. Ein einfaches Beispiel ist folgendes:



Hier werden zwei Integer (leicht über einen Slider veränderbar) miteinander addiert und dann gecheckt, ob das Ergebnis kleiner als 20 ist.

Die Komplexität der Blöcke kann natürlich variieren, von einfachen Summenoperatoren bis hin zu Skizzen, welche Constraints verarbeiten und dann eine Lösungsmenge dazu ausgeben können. Die in diesem Praktikum betrachtete visuelle Programmiersprache war Dynamo, ein Open Source Projekt basierend auf Autodesk Revit.

Verlauf des Praktikums

Das Forschungspraktikum war zu Beginn darauf ausgelegt, einen existierenden Open Source Geometric Constraint Solver in ein C#-Projekt zu implementieren.

Das daraus entstehende Projekt sollte die Geometrische Grundlage für einen Knoten in Dynamo bilden. Dieser Knoten sollte dann die Möglichkeit bieten

1. eingehende Verbindungen als Parameter verarbeiten zu können,
2. direkt in der Darstellung Constraints einzufügen,
3. geometrische Konstrukte im Rahmen der ihnen gestellten Bedingungen zu verschieben und
4. die entstehende Lösungsmenge als System von Punkten auszugeben.

Im Verlauf des Praktikums stellte sich heraus, dass der Arbeitsaufwand für die Erstellung des Skizzenknotens die angesetzten 60 Arbeitsstunden für ein Forschungspraktikum übersteigen würden, somit wurde der Schwerpunkt auf die Implementierung eines Solvers in ein C#-Projekt gelegt.

Vergleich der Constraint Solver

Folgende Solver wurden in Betracht gezogen und miteinander verglichen:

1. Free GCS
2. GeoSolver
3. SolveSpace

Der Schwerpunkt wurde auf die Analyse von SolveSpace gelegt, da bereits Anfänge eines Projekts von Herrn Singer mit diesem Solver existierten.

Free GCS

Programmiersprache	C++
Mögliche Constraints	Equal (inkl. Horizontal, Vertikal), Differnce, P2PDistance, P2PAngle, P2LDistance, PointonLine, Parallel, Perpendicular, L2LAngle, MidpointOnLine

Lösungsverfahren	- unterteilt das System in bis zu drei Untersysteme
	- Zur Lösung können drei Algorithmen angegeben werden:
	  ○ BFGS (ein Quasi-Newton)
	  ○ Levenberg-Marquart
	  ○ DogLeg (Trust Region)

Geo Solver

Programmiersprache	Python
Mögliche Constraints	Jegliche Varianten von ParametricConstraint und insbesondere von dessen folgenden Unterklassen: - FixConstraint (fester Abstand eines Punktes relativ zum Koordinatensystem) - DistanceConstraint (zwischen zwei Punkten) - AngleConstraint
Lösungsverfahren	- Erstellen eines bipartiten Graphs (Cluster) (Variablen<-> Constraints) - SolverCluster mit GeoCluster mappen (und vice versa) - Subcluster bestimmen und lösen - Cluster kombinieren - Top-level results bestimmen

SolveSpace

Programmiersprache	C++
Mögliche Constraints	POINTS_COINCIDENT, PT_PT_DISTANCE, PT_PLANE_DISTANCE, PT_LINE_DISTANCE, PT_FACE_DISTANCE, PROJ_PT_DISTANCE , PT_IN_PLANE, PT_ON_LINE, PT_ON_FACE, EQUAL_LENGTH_LINES, LENGTH_RATIO, EQ_LEN_PT_LINE_D, EQ_PT_LN_DISTANCES, EQUAL_ANGLE, EQUAL_LINE_ARC_LEN, SYMMETRIC, SYMMETRIC_HORIZ, SYMMETRIC_VERT, SYMMETRIC_LINE, AT_MIDPOINT, HORIZONTAL, VERTICAL, DIAMETER, PT_ON_CIRCLE, SAME_ORIENTATION, ANGLE, PARALLEL, PERPENDICULAR, ARC_LINE_TANGENT, CUBIC_LINE_TANGENT, CURVE_CURVE_TANGENT, EQUAL_RADIUS, WHERE_DRAGGED

Lösungsverfahren	- Bestimmen und Lösen von alleinstehenden/unabhängigen Gleichungen - Modifizierter Newton für den Rest
Input/Output	- Eingabe von Constraints über addConstraint Methode - Die Koordinaten der eingegeben SolverPoints werden den Constraints entsprechend geändert

Fazit

Da alle Solver die wichtigsten Constraints lösen konnten und der Implementierung her keine großen Unterschiede ergaben, wurde mit dem bestehenden SolveSpace Projekt weitergearbeitet.

