Universität Leipzig
Fakultät für mathematik und Informatik
Institut für Informatik

Dynamische Geometriesoftware auf Java-Basis

Bachelor-Arbeit

Leipzig, November 2004

vorgelegt von

Stock, Andy

geb. am 16.08.1981

Studiengang: Informatik

Zusammenfassung

Ziel dieser Arbeit ist es, einen Überblick über die theoretischen und praktischen Grundlagen der Entwicklung dynamischer Geometriesoftware auf Basis von Java zu geben. Im Rahmen meines Bachelor-Praktikums habe ich selbst ein dynamisches Geometrieprogramm, Geofuchs, entwickelt. Obwohl ich für viele der dabei aufgetretenen Probleme auch auf alternative Lösungen hinweise, bildet die Beschreibung von Geofuchs und seinen Grundlagen den zentralen Teil dieser Arbeit.

Nach einer Einführung in das Gebiet der dynamischen Geometrie, ihre Möglichkeiten, Probleme und ausgewählte Programme beschreibe ich die reelle projektive Geometrie, auf der Geofuchs basiert, aus analytischer Sicht. Sobald aber nicht nur ihre mathematischen Grundlagen, sondern dynamische Geometriesoftware selbst beschrieben werden soll, ist das bekannte geometrische Vokabular nicht mehr ausdrucksfähig genug. Daher stelle ich eine Terminolgie, die den Unterschieden zwischen statischer und dynamischer Geometrie Rechnung trägt, vor. Aufbauend darauf erläutere ich, wie Graphenalgorithmen in dynamischer Geometriesoftware eingesetzt werden können und stelle einen vergleichsweise einfachen Algorithmus vor, mit dem sich Sprünge von Schnittpunkten, ein Problem vieler dynamischer Geometrieprogramme, vermeiden lassen. Geofuchs ist in Java implementiert und nutzt XML zum Speichern und Laden von Konstruktionen. Ich erkläre, warum Java und XML sich für die Entwicklung dynamischer Geometriesoftware besonders eignen, und mit Hilfe welcher Techniken XML-Dokumente mit Java verarbeitet werden können.

Der letzte Teil der Arbeit beschäftigt sich in erster Linie mit Geofuchs. Zunächst beschreibe ich die grafische Benutzeroberfläche von Geofuchs aus der Sicht des Benutzers und erläutere ihre Vor- und Nachteile. Dann beschreibe ich den Aufbau des von Geofuchs zum Speichern von Konstruktionen verwendeten XML-basierten Formats GeoXML und gebe einen groben Überblick über die programminternen Abläufe beim Laden und Speichern von Konstruktionen. Abschließend erläutere ich einige Details der Implementierung von Geofuchs und bewerte Geofuchs im Vergleich zu anderen dynamischen Geometrieprogrammen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Was ist dynamische Geometriesoftware?

Die Geometrie (griechisch: Landvermessung) ist ein Teilgebiet der Mathematik. Sie entstand bereits in der Antike als Lehre vom Raum und den räumlichen Objekten. Ihre Grundlagen wurden durch Abstraktion von Beobachtungen der Außenwelt gewonnen. Obwohl die Geometrie sich mit zunehmender Axiomatisierung immer mehr von der räumlichen, visuellen Vorstellung löste, findet der erste Kontakt eines Schülers zur Geometrie normalerweise nach wie vor über einen visuellen Zugang statt. Dabei werden geometrische Gebilde wie Punkte, Geraden und Kreise, die in bestimmten Beziehungen (z. B. Orthogonalität im Falle von Geraden) zueinander stehen, mit Hilfe von Stift, Lineal und Zirkel schrittweise zu Papier gebracht. Das Ergebnis nennt man eine geometrische Konstruktion. Mit zunehmender Verbreitung und Leistungsfähigkeit von Computern entstanden bereits in den frühen 80ern erste Geometrieprogramme.

Dynamische Geometriesoftware stellt nicht nur einen präziseren Ersatz für Lineal und Zirkel zur Verfügung, sondern erlaubt auch die nachträgliche, interaktive Manipulation von Konstruktionen. Dabei wird zunächst eine Konstruktion erzeugt. Ausgehend von einigen vom Benutzer auf einer Zeichenfläche (die einen Ausschnitt einer Ebene repräsentiert) platzierten Punkten können abhängige geometrische Gebilde, beispielsweise Geraden durch zwei Punkte, Kreise durch Angabe ihres Mittelpunktes und eines Punktes auf ihrer Peripherie oder Schnittpunkte von Kreisen und Geraden konstruiert werden. Dabei ist es dem Benutzer zu jedem Zeitpunkt möglich, alle Punkte, deren Lage nicht vollständig durch ihre Beziehungen zu anderen geometrischen Gebilden bestimmt ist, zu bewegen. Das Programm stellt sicher, dass bestehende Beziehungen zwischen den geometrischen Gebilden erhalten bleiben, und bewegt die betroffenen geometrischen Gebilde mit. Werden also zwei Geraden durch je zwei Punkte konstruiert und wird dann ihr Schnittpunkt bestimmt, so verändern beim Bewegen erstgenannter Punkte sowohl die beiden Geraden als auch ihr Schnittpunkt ihre Lage (siehe Abbildung 1.1).

1.2 Funktionalität und Anwendungsgebiete

Damit der Benutzer Konstruktionen erstellen kann, muss ein dynamisches Geometrieprogramm ihm Konstruktionswerkzeuge und eine Zeichenfläche, die Lineal, Stift, Zirkel und Papier ersetzen, zur Verfügung stellen. Es müssen also Grundoperationen verfügbar sein, die

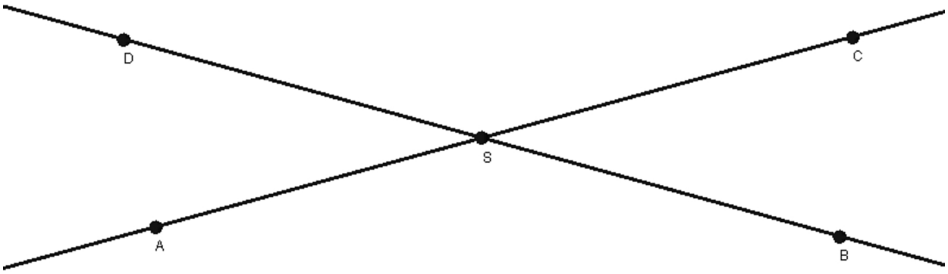

Abbildung 1.1: Die Punkte A und C sowie B und D definieren jeweils eine Gerade eindeutig. Der Punkt S ist der Schnittpunkt beider Geraden. Dieses Konstruktion kann mit einem dynamischen Geometrieprogramm wie folgt erstellt werden: Zunächst muß der Benutzer wie in einem Zeichenprogramm ein Werkzeug „Punkt hinzufügen“ auswählen. Dann platziert er durch mehrere Mausclicks die Punkte auf einer Zeichenfläche. Nun wählt er ein anderes Werkzeug zum Konstruieren von Geraden aus. Die Geraden AC und BD kann der Benutzer jetzt durch Mausclicks auf die entsprechenden Punkte konstruieren. Letztendlich wählt er ein Werkzeug zum Konstruieren des Schnittpunkts zweier Geraden aus. Durch einen Mausclick auf beide Geraden AC und BD gleichzeitig (oder, je nach verwendetem Programm, zwei aufeinanderfolgende Mausclicks auf jeweils eine der Geraden) konstruiert er den Schnittpunkt S. Ein dynamisches Geometrieprogramm stellt auch ein Werkzeug zum Bewegen von Punkten zur Verfügung. Nachdem der Benutzer dieses Werkzeug ausgewählt hat, kann er die Punkte A, B, C und D mit der Maus verschieben. Dabei bewegen sich auch die Geraden AC und BD und ihr Schnittpunkt S so, dass AC und BD weiterhin durch die Punkte A und C bzw. B und D verlaufen und S weiterhin der Schnittpunkt von AC und BD ist.

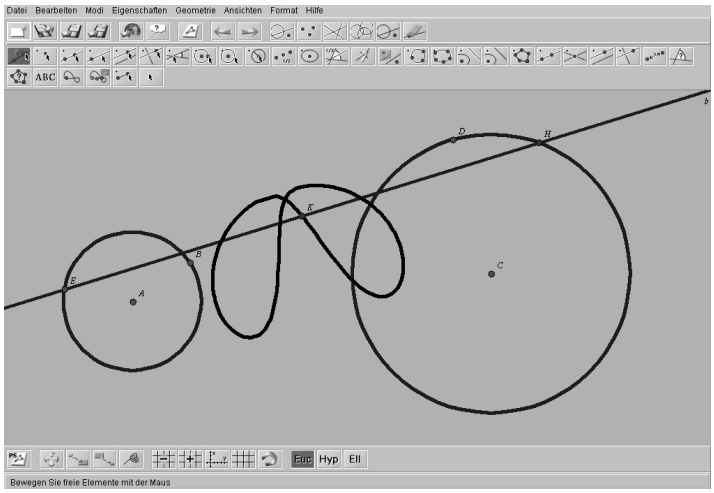

Abbildung 1.2: Eine Ortskurve mit Cinderella (siehe Abschnitt 1.4.2).

das Erstellen aller mit Zirkel und Lineal erstellbaren Konstruktionen ermöglichen. Zu diesen Grundoperationen gehört beispielsweise das Konstruieren einer Gerade, die durch zwei gegebene nicht identische Punkte verläuft. Normalerweise bieten dynamische Geometrieprogramme auch Werkzeuge zum direkten Erzeugen von beispielsweise Mittelpunkten, Parallelen und Senkrechten an, um das Erstellen von Konstruktionen zu vereinfachen. Viele dynamische Geometrieprogramme unterstützen auch Kegelschnitte, und Unterstützung für nicht-euklidische Geometrien ist ebenfalls verfügbar (siehe Abschnitt 1.4.2). Außerdem müssen jene Punkte, deren Position nicht durch ihre Beziehungen zu anderen geometrischen Gebilden vollständig vorgegeben ist, bewegt werden können. Dabei können Punkte normalerweise auch an geometrische Gebilde wie Geraden oder Kreise gebunden und dann nur noch auf diesen bewegt werden. Solche Punkte nennt man Gleiter. Viele dynamische Geometrieprogramme ermöglichen es zudem, die Positionen von Punkten im zeitlichen Verlauf – ihre Spuren oder Ortskurven – aufzuzeichnen und darzustellen (siehe Abbildung 1.2).

Ulrich H. Kortenkamp und Jürgen Richter-Gebert nennen in [24] drei Anwendungsgebiete dynamischer Geometriesoftware: Das Erstellen von Zeichnungen, den Einsatz als „geometrisches Experimentierfeld“ und das Vermitteln von Wissen.

Das Erstellen von Zeichnungen mittels dynamischer Geometriesoftware bietet einen entscheidenden Vorteil: Die Zeichnung kann im Nachhinein durch Bewegen der freien Punkte so manipuliert werden, dass der zu veranschaulichende Sachverhalt besonders hervorgehoben wird (siehe Abbildung 1.3). Um ansprechende Zeichnungen erstellen zu können, sollte das eingesetzte dynamische Geometrieprogramm umfassende Optionen für die Darstellung von Konstruktionen zur Verfügung stellen, beispielsweise das Ändern der Darstellungsfarben oder das Beschriften geometrischer Gebilde. Eine Exportfunktion in gängige Grafikformate ist ebenfalls wünschenswert. Viele der in dieser Bachelor-Arbeit verwendeten Zeichnungen sind mit dem von mir entwickelten dynamischen Geometrieprogramm Geofuchs (siehe Abschnitt 1.5) erstellt worden.

Der Einsatz dynamischer Geometriesoftware als „geometrisches Experimentierfeld“ dient dem Erlangen neuer Erkenntnisse. Der Benutzer kann Hypothesen über geometrische und

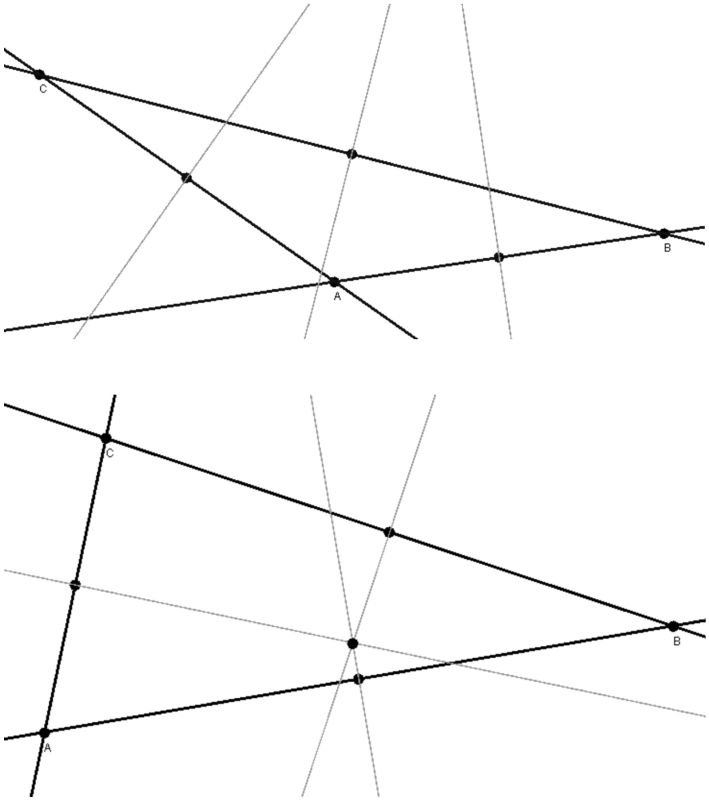

Abbildung 1.3: Die Mittelsenkrechten des Dreiecks ABC schneiden sich in einem Punkt. Die obere Zeichnung kann diesen Sachverhalt nicht hinreichend darstellen, da der Schnittpunkt der Mittelsenkrechten außerhalb der Zeichnung liegt. Wäre die Zeichnung auf Papier entstanden, müsste sie von Anfang an neu erstellt werden, und mit einem einfachen Zeichenprogramm müssten alle Punkte und Geraden verschoben werden. Mit einem dynamischen Geometrieprogramm dagegen müssen nur die Punkte A, B und C bewegt werden, um ein anschaulicheres Resultat wie z. B. die untere Zeichnung zu erhalten.

sonstige Sachverhalte, die sich geometrisch modellieren lassen, durch die interaktive Manipulation von Konstruktionen gewinnen und anschaulich überprüfen. Durch die Nutzung eines automatischen Geometrie-Beweisers kann gegebenenfalls die Korrektheit der Hypothese bewiesen werden.

Die Vermittlung von Wissen stellt schließlich aus meiner Sicht das Haupteinsatzgebiet dynamischer Geometrieprogramme dar (siehe auch [27]). Neben den genannten Visualisierungs- und Experimentiermöglichkeiten, die sich auch im Mathematikunterricht nutzen lassen, ermöglicht Cinderella (siehe Abschnitt 1.4.2) beispielsweise das Erstellen interaktiver Aufgabenblätter, die auch in Internetseiten eingebunden werden können. Lösungsversuche werden zudem durch einen randomisierten Geometrie-Beweiser auf ihre Korrektheit überprüft.

1.3 Anforderungen und Probleme

Aus den in Abschnitt 1.2 genannten Anwendungsgebieten ergeben sich Anforderungen an dynamische Geometriesoftware, deren Gewichtung von der anvisierten Zielgruppe abhängt. Entwickelt man ein dynamisches Geometrieprogramm hauptsächlich als Lernsoftware auf Schulniveau, so können von den Benutzern weder Übung im Umgang mit Computern noch weitergehende Geometriekenntnisse erwartet werden. Dementsprechend wichtig ist eine attraktive, intuitiv verständliche grafische Benutzeroberfläche, um die Konzentration auf das Wesentliche – das zu vermittelnde Wissen – zu ermöglichen. Für ein dynamisches Geometrieprogramm, das im akademischen Bereich eingesetzt werden soll, rückt demgegenüber der Funktionsumfang in den Vordergrund. Beide Forderungen müssen sich jedoch keineswegs ausschließen (siehe auch Abschnitt 1.4).

Ulrich Kortenkamp stellt in seiner Dissertation [19] die Notwendigkeit einer konsistenten mathematischen Grundlage für dynamische Geometrieprogramme heraus. Soll der Benutzer durch die Anwendung des Programms neues Wissen erlangen, so kann nicht vorausgesetzt werden, dass er in der Lage ist, zwischen eigenen Fehlern beim Konstruieren und Fehlern des Programms zu unterscheiden. Durch das Geometrieprogramm selbst bedingte Inkonsistenzen ergeben sich zum einen aus numerischen Problemen, zum anderen durch geometrische Gebilde, deren Lage durch Bedingungen teilweise, aber nicht vollständig festgelegt ist. In diesem Fall kann es zu unvorhersagbarem und unstetigem Verhalten kommen, das heißt eine kleine Änderung der Position eines geometrischen Gebildes kann einen großen „Sprung“ davon abhängiger geometrischer Gebilde auslösen. Nachdem Harald Winroth im Rahmen seiner Dissertation [27] dieses Problem nur teilweise lösen konnte, gelang es Ulrich Kortenkamp mit der seinen, einen konsistenten mathematischen Rahmen für dynamische Geometrieprogramme, in dem derartige Sprünge (zum Preis einer aufwändigen Implementierung) weitgehend vermeidbar sind, zu beschreiben.

1.4 Ausgewählte Programme

Eine große Zahl dynamischer Geometrieprogramme ist verfügbar, und viele weitere befinden sich in Entwicklung. Im Folgenden möchte ich lediglich drei Programme vorstellen: „Zirkel und Lineal“, Cinderella und GeoGebra. Zu diesen in Java implementierten Programmen sind umfangreiche Informationen über ihre technische Umsetzung verfügbar. Jedoch ist „Zirkel und Lineal“ in Bezug auf seine mathematische Grundlage und seine Programmstruktur eher konservativ, während Cinderella und GeoGebra diesbezüglich einem moderneren und inno-

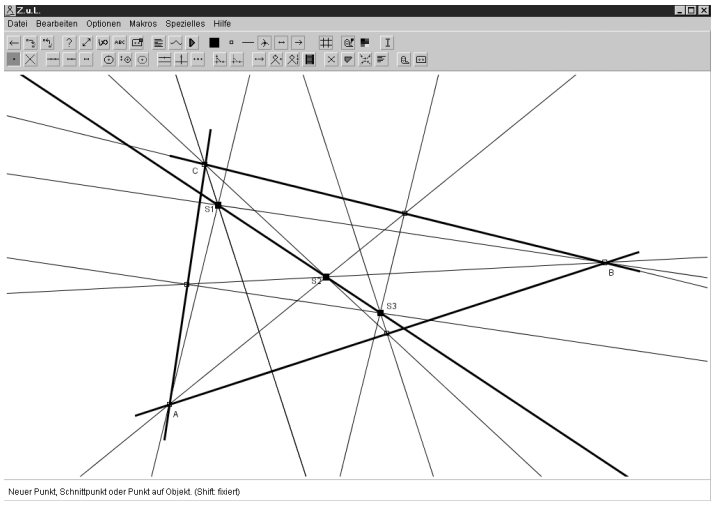

Abbildung 1.4: Der Schnittpunkt der Mittelsenkrechten, der Schnittpunkt der Seitenhalbierenden und der Schnittpunkt der Höhen eines Dreiecks liegen auf einer Geraden, der sogenannten Euler-Geraden. Diese wurde hier mit „Zirkel und Lineal“ konstruiert.

vativeren Ansatz folgen. Gleichzeitig zeigen diese Programme auf, welcher Funktionsumfang von einem dynamischen Geometrieprogramm zu erwarten ist.