DynamoSketchNode

Grundlegendes Konzept

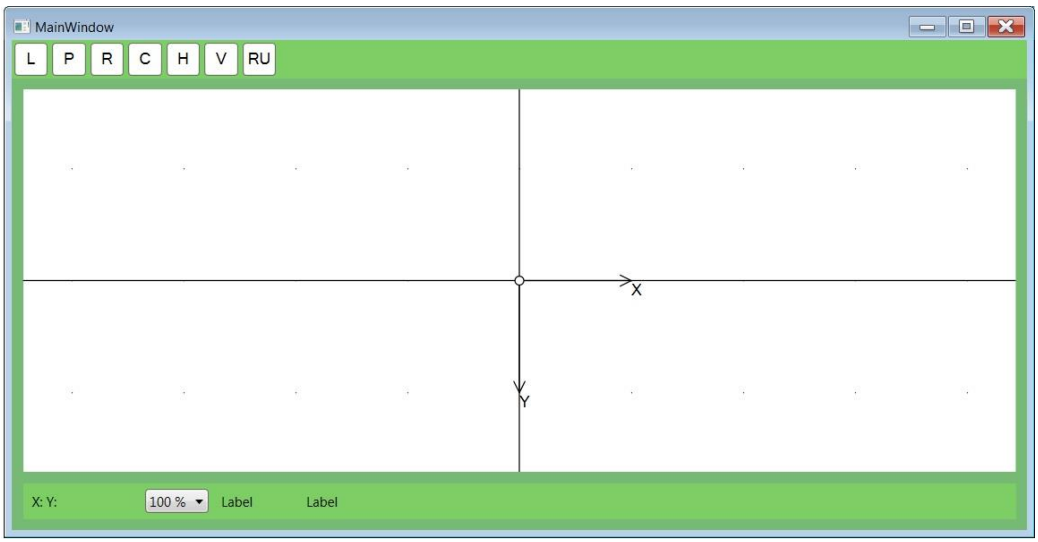
Das Prinzip von DynamoSketchNode ist, dass zu jedem gezeichneten Objekt auf der Skizzenfläche (SketchPoint, SketchLine) ein entsprechendes Solver-Objekt (SolverPoint, SolverLine) erstellt wird, welches dann dem implementierten Solver übergeben werden kann. Dem Solver werden ebenfalls die gewünschten Contraints übergeben. Aus den Solver-Objekten und den Constraints berechnet dann der Solver wenn möglich eine Lösung des Systems und verändert die Koordinaten der Solver-Objekte. Dementsprechend müssen dann die Koordinaten der Skizzenobjekte angepasst werden.

Wird das gezeichnete Objekt verändert (z.B. durch ziehen auf der Oberfläche) müssen die Koordinaten den Solver-Objekts angepasst werden. Außerdem sollen beim Ziehen keine bereits gelösten Constraints verletzt werden können. Dies beinhaltet sowohl, dass z.B. horizontale Linien nicht schräg werden können, als auch dass Polygone nicht aufreißen.

Zu jedem sichtbaren Objekt gibt es ein virtuelle Objekt (VirtualPoint, VirtualLine) mit größeren Ausmaßen, um das anklicken zu erleichtern.

Anfangszustand

In dem bestehenden Projekt war eine graphische Oberfläche wie folgt modelliert:



Ebenfalls implementiert waren bereits die notwendigen Klassen sowie ein Wrapper für den Solver von SolveSpace. Das Programm warf keine Compilerfehler, die Funktionalität war jedoch sehr eingeschränkt:

Was funktionierte	Was nicht funktionierte
Zeichnen von Punkten	Zeichnen von Kreisen/Bögen
Zeichnen von Linien	Löschen von Objekten
Zeichen von EINEM Polygon (Schließen über Rechtsklick)	Schließen weiterer Polygone
Ziehen einer Linie	Ziehen ohne bestehende Constraints zu verletzen
Mehrere Objekte selektieren	Deselektieren von Objekten

Das Hauptproblem stellte die Tatsache dar, dass bestehende Constraints nicht fix waren, d.h. sie konnten beim Ziehen von Objekten verletzt werden und bei einem überbestimmten System gab es keine Fehlermeldung. Das Ziehen allgemein wurde im Laufe des Praktikums zum Hauptschwerpunkt.

Problemlösung

Als erstes wurde versucht, kleinere Probleme zu beheben, um das Programm bedienbar zu machen. Das Zeichnen von Kreisen/Bögen sowie das Löschen von Objekten wurde vernachlässigt, da das primäre Ziel mittlerweile die korrekte Implementierung eines Constraint Solvers war.

Deselektieren von Objekten

Problembeschreibung: Objekte waren farblich noch als markiert gemeldet, wenn auf andere Objekte geklickt wurde. Das Deselektieren funktionierte bei Klick auf die Leinwand.

Ursache: War eine VirtualLine angeklickt worden, wurde auch die SketchLine als selektiert gesetzt. Beim Klicken auf ein anderes Objekt, wurde die SketchLine nicht wieder deselektiert (beim Klicken auf die Leinwand funktionierte das Deselektieren).

Lösung: Das Attribut „isSelected“ in oben beschriebenem Fall auf false setzen.

Schließen von Polygonen (über Rechtsklick)

Problembeschreibung: Beim Schließen von Polygonen über Rechtsklick wurde immer zum Startpunkt des ersten Polygons hin geschlossen und nicht zum Startpunkt des gerade gezeichneten Polygons.

Ursache: Für die Linien aller gezeichneten Polygone gab es nur eine Liste (Polyline). Beim Zeichnen eines Polygons wurden die Linien über Polyline.add(Line) zu dieser Liste hinzugefügt. Beim Zeichnen eines zweiten Polygons wurden auch diese Linien zu Polyline hinzugefügt. Das Schließen über Rechtsklick erfolgte immer zum Startpunkt der ersten Linie in Polyline und somit zum Startpunkt der ersten Linie im ersten Polygon.

Lösung: Die Liste Polyline nach dem Schließen eines Polygons leeren.
Somit wird bei jedem neuen Polygon eine neue Liste angelegt und zum Startpunkt der ersten Linie geschlossen.
ABER: Polyline enthält damit nur temporäre Daten und es existiert keine Liste mit Polygonen.

Fixieren von Objekten über Constraints

Problembeschreibung: Linien konnten zwar als Horizontal/Vertikal deklariert werden (und wurden auch auf der Skizze so verändert), waren jedoch durch Ziehen weiterhin beliebig veränderbar. Wurde der Solver nach dem Ziehen erneut aufgerufen, sprang die Linie auf ihre horizontale/vertikale Position zurück, anstatt die Länge und Höhe der neuen Position anzupassen.
Beim Ziehen von Punkten/Linien in einem Polygon, wurde der Koinzidenz-Constraint der Eckpunkte nicht eingehalten.
Ursache: Die Ursachen konnten nicht zu 100% festgestellt werden, es gab jedoch einige Problemzonen und Vermutungen:

1. Wenn das virtuelle Objekt gezogen wird, müssen die Koordinaten des gezeichneten Objekts angepasst werden.
2. Wenn das gezeichnete Objekt seine Koordinaten verändert, muss der SolverPoint daran angepasst werden.
3. Da beim Ziehen einer Polygonlinie ein Constraint verletzt werden konnte, hätte der Solver in Realtime lösen müssen.

Die Hauptursache schien also das fehlerhafte Update der verschiedenen Punktvarianten (Virtuell, gezeichnet, Solver) untereinander, sowie der punktuelle Aufruf des Solvers.

Lösungsansätze: Von den Update-Funktionen waren folgende implementiert:

1. SolverStartPointUpdate
    a. Wenn ein SketchPoint eine Positionsänderung erfährt, soll er seine neuen Koordinaten an den SolverPoint schicken.
    b. Dies geschah in Line.cs
2. HandleSolverSolved
    a. Wenn der Solver eine Lösung ausgiebt, dann müssen die Koordinaten des gezeichneten Punktes angepasst werden.
    b. Dies geschah in Line.cs
3. HandlePointPositionChanged
    a. Ein virtueller Punkt ändert seine Koordinaten, wenn sein gezeichneter Punkt eine Änderung erfährt.
    b. Dies geschah in VirtualSketchPoint.cs

Nicht implementiert waren:

1. SolverEndPointUpdate (in Line.cs)
2. HandleVirtualPointPositionChanged (in SketchPoint.cs)

Zu diesem Punkt wurde schon deutlich, dass das ganze Update System großes Fehlerpotenzial hatte. Es wurde weiter versucht, die fehlenden Updates zu implementieren und Fehler in den bestehenden Updates zu korrigieren, aber es sei vorweg genommen, dass letzten Endes vom Betreuer ein neues Grundkonzept des gesamten Programms entworfen und umgesetzt wurde.