1.4.1 „Zirkel und Lineal“

„Zirkel und Lineal“ [14] wurde von René Grothmann an der Katholischen Universität Eichstätt entwickelt. Das Programm ist, auch im Quellcode, kostenlos verfügbar. Eine kurze technische Beschreibung findet sich online unter [13]. Das Programm ist hauptsächlich für die Verwendung im Schulunterricht vorgesehen und bietet nur grundlegende Konstruktionswerkzeuge an. Beispielsweise können zwar Kegelschnitte durch fünf gegebene Punkte konstruiert werden, es ist jedoch nicht möglich, deren Schnittpunkte mit anderen geometrischen Gebilden zu bestimmen. Abgesehen davon stehen umfangreiche Funktionen zur Verfügung. Beispielsweise ist das Erstellen von Makros oder das Anzeigen der Spuren von Punkten möglich. Die Darstellung der geometrischen Gebilde, beispielsweise ihre Farbe, lässt sich beeinflussen. Ebenso kann man die grafischen Repräsentationen von Konstruktionen in verschiedene Grafikformate exportieren. Problematisch ist das vergleichsweise häufige Auftreten von Sprüngen. Die grafische Benutzeroberfläche erlaubt zwar eine sehr komfortable Bedienung, ist aber wenig attraktiv. Zu jedem Zeitpunkt kann nur eine Konstruktion, die nur auf einer Zeichenfläche dargestellt werden kann, geöffnet sein. Das automatische Erstellen von Java-Applets, die dann in Webseiten eingebunden werden können, ist möglich. Es können „dynamische Aufgabenblätter“ erstellt werden, Lösungsversuche werden jedoch nicht automatisch auf ihre Korrektheit überprüft.

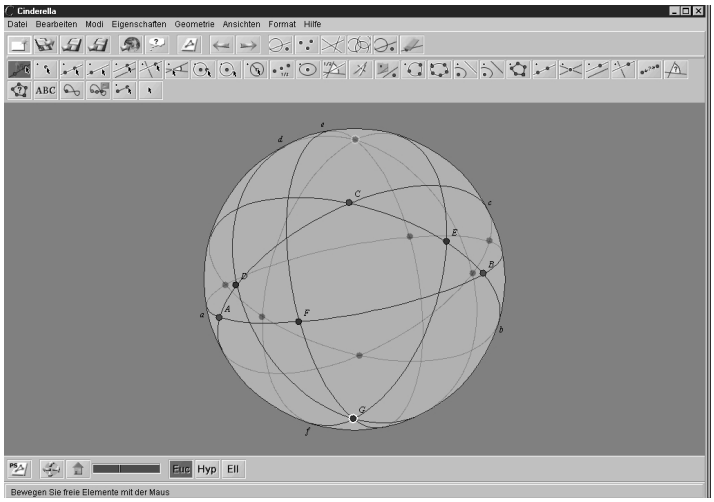

Abbildung 1.5: Cinderella unterstützt auch nicht-euklidische Geometrien.

1.4.2 Cinderella

Cinderella [20] wurde von Ulrich Kortenkamp und Jürgen Richter-Gebert entwickelt. Die Universitäts-Version 1.4 ist kostenlos im Internet, der Quellcode von Cinderella gemäß [19] für Forschungszwecke auf Anfrage bei den Entwicklern auszugsweise verfügbar. Im Gegensatz zu „Zirkel und Lineal“ (siehe Abschnitt 1.4.1) konnte ich bei Cinderella keine Sprünge geometrischer Gebilde oder sonstige mathematische Inkonsistenzen finden (zur zu Grunde liegenden Mathematik siehe Kapitel 2.3). Cinderella unterstützt euklidische, elliptische und hyperbolische Geometrie. Konstruktionen können (gegebenenfalls in unterschiedlichen geometrischen Interpretationen) in mehreren Zeichenfenstern gleichzeitig dargestellt werden, wobei es kein übergeordnetes Hauptfenster gibt. Sofern das Programm nicht mehrfach gestartet wurde, kann zu jedem Zeitpunkt jedoch nur eine Konstruktion geöffnet sein. Die grafische Benutzeroberflächer ist attraktiv. Das Erstellen von Konstruktionen ist sehr komfortabel, das Bewegen von Punkten dagegen etwas umständlich gelöst (zweiteres erfordert die Auswahl eines eigenen Werkzeugs). Neben einer Vielzahl von Konstruktionswerkzeugen gibt es auch Werkzeuge zum Messen von Längen, Winkeln und Flächen. Die Darstellung der geometrischen Gebilde (Farbe, Größe usw.) kann beeinflusst werden. Des weiteren bietet Cinderella einen randomisierten Beweiser, Export von Konstruktionen als Encapsulated Postscript- und HTML-Dateien sowie das Erstellen interaktiver Aufgabenblätter (wobei Lösungsversuche vom randomisierten Beweiser auf ihre Korrektheit geprüft werden). In Bezug auf die mathematische Basis und den Funktionsumfang ist Cinderella aus meiner Sicht als „Stand der Technik“ zu betrachten.

1.4.3 GeoGebra

GeoGebra [15] entstand im Rahmen der Diplomarbeit [16] von Markus Hohenwarter an der Universität Salzburg und wird momentan im Rahmen einer Dissertation weiterentwickelt. Dabei versucht GeoGebra, Algebra, Anaylsis und Geometrie zu verbinden. So ist es z. B.

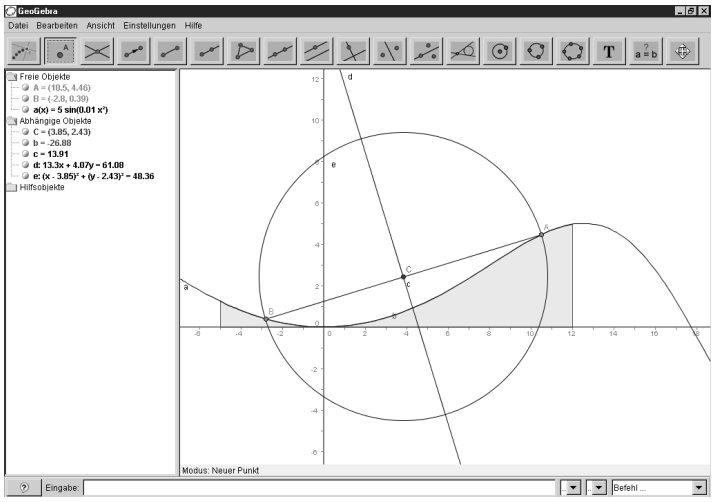

Abbildung 1.6: GeoGebra verbindet Algebra, Analysis und Geometrie.

möglich, die Ableitung einer Funktion zu bestimmen, eine entsprechende Kurve darzustellen, einen Gleiter daran zu binden und eine Gerade durch diesen und einen anderen Punkt zu konstruieren. Der Geometrieteil des Programms bietet alle notwendigen Grundfunktionen einschließlich Kegelschnitten. Die aktuelle Version 2.4 bietet vielfältige Möglichkeiten. Beispielsweise können Konstruktionen in verschiedene Grafikformate exportiert und als Internetseiten (im HTML-Format) gespeichert werden. Der Funktionsumfang von „Zirkel und Lineal“ und Cinderella wird allerdings nicht erreicht. Es ist z. B. nicht möglich, die grafische Darstellung geometrischer Gebilde, z. B. ihre Farbe, zu beeinflussen. Zu jedem Zeitpunkt kann nur eine Konstruktion, die nur auf einer Zeichenfläche dargestellt wird, geöffnet sein. Das Erstellen von Konstruktionen ist nicht so komfortabel und zügig wie mit „Zirkel und Lineal“ möglich. Beispielsweise können mit dem „Punkt hinzufügen“-Werkzeug zwar freie Punkte und Gleiter, aber keine Schnittpunkte konstruiert werden. Dazu, ebenso wie für das Bewegen von Punkten, ist ein eigenes Werkzeug auszuwählen. Die mathematische Basis von GeoGebra ist weit weniger komplex als die von Cinderella. Dennoch verhalten sich geometrische Gebilde in GeoGebra meist erwartungskonform. GeoGebra ist ein durchdachtes, innovatives dynamisches Geometrieprogramm, das im Übrigen mehrere Auszeichnungen, unter anderem den „European Academic Software Award“ im Jahr 2002 und den deutschen Bildungssoftware-Preis „digita 2004“, gewonnen hat. Auch GeoGebra ist (allerdings nicht im Quellcode) kostenlos verfügbar.

1.5 Über Geofuchs

Ziel meines Bachelor-Praktikums war die Entwicklung eines attraktiven dynamischen Geometrieprogramms auf konsistenter mathematischer Grundlage. Allerdings war zu Beginn des Praktikums der genaue Einsatzbereich für dieses Programm noch nicht klar. Deshalb sollte es zunächst nur grundlegende Funktionen zur Verfügung stellen. Konstruktionen sollten in einem XML-basierten Format gespeichert werden können. Darüber hinaus sollte das Programm leicht erweiterbar sein.

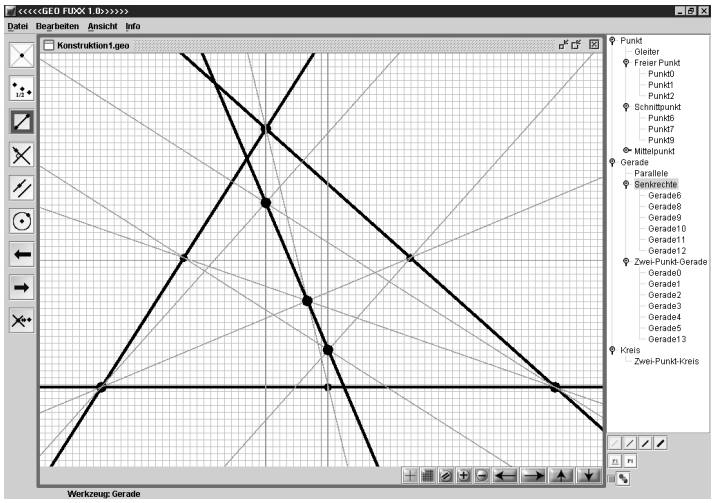

Abbildung 1.7: Die Euler-Gerade eines Dreiecks mit Geofuchs.

Zunächst habe ich „Zirkel und Lineal“, das im Quellcode verfügbar ist, auf seine Erweiterbarkeit untersucht. Jedoch erwiesen sich zentrale Teile des Programms, u. a. das gesamte mathematische Modell, als nicht den gestellten Anforderungen entsprechend. Es wären also umfangreiche Änderungen notwendig gewesen. Da der Quellcode von „Zirkel und Lineal“ unübersichtlich und kaum kommentiert ist, war der erwartete Aufwand nicht zu rechtfertigen. Daher traf ich die Entscheidung, stattdessen das dynamische Geometrieprogramm Geofuchs zur Grundlage meiner weiteren Arbeit zu machen.

Die erste Version von Geofuchs wurde in einem Softwaretechnik-Praktikum im Sommersemester 2004 von A.Aderhold, J.Bachmann, T.Chiacos, Y.Metzner, M.Todorov und A.Vollmer unter Betreuung von Prof. Dr. Hans-Gert Gräbe an der Universität Leipzig implementiert. Diese frühe Version zeichnete sich durch ihre attraktive Benutzeroberfläche aus.

Im Rahmen meines eigenen Praktikums habe ich zunächst das zu Grunde liegende geometrische Modell vollständig ersetzt und große Teile der grafischen Benutzeroberfläche geändert. Dann habe ich das Format zum Speichern und Laden von Konstruktionen entworfen und die entsprechenden Funktionen implementiert. Außerdem habe ich einige weitergehende Funktionen, z. B. das nachträgliche Ändern von Beziehungen zwischen geometrischen Gebilden und diverse Möglichkeiten zur Beeinflussung der Darstellung von geometrischen Gebilden, hinzugefügt.

Geofuchs unterstützt (Stand 25.10.2004):

- Freie Punkte
- Gleiter auf einer Geraden
- Gleiter auf einem Kreis
- Mittelpunkte zweier Punkte

- Schnittpunkte zweier Geraden
- Schnittpunkte zweier Kreise
- Schnittpunkte je einer Gerade und eines Kreises
- Geraden durch zwei Punkte
- Geraden, die zu einer gegebenen Geraden parallel sind und durch einen gegebenen Punkt verlaufen
- Geraden, die zu einer gegebenen Geraden orthogonal sind und durch einen gegebenen Punkt verlaufen
- Kreise mit gegebenem Mittelpunkt und einem Punkt auf der Peripherie
- Bewegen von freien Punkten und Gleitern
- Rückgängigmachen und Wiederholen von Konstruktionsschritten
- Nachträgliches Ändern von Beziehungen zwischen geometrischen Gebilden
- Ändern der Darstellungsfarbe geometrischer Gebilde
- Ändern, Anzeigen und Ausblenden der Beschriftung geometrischer Gebilde
- verschiedene Darstellungsmodi (versteckt, normal, hervorgehoben, stark hervorgehoben) für geometrische Gebilde
- Speichern und Laden von Konstruktionen in einem XML-basierten Format

Kapitel 2

Mathematische Grundlagen

2.1 Punkte und Geraden

Punkte sind die Grundbausteine der Elementargeometrie [10]. Bei der Entwicklung eines dynamischen Geometrieprogramms muss daher früh entschieden werden, wie Punkte im Programm repräsentiert werden sollen. Dabei liegt es scheinbar nahe, Punkte durch ihre kartesischen Koordinaten $(x, y) \in \mathbb{R}^2$ zu repräsentieren. Ausgehend davon könnten Geraden durch die Parameter m und n ihrer kartesischen Normalform $y = m\,x + n$ oder die Parameter a,b und c ihrer allgemeinen Gleichung

$$a\,x + b\,y + c = 0 \tag{2.1}$$

dargestellt werden. Dieser Ansatz ist jedoch nicht nur unflexibel, sondern auch fehleranfällig, da eine Reihe von Sonderfällen auftreten können, die erkannt und im Programm abgefangen werden müssen.

Was soll beispielsweise mit dem Schnittpunkt paralleler Geraden geschehen? Mit jedem dynamischen Geometrieprogramm ist es möglich, zwei sich schneidende Geraden und deren Schnittpunkt zu konstruieren und die Lage der Geraden danach so zu verändern, dass sie parallel sind.

Seien die beiden Geraden durch ihre allgemeinen Gleichungen

$$a_1\,x + b_1\,y + c_1 = 0$$

und

$$a_2\,x + b_2\,y + c_2 = 0$$

gegeben. Dann haben sie den Schnittpunkt

$$\left(\frac{b_1\,c_2 - c_1\,b_2}{a_1\,b_2 - a_2\,b_1}, -\frac{a_1\,c_2 - a_2\,c_1}{a_1\,b_2 - a_2\,b_1}\right).$$

Sind die Geraden parallel, so ist $a_1\,b_2 - a_2\,b_1 = 0$, der Schnittpunkt ist also erwartungsgemäß nicht definiert. Was soll in diesem Fall mit geometrischen Gebilden passieren, deren Lage von den Koordinaten dieses Schnittpunkts abhängt? Seine Koordinaten können jedenfalls nicht zu Berechnungen herangezogen werden. Eine naheliegende Möglichkeit wäre es, sowohl den Schnittpunkt als auch alle davon abhängigen geometrischen Gebilde als „ungültig“

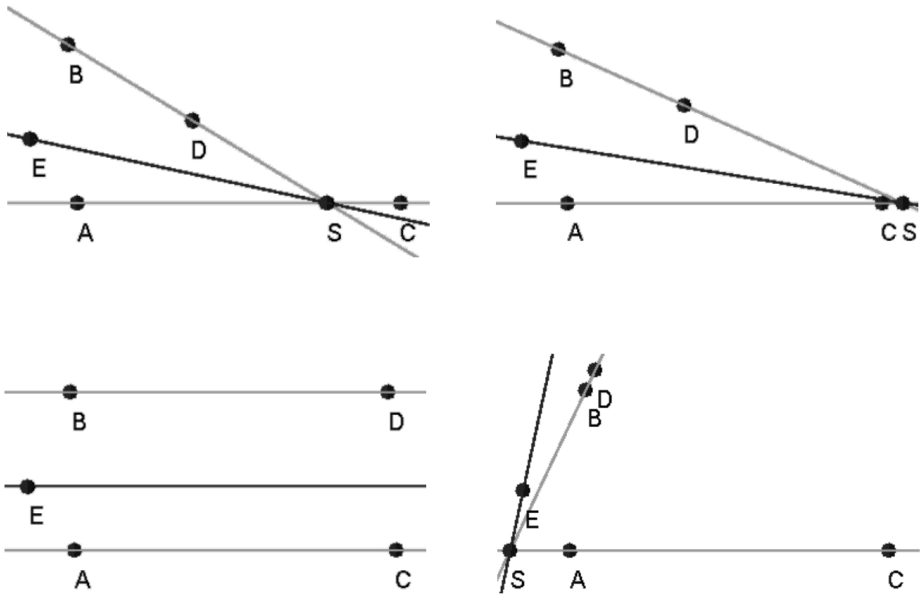

Abbildung 2.1: Die Punkte A und C sowie B und D definieren jeweils eine Gerade eindeutig. S ist der Schnittpunkt beider Geraden (oben links). Bewegt man D nun nach oben, so bewegt sich S nach rechts und verschwindet von der Zeichenfläche (oben rechts). Nachdem die Parallelsituation (unten links) überwunden ist, taucht S von der anderen Seite her wieder auf der Zeichenfläche auf (unten rechts). Was aber soll mit der durch S und E verlaufenden Geraden geschehen, wenn die Geraden AC und BD parallel sind?

zu markieren, solange die Geraden parallel sind. Das würde jedoch bedeuten, dass gegebenenfalls große Teile der Konstruktion auf einmal einfach verschwinden (und hoffentlich wieder auftauchen, sobald die Situation es zulässt). Darüber hinaus können nicht definierte geometrische Gebilde zu Stetigkeitsproblemen führen (siehe Abschnitt 4.2).

Ändert sich die Lage zweier Geraden AC und BD so, dass sie nahezu parallel sind, wird mit zunehmend kleinerem eingeschlossenen Winkel ihr Schnittpunkt S früher oder später den auf der Zeichenfläche dargestellten Bereich der Konstruktion verlassen. S entfernt sich dann immer weiter vom dargestellten Konstruktionsausschnitt. Verläuft eine weitere Gerade ES durch den Schnittpunkt S und einen weiteren Punkt E, werden auch die Winkel, die ES mit AC und BD einschließt, immer kleiner. Sind AC und BD schließlich parallel, kann man annehmen, dass ihr Schnittpunkt S „im Unendlichen verschwunden“ ist. In diesem Fall ist ES parallel zu AC und BD (siehe Abbildung 2.1). Verwendet man kartesische Koordinaten, lässt sich dieses Verhalten nur umständlich durch die Behandlung von Sonderfällen ausdrücken.

Die Lösung besteht darin, keine euklidische, sondern eine projektive Geometrie zu verwenden [21]. Im Folgenden stelle ich die reelle projektive Geometrie, die in Geofuchs verwendet wird, aus analytischer Sicht vor. Ein Punkt wird dabei durch einen von $(0, 0, 0)$ verschiedenen Vektor (x, y, h), seine homogenen oder projektiven Koordinaten, dargestellt. Dabei entsprechen die kartesischen Koordinaten (x, y) den homogenen Koordinaten $(x, y, 1)$. Die homogenen Koordinaten (x, y, h) mit $h \neq 0$ entsprechen den kartesischen Koordinaten $(\frac{x}{h}, \frac{y}{h})$. Die Vektoren $(\lambda\, x, \lambda\, y, \lambda\, h)$ mit $\lambda \neq 0$ stellen den gleichen Punkt dar. Daher schreibt man auch $(x : y : h)$.