Zu SolverEndPointUpdate:

Da schon eine Implementierung zur Anpassung des Start-Punktes einer Linie gemäß seinem Solver-Punkt vorhanden war, wurde diese für den Endpunkt übernommen und angepasst. Das Ergebnis war, dass die Linie nach einem Solveraufruf fest wurde, d.h. in keine Richtung gezogen werden konnte, selbst wenn es innerhalb der bestehenden Constraints möglich sein sollte.

Zu HandleVirtualPositionChanged:

Anfangs bestand eine Verbindung zwischen einem SketchPoint und seinem VirtualPoint nur einseitig vom VirtualPoint aus gesehen. Damit vom Sketchpoint aus auf die Koordinaten seines VirtualPoints zugegriffen werden konnte, wurde jedem Sketchpoint ein VirtualPoint als Attribut zugewiesen. Nach der Anpassung aller Methoden- und Konstruktorköpfe auf das neue Attribut funktionierte das Updaten der Koordinaten immer noch nicht richtig.

Fazit

Das ganze Programm wurde durch die Änderungen nicht verwendbar, was in Kombination mit der Unübersichtlichkeit zu einen neuen Konzept führte.

Konzept mit Bindings

Änderungen gegenüber DynamoSketchNode

Die große Änderung bestand darin, dass beim Ziehen von Elementen nicht der Solver über ein Koinzidenz-Constraint das Zusammenbleiben von einer Linie/einem Polygon garantiert, sondern Bindings.

Zur Vereinfachung wurden virtuelle Objekte komplett weggelassen und die gezeichneten Objekte so groß festgelegt, dass ein Anklicken ohne Probleme möglich ist.

Implementierung eines Solvers

Als Ziel wurde festgelegt, dass Horizontal- und Vertikal-Constraints gelöst werden können. Das Lösen muss nicht permanent (während des Ziehens) stattfinden, es reicht, wenn auf Befehl gelöst werden kann. Zu diesem Zweck wurde das Projekt „Constraint Solver“ aus DynamoSketchNode übernommen.

Schritt 1:

Jedem gezeichneten Objekt wird ein Solver-Objekt als Attribut zugewiesen. Das Updaten der Solver-Objekt-Koordinaten geschieht direkt, wenn die Koordinaten des gezeichneten Objekts geändert werden.

Ein globaler Solver und eine globale Workplane werden vorausgesetzt und jedem gezeichneten Objekt als Attribut zugewiesen.

Der Solver und die Workplane werden in der Main-Methode initialisiert.

Schritt 2:

Bei einem Klick auf AddHorizontalConstraint oder AddVertikalConstraint wird ein Horizontal/Vertikal Constraint erstellt. Bei Abschluss des Praktikums war dies nur mit einer markierten Linie möglich.

Eine If-Abfrage, welche Objekte markiert sind und ein dementsprechend anderer Methodenaufruf von AddConstraint sollten das Problem beheben.

WICHTIG: Der Constraint-Count muss hoch genug angesetzt werden, da sonst Konflikte mit solverinternen Constraints entstehen können.

Schritt 3:

Bei einem Klick auf den Solve-Button wird die Solve-Methode des Solvers aufgerufen.

In der Version, die dem Betreuer vorliegt, werden nur die Koordinaten von markierten Objekten angepasst. Dies ist leicht zu beheben, indem in MainWindow.xaml.cs die If-Abfrage aus Zeile 192 entfernt wird.

Ergebnis

Das Ziehen klappt mit den Bindings sehr gut und Horizontal/Vertikal-Constraints können eingegeben werden.

Es bestehen insgesamt noch einige Probleme, diese konnten aber im Rahmen des Praktikums nicht mehr gelöst werden.

Persönliches Fazit

Ein unbekanntes Projekt in einer unbekannten Programmiersprache war eine große Herausforderung und barg einige Stolpersteine. Dies war aber eine gute Gelegenheit mit der Arbeit in eben unbekanntem Gebiet Erfahrung zu sammeln, z.B. wo in fremden Code Fehlerpotenzial liegt, wie man sich notwendige Fähigkeiten schnell aneignet und selbstständig Lösungsansätze entwickelt.

An dieser Stelle möchte ich noch meinem Betreuer für die gute Zusammenarbeit und Unterstützung danken und hoffe mit diesem Praktikum seinem Projekt geholfen zu haben.