Die Abbildung

$$(x, y, h) \mapsto \left(\frac{x}{h}, \frac{y}{h}\right), \tag{2.2}$$

welche homogene auf kartesische Punktkoordinaten abbildet, ist für $h = 0$ nicht definiert. Punkte der Form $(x, y, 0)$ haben daher keine Entsprechung in kartesischen Koordinaten und der euklidischen Geometrie. Diese Punkte nennt man Punkte im Unendlichen oder uneigentliche Punkte. Alle anderen Punkte nennt man eigentliche Punkte.

Ausgehend von der allgemeinen Geradengleichung 2.1 und der Abbildung 2.2 ergibt sich die Gleichung $a\,\frac{x}{h} + b\,\frac{y}{h} + c = 0$. Multipliziert man diese mit h, erhält man die homogene Geradengleichung

$$a\,x + b\,y + c\,h = 0. \tag{2.3}$$

Das Tupel (a, b, c) nennt man homogene oder Plücker'sche Geradenkoordinaten [10] . Ebenso wie bei den homogenen Punktkoordinaten stellen die homogenen Geradenkoordinaten $(\lambda\, a, \lambda\, b, \lambda\, c)$ für jedes $\lambda \neq 0$ die gleiche Gerade dar. Der Vektor $(0, 0, 0)$ entspricht keiner Geraden. Der Punkt (x, y, h) liegt genau dann auf der Geraden (a, b, c), wenn Gleichung 2.3 erfüllt ist. Auf der Geraden $(0, 0, 1)$ liegen genau die Punkte, bei denen $h = 0$ ist, also die uneigentlichen Punkte. Diese Gerade nennt man die uneigentliche Gerade oder Ferngerade. Alle anderen Geraden heißen eigentliche Geraden. Auf jeder eigentlichen Geraden liegt genau ein uneigentlicher Punkt, ihr Fernpunkt.

Soll der Schnittpunkt zweier Geraden (a_1, b_1, c_1) und (a_2, b_2, c_2) berechnet werden, ist das folgende lineare Gleichungssystem zu lösen, wobei nur die nichttrivialen Lösungen von Interesse sind:

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix} \begin{pmatrix} x \\ y \\ h \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Die Lösungsmenge ist

$$\mathcal{L} = \left\{ (x, y, h)^T \,\middle|\, x = \frac{h\,(b_1\,c_2 - c_1\,b_2)}{a_1\,b_2 - a_2\,b_1}, y = -\frac{(a_1\,c_2 - a_2\,c_1)\,h}{a_1\,b_2 - a_2\,b_1} \right\}.$$

Da die homogenen Koordinaten von Punkten und Geraden nur bis auf einen Faktor eindeutig bestimmt sind, kann h beliebig, aber ungleich Null gewählt werden. Mit $h = a_1\,b_2 - a_2\,b_1$ ergibt sich der Schnittpunkt

$$(b_1\,c_2 - c_1\,b_2, -a_1\,c_2 + a_2\,c_1, a_1\,b_2 - a_2\,b_1). \tag{2.4}$$

In der projektiven Geometrie schneiden sich zwei verschiedene Geraden immer in genau einem Punkt. Sind die Geraden parallel, so liegt ihr Schnittpunkt auf der Ferngeraden.

An dieser Stelle möchte ich auf die Beziehung der Punkt- und Geradenkoordinaten zum Skalar- und Vektorprodukt hinweisen. Gerd Fischer gibt in [9] folgende Definition für das kanonische Skalarprodukt der Vektoren $x = (x_1, x_2, x_3)$ und $y = (y_1, y_2, y_3)$ im $\mathbb{R}^3$ an:

$$\langle x, y \rangle = x_1\,y_1 + x_2\,y_2 + x_3\,y_3. \tag{2.5}$$

Das Vektorprodukt ist

$$x \times y = (x_2\,y_3 - x_3\,y_2, x_3\,y_1 - x_1\,y_3, x_1\,y_2 - x_2\,y_1). \tag{2.6}$$

Dabei gelten die Gleichungen

$$\langle x \times y, x \rangle = 0 \tag{2.7}$$

und

$$\langle x \times y, y \rangle = 0. \tag{2.8}$$

Den Term $\langle x \times y, z \rangle$ bezeichnet man übrigens als das Spatprodukt von x, y und z. Des weiteren heißen zwei Vektoren genau dann orthogonal, wenn ihr Skalarprodukt gleich Null ist. An den Gleichungen 2.7 und 2.8 kann man leicht erkennen, dass der Schnittpunkt zweier durch die Vektoren x und y gegebener Geraden der durch den Vektor $x \times y$ dargestellte Punkt ist.

Für ein dynamisches Geometrieprogramm auf projektiver Grundlage ist es notwendig, die homogenen Koordinaten (a, b, c) der durch zwei gegebene Punkte (x_1, y_1, h_1) und (x_2, y_2, h_2) verlaufenden Geraden berechnen zu können. Dazu ist das folgende lineare Gleichungssystem zu lösen:

$$\begin{pmatrix} x_1 & y_1 & h_1 \\ x_2 & y_2 & h_2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Die Lösungsmenge ist

$$\mathcal{L} = \left\{ (a, b, c)^T \,\middle|\, a = \frac{c\,(y_1\,h_2 - h_1\,y_2)}{x_1\,y_2 - x_2\,y_1}, b = -\frac{c\,(x_1\,h_2 - x_2\,h_1)}{x_1\,y_2 - x_2\,y_1} \right\}.$$

Dabei kann c beliebig, aber ungleich Null gewählt werden. Mit $c = x_1\,y_2 - x_2\,y_1$ ergeben sich die homogenen Geradenkoordinaten

$$(y_1\,h_2 - h_1\,y_2, -x_1\,h_2 + x_2\,h_1, x_1\,y_2 - x_2\,y_1), \tag{2.9}$$

oder einfacher formuliert $(x_1, y_1, h_1) \times (x_2, y_2, h_2)$. Harold S. M. Coxeter beschreibt in [5] das Prinzip der Dualität. Demnach bleibt ein Satz der projektiven Geometrie auch dann gültig, wenn die Begriffe „Punkt“ und „Gerade“ sowie damit assoziierte Begriffe wie „kollinear“ und „konkurrent“ vertauscht werden. Bezogen auf dynamische Geometriesoftware bedeutet das beispielsweise, dass nur eine Methode notwendig ist, um sowohl den Schnittpunkt zweier gegebener Geraden als auch eine Gerade, die durch zwei gegebene Punkte verläuft, zu berechnen. Innerhalb dieser müssen keinerlei Sonderfälle abgefangen werden, da zwei verschiedene Punkte stets genau eine Gerade definieren und zwei verschiedene Geraden immer genau einen Schnittpunkt haben.

Wie bereits erwähnt, schneiden sich parallele Geraden in ihrem Fernpunkt. Möchte man zu einer gegebenen Geraden (a, b, c) die Koordinaten einer Parallele bestimmen, muss zunächst der gemeinsame Fernpunkt bestimmt werden. Dazu ist die Gleichung

$$a\,x + b\,y + c\,0 = a\,x + b\,y = 0$$

zu lösen. Diese wird von dem Punkt $(b, -a, 0)$ erfüllt. Das selbe Ergebnis liefert $(a_1, b_1, c_1) \times (0, 0, 1)$, also der Schnitt der gegebenen Geraden mit der Ferngeraden. Ist außerdem noch ein Punkt (x_p, y_p, h_p) auf der Parallelen gegeben, so hat die Parallele die Koordinaten

$$(a\,h_p, b\,h_p, -b\,y_p - a\,x_p) = ((a, b, c) \times (0, 0, 1)) \times (x_p, y_p, h_p). \tag{2.10}$$

In der zweidimensionalen reellen euklidischen Ebene ist ein Richtungsvektor $v = (v_1, v_2)$ der durch zwei Punkte $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ definierten Geraden durch

$$v = P_1 - P_2 = (x_1 - x_2, y_1 - y_2) \tag{2.11}$$

gegeben. Der Richtungsvektor der durch die beiden Punkte mit den homogenen Koordinaten (x_1, y_1, h_1) und (x_2, y_2, h_2) definierten Geraden ist dementsprechend

$$v_h = \left(\frac{x_1}{h_1} - \frac{x_2}{h_2}, \frac{y_1}{h_1} - \frac{y_2}{h_2}, 1\right).$$

Eine Multiplikation mit $h_1\,h_2$ liefert

$$v_h = (x_1\,h_2 - x_2\,h_1, y_1\,h_2 - y_2\,h_1, h_1\,h_2). \tag{2.12}$$

Angenommen, genau einer der gegebenen Punkte ist uneigentlich, also z. B. $h_1 = 0$. Dann ist

$$v_h = (x_1\,h_2, y_1\,h_2, 0) = (x_1, y_1, 0). \tag{2.13}$$

Ist also $(x_\infty, y_\infty, 0)$ der Fernpunkt einer eigentlichen Geraden, so ist (x_∞, y_∞) ein Richtungsvektor ihrer Entsprechung in der euklidischen Ebene. Für eine Gerade (a, b, c) ist der Richtungsvektor dementsprechend $(a, b, c) \times (0, 0, 1) = (b, -a, 0)$. Für einen dazu orthogonalen Vektor (x_o, y_o, h_o) gilt $\langle (b, -a, 0), (x_o, y_o, h_o) \rangle = 0$. Der Vektor $(a, b, 0)$ erfüllt diese Bedingung und stellt damit den uneigentlichen Punkt aller zur Geraden (a, b, c) orthogonalen Geraden dar. Ist also zu einer gegebenen Geraden $g = (a, b, c)$ und einem gegebenen Punkt $P = (x_p, y_p, h_p)$ eine Gerade gesucht, die senkrecht auf g steht und durch P verläuft, so hat diese die Koordinaten

$$(a, b, 0) \times (x, y, h) = (-h_p\,b, a\,h_p, b\,x_p - a\,y_p). \tag{2.14}$$

Der Mittelpunkt zweier Punkte kann ebenfalls einfach bestimmt werden. Ausgangspunkt ist wieder die Berechnungsvorschrift für den Mittelpunkt

$$M = (x_m, y_m) = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

der in kartesischen Koordinaten gegebenen Punkte $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$. Für den Mittelpunkt $M = (x_m, y_m, h_m)$ der Punkte $P_1 = (x_1, y_1, h_1)$ und $P_2 = (x_1, y_1, h_1)$ ergibt sich daraus mit der Funktion 2.2

$$M = (x_m, y_m, h_m) = \left(\frac{x_1\, h_2 + x_2\, h_1}{2\, h_1\, h_2}, \frac{y_1\, h_2 + y_2\, h_1}{2\, h_1\, h_2}, 1 \right).$$

Eine Multiplikation mit $2\, h_1\, h_2$ liefert

$$M = (x_1\, h_2 + x_2\, h_1, y_1\, h_2 + y_2\, h_1, 2\, h_1\, h_2). \tag{2.15}$$

2.2 Kreise

Alle mir bekannten dynamischen Geometrieprogramme bieten ein Werkzeug an, mit dem aus zwei gegebenen Punkten, dem Mittelpunkt und einem Punkt auf der Peripherie, ein Kreis konstruiert werden kann. In der reellen projektiven Geometrie gibt es streng genommen keine Metrik (und auch keine Winkel), so dass auch die Einführung von Kreisen über die Wechselbeziehungen zwischen homogenen und kartesischen Koordinaten erfolgen muss. Ausgangspunkt ist die kartesische Gleichung für einen Kreis mit Mittelpunkt (x_0, y_0) und dem Radius r

$$(x - x_0)^2 + (y - y_0)^2 - r^2 = 0. \tag{2.16}$$

Mit den Parametern

$$c_1 = 1, c_2 = -2\, x_0, c_3 = -2\, y_0, c_4 = x_0^2 + y_0^2 - r^2 \tag{2.17}$$

ergibt sich nach [12] die Gleichung

$$c_1\, (x^2 + y^2) + c_2\, x + c_3\, y + c_4 = 0. \tag{2.18}$$

Dabei tritt im Parameter c_4 noch der Square-Radius r^2 auf. Für einen gegebenen Punkt (x_p, y_p) auf der Peripherie des Kreises gilt nach Gleichung 2.16

$$(x_p - x_0)^2 + (y_p - y_0)^2 = r^2. \tag{2.19}$$

Damit ist der Parameter c_4

$$c_4 = x_0^2 + y_0^2 - r^2 = x_0^2 + y_0^2 - ((x_p - x_0)^2 + (y_p - y_0)^2) = -{x_p}^2 - {y_p}^2 + 2\, x_p\, x_0 + 2\, y_p\, y_0. \tag{2.20}$$

Aus der Kreisgleichung 2.18 und Funktion 2.2 lässt sich die homogene Kreisgleichung

$$c_1 \left(\left(\frac{x}{h} \right)^2 + \left(\frac{y}{h} \right)^2 \right) + c_2\, \frac{x}{h} + c_3\, \frac{y}{h} + c_4 = 0$$

bestimmen. Eine Multiplikation mit h^2 ergibt

$$c_1\, (x^2 + y^2) + c_2\, x\, h + c_3\, y\, h + c_4\, h^2 = 0. \tag{2.21}$$

Für einen Kreis mit dem homogenen Mittelpunkt (x_0, y_0, h_0) und dem homogenen Punkt (x_p, y_p, h_p) auf der Peripherie ergeben sich aus den Gleichungen 2.17 und 2.20 und der Abbildung 2.2 die homogenen Parameter

$$c_1 = 1, c_2 = -2\,\frac{x_0}{h_0}, c_3 = -2\,\frac{y_0}{h_0}, c_4 = -\left(\frac{x_p}{h_p}\right)^2 - \left(\frac{y_p}{h_p}\right)^2 + 2\,\frac{x_p}{h_p}\,\frac{x_0}{h_0} + 2\,\frac{y_p}{h_p}\,\frac{y_0}{h_0}.$$

Eine Multiplikation mit $h_p^2\,h_0$ liefert schließlich

$$c_1 = h_p^2\,h_0, c_2 = -2\,x_0\,h_p^2, c_3 = -2\,y_0\,h_p^2, c_4 = -x_p^2\,h_0 - y_p^2\,h_0 + 2\,x_0\,x_p\,h_p + 2\,y_0\,y_p\,h_p. \quad (2.22)$$

Ein Kreis ist also durch seine vier Parameter (c_1, c_2, c_3, c_4) der Gleichung 2.21 eindeutig bestimmt, jedoch repräsentieren wieder die Parameter $(\lambda\,c_1, \lambda\,c_2, \lambda\,c_3, \lambda\,c_4)$ mit $\lambda \neq 0$ den gleichen Kreis. Für einen Kreis mit uneigentlichem Mittelpunkt und eigentlichem Punkt auf der Peripherie ist $c_1 = 0$ und der Kreis entartet zu dem Geradenpaar $(0, 0, 1)$ und (c_2, c_3, c_4). Diese Geraden fallen zusammen, falls $c_1 = c_2 = c_3 = 0$ ist. Die Parameter $(0, 0, 0, c_4)$ mit $c_4 \neq 0$ entsprechen also der Ferngeraden.

Um die reellen Schnittpunkte eines Kreises (c_1, c_2, c_3, c_4) mit einer Geraden (a, b, c) zu bestimmen, ist das nichtlineare Gleichungssystem

$$c_1\,(x^2 + y^2) + c_2\,x\,h + c_3\,y\,h + c_4\,h^2 = 0$$

$$a\,x + b\,y + c\,h = 0$$

zu lösen. Falls $a \neq 0$ und $c_1 \neq 0$, ergeben sich die beiden Schnittpunkte

$$\left(-c_2\,ab^2 + bc_3\,a^2 - b\,\sqrt{k} - 2\,cc_1\,a^2, -\left(-c_2\,ab + 2\,cbc_1 + c_3\,a^2 - \sqrt{k}\right)a, 2\,a\,c_1\,b^2 + 2\,c_1\,a^3\right)$$

und

$$\left(-c_2\,ab^2 + bc_3\,a^2 + b\,\sqrt{k} - 2\,cc_1\,a^2, -\left(-c_2\,ab + 2\,cbc_1 + c_3\,a^2 + \sqrt{k}\right)a, 2\,a\,c_1\,b^2 + 2\,c_1\,a^3\right)$$

mit

$$k = a^2\left({c_2}^2b^2 - 2\,c_2\,abc_3 + 4\,cbc_1\,c_3 + {c_3}^2a^2 - 4\,c_1\,b^2c_4 - 4\,{c_1}^2c^2 + 4\,c_1\,ac_2\,c - 4\,c_1\,a^2c_4\right).$$

Für $k > 0$ gibt es zwei reelle Schnittpunkte, ist $k = 0$, fallen diese zusammen, und für $k < 0$ gibt es keine reellen Schnittpunkte. Für $a = 0$ und $c_1 \neq 0$ sind die Schnittpunkte

$$\left(-bc_2 + \sqrt{{c_2}^2b^2 - 4\,{c_1}^2c^2 + 4\,cbc_1\,c_3 - 4\,c_1\,b^2c_4}, -2\,c_1\,c, 2\,b\,c_1\right)$$

und

$$\left(-bc_2 - \sqrt{{c_2}^2b^2 - 4\,{c_1}^2c^2 + 4\,cbc_1\,c_3 - 4\,c_1\,b^2c_4}, -2\,c_1\,c, 2\,b\,c_1\right). \quad (2.23)$$

Für $c_1 = 0$ sind die Schnittpunkte der Geraden (a, b, c) mit der Geraden (c_2, c_3, c_4) und der Ferngeraden zu bestimmen.

Die Schnittpunkte zweier Kreise kann man ebenfalls durch das Lösen eines nichtlinearen Gleichungssystems bestimmen. Dabei treten jedoch numerische Probleme auf. Daher möchte zeigen, wie die Bestimmung der Schnittpunkte zweier Kreise durch die Bestimmung

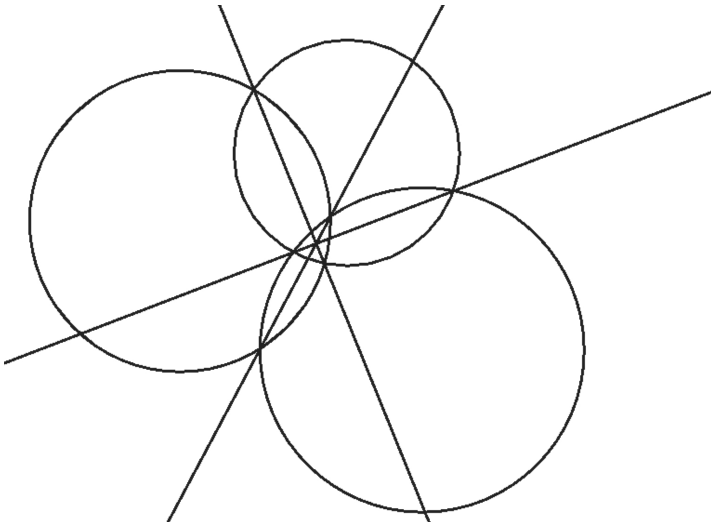

Abbildung 2.2: Drei Kreise haben paarweise drei Potenzgeraden. Diese treffen sich in einem Punkt, dem Potenzmittelpunkt.

der Schnittpunkte eines Kreises und einer Gerade konstruktiv und analytisch gelöst werden kann.

Zwei Kreise mit verschiedenen Mittelpunkten bestimmen eine Gerade, ihre Potenzgerade, eindeutig. Die Potenzgerade verläuft durch die Schnittpunkte beider Kreise und steht senkrecht auf der Verbindungsgerade der Mittelpunkte beider Kreise [18]. Des weiteren schneiden sich die drei Potenzgeraden dreier Kreise, deren Mittelpunkte nicht kollinear sind, in einem eigentlichen Punkt, dem Potenzmittelpunkt (siehe Abbildung 2.2). Andernfalls sind sie parallel.

Soll nun der Schnittpunkt zweier Kreise C_1 und C_2 mit den Mittelpunkten M_1 und M_2 und den Punkten P_1 und P_2 auf der Peripherie konstruiert werden, so bestimmt man zunächst den Mittelpunkt M_3 von P_1 und P_2. Man kann sich nun einen dritten Kreis C_3 mit dem Mittelpunkt M_3, auf dessen Peripherie sowohl P_1 als auch P_2 liegen, vorstellen. Man konstruiert die Verbindungsgerade von M_1 und M_3 und eine dazu orthogonale Gerade, die durch P_1 verläuft. Diese Gerade ist die Potenzgerade $g_{1,3}$ von C_1 und C_3. Auf die gleiche Art und Weise kann man nun die Potenzgerade $g_{2,3}$ von C_2 und C_3 konstruieren. Der Schnittpunkt S von $g_{1,3}$ und $g_{2,3}$ ist der Potenzmittelpunkt der Kreise C_1, C_2 und C_3. Nun konstruiert man die Verbindungsgerade M_1M_2 von M_1 und M_2 sowie die Potenzgerade $g_{1,2}$, die orthogonal zu M_1M_2 ist und durch S verläuft. Die Schnittpunkte von $g_{1,2}$ mit entweder C_1 oder C_2 sind die Schnittpunkte der Kreise C_1 und C_2 (siehe Abbildung 2.3).

Die beschriebene Konstruktion der Potenzgerade zweier Kreise setzt jedoch voraus, dass die Mittelpunkte M_1 und M_2 der Kreise und der Mittelpunkt M_3 der für die Konstruktion gewählten Punkte auf der Peripherie nicht kollinear sind. Andernfalls liegt der Potenzmittelpunkt auf der Ferngeraden. In einem dynamischen Geometrieprogramm wäre dieser Sonderfall abzufangen. Daher empfiehlt sich die rein analytische Bestimmung der Potenzgerade zweier Kreise (für eine ausführlichere Darstellung und den Begriff der Potenz siehe z. B. [12] oder [18]).

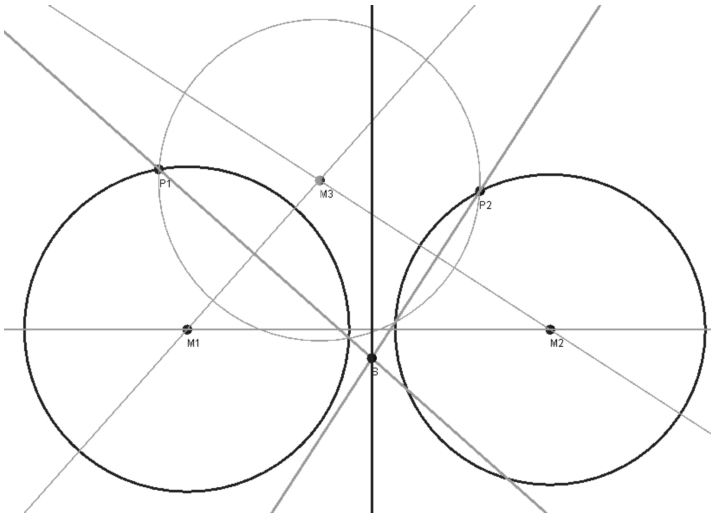

Abbildung 2.3: Die Potenzgerade zweier Kreise C_1 und C_2 kann auch konstruiert werden, wenn die Kreise keine (reellen) Schnittpunkte haben. Dazu wird ein dritter Kreis konstruiert, der C_1 und C_2 schneidet. Dessen Potenzgeraden mit C_1 und C_2 schneiden sich im Potenzmittelpunkt, durch den auch die Potenzgerade von C_1 und C_2 verläuft. Diese steht senkrecht auf der Verbindungsgeraden der Mittelpunkte von C_1 und C_2.

Seien $C_1 = (c_{11}, c_{12}, c_{13}, c_{14})$ und $C_2 = (c_{21}, c_{22}, c_{23}, c_{24})$ Kreise. Die Potenzgerade g von C_1 und C_2 ist dann

$$g = C_1 - C_2 = (c_{11} - c_{21})\,(x^2 + y^2) + (c_{12} - c_{22})\,x\,h + (c_{13} - c_{23})\,y\,h + (c_{14} - c_{24})\,h^2.$$

Um eine Gerade zu erhalten, muss $c_{11} - c_{21} = 0$ sein. Daher multipliziert man C_1 mit c_{21} und C_2 mit c_{11}. Dann ist $g = c_{21}\,C_1 - c_{11}\,C_2$, also

$$g = (c_{21}\,c_{12} - c_{11}\,c_{22}, c_{21}\,c_{13} - c_{11}\,c_{23}, c_{21}\,c_{14} - c_{11}\,c_{24}). \tag{2.24}$$

2.3 Die Mathematik von Cinderella

Die mathematischen Grundlagen von Cinderella, die Ulrich Kortenkamp in [19] beschreibt, sind im Vergleich zur vorgestellten reellen projektiven Geometrie leistungsfähiger, aber auch viel komplexer. In diesem Abschnitt möchte ich auf die wichtigsten Unterschiede hinweisen.

Mit einem dynamischen Geometrieprogramm kann man beispielsweise die Schnittpunkte zweier sich schneidender Kreise konstruieren und die Konstruktion im Nachhinein so verändern, dass die Kreise sich nicht mehr schneiden. In diesem Fall sind die Koordinaten der Schnittpunkte komplex. Verwendet man reelle Koordinaten, müssen die Schnittpunkte als ungültig markiert und können nicht mehr gezeichnet werden. Da man ihre Koordinaten nicht mehr zu weiteren Berechnungen heranziehen kann, müssen alle geometrischen Gebilde, deren Lage von den Koordinaten der Schnittpunkte abhängt, ebenfalls als ungültig markiert werden. Die abhängigen Gebilde müssen aber nicht zwangsläufig auch komplex sein.

Beispielsweise sind die Koordinaten der durch die beiden Schnittpunkte verlaufenden Gerade, der Potenzgerade der beiden Kreise, immer reell, auch wenn die Schnittpunkte selbst nicht reell sind. Außerdem führen „ungültige Koordinaten“ zu Problemen bei der Verwendung distanz-basierter Algorithmen zum Vermeiden von Sprüngen (siehe Abschnitt 4.2 für eine zufriedenstellende Lösung dieses Problems). Cinderella verwendet daher komplexe homogene Koordinaten. Damit sind die Koordinaten von Punkten grundsätzlich definiert. Nur nicht-reelle geometrische Gebilde werden nicht gezeichnet, aber deren komplexe Koordinaten und Parameter können unabhängig davon für Berechnungen verwendet werden. Außerdem wären ohne komplexe Koordinaten weder der von Cinderella zur Vermeidung von Sprüngen verwendete „Complex-Tracing“-Algorithmus noch die verwendete Definition von Längen und Winkeln möglich.

In der projektiven Geometrie sind Längen und Winkel, da sie unter projektiven Abbildungen nicht erhalten bleiben, nicht definiert. In den Abschnitten 2.1 und 2.2 habe ich beschrieben, wie darauf aufbauende Konzepte wie Orthogonalität und Kreise durch die Wechselbeziehungen zur euklidischen Geometrie beschrieben werden können. Cinderella geht diesbezüglich einen anderen Weg, basierend auf den sogenannten Cayley-Klein-Geometrien. Dabei definiert man Längen und Winkel in Bezug auf einen Kegelschnitt, das Fundamentalgebilde. Je nachdem, welchen Kegelschnitt man als Fundamentalgebilde wählt, ergeben sich verschiedene (u. a. euklidische, hyperbolische und elliptische) Geometrien, die im Programm gleich behandelt werden können.

Außerdem unterstützt Cinderella auch Kegelschnitte. Kreise werden als Spezialfälle von Kegelschnitten betrachtet.

Kapitel 3

Begriffsbildung

3.1 Grundlegende Begriffe

Bisher habe ich Begriffe wie „geometrisches Gebilde“ und „Konstruktion“ verwendet, ohne genau geklärt zu haben, was darunter zu verstehen ist. In der darstellenden und analytischen Geometrie können diese Begriffe als Grundbegriffe angesehen werden. Da sich statische und dynamische Geometrie in einigen grundlegenden Aspekten unterscheiden, halte ich es für notwendig, die zur Beschreibung dynamischer Geometriesoftware verwendete Terminologie zu präzisieren und zu erweitern. Zunächst möchte ich zeigen, wie geometrische Gebilde in einem dynamischen Geometrieprogramm repräsentiert werden können. Dabei möchte ich mich auf die Betrachtung von Punkten, Geraden und Kreisen beschränken. In diesem Sinne ist ein geometrisches Gebilde entweder ein Punkt, eine Gerade oder ein Kreis in der Ebene $\mathcal{E}$. In der analytischen Geometrie stellt man geometrische Gebilde durch Zahlen, ihre Koordinaten, dar (siehe Kapitel 2). Da ein Computer nur mit Zahlen umgehen kann, bietet sich diese Art der Darstellung auch für dynamische Geometriesoftware an. In einem dynamischen Geometrieprogramm kann der Benutzer zumindest freie Punkte sowie Kreis- und Geradengleiter bewegen, was die Bewegung anderer geometrischer Gebilde zur Folge haben kann. Punkte, Geraden und Kreise in einem dynamischen Geometrieprogramm können daher zu verschiedenen Zeitpunkten verschiedene Punkte, Geraden oder Kreise der Ebene $\mathcal{E}$ repräsentieren. Im Folgenden gehe ich davon aus, dass sowohl Punkte, Geraden als auch Kreise durch geeignete Koordinaten repräsentiert werden. Eine solche Repräsentation bezeichne ich als geometrisches Objekt.

Definition 1 (Klassen geometrischer Objekte) *Sei o ein Datenobjekt in einem dynamischen Geometrieprogramm.*

- *Falls o einen Punkt repräsentiert, dann ist o eine Instanz der Klasse* `Punkt`*.*
- *Falls o eine Gerade repräsentiert, dann ist o eine Instanz der Klasse* `Gerade`*.*
- *Falls o einen Kreis repräsentiert, dann ist o eine Instanz der Klasse* `Kreis`*.*

Definition 2 (Allgemeines geometrisches Objekt) *Ein allgemeines geometrisches Objekt (AGO) ist eine Instanz genau einer der Klassen* `Punkt`*,* `Gerade` *oder* `Kreis`*. Ist o eine Instanz von* `Punkt`*, dann heißt o allgemeiner Punkt. Ist o eine Instanz von* `Gerade`*, dann heißt o allgemeine Gerade. Ist o eine Instanz von* `Kreis`*, dann heißt o allgemeiner Kreis.*

Der Zustand eines AGOs ist durch die Werte seiner Attribute gegeben. Ein AGO in einem dynamischen Geometrieprogramm hat u. a. Attribute, deren Werte, normalerweise Gleitkommazahlen, seine Lage beschreiben. Das können z. B. homogene Koordinaten, wie in Kapitel 2 beschrieben, sein. Andere mögliche Attribute, z. B. die Farbe der grafischen Repräsentation eines AGOs, vernachlässige ich an dieser Stelle. Ein AGO in einem bestimmten Zustand bezeichne ich als spezielles geometrisches Objekt (SGO). Ein AGO ist somit eine Variable (im Sinne der Theorie der Programmiersprachen), ein SGO ein Wert. Analog zu allgemeinen Punkten, Geraden und Kreisen verwende ich für SGOs der entsprechenden Typen die Begriffe „spezieller Punkt“, „spezielle Gerade“ und „spezieller Kreis“. Zwei SGOs sind gleich, wenn die Werte der Attribute, die ihre Lage bestimmen, gleich sind. Das heißt aber nicht, dass sie identisch im Sinne der Objektidentität sind.

Definition 3 (Konstruktionswerkzeug) *Seien* $t_1, \ldots, t_n, t_a \in \{\texttt{Punkt}, \texttt{Gerade}, \texttt{Kreis}\}$. *Dann ist ein Konstruktionswerkzeug eine Abbildung* $w \in (t_1 \times \ldots \times t_n) \to t_a$. *Die Typen* $t_1, \ldots, t_n$ *heißen erwartete Typen von* w, t_a *heißt erzeugter Typ von* w.

Ein Konstruktionswerkzeug[1] bildet AGOs der erwarteten Typen auf ein AGO vom erzeugten Typ ab, z. B. zwei verschiedene allgemeine Punkte auf die allgemeine Gerade, die durch beide allgemeinen Punkte verläuft. Eine allgemeine Gerade `g` verläuft durch zwei allgemeine Punkte `P1` und `P2`, falls für alle speziellen Punkte `P1` und `P2` die spezielle Gerade `G` durch `P1` und `P2` verläuft. Ein Konstruktionswerkzeug bestimmt somit das Verhalten des erzeugten AGOs. Die Abbildung zweier allgemeiner Punkte auf die durch sie verlaufende allgemeine Gerade ist allerdings nur möglich, wenn die beiden allgemeinen Punkte nicht identisch sind. Ein Konstruktionswerkzeug ist daher eine partielle Funktion. Für jedes Konstruktionswerkzeug muss angegeben werden, welche Bedingungen die AGOs, die als Eingabe dienen sollen, erfüllen müssen, damit diese Funktion definiert ist. Das geschieht durch eine sogenannte Nichtdegenerationsbedingung. An dieser Stelle möchte ich einige konkrete Konstruktionswerkzeuge als Beispiel vorstellen.

- `FPunkt` $\in () \to$ `Punkt`, wobei `FPunkt()` ein freier Punkt ist
- `PPGerade` $\in$ (`Punkt` $\times$ `Punkt`) $\to$ `Gerade`, wobei `PPGerade(P1,P2)` die durch die Punkte `P1` und `P2` verlaufende Gerade ist (Nichtdegenerationsbedingung: `P1` und `P2` sind nicht identisch)
- `PPKreis` $\in$ (`Punkt` $\times$ `Punkt`) $\to$ `Kreis`, wobei `PPKreis(P1,P2)` der Kreis mit dem Mittelpunkt `P1` und dem Punkt `P2` auf der Peripherie ist (Nichtdegenerationsbedingung: `P1` und `P2` sind nicht identisch)
- `GGSchnittpunkt` $\in$ (`Gerade` $\times$ `Gerade`) $\to$ `Punkt`, wobei `GGSchnittpunkt(G1,G2)` der Schnittpunkt der Geraden `G1` und `G2` ist (Nichtdegenerationsbedingung: `G1` und `G2` sind nicht identisch)

[1] Der Benutzer kann in einem dynamischen Geometrieprogramm nicht nur AGOs konstruieren, sondern beispielsweise auch deren grafische Darstellung beeinflussen. Dazu stellt das dynamische Geometrieprogramm ihm Werkzeuge zur Verfügung. Ich werde diesen für die funktionale Beschreibung von Software gebräuchlichen Begriff weiterhin in seiner bekannten Bedeutung verwenden. Konstruktionswerkzeuge sind spezielle Werkzeuge. Sofern sich Werkzeuge auf AGOs anwenden lassen, ist es sinnvoll, analog zu Konstruktionswerkzeugen von „erwarteten Typen“ zu sprechen.

Definition 4 (Unmittelbar abhängige und unmittelbar definierende AGOs) *Seien $d_1, \ldots d_n$ AGOs der Typen $t_1, \ldots, t_n$, $w \in (t_1 \times \ldots \times t_n) \to t_a$ ein Konstruktionswerkzeug und $a = w(d_1, \ldots, d_n)$. Dann heißt a unmittelbar abhängig von $d_1, \ldots, d_n$ und $d_1, \ldots, d_n$ heißen unmittelbar definierende AGOs von a.*

Definition 5 (Abhängige AGOs) *Sei d ein AGO. Dann sind ausschließlich die folgenden AGOs abhängig von d:*

1. *Jedes unmittelbar von d abhängige AGO ist auch abhängig von d.*
2. *Ist ein AGO a abhängig von d, dann ist auch jedes von a abhängige AGO abhängig von d.*

Definition 6 (Definierende AGOs) *Sei a ein AGO. Dann sind ausschließlich die folgenden AGOs definierende AGOs von a:*

1. *Jedes unmittelbar definierende AGO von a ist auch ein definierendes AGO von a.*
2. *Ist d ein definierendes AGO von a, dann ist auch jedes definierende AGO von d ein definierendes AGO von a.*

Anschaulich ist ein AGO a vom AGO d genau dann abhängig, wenn eine Änderung der Lage von d eine Änderung der Lage von a zur Folge haben kann. In diesem (und nur in diesem) Fall ist d ein definierendes AGO von a.

Definition 7 (Freie, halbfreie und feste AGOs) *Sei o ein AGO. Dann heißt o frei, falls o keine definierenden AGOs hat. Falls die Lage von o eindeutig durch die Lage der definierenden AGOs von o festgelegt ist, heißt o fest. Ist o weder fest noch frei, dann ist o halbfrei.*

Das Verhalten eines AGOs a ist durch eine Methode gegeben, die aus den Werten der Attribute seiner definiernden AGOs, die deren Lage bestimmen und, falls a frei oder halbfrei ist, den Werten weiterer Parameter die Werte der Attribute, die die Lage von a bestimmen, berechnet. Eine solche Methode bezeichne ich als Algorithmus. Algorithmen beschreiben somit die Beziehung zwischen einem AGO und seinen definierenden AGOs. AGOs, die mit dem gleichen Konstruktionswerkzeug konstruiert wurden, haben gleiche Algorithmen. Eine Bewegung eines AGOs entspricht einer Änderung des Zustands des AGOs.

Definition 8 (Auswahlfunktion) *Sei $\mathcal{K} = \{o_1, \ldots, o_m\}$ eine Menge von AGOs. Seien außerdem $t_1, \ldots, t_n$ und t_a Klassen und $w \in (t_1 \times \ldots \times t_n) \to t_a$ ein Konstruktionswerkzeug. Dann heißt die Funktion $f : \{1, \ldots, n\} \to \{1, \ldots, m\}$ Auswahlfunktion aus $\mathcal{K}$ für w, falls $w(o_{f(1)}, \ldots, o_{f(n)})$ definiert ist und für alle $i \in \{1, \ldots, n\}$ gilt: $o_{f(i)} \in t_i$.*

Definition 9 (Anwendbarkeit eines Konstruktionswerkzeugs) *Ein Konstruktionswerkzeug w heißt genau dann anwendbar auf eine Menge $\mathcal{K} = \{o_1, \ldots, o_n\}$ und das Tupel $K = (o_1, \ldots, o_n)$ von AGOs, wenn es eine Auswahlfunktion aus $\mathcal{K}$ für w gibt.*

Anschaulich ist ein Konstruktionswerkzeug genau dann auf eine Menge von AGOs anwendbar, wenn es in dieser Menge genug AGOs gibt, die als „Eingabe“ für das Konstruktionswerkzeug dienen können. Es wäre im Übrigen grundsätzlich denkbar, nur Injektionen als Auswahlfunktionen zuzulassen. Allerdings gibt es dynamische Geometrieprogramme, die

beispielsweise ein Konstruktionswerkzeug `Senkrechte` $\in$ (`Punkt` $\times$ `Punkt` $\times$ `Punkt`) $\rightarrow$ `Gerade` zur Verfügung stellen, wobei `Senkrechte(P1,P2,P3)` die allgemeine Gerade ist, die durch `P1` verläuft und senkrecht zur durch `P2` und `P3` verlaufenden Gerade ist. Fordert man von einer Auswahlfunktion Injektivität, dann könnte mit diesem Konstruktionswerkzeug nicht die durch einen allgemeinen Punkt `P1` verlaufende Gerade, die senkrecht zur durch `P1` und einen zweiten Punkt `P2` verlaufenden Gerade ist, konstruiert werden. Es ist daher flexibler, die Injektivität der Auswahlfunktion nicht grundsätzlich, sondern in den Nichtdegenerationsbedingungen der Konstruktionswerkzeuge zu fordern.

Definition 10 (Allgemeine Konstruktion) *Ausschließlich folgende Tupel sind allgemeine Konstruktionen:*

1. *Die leere Menge ist eine allgemeine Konstruktion, die leere Konstruktion.*
2. *Seien $o_1, \ldots, o_m$ AGOs und $K = (o_1, \ldots, o_m)$ eine allgemeine Konstruktion. Sei außerdem $\mathcal{K} = \{o_1, \ldots, o_m\}$, $w \in (t_1 \times \ldots \times t_n) \rightarrow t_a$ ein auf $\mathcal{K}$ anwendbares Konstruktionswerkzeug und f eine Auswahlfunktion aus $\mathcal{K}$ für w. Dann ist auch $K' = (o_1, \ldots, o_m, w(o_{f(1)}, \ldots, o_{f(n)}))$ eine allgemeine Konstruktion.*

Definition 11 (Enthaltene AGOs) *Sei $K = (o_1, \ldots, o_n)$ eine allgemeine Konstruktion. Dann enthält K die AGOs $o_1, \ldots, o_n$.*

Definition 12 (Hinzufügen von AGOs) *Seien $K = (o_1, \ldots, o_m)$ und $K' = (o_1, \ldots, o_m, o_{m+1})$ allgemeine Konstruktionen. Dann entsteht K' aus K durch Hinzufügen des AGOs o_{m+1}.*

Ein SGO ist ein AGO mit festgelegtem Zustand, also zu einem bestimmten Zeitpunkt. Eine spezielle Konstruktion ist demnach ein Tupel von SGOs. In einem dynamischen Geometrieprogramm finden alle Aktionen des Benutzers zu einem bestimmten Zeitpunkt statt. Fügt der Benutzer einer allgemeinen Konstruktion einen allgemeinen Punkt hinzu, so haben die allgemeine Konstruktion und der allgemeine Punkt zum Zeitpunkt des Hinzufügens einen bestimmten Zustand. Somit hat der Benutzer auch einer speziellen Konstruktion einen speziellen Punkt hinzugefügt. Eine grafische Darstellung kann streng genommen nur für SGOs erfolgen. Clickt der Benutzer eines dynamischen Geometrieprogramms mit der Maus auf die grafische Repräsentation eines SGOs, so hat er damit auch ein AGO ausgewählt (unter Umständen kann der Mausclick sogar zu einer Zustandsänderung des AGOs führen). Dennoch ist diese Unterscheidung notwendig, um dynamische Geometriesoftware möglichst exakt beschreiben zu können. Drei allgemeine freie Punkte M, P_1 und P_2 können vom Benutzer so bewegt werden, dass zu einem konkreten Zeitpunkt der spezielle Punkt M von den speziellen Punkten P_1 und P_2 den gleichen Abstand hat, also im geometrischen Sinn ihr Mittelpunkt ist. Andererseits könnte M kein freier, sondern ein fester, von P_1 und P_2 abhängiger allgemeiner Punkt sein, der Aufgrund seines Algorithmus in jeder speziellen Konstruktion der geometrische Mittelpunkt von P_1 und P_2 ist. In diesem Fall ist M der allgemeine Mittelpunkt der allgemeinen Punkte P_1 und P_2. Zu jedem konkreten Zeitpunkt ist M dann der spezielle Mittelpunkt der speziellen Punkte P_1 und P_2.

Zur Verbesserung der Lesbarkeit werde ich im Folgenden häufig einfach Begriffe wie Konstruktion, Punkt, Gerade, Mittelpunkt und Schnittpunkt verwenden. Sofern nicht explizit anders angegeben, ist damit stets eine allgemeine Konstruktion, ein allgemeiner Punkt, eine allgemeine Gerade usw. gemeint.

Der Benutzer eines dynamischen Geometrieprogramms beginnt mit einer leeren Konstruktion. Auf diese ist nur das Konstruktionswerkzeug `FPunkt` anwendbar. Der Benutzer muss also zunächst einige freie Punkte konstruieren. Dazu wählt er das Konstruktionswerkzeug `FPunkt` aus und clickt dann mit der Maus auf eine Zeichenfläche, die einen Ausschnitt der Ebene $\mathcal{E}$ grafisch darstellt. Die Darstellung auf der Zeichenfläche beruht auf einem Koordinatensystem, dem Zeichenkoordinatensystem. Die Speicherung der AGOs und alle Berechnungen erfolgen in einem anderen Koordinatensystem, dem Weltkoordinatensystem. Die Transformation zwischen Zeichen- und Weltkoordinaten, normalerweise eine kombinierte Translation und Skalierung, kann vom Benutzer manipuliert werden. Daher kann eine Zeichenfläche zu verschiedenen Zeitpunkten verschiedene Ausschnitte der Ebene $\mathcal{E}$ darstellen. Mit einem Mausclick auf der Zeichenfläche gibt der Benutzer Zeichenkoordinaten an, die dann in Weltkoordinaten transformiert werden. Ist das Konstruktionswerkzeug `FPunkt` ausgewählt, wird der Konstruktion ein freier Punkt hinzugefügt. Dessen Lage hängt ausschließlich von den vom Benutzer durch den Mausclick angegebenen Zeichenkoordinaten und der Transformation von Zeichen- in Weltkoordinaten ab. Zur grafischen Darstellung einer speziellen Konstruktion werden die Weltkoordinaten der enthaltenen SGOs wieder in Zeichenkoordinaten transformiert.

Hat der Benutzer mindestens zwei Punkte konstruiert, dann ist das Konstruktionswerkzeug `PPGerade` anwendbar. Wählt der Benutzer dieses aus, erwartet das dynamische Geometrieprogramm die Auswahl zweier Punkte. Hat er diese Punkte durch Mausclicks auf der Zeichenfläche ausgewählt, dann wird die durch beide Punkte verlaufende Gerade der Konstruktion hinzugefügt. Aus Sicht des Benutzers besteht dieser Konstruktionsschritt aus drei Teilschritten: Der Auswahl des Konstruktionswerkzeugs, der Auswahl des ersten und der Auswahl des zweiten definierenden AGOs. Erwartet ein Konstruktionswerkzeug einen Punkt und eine Gerade, dann sind seine erwarteten Typen `Punkt` und `Gerade`. Nach der Auswahl eines Punktes erwartet das dynamische Geometrieprogramm dann nur noch die Auswahl einer Gerade. Mit der Auswahl einer Gerade wird das neue AGO der Konstruktion hinzugefügt. Damit ist dieser Konstruktionsschritt abgeschlossen.

Der Benutzer kann die beiden freien Punkte so bewegen, dass sie zu einem bestimmten Zeitpunkt gleiche spezielle Punkte sind. Dann definieren diese beiden speziellen Punkte aber keine spezielle Gerade. Für das Werkzeug `PPGerade` forderte die Nichtdegenerationsbedingung, dass die beiden als Eingabe dienenden allgemeinen Punkte nicht identisch sein dürfen. Das impliziert auch, dass die spezielle Gerade `PPGerade(P1, P2)` nicht definiert ist, falls die speziellen Punkte `P1` und `P2` zu einem konkreten Zeitpunkt gleich sind. Falls ein SGO aufgrund der Zustände seiner definierenden AGOs nicht definiert ist, dann bezeichnet man es als degeneriert. Eine spezielle Konstruktion, die mindestens ein degeneriertes SGO enthält, bezeichnet man ebenfalls als degeneriert.

3.2 Geometric Straight Line Programs

Ulrich Kortenkamp stellt in seiner Dissertation [19] mit den sogenannten Geometric Straight Line Programs (GSPs) eine intuitiv verständliche Notation für allgemeine Konstruktionen vor. Eine kurze Einführung in GSPs findet sich zudem in [23]. Demnach ist ein GSP eine Folge von Anweisungen. Solche Anweisungen sind beispielsweise (ich habe gegenüber [23] geringfügige syntaktische Änderungen vorgenommen):

- `L=Join(P1,P2);` (die Gerade `L` verläuft durch die Punkte `P1` und `P2`)

- `P=Meet(L1,L2);` (der Punkt `P` ist der Schnittpunkt der Geraden `P1` und `P2`)
- `C=Circle(M,P);` (der Kreis `K` hat den Punkt `M` als Mittelpunkt und den Punkt `P` auf der Peripherie)
- `P=FreePoint();` (der Punkt `P` ist frei)
- `P=IntersectionCL(C,L);` (der Punkt `P` ist der Schnittpunkt des Kreises `C` und der Gerade `L`)
- `P=IntersectionCC(C1,C2);` (der Punkt `P` ist der Schnittpunkt der Kreise `C1` und `C2`)

In jeder Anweisung eines GSPs wird einem AGO ein Bezeichner zugeordnet, über den in anderen Anweisungen auf das AGO verwiesen werden kann. Jede Anweisung gibt zu einem AGO a die unmittelbar definierenden AGOs $d_1, \ldots, d_n$ und die Beziehung an, in denen a zu $d_1, \ldots, d_n$ steht. Allerdings sind `Join`, `Meet`, `Circle` usw. keine Funktionen, sondern Relationen. Die Relation `Join` ist beispielsweise formal wie folgt definiert (siehe [19]):

$$\texttt{Join} := \{(p_1, p_2, l) |\ l \textit{ ist die Gerade durch } p_1 \textit{ und } p_2 \textit{ und } p_1 \neq p_2\}.$$

Demnach ist eine Anweisung in einem GSP als Aussage über die Beziehungen von AGOs zu verstehen. Eine Anweisung in einem GSP enthält implizit sowohl den Typ der AGOs, über die eine Aussage gemacht wird, als auch eine Nichtdegenerationsbedingung. Ich halte es für angebracht, jeder Anweisung den Typ des abhängigen AGOs voranzustellen. Beispielsweise schreibe ich `Point P=Meet(L1,L2)` anstelle von `P=Meet(L1,L2)`. Die Typen `Punkt`, `Gerade` und `Kreis` bezeichne ich in GSPs als `Point`, `Line` und `Circle`. Eine Konstruktion, die ausschließlich die Punkte `P1`, `P2`, `P3` und `P4`, die durch jeweils zwei dieser Punkte verlaufenden Geraden `g1` und `g2` sowie deren Schnittpunkt `S` enthält, kann als GSP wie folgt dargestellt werden:

```
Point P1=FreePoint();
Point P2=FreePoint();
Line  g1=Join(P1,P2);
Point P3=FreePoint();
Point P4=FreePoint();
Line  g2=Join(P3,P4);
Point S =Meet(g1,g2);
```

3.3 Abhängigkeitsgraphen

Dieser Abschnitt setzt die Kenntnis grundlegender Begriffe der Graphentheorie voraus. Ich verwende die in [6] beschriebene Terminologie.

Definition 13 (Abhängigkeitsgraph) *Den Abhängigkeitsgraphen einer allgemeinen Konstruktion K bezeichne ich mit $\gamma(K)$.*

1. *Der Abhängigkeitsgraph der leeren Konstruktion ist der leere Graph.*

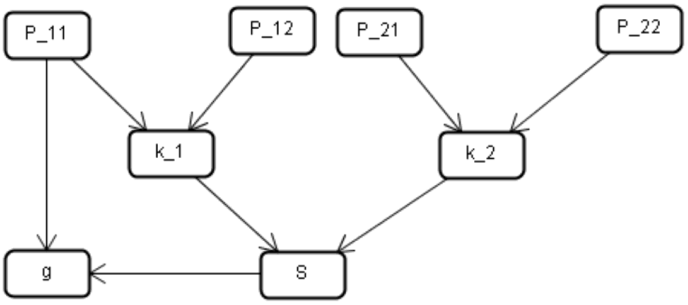

Abbildung 3.1: Die Beziehungen der in einer Konstruktion enthaltenen AGOs bilden einen gerichteten azyklischen Graphen.

2. *Sei $K = (o_1, \ldots, o_n)$ eine allgemeine Konstruktion und $(\mathcal{V}, \mathcal{E})$ der Abhängigkeitsgraph von K. Sei auch $K' = (o_1, \ldots, o_n, a)$ eine allgemeine Konstruktion und seien $d_1, \ldots, d_n$ die unmittelbar definierenden AGOs von a. Dann ist $(\mathcal{V} \cup \{a\}, \mathcal{E} \cup \{(d_1, a), \ldots, (d_n, a)\})$ der Abhängigkeitsgraph von K'.*

Die Knoten des Abhängigkeitsgraphen $\gamma(K)$ einer allgemeinen Konstruktion K sind also die in K enthaltenen AGOs. Für jedes in K enthaltene AGO a gibt es jeweils genau eine Kante von jedem definierenden AGO von a zu a. Eine Kante da ist also als „d ist ein definierendes AGO von a“ zu interpretieren. Abbildung 3.1 zeigt den Abhängigkeitsgraphen der allgemeinen Konstruktion

```
Point  P_11:=FreePoint();
Point  P_12:=FreePoint();
Point  P_21:=FreePoint();
Point  P_22:=FreePoint();
Circle k_1 :=Circle(P_11, P_12);
Circle k_2 :=Circle(P_21, P_22);
Point  S   :=IntersectionCC(k_1, k_2);
Line   g   :=Join(S, P_11);
```

Aus der Definition des Begriffes der allgemeinen Konstruktion folgt, dass die Abhängigkeitsgraphen von Konstruktionen in jedem Fall azyklisch sind. Allerdings gibt es Konstruktionen, deren Abhängigkeitsgraph nicht zusammenhängend ist. Eine solche Konstruktion ist z. B. jede Konstruktion, die ausschließlich aus mehreren freien Punkten besteht. In jedem Abhängigkeitsgraphen gibt es Knoten mit Eingangsgrad Null und Knoten mit Ausgangsgrad Null. Erstere Knoten sind AGOs, von denen keine anderen AGOs abhängig sind. Zweitere sind AGOs, die keine definierenden AGOs haben, also freie AGOs.

Kapitel 4

Ausgewählte Algorithmen

4.1 Algorithmen für den Abhängigkeitsgraphen

Geofuchs erlaubt es dem Benutzer, die Beziehungen von AGOs im Nachhinein zu verändern. So ist es z. B. möglich, einen freien Punkt zum Schnittpunkt zweier Geraden zu machen. Dabei darf keine der Geraden von dem Punkt abhängen. Kein AGO darf durch ein davon abhängiges AGO oder sich selbst definiert werden. Hat der Benutzer ein AGO o, dessen Beziehungen zu anderen AGOs geändert werden soll, ausgewählt, müssen daher sowohl o als auch alle davon abhängigen AGOs vorübergehend „ausgeblendet“ werden, so dass es dem Benutzer nicht möglich ist, sie als definierende AGOs zu wählen. Dazu ist zunächst der Abhängigkeitsgraph der Konstruktion zu bestimmen. Aus Definition 13 (Seite 24) ist direkt ersichtlich, wie das geschehen kann.

Algorithmus: Den Abhängigkeitsgraphen einer Konstruktion bestimmen

Sei $K = (o_1, \ldots, o_n)$ eine Konstruktion. Die definierenden AGOs jedes AGOs o_i seien $d_{i,1}, \ldots, d_{i,f(i)}$. Dieser Algorithmus berechnet den Abhängigkeitsgraphen $\gamma(K) = (\mathcal{V}, \mathcal{E})$ von K.

1. $\mathcal{V} = \emptyset$, $\mathcal{E} = \emptyset$
2. Für alle $i = 1, \ldots, n$:
   (a) $\mathcal{V} = \mathcal{V} \cup \{o_i\}$
   (b) Für alle $j = 1, \ldots, f(i) : \mathcal{E} = \mathcal{E} \cup \{(d_{i,j}, o_i)\}$

Ausgehend vom leeren Graphen werden die AGOs sequentiell in der Reihenfolge, in der sie der allgemeinen Konstruktion hinzugefügt wurden, durchlaufen. Für jedes AGO o_i werden dem Abhängigkeitsgraphen ein Knoten o_i und Kanten von allen definierenden AGOs von o_i zu o_i hinzugefügt.

Ist das AGO a vom AGO d abhängig, dann ist a im Abhängigkeitsgraphen von d aus erreichbar. Zum Bestimmen der von einem AGO d abhängigen AGOs müssen also alle von d aus erreichbaren Knoten bestimmt werden. Wie genau das geschehen kann, ist von der gewählten programminternen Repräsentation des Graphen abhängig. Mark Allen Weiss beschreibt in [26] zwei grundsätzliche Möglichkeiten für die Repräsentation von Graphen: Die Speicherung als Adjazenzmatrix und die die Speicherung in Form einer Adjazenzliste. In Geofuchs kommt

die zweite Variante zum Einsatz, zu jedem Knoten wird eine Liste der auf einem Weg der Länge Eins erreichbaren Knoten gespeichert. Die von einem Knoten aus erreichbaren Knoten können dann einfach rekursiv bestimmt werden.

Einige andere grundlegende Graphenalgorithmen, wie sie z. B. in [1] und [22] beschrieben werden, können in dynamischen Geometrieprogrammen gewinnbringend eingesetzt werden. Soll beispielsweise aus einer Menge $\{o_1, \ldots, o_n\}$ von AGOs eine Konstruktion, die genau die AGOs $o_1, \ldots, o_n$ enthält, bestimmt werden, so ist zunächst der Abhängigkeitsgraph von $(o_1, \ldots, o_n)$ wie oben beschrieben zu bestimmen[1]. Dabei ist $(o_1, \ldots, o_n)$ ein Tupel von AGOs, aber nicht zwangsläufig eine Konstruktion. Zur Lösung dieses Problems ist eine topologische Sortierung für den Abhängigkeitsgraphen zu finden. Existiert keine solche Sortierung, dann ist der Abhängigkeitsgraph zyklisch, andernfalls gibt sie eine Reihenfolge an, in der die AGOs $o_1, \ldots, o_n$ eine allgemeine Konstruktion bilden. Auf diese Art und Weise kann z. B. die Stabilität eines dynamischen Geometrieprogramms beim Laden allgemeiner und spezieller Konstruktionen aus Dateien (siehe Kapitel 7) erhöht werden.

4.2 Vermeiden von Sprüngen

Wie bereits in Abschnitt 1.3 erwähnt, treten in vielen dynamischen Geometrieprogrammen Situationen auf, in denen eine kleine, kontinuierliche Bewegung eines AGOs einen großen „Sprung“ eines davon abhängigen AGOs auslöst (Abbildung 4.1 zeigt ein mit „Zirkel und Lineal“ konstruiertes Beispiel). Harald Winroth beschreibt in [27] das Problem wie folgt: Eine allgemeine Konstruktion kann man als Menge von AGOs und Gleichungen, die die Beziehungen zwischen den AGOs darstellen, auffassen[2]. Ein solches System von Gleichungen hat nicht zwangsläufig eine eindeutige Lösung. Für einen Geradengleiter gibt es unendlich viele mögliche Lösungen, und für den Schnittpunkt zweier sich schneidender Kreise gibt es zwei (gegebenenfalls gleiche) Lösungen. Wenn in einem solchen Fall eine Neuberechnung der Position des Gleiters bzw. des Schnittpunkts notwendig ist, weil sich die Lage eines definierenden AGOs geändert hat, sind die gegebenen Bedingungen nicht ausreichend, um die neue Position des Gleiters bzw. Schnittpunkts eindeutig zu bestimmen. Um ein erwartungskonformes Verhalten der AGOs zu erzwingen, ist es daher notwendig, weitere Bedingungen anzugeben. Ich möchte einen Algorithmus beschreiben, der Sprünge von Schnittpunkten in den meisten Fällen vermeiden kann.

Ulrich Kortenkamp zeigt in [19], dass der Zustand eines AGOs nicht ausschließlich vom Zustand seiner definierenden AGOs abhängen darf. Man stelle sich die in Abbildung 4.2 gezeigte Situation vor. Bewegt man jetzt den Punkt A, ändert sich der Winkel zwischen AB und BC um einen Wert δ. Der Winkel zwischen AB und BD ändert sich um $\frac{1}{2}\,\delta$, während sich der Winkel zwischen AB und BE nur um $\frac{1}{4}\,\delta$ ändert. Rotiert man den Punkt A einmal um den speziellen Punkt B, bis er wieder seine Ausgangslage erreicht hat, kann die Gerade BE ihre Ausgangslage nicht ohne Sprung wieder erreichen. Das Problem in diesem Fall ist, dass zwei spezielle Geraden zwei verschiedene Winkel einschließen und das Programm eine der Winkelhalbierenden auswählen muss. Konstruiert man die Winkelhalbierenden wie mit Zirkel und Lineal, ist es notwendig, die Schnittpunkte von Kreisen und Geraden zu bestimmen.

[1] Streng genommen ist der vorgestellte Algorithmus nur für Konstruktionen definiert, kann aber in ähnlicher Form auch für Tupel von AGOs verwendet werden.

[2] In der hier verwendeten Terminologie sind die Gleichungen implizit in den Algorithmen der AGOs enthalten.

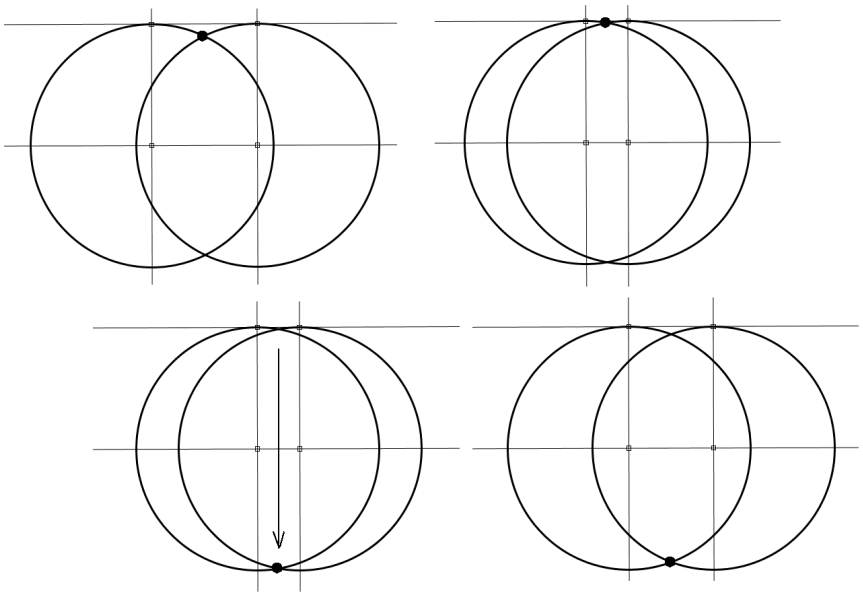

Abbildung 4.1: Die Mittelpunkte der beiden Kreise K_1 (links) und K_2 sind Gleiter auf einer Geraden. Außerdem haben K_1 und K_2 den gleichen Radius (oben links). Die Konstruktion enthält zusätzlich einen der beiden Schnittpunkte von K_1 und K_2. Verschiebt man K_1 etwas nach rechts (oben rechts), wird der Schnittpunkt erwartungsgemäß mitbewegt. Sobald aber K_1 K_2 „passiert“, springt der Schnittpunkt an die Position, an der sich der andere Schnittpunkt von K_1 und K_2 befände (unten links).

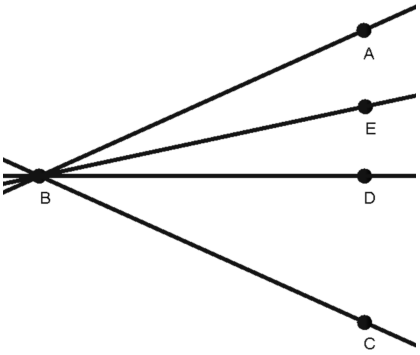

Abbildung 4.2: Die Gerade BD ist eine Winkelhalbierende der Geraden AB und BC. Die Gerade BE ist eine Winkelhalbierende von AB und BD.

Auch ein spezieller Kreis und eine spezielle Gerade können zwei verschiedene Schnittpunkte haben, in diesem Fall tritt also prinzipiell das gleiche Problem auf. Um aus einer Menge möglicher Folgezustände für ein AGO den erwartungskonformen auszuwählen, muss deshalb die Vorgeschichte der Konstruktion betrachtet werden.

Ulrich Kortenkamp beschreibt in seiner Dissertation ein Verfahren namens „Complex Tracing“. Dieses Verfahren beruht darauf, aus mehreren möglichen Folgezuständen für ein AGO denjenigen auszuwählen, mit dem die geringste Lageänderung einhergeht. Für die beiden Schnittpunkte eines Kreises und einer Gerade bedeutet das, dass sie bei Bewegungen der Gerade oder des Kreises jeweils an die Position bewegt werden, die ihrer alten Position am nächsten ist. Für Geraden können solche Nähe-Beziehungen z. B. über die Richtung realisiert werden. Auf diese Art und Weise kann es offensichtlich zu Fehlentscheidungen kommen, z. B. wenn die beiden Schnittpunkte zu einem Zeitpunkt gleich sind und dann weiterbewegt werden. Eine solche Situation bezeichnet man als Singularität. Beim „Complex Tracing“ wird die Bewegung eines Punktes von A nach B durch einen Parameter dargestellt. Anhand dieses Parameters werden „Zwischenschritte“ berechnet, mit denen Singularitäten zuverlässig umgangen werden können.

Markus Hohenwarter beschreibt in [16] den in GeoGebra verwendeten, einfacheren und dennoch aus meiner Sicht ausreichend leistungsfähigen Algorithmus zum Vermeiden von Sprüngen von Schnittpunkten. Dieser Algorithmus kommt in geringfügig geänderter Form auch in Geofuchs zum Einsatz. Für n spezielle Punkte und n mögliche neue Positionen gibt es $n!$ verschiedene Möglichkeiten, die speziellen Punkte auf die neuen Positionen abzubilden. Da GeoGebra nicht mit komplexen Zahlen rechnet, sind die speziellen Punkte nicht in jedem Fall definiert. Deshalb werden die letzten definierten Positionen aller Punkte gespeichert und für die Nähe-Beziehung verwendet. Dann werden die Abstände der zuletzt definierten Positionen der Punkte zu den möglichen neuen Positionen berechnet und ein von der Anzahl der Neuberechnungen, seit denen der entsprechende Punkt undefiniert ist, abhängiger Summand dazuaddiert. Auf diese Weise werden definierte spezielle Punkte stets undefinierten vorgezogen. Dann werden die Abstandssummen aller $n!$ Abbildungen von speziellen Punkten auf neue Positionen berechnet. Da in Geofuchs immer $n <= 2$ und in GeoGebra (das auch Kegelschnitte unterstützt) $n <= 4$ ist, ist der fakultative Aufwand vertretbar. Für größere n ist es erwägenswert, nicht alle Abstandssummen zu berechnen. Stattdessen kann man einen bipartiten gewichteten Graphen konstruieren, für den dann ein minimales Matching bestimmt wird. Algorithmen zum Lösen solcher Zuordnungsprobleme werden beispielsweise in [1] und [22] beschrieben. Für kleines n wiegt der entstehende Overhead jedoch deutlich schwerer als die geringere Komplexität. Die Abbildung mit der kleinsten Abstandssumme wird schließlich verwendet, um jedem speziellen Punkt eine neue Position zuzuordnen. Dieses Verfahren funktioniert gemäß [16] immer, wenn die definierenden Punkte der sich schneidenden AGOs sich kontinuierlich bewegen. Allerdings ist die Bewegung von Punkten in einem dynamischen Geometrieprogramm niemals kontinuierlich im mathematischen Sinn, da der auf dem Computer darstellbare diskrete Zahlbereich eine endliche Menge möglicher Positionen vorgibt. Eine kontinuierliche Bewegung ist daher in diesem Zusammenhang eine Bewegung, bei der der Benutzer keine Sprünge wahrnehmen kann. In Geofuchs kommt folgender Algorithmus zum Einsatz:

Algorithmus: Abbildung von Schnittpunkten auf neue Positionen

Seien $I_1, \ldots, I_n$ die Schnittpunkte, $P_1^{alt}, \ldots, P_n^{alt}$ ihre zuletzt definierten Positio-

nen und a_k die Anzahl der Durchläufe, seit denen I_k nicht definiert ist. $\langle P_1, P_2 \rangle$ bezeichne den euklidischen Abstand der speziellen Punkte P_1 und P_2.

1. Bestimme die neuen Positionen $P_1^{neu}, \ldots, P_n^{neu}$ der Schnittpunkte
2. Für alle $i = 1 \ldots n$: Für alle $j = 1 \ldots n$: $d_{j,i} = \langle P_j^{alt}, P_i^{neu} \rangle + a_i^2$, falls P_i^{neu} definiert ist, und sonst $d_{j,i} = -1$
3. Bestimme $d_{max} = \max(d_{j,i})$
4. Für alle $i = 1 \ldots n$: Für alle $j = 1 \ldots n$: Falls $d_{j,i} = -1$, setze $d_{j,i} = d_{max} + 1$.
5. Bestimme eine Abbildung $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$, so dass $\sum_{i=1}^{n} \sqrt{d_{i,f(i)}}$ minimal ist
6. Für alle $k = 1 \ldots n$:
   (a) Bewege den Punkt I_k an die Position $P_{f(k)}^{neu}$
   (b) $a_k = a_k + 1$
   (c) Falls $P_{f(k)}^{neu}$ definiert ist, setze $a(k) = 0$ und $P_k^{alt} = P_{f(k)}^{neu}$

Dieser Algorithmus unterscheidet sich von dem von Markus Hohenwarter angegebenen nur im fünften Schritt. In Geofuchs verwende ich nicht die Abbildung mit der kleinsten Abstandssumme, sondern die Abbildung, bei der die Summe der Quadratwurzeln der Abstände minimal ist, als Zuordnungsfunktion der Schnittpunkte zu ihren neuen Positionen. Verwendet man für die Abstandsberechnung den euklidischen Abstand, betrachtet uneigentliche spezielle Punkte für den Algorithmus als nicht definiert und verwendet die Abbildung mit der kleinsten Abstandssumme, gibt es nämlich ein Problem: Wenn ein Punkt wie in Abbildung 2.1 (Seite 11) den sichtbaren Bereich verläßt, im Unendlichen verschwindet und von der anderen Seite her wieder auftaucht, „springt“ er von seiner letzten definierten zur nächsten definierten Position. Diese liegt aber am entgegengesetzten Ende des darstellbaren Weltkoordinatenbereichs. Das führt beispielsweise bei der Konstruktion

```
Point  P_1:=FreePoint();
Point  P_2:=FreePoint();
Line   g_1:=Join(P_1,P_2);
Point  P_3:=FreePoint();
Point  P_4:=FreePoint();
Line   g_2:=Join(P_3,P_4);
Point  P_5:=FreePoint();
Point  P_6:=Meet(g_1,g_2);
Circle k  :=Circle(P_6, P_5);
Point  P_7:=FreePoint();
Point  P_8:=FreePoint();
Line   g_3:=Join(P_7,P_8);
Point  S  :=IntersectionCL(k,g_3);
```

zu Problemen. Verschwindet der Schnittpunkt `P_6` der Geraden `g_1` und `g_2` im Unendlichen, dann entartet der Kreis `k` zu einem Geradenpaar (im mathematischen Sinn). In dieser Situation ist einer der Schnittpunkte von `k` und `g_3` uneigentlich (also für den Algorithmus

nicht definiert). Wird der Parallelzustand „überschritten“, trifft der beschriebene Algorithmus häufig nicht die erwartungskonforme Entscheidung: Der Schnittpunkt S springt.

Dieses Problem kann wie oben erwähnt vermieden werden, wenn man statt der Abbildung mit minimaler Abstandssumme die Abbildung mit minimaler Summe der Quadratwurzeln der Abstände als Zuordnung der Schnittpunkte zu ihren neuen Positionen verwendet. Anschaulich fallen dabei durch die mit zunehmend größerem Funktionswert schwächer ansteigende Wurzelfunktion sehr große Abstände weniger ins Gewicht. In der aktuellen Version von GeoGebra treten in der beschriebenen Situation übrigens ebenfalls keine Sprünge auf.

Kapitel 5

Java und XML

5.1 Java

Als ich mit dem Entwurf von Geofuchs begann, stand bereits fest, dass die Implementierung in Java erfolgen sollte. Java ist eine von Sun entwickelte, objektorientierte Programmiersprache. Für Geofuchs habe ich die aktuelle Version 1.4 verwendet. James Gosling und Henry McGilton stellen im bereits 1996 veröffentlichten White Paper [11] die Entwicklungsziele von Java vor. Darunter finden sich unter anderem Objektorientierung, Einfachheit und Plattformunabhängigkeit. Mark Allen Weiss schreibt in [26]:

> Java offers many benefits, and programmers often view Java as a safer, more portable, and easier-to-use language than C++.

Tatsächlich ist Java auch aus meiner Sicht relativ einfach zu verwenden. So wird beispielsweise jegliche Zeigerarithmetik vermieden, die in C++ für häufige (und schwer zu findende) Fehler sorgt. Außerdem stellt Java eine Vielzahl von Klassen zur Verfügung, die beispielsweise die Programmierung einer grafischen Benutzeroberfläche (einschließlich vorgefertigter Dialogfelder etc.) oder den Zugriff auf Dateien stark vereinfachen. Dadurch ermöglicht es Java, komplexe (und dennoch robuste) Anwendungen in relativ kurzer Zeit zu implementieren. Ulrich Kortenkamp schreibt dazu in [19]:

> The most important „feature“ of Java that is often underestimated is that it is very easy to write Java software... Here we just want to recommend Java for both academic and commercial purposes. The total cost of programming work ... is very low according to our experience.

Die ausschlaggebenden Argumente für Java sind jedoch die Plattformunabhängigkeit und die Internet-Fähigkeiten von Java-Programmen. Java selbst enthält keine plattformspezifischen Sprachbestandteile. Java-Programme werden in eine Zwischensprache, den Java-Bytecode, übersetzt. Dieser wird dann von einer virtuellen Maschine (der „Java Virtual Machine“) interpretiert. Damit sind Java-Programme auf jeder Plattform ausführbar, auf der eine solche virtuelle Maschine zur Verfügung steht. Des weiteren kann man Java-Programme nicht nur als eigenständige Programme auf einem Rechner ausführen, sondern auch in Form sogenannter Applets in HTML-Seiten einbinden. Beim Aufruf der Seite wird das Applet auf den aufrufenden Rechner geladen und dort im Browser durch eine virtuelle Maschine ausgeführt. Geofuchs ist sowohl als eigenständiges Programm als auch als Applet ausführbar. Abgesehen

davon unterstützt Java auch die Entwicklung verteilter Anwendungen auf hoher Ebene. In [19] und [24] gibt Ulrich Kortenkamp einen Überblick über die Möglichkeiten, die die kombinierte Nutzung von dynamischer Geometriesoftware und Internet bietet.

Auch Harald Winroth beschreibt in seiner Dissertation [27] einige dieser Vorteile von Java. Allerdings wurde das von ihm entwickelte dynamische Geometrieprogramm „pdb“ (projective drawing board) in C++ implementiert:

> The main reason was that we anticipated a need for extensive code optimization in order to achieve an acceptable performance for complicated drawings... C++ has been designed to allow for that type of low-level optimization, and we do not believe that a comparable speed-up could have been achieved with a language such as e.g. Java.

Tatsächlich ist die Geschwindigkeit von Java-Programmen nach wie vor problematisch. Allerdings zeigen die im Rahmen dieser Arbeit vorgestellten Programme, dass auch in Java implementierte dynamische Geometrieprogramme durchaus ausreichende Antwortzeiten erreichen können.

5.2 Die Extensible Markup Language (XML)

Unter dem Begriff XML wird häufig eine ganze Familie von Techniken zusammengefasst. Diese basieren auf einer Menge von Regeln für den Aufbau von Datenobjekten, die semistrukturierte Daten enthalten. Diese Regeln sind in der W3C-Empfehlung „Extensible Markup Language (XML) 1.0“ [3] beschrieben. Ein Datenobjekt, das diesen Regeln entspricht, nennt man ein XML-Dokument und wohlgeformt. Die Sprachfamilie XML umfasst viele weitere Standards, für die Entwicklung von Geofuchs waren bisher aber nur die Kernspezifikation XML 1.0 und die Spezifikationen von Programmierschnittstellen (siehe Abschnitt 5.3) von großer Bedeutung. Zwischenzeitlich ist auch der XML-Standard 1.1 [4] veröffentlicht worden, der sich von der Version 1.0 in erster Linie durch weniger strenge Regeln bezüglich des verwendbaren Zeichensatzes unterscheidet. Auf den Internet-Seiten der W3C XML Core Working Group [28] findet sich dazu folgender Hinweis:

> XML 1.1 updates XML so that it no longer depends on the specific Unicode version: you can always use the latest. It also adds checking of normalization, and follows the Unicode line ending rules more closely. You are encouraged to create or generate XML 1.0 documents if you do not need the new features in XML 1.1; XML Parsers are expected to understand both XML 1.0 and XML 1.1.

Als ich mit der Implementierung von Geofuchs begann, war der neue XML-Standard noch keine zwei Monate alt. Zudem war und ist nicht zu erwarten, dass die erweiterten Möglichleiten des neuen Standards für Geofuchs in absehbarer Zeit von Interesse sein werden. Daher verwendet Geofuchs die „alte“ Version 1.0, die sich zum plattformunabhängigen Datenaustausch zwischen verschiedenen Anwendungen etabliert hat und durch eine Vielzahl von Werkzeugen unterstützt wird.

5.3 Verarbeitung von XML-Dokumenten mit Java

Nachdem feststand, dass Geofuchs in Java implementiert werden und XML zur Speicherung von Konstruktionen nutzen sollte, musste ich entscheiden, wie die XML-Dokumente geschrie-

ben und gelesen werden sollten. „Zirkel und Lineal“ speichert Konstruktionen ebenfalls in einem XML-basierten Format. Dazu werden Zeichenketten erzeugt, die dann mittels der in Java zur Verfügung stehenden Klassen zum Zugriff auf das Dateisystem zeilenweise in eine Datei geschrieben werden, so dass die resultierende Datei ein XML-Dokument ist. Diese Variante auf Zeichenebene schied für Geofuchs von vornherein aus, da sie umständlich zu implementieren, fehleranfällig und schwer zu warten ist. Stattdessen wollte ich einen auf höherer Ebene angesiedelten Ansatz verwenden. In Betracht kamen der W3C-Standard DOM (Document Object Model) und die De-facto-Standards SAX (Simple API for XML) und JDOM (Java Document Object Model). Alle drei ermöglichen die Verarbeitung von XML-Dokumenten über ein logisches Modell.

SAX [25] wurde ursprünglich als reine Java-API entwickelt, ist zwischenzeitlich aber auch für andere Programmiersprachen verfügbar. Die Verarbeitung von XML-Dokumenten mit SAX erfolgt ereignisorientiert. Das XML-Dokument wird sequenziell gelesen. Jedesmal, wenn dabei ein Ereignis (wie z. B. ein Start-Tag) auftritt, wird eine bestimmte Methode aufgerufen, die dieses Ereignis verarbeitet. Schon die ersten Entwürfe für das von Geofuchs verwendete Speicherformat GeoXML hatten jedoch eine verschachtelte Struktur, die sequenziell schwierig zu verarbeiten ist. Erweiterungen und sonstige Änderungen wären dementsprechend ebenfalls unnötig kompliziert geworden. Daher kamen nur noch DOM und JDOM ernsthaft in Frage.

DOM ist in mehreren W3C-Empfehlungen, die unter [7] online verfügbar sind, spezifiziert. Ebenso wie SAX definiert DOM eine Menge von Schnittstellen, mit denen auf XML-Dokumente zugegriffen werden kann. DOM verwendet dazu allerdings keinen ereignis-, sondern einen baumorientierten Ansatz. Dabei wird das XML-Dokument im Programm durch einen Baum mit verschiedenen Arten von Knoten repräsentiert, in dem frei navigiert werden kann. Daher ist DOM flexibler als SAX, allerdings muss bei der Verarbeitung eines XML-Dokuments der komplette Baum im Arbeitsspeicher gehalten werden. Da aber mit einem dynamischen Geometrieprogramm erstellte Konstruktionen selten hunderte, tausende oder noch mehr AGOs enthalten, sind diesbezügliche Probleme nicht zu erwarten.

Während DOM unabhängig von einer konkreten Programmiersprache spezifiziert ist, ist JDOM [17] eine an DOM angelehnte API speziell für Java. Dadurch ist JDOM leichter zu verwenden als DOM. Gegen die Verwendung von JDOM spricht jedoch die fehlende formale Spezifikation. Daher habe ich mich entschieden, für die Verarbeitung von XML-Dateien in Geofuchs eine Implementierung des DOM zu verwenden. Unter den verfügbaren Implementierungen fiel die Entscheidung auf das in der Version 1.4 der J2SE (Java 2 Platform, Standard Edition) enthaltene JAXP (Java API for XML Processing).

Kapitel 6

Die grafische Benutzeroberfläche von Geofuchs

Wie bereits in Abschnitt 1.3 erwähnt, ist die grafische Benutzeroberfläche eines dynamischen Geometrieprogramms ausschlaggebend für seine praktische Einsetzbarkeit als Lernprogramm. Auch ein im Umgang mit Computern und dynamischen Geometrieprogrammen unerfahrener Benutzer sollte das Programm ohne lange Einarbeitungszeit verwenden können. Das Programm muss deshalb einfach und intuitiv bedienbar sein. Außerdem sollte der Benutzer beim Konstruieren gemachte Fehler einfach korrigieren können. Ein geübter Benutzer soll möglichst zügig und ohne Unterbrechnungen des Arbeitsflusses Konstruktionen erstellen können. In diesem Kapitel möchte ich darstellen, wie ich versucht habe, diesen Anforderungen mit Geofuchs gerecht zu werden, und die Bedienung von Geofuchs aus Sicht des Benutzers beschreiben.

In dynamischen Geometrieprogrammen werden spezielle Konstruktionen auf sogenannten Zeichenflächen wie auf einem Blatt Papier dargestellt. Der Benutzer sieht die in einer speziellen Konstruktion enthaltenen SGOs als grafische Punkte, Geraden und Kreise, wie sie mit Stift, Lineal und Zirkel gezeichnet werden könnten. Die Bewegung eines AGOs macht sich für den Benutzer dadurch bemerkbar, dass das AGO zu verschiedenen Zeitpunkten durch verschiedene grafische Objekte auf der Zeichenfläche repräsentiert wird. Der Benutzer sieht dabei statt einer Abfolge verschiedener grafischer Objekte jedoch nur ein grafisches Objekt, dass sich bewegt. Aus der Sicht des Benutzers besteht eine Konstruktion ausschließlich aus solchen grafischen Objekten, die bewegt werden können. Daher halte ich es für gerechtfertigt, in diesem Kapitel auf die Unterscheidung von AGOs, SGOs und ihren grafischen Repräsentationen zu verzichten.

6.1 Aufbau

Abbildung 6.1 zeigt den Aufbau des Hauptfensters von Geofuchs. Das Hauptfenster enthält einen Menübalken, einen Statusbalken, mehrere Werkzeugbalken und eine nach Typen und Konstruktionswerkzeugen gegliederte Auflistung von AGOs (bzw. deren Namen[1]). Im Zentrum des Hauptfensters befindet sich Platz für beliebig viele Kindfenster, die ich im Folgenden als Zeichenfenster bezeichne. Die Zeichenfenster enthalten jeweils eine Zeichenfläche, auf der eine Konstruktion grafisch dargestellt wird und bearbeitet werden kann, und einen Werkzeugbalken. In diesem finden sich Schaltflächen, mit denen der durch die Zeichenfläche darge-

[1] In Geofuchs hat jedes AGO einen eindeutigen Namen.

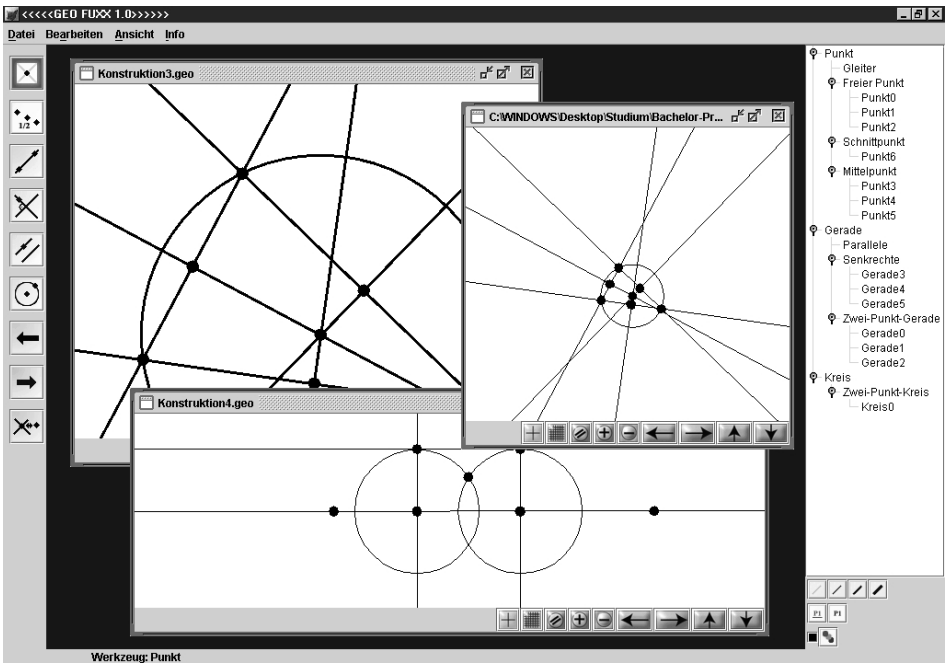

Abbildung 6.1: Das Hauptfenster von Geofuchs mit drei Kindfenstern, die Zeichenflächen beinhalten. Die beiden oberen Kindfenster zeigen dieselbe, das untere Kindfenster eine zweite Konstruktion. Am oberen Rand des Hauptfensters befindet sich wie gewöhnlich die Menüleiste, am unteren Rand eine Statusleiste. Am linken Rand befindet sich ein Werkzeugbalken mit Konstruktionswerkzeugen. Am rechten Rand befindet sich eine nach Typen und Werkzeugen gegliederte Liste aller AGOs der aktiven Konstruktion. Unterhalb davon befinden sich weitere Werkzeugbalken, die Werkzeuge zum Beeinflussen der grafischen Darstellung der AGOs enthalten.

stellte Konstruktionsausschnitt verschoben und skaliert sowie die Koordinatenachsen und das Koordinatengitter ein- und ausgeblendet werden können. Die Zeichenfenster können einzeln skaliert, minimiert und maximiert werden. Zu jedem Zeitpunkt können mehrere Konstruktionen, die jeweils durch ein oder mehrere Zeichenfenster dargestellt werden, geöffnet sein. Jedes Zeichenfenster gehört zu genau einer Konstruktion. Das Zeichenfenster, in dem zuletzt ein Mausclick erfolgt ist, wird optisch hervorgehoben. Die durch dieses Zeichenfenster dargestellte Konstruktion ist „aktiv“, d. h. Programmfunktionen wie „Rückgängig“, „Wiederholen“ und „Konstruktion speichern“ beziehen sich auf diese Konstruktion. Zur Auswahl von Farben und Dateien (zum Laden und Speichern von Konstruktionen) stehen in Geofuchs Dialogfelder zur Verfügung. Die am rechten Rand des Hauptfensters befindliche Auflistung der Namen der in der aktiven Konstruktion enthaltenen AGOs hat die Gestalt eines Waldes: Die Wurzeln sind die Typen von AGOs (Punkt, Gerade und Kreis). Die inneren Knoten sind Bezeichner für die Konstruktionswerkzeuge, die ein AGO des durch den Elternknoten vorgegebenen Typs erzeugen. Die Blätter sind die Namen der in der aktiven Konstruktion enthaltenen AGOs. Ich bezeichne diesen Bestandteil der grafischen Benutzeroberfläche von Geofuchs ab sofort als „Walddarstellung“.

Helmut Balzert empfiehlt in [2] die logische Aufteilung der Bestandteile einer grafischen Benutzeroberfläche in optisch deutlich unterscheidbare Gruppen. Die Schaltflächen zur Auswahl eines Werkzeugs sind in zwei Gruppen unterteilt: Werkzeuge zur Manipulation der grafischen Darstellung einer Konstruktion und Werkzeuge zur Manipulation der Konstruktion an sich, z. B. Konstruktionswerkzeuge. Die erste Gruppe findet sich unten am rechten, die zweite Gruppe am linken Rand des Hauptfensters. Die Schaltflächen der zweiten Gruppe sind von oben nach unten geordnet: Zuerst die Schaltflächen zum Hinzufügen von Punkten, dann die Schaltflächen zum Hinzufügen von Geraden, dann die bisher einzige Schaltfläche zum Hinzufügen von Kreisen und letztendlich die Schaltflächen zum Korrigieren von Konstruktionsfehlern. Diese vier Schaltflächengruppen sind farblich voneinander abgehoben. Somit erhält der Benutzer allein durch die Position und Farbe einer Schaltfläche schon Informationen über deren Bedeutung, was die Einarbeitung in das Programm erheblich erleichtert.

6.2 Erstellen von Konstruktionen

Um Konstruktionen zu erstellen, stehen verschiedene Werkzeuge zur Verfügung, die der Benutzer mit einem Mausclick auf eine Schaltfläche auswählen kann. Sobald der Mauscursor sich kurzzeitig ruhig über einer Schaltfläche befindet, wird eine kurze Beschreibung des entsprechenden Werkzeugs wie z. B. „Mittelpunkt bestimmen“ in unmittelbarer Nähe der Schaltfläche angezeigt (das gilt für alle Schaltflächen im Hauptfenster und in den Zeichenfenstern). Nach dem Mausclick wird die Schaltfläche optisch hervorgehoben, bis der Benutzer ein anderes Werkzeug auswählt. Im Statusbalken wird stets der Name des ausgewählten Werkzeugs angegeben. Bewegt der Benutzer dann die Maus über die aktive Zeichenfläche, so wird jedes AGO, das sich in unmittelbarer Nähe der Mauscursors befindet, grafisch hervorgehoben, falls der Typ des AGOs vom Werkzeug erwartet wird und das AGO im aktuellen Konstruktionsschritt noch nicht ausgewählt wurde. Betrifft das nur ein AGO, wird sein Name und eine kurze Beschreibung (z. B. „g1: Gerade durch P1 und P2“) im Statusbalken angezeigt. Auf diese Art und Weise wird dem Benutzer verdeutlicht, dass ein Mausclick mit der linken Maustaste die Auswahl dieses AGOs zur Folge hätte. Befinden sich mehrerer „passende“ AGOs in der Nähe des Mauscursors, werden alle hervorgehoben. Nach einem Mausclick mit der linken

Maustaste erscheint dann normalerweise eine Liste mit den Namen dieser AGOs, aus der der Benutzer das gewünschte auswählen kann. Das gilt für alle Konstruktionswerkzeuge mit einer Ausnahme, dem Werkzeug für freie Punkte, Gleiter und Schnittpunkte.

Freie Punkte, Gleiter und Schnittpunkte können mit einem einzigen Werkzeug konstruiert werden. Bewegt der Benutzer den Mauscursor nach der Auswahl dieses Werkzeugs über die aktive Zeichenfläche, werden alle Geraden und Kreise in Cursornähe hervorgehoben. Enthält die auf der Zeichenfläche dargestellte Konstruktion keine Gerade und keinen Kreis, die sich in der Nähe des Mauscursors befinden, dann wird der Konstruktion ein freier Punkt hinzugefügt. Befindet sich zum Zeitpunkt des Mausclicks genau eine Gerade oder genau ein Kreis hinreichend nah am Mauscursor, dann wird der Konstruktion ein Gleiter auf der Gerade oder dem Kreis hinzugefügt. Befinden sich zum Zeitpunkt des Mausclicks zwei Geraden in der Nähe des Mauscursors, dann wird der Schnittpunkt beider Geraden der Konstruktion hinzugefügt. Falls sich je eine Gerade und ein Kreis oder zwei Kreise in der Nähe des Mauscursors befinden, wird der Schnittpunkt, der den geringsten Abstand zum Mauscursor hat, der Konstruktion hinzugefügt. Wurde dieser bereits zuvor konstruiert, wird der andere Schnittpunkt der zu schneidenden AGOs der Konstruktion hinzugefügt. Falls sich zum Zeitpunkt des Mausclicks mehr als zwei Geraden oder Kreise in der Nähe des Mauscursors befinden, wird der Schnittpunkt der beiden zuerst konstruierten dieser AGOs der Konstruktion hinzugefügt. Dieses für den Benutzer undurchsichtige Verhalten ist der Nachteil dieses „kombinierten Werkzeugs“. Allerdings können Situationen, in denen mehrere AGOs sich in einem Punkt schneiden, durch Verschieben eines freien oder halbfreien Punkts vermieden werden, sofern sie sich nicht unabhängig von ihrem Zustand in einem Punkt schneiden. Das Bewegen freier Punkte und Gleiter ist in Geofuchs schnell und einfach ohne die Auswahl eines anderen Werkzeugs möglich (siehe Abschnitt 6.3). Schneiden sich mehr als zwei AGOs unabhängig von ihrem Zustand immer in einem Punkt, ist es für den Benutzer irrelevant, welche der AGOs den Schnittpunkt definieren. Sollte der Einsatz des Werkzeugs zum Hinzufügen freier Punkte, Gleiter und Schnittpunkte nicht zum vom Benutzer beabsichtigten Ergebnis führen, stehen weitgehende Möglichkeiten, Konstruktionsfehler zu korrigieren, zur Verfügung (siehe Abschnitt 6.5). Die Verwendung eines einzigen Werkzeugs zum Konstruieren von freien Punkten, Gleitern und Schnittpunkten hat den Vorteil, dass die Werkzeugleiste „schlanker“ und damit übersichtlicher ist und einige Werkzeugwechsel, die aus meiner Sicht den Arbeitsfluss behindern, entfallen können.

Wählt der Benutzer das Werkzeug aus, mit dem er die durch zwei Punkte verlaufende Gerade konstruieren kann, so erwartet Geofuchs die Auswahl zweier im Sinne der Objektidentität nicht identischer Punkte durch Mausclicks auf der Zeichenfläche. Ein Mausclick mit der linken Taste in unmittelbarer Nähe eines Punkts wählt diesen aus. Erfolgt der erste Mausclick, wenn sich kein Punkt hinreichend nah am Mauscursor befindet, so wird in Abhängigkeit der Anzahl von Geraden und Kreisen, die sich hinreichend nah am Mauscursor befinden, der Konstruktion wie oben beschrieben ein freier Punkt, ein Gleiter oder ein Schnittpunkt hinzugefügt und zugleich ausgewählt. Ist bereits ein Punkt ausgewählt, wird die Gerade, die im Falle eines zweiten Mausclicks der Konstruktion hinzugefügt werden würde, sofort auf der Zeichenfläche angezeigt. Wieder kann ein bereits existierender Punkt ausgewählt oder ein neuer Punkt der Konstruktion hinzugefügt und zugleich ausgewählt werden. Dabei folgt die „Vorschau“ auf die Gerade, die der Konstruktion im Falle eines Mausckicks hinzugefügt werden würde, dem Mauscursor. Derartiges Verhalten zeigen auch alle anderen Werkzeuge, die zwei Punkte als Eingabe erwarten, also die Werkzeuge zum Konstruieren von Mittelpunkten und Kreisen.

Die Werkzeuge zum Konstruieren von Parallelen und Senkrechten erwarten als Eingabe

je einen Punkt und eine Gerade. Dabei kann sowohl der Punkt als auch die Gerade zuerst ausgewählt werden. Wählt der Benutzer zuerst eine Gerade aus, dann wird die Paralelle bzw. Senkrechte, die der Konstruktion im Falle eines Mausclicks zu diesem Zeitpunkt hinzugefügt werden würde, angezeigt. Ein zweiter Mausclick wählt dann einen Punkt aus oder fügt der Konstruktion einen neuen Punkt wie oben beschrieben hinzu. Wählt der Benutzer zuerst einen Punkt aus, so wird nur dann eine Gerade als Vorschau angezeigt, wenn sich der Mauscursor in der Nähe einer bereits konstruierten Geraden befindet.

Erwartet das ausgewählte Werkzeug einen oder mehrere Punkte als Eingabe, dann müssen diese also nicht zuvor konstruiert worden sein, sondern können der Konstruktion „just in time“ hinzugefügt werden. Das steigert die Geschwindigkeit, mit der Konstruktionen erstellt werden können. Dieser Vorteil macht sich aber erst dann bemerkbar, wenn der Benutzer schon einige Übung im Umgang mit Geofuchs hat. Auf diesem Weg unterstützt ihn die beschriebene Vorschau und die grafische Hervorhebung auswählbarer AGOs eines erwarteten Typs in Cursornähe.

6.3 Bewegen von Punkten

Sowohl in GeoGebra als auch in Cinderella ist es notwendig, ein eigenes Werkzeug zum Bewegen von Punkten auszuwählen. Das stellt aus meiner Sicht eine unangenehme Unterbrechung des Arbeitsflusses dar. Der Benutzer bewegt freie Punkte und Gleiter nicht nur, nachdem die Konstruktion fertiggestellt ist. Aus Übersichtsgründen oder zur gezielten Auswahl von AGOs für ein Werkzeug ist es häufig notwendig, schon vor Fertigstellung der angestrebten Konstruktion freie Punkte und Gleiter zu bewegen. Daher kann der Benutzer solche Punkte in Geofuchs jederzeit bewegen, indem er den Mauscursor auf der Zeichenfläche in unmittelbare Nähe eines Punkts bewegt und dann die rechte Maustaste drückt. Solange der Benutzer die rechte Maustaste gedrückt hält, folgt der beim Drücken der Taste ausgewählte Punkt dem Mauscursor, der Punkt wird bewegt. Befinden sich beim Drücken der Maustaste mehrere freie Punkte oder Gleiter in der Nähe des Mauscursors, dann wird der Punkt bewegt, der der Konstruktion zuerst hinzugefügt wurde. Ein Gleiter folgt dem Mauscursor nur insofern, dass der Gleiter von der Position des Mauscursors in Weltkoordinaten auf das AGO, von dem er unmittelbar abhängt (also eine Gerade oder einen Kreis), projeziert wird. Sei C der durch die Position des Mauscursors gegebene Punkt. Ein Geradengleiter P wird dann auf den Fußpunkt des durch C verlaufenden Lots auf der definierenden Geraden von P projeziert. Ein Kreisgleiter auf einem Kreis mit Mittelpunkt M wird auf den Schnittpunkt des Kreises mit der Gerade MC projeziert, der C am nächsten ist. Beim Loslassen der rechten Maustaste gelangt das Programm in seinen vorherigen Zustand zurück. Das vor der Bewegung ausgewählte Werkzeug ist nach wie vor ausgewählt. Auch im aktuellen Konstruktionsschritt bereits ausgewählte AGOs bleiben ausgewählt.

6.4 Beeinflussen der grafischen Darstellung von Konstruktionen

Geofuchs ermöglicht es dem Benutzer, Einfluss auf die grafische Darstellung von Konstruktionen zu nehmen. Zum einen kann der Benutzer den durch eine Zeichenfläche dargestellten Ausschnitt der Ebene beeinflussen, indem er die Transformation zwischen Zeichen- und Weltkoordinaten manipuliert. Zum anderen können Eigenschaften wie die Darstellungsfarbe von

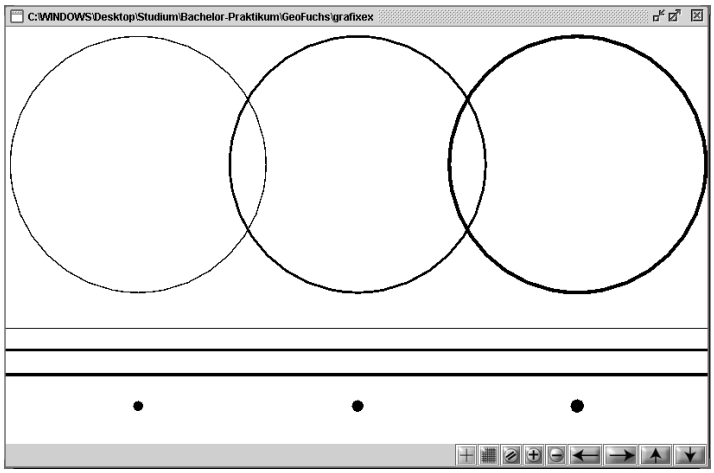

Abbildung 6.2: Ein Zeichenfenster mit unterschiedlich stark hervorgehobenen Kreisen, Geraden und Punkten.

AGOs manipuliert werden.

Die Transformation zwischen Zeichen- und Weltkoordinaten ist normalerweise eine Kombination aus einer Translation und einer Skalierung. Beide können vom Benutzer manipuliert werden. Eine Manipulation der Translation entspricht einer Verschiebung des auf der Zeichenfläche dargestellten Ausschnitts der Konstruktion. Eine Manipulation der Skalierung entspricht dem Vergrößern oder Verkleinern des dargestellten Ausschnitts der Konstruktion. Dazu drückt der Benutzer die linke Maustaste und hält sie, während der Mauscursor sich über einer Zeichenfläche befindet. Bewegt er währenddessen den Mauscursor nach oben oder unten, wird der Skalierungsfaktor der Zeichenfläche vergrößert oder verkleinert. Die Änderung der Translation geschieht wie das Bewegen eines Punktes durch Drücken und Halten der rechten Maustaste, während sich der Mauscursor über einer Zeichenfläche befindet. Befindet sich zu diesem Zeitpunkt kein freier Punkt oder Gleiter in unmittelbarer Nähe des Mauscursors, dann wird anstelle eines Punkts der durch die Zeichenfläche dargestellte Ausschnitt der Konstruktion verschoben. Da diese Möglichkeiten zur Beeinflussung der Transformation nicht intuitiv ersichtlich sind, gibt es zusätzlich Schaltflächen, die die gleichen Funktionen zur Verfügung stellen. Außerdem kann der Benutzer mittels entsprechender Schaltflächen sowohl Koordinatenachsen als auch ein Koordinatengitter auf einer Zeichenfläche anzeigen oder ausblenden.

Geofuchs ermöglicht es dem Benutzer auch, die grafische Darstellung einzelner AGOs zu beeinflussen. Die dafür zuständigen Werkzeuge erwarten genau ein AGO beliebigen Typs als Eingabe. Der Durchmesser von Punkten und die Dicke von Geraden und Kreisen können in vier Stufen von „versteckt“ bis „stark hervorgehoben“ variiert werden (siehe Abbildung 6.2). Für jede der vier Stufen gibt es ein eigenes Werkzeug. Hat der Benutzer eines davon ausgewählt, dann führt ein Mausclick mit der linken Taste auf der Zeichenfläche dazu, dass alle AGOs, die sich in unmittelbarer Nähe des Mauscursors befinden, ihr Erscheinungsbild der Stufe entsprechend ändern. Alternativ kann der Benutzer mit der rechten Maustaste einen Knoten der Walddarstellung der aktiven Konstruktion auswählen. Ein Mausclick auf ein Blatt

wählt das entsprechende AGO aus. Ein Mausclick auf einen inneren Knoten wählt alle AGOs, die mit dem dem Knoten entsprechenden Konstruktionswerkzeug konstruiert wurden, aus. Ein Mausclick auf eine Wurzel wählt alle AGOs vom entsprechenden Typ aus. Hat der Benutzer z. B. das Werkzeug „Verstecken“ ausgewählt, dann führt ein Mausclick mit der rechten Taste auf den Wurzelknoten „Punkt“ dazu, dass die in der aktiven Konstruktion enthaltenen Punkte auf allen der Konstruktion zugeordneten Zeichenflächen nicht mehr grafisch dargestellt werden. Diese Möglichkeiten der Auswahl von AGOs auf einer Zeichenfläche oder in der Waldansicht bestehen auch für das Werkzeug zum Ändern der Darstellungsfarben und das Werkzeug zum Ein- und Ausblenden der Namen von AGOs. Das Werkzeug zum Umbenennen von AGOs erfordert die Auswahl genau eines AGOs. Das kann in der Walddarstellung der aktiven Konstruktion oder auf der Zeichenfläche geschehen. Außerdem gibt es eine Schaltfläche, durch die die Farbe für zukünftige Anwendungen des Werkzeugs zum Farben ändern festgelegt wird. Clickt der Benutzer auf diese Schaltfläche, wird ein Farbauswahldialog angezeigt. Hat der Benutzer eine Farbe ausgewählt, nimmt die Schaltfläche zum Farben auswählen diese Farbe an, und das (auch unabhängig auswählbare) „Farbe ändern“-Werkzeug wird aktiviert.

6.5 Korrektur von Konstruktionsfehlern

Es ist davon auszugehen, dass der Benutzer eines dynamischen Geometrieprogramms Fehler beim Konstruieren macht. Es sind zwei Arten von Konstruktionsfehlern zu erwarten:

- Bedienungsfehler, die auf nicht der Intention des Benutzers entsprechender Anwendung der vom Programm zur Verfügung gestellten Funktionen beruhen
- Inhaltliche Fehler, die auf mangelndem Wissen des Benutzers über die angestrebte Konstruktion beruhen

Ein Bedienungsfehler liegt z. B. vor, wenn der Benutzer den Schnittpunkt zweier Geraden bestimmen möchte, das Werkzeug zum Hinzufügen freier Punkte, Gleiter und Schnittpunkte auswählt und statt auf beide Geraden gleichzeitig auf jede der Geraden einzeln clickt. In diesem Fall werden der Konstruktion zwei Gleiter anstelle des Schnittpunktes hinzugefügt. Bedienungsfehler werden vom Benutzer in den meisten Fällen sofort erkannt. Sie können einfach korrigiert werden, falls das dynamische Geometrieprogramm es erlaubt, die letzten Konstruktionsschritte ihrer Reihenfolge entsprechend rückgängig zu machen. Die meisten dynamischen Geometrieprogramme einschließlich Geofuchs bieten eine solche Funktion an. Geofuchs erlaubt es, alle Konstruktionsschritte einer Konstruktion rückgängig zu machen. Rückgängig gemachte Konstruktionsschritte werden erst endgültig verworfen, wenn ein neuer Konstruktionsschritt ausgeführt wird, und können bis zu diesem Zeitpunkt wiederhergestellt werden. Dieser Mechanismus allein ist allerdings nur von Nutzen, wenn der zu korrigierende Konstruktionsfehler unmittelbar nach seinem Auftreten erkannt wird. Dennoch stellt er beispielsweise in GeoGebra die einzige Möglichkeit zur Korrektur von Konstruktionsfehlern dar. Cinderella und „Zirkel und Lineal“ ermöglichen es auch, AGOs zu löschen. Dann werden aber alle davon abhängigen AGOs auch gelöscht. Damit ist diese Funktion zur Korrektur inhaltlicher Fehler, die oft relativ spät erkannt werden, nur eingeschränkt einsetzbar.

Nach einer kurzen Einarbeitungszeit ist in Hinblick auf den Einsatz als Lern- und Experimentierprogramm zu erwarten, dass der größere Teil der Konstruktionsfehler inhaltiche Fehler sein werden. Der Benutzer wird häufig ein Ziel, zum Beispiel die Konstruktion der Potenzgerade zweier sich nicht schneidender Kreise, vor Augen haben, sich aber nicht sicher

sein, wie er dieses Ziel erreichen kann. Daher wird er mehrere Versuche brauchen, bis er eine seinem Ziel entsprechende Konstruktion erstellt hat. Fällt dem Benutzer auf, dass er mit der von ihm erstellten Konstruktion sein Ziel nicht erreichen kann, muss er von Neuem beginnen, ein AGO und alle davon abhängigen AGOs löschen oder die letzten Konstruktionsschritte rückgängig machen. Dabei werden häufig AGOs gelöscht, die der Benutzer dann erneut konstruieren muss. Das gleiche Problem tritt auf, falls er beispielsweise versuchen möchte, welche Auswirkungen es hat, wenn ein zuvor freier Punkt sich nur noch auf einem Kreis bewegen darf: Dann muss der Benutzer neben dem Punkt zumindest alle davon abhängigen AGOs löschen, einen Kreisgleiter konstruieren und die abhängigen AGOs erneut konstruieren. Einfacher ist es, wenn der Benutzer den freien Punkt einfach in einen Kreisgleiter „verwandeln“ kann, ohne die abhängigen AGOs zu beeinflussen.

Geofuchs erlaubt es daher, die Beziehungen zwischen den AGOs einer Konstruktion im Nachhinein zu ändern. Dazu wählt der Benutzer ein spezielles Werkzeug aus, das genau einen Punkt, genau eine Gerade oder genau einen Kreis als Eingabe erwartet. Dann wählt er ein zu änderndes AGO durch einen Mausclick auf einer Zeichenfläche aus. Um zyklische Abhängigkeiten auszuschließen, werden das ausgewählte und alle davon abhängigen AGOs vorübergehend aus der Konstruktion entfernt. Der Benutzer wählt dann in einem Dialogfeld ein Konstruktionswerkzeug aus, das AGOs vom Typ des ausgewählten AGOs erzeugt. Dann wählt er AGOs der vom Konstruktionswerkzeug erwarteten Typen durch Mausclicks auf der Zeichenfläche aus. Sobald genug definierende AGOs ausgewählt sind, werden das geänderte AGO und alle davon abhängigen AGOs wieder der Konstruktion hinzugefügt. So kann der Benutzer z. B. aus einem freien Punkt einen Gleiter oder einen Schnittpunkt (aber weder eine Gerade noch einen Kreis) machen, ohne dass davon abhängige AGOs gelöscht werden.

Die Möglichkeit der nachträglichen Änderung von Beziehungen zwischen AGOs stellt aus meiner Sicht einen wesentlichen Schritt in Richtung Benutzerfreundlichkeit dar. Überraschenderweise ist mir mit Ausnahme von „pdb“ (dem in [27] beschriebenen dynamischen Geometrieprogramm von Harald Winroth) und Geofuchs kein dynamisches Geometrieprogramm bekannt, das eine derartige Funktion anbietet.

Kapitel 7

GeoXML

Die Speicherung spezieller Konstruktionen durch Geofuchs erfolgt in einem XML-basierten Format, das ich GeoXML genannt habe. In diesem Kapitel beschreibe ich den Aufbau von GeoXML-Dokumenten und gebe einen groben Überblick über die Implementierung des Ladens und Speicherns spezieller Konstruktionen. Dabei setze ich Kenntnisse über grundlegende XML-spezifische Begriffe wie „Element“ und „Attribut“ (siehe z. B. [3]) voraus. Zum Erstellen der UML-Diagramme habe ich das Werkzeug „Java and UML Developer's Environment“ (Jude) verwendet. Ich verwende die in [2] beschriebene Notation mit folgenden Änderungen, die die Verwendung von Jude mit sich bringt: Konstruktoren bezeichne ich nicht mit dem Namen der Klasse, von der eine Instanz erzeugt werden soll, sondern mit `<<create>>`. Die Nachrichten in Sequenzdiagrammen sind numeriert, Wiederholungen in Sequenzdiagrammen stelle ich nicht durch `* operation()`, sondern durch `operation() *` dar. Ursache der ersten in Sequenzdiagrammen auftretenden Nachricht ist stets eine Eingabe des Benutzers. Diesen Fakt vernachlässige ich in den Sequenzdiagrammen.

7.1 Struktur eines GeoXML-Dokuments

In diesem Abschnitt möchte ich den grundlegenden Aufbau eines GeoXML-Dokuments darstellen. Ein GeoXML-Dokument hat folgende Form:

```
<?xml version="1.0"?>
<construction format="geoxml" version="1.0">
  <variables>
    <!--...-->
  </variables>
  <objects>
    <!--...-->
  </objects>
  <display-options>
    <!--...-->
  </display-options>
</construction>
```

Jedes GeoXML-Dokument beginnt mit einem (minimalen) Prolog. Das Dokumentelement ist `<construction>` und enthält genau drei Kindelemente: `<variables>`, `<objects>` und

<display-options>. Diese unterteilen das Dokument in drei semantische Bereiche: Einen Bereich für Parameterwerte (also Zahlen), einen Bereich, der die Konstruktion beschreibt, und einen Bereich, der Informationen zur grafischen Darstellung der Konstruktion enthält. Die ersten beiden Bereiche beschreiben zusammen eine spezielle Konstruktion.

Die Koordinaten von SGOs werden als Kindelemente von <variables> gespeichert:

```
<!--...-->
  <variables>
    <variable id="var0">-254.0</variable>
    <variable id="var1">41.0</variable>
    <variable id="var2">1.0</variable>
    <!--...-->
  </variables>
<!--...-->
```

Jedes der <variable>-Elemente hat ein eindeutiges id-Attribut und enthält eine Zahl. Die AGOs werden als Kindelemente von <objects> gespeichert:

```
<!--...-->
  <objects>
    <point id="Punkt0" name="Punkt0" function="free">
      <param ref="var0"/>
      <param ref="var1"/>
      <param ref="var2"/>
    </point>
    <point id="Punkt1" name="Punkt1" function="free">
      <param ref="var3"/>
      <param ref="var4"/>
      <param ref="var5"/>
    </point>
    <point id="Punkt2" name="Punkt2" function="midpoint">
      <param ref="Punkt1"/>
      <param ref="Punkt0"/>
    </point>
    <line id="Gerade0" name="Gerade0" function="twopointline">
     <param ref="Punkt1"/>
     <param ref="Punkt0"/>
    </line>
    <!--...-->
  </objects>
<!--...-->
```

Jedes Kindelement von <objects> entspricht genau einem AGO. Der Name des Kindelements gibt den Typ des AGOs an. Es gibt zwei Attribute name und id, um es zu ermöglichen, sowohl eine (gegebenenfalls automatisch generierte) Kennung als auch einen sprechenden Namen für jedes AGO anzugeben. Da in Geofuchs jedes AGO einen eindeutigen Namen hat, der gleichzeitig als Kennung dient, sind in von Geofuchs erzeugten GeoXML-Dokumenten die Werte von name und id stets gleich. Der Wert des Attributs function bezeichnet den

Algorithmus des AGOs (bzw. gibt an, durch welches Konstruktionswerkzeug es beim Laden zu konstruieren ist). Als einzige Kindelemente enthalten die Elemente `<point>`, `<line>` usw. beliebig viele `<param>`. Jedes der `<param>`-Elemente hat ein Attribut `ref`, das den Wert des `id`-Attributs von entweder einem Kindelement von `<objects>` oder einem `<variable>`-Element enthält. Jedes Element `<param>` verweist also auf entweder eine Zahl oder ein (definierendes) AGO.

Die Kindelemente von `<display-options>` enthalten Informationen über die grafische Darstellung von AGOs:

```
<!--...-->
  <display-options>
    <option ref="Punkt0" showname="false">
      <color>0.0</color>
      <color>0.0</color>
      <color>250.0</color>
      <size>1</size>
    </option>
    <!--...-->
    <option ref="Gerade0" showname="false">
      <color>250.0</color>
      <color>0.0</color>
      <color>0.0</color>
      <size>1</size>
    </option>
    <!--...-->
  </display-options>
<!--...-->
```

Das Element `<display-options>` enthält genau ein Kindelement `<option>` für jedes in der Konstruktion enthaltene AGO. Das Attribut `ref` enthält den Wert des `id`-Attributs eines Kindelements von `<objects>` und verweist somit auf das AGO, auf das sich die Darstellungsoptionen beziehen. Das Attribut `<showname>` gibt an, ob der Name des AGOs angezeigt werden soll. Jedes Element `<option>` kann beliebig viele Unterelemente `<color>` und ein Unterelement `<size>` enthalten. Die `<color>`-Elemente enthalten Zahlen, die gemeinsam eine Farbe in einem Farbsystem repräsentieren. Geofuchs speichert Farben im RGB-System, daher hat in von Geofuchs erzeugten GeoXML-Dokumenten jedes `<option>`-Element drei Kindelemente `<color>`. Das AGO `Gerade0` im obigen Beispiel ist rot, `Punkt0` ist blau. Das Element `<size>` enthält eine Zahl, die den Durchmesser der grafischen Repräsentation von Punkten und die Dicke der grafischen Repräsentation von Geraden und Kreisen beeinflusst. Geofuchs benutzt dafür Vorgabewerte, die dann mit dieser Zahl multipliziert werden. Ein Element `<size>0</size>` beschreibt somit ein verstecktes AGO.

7.2 Implementierung und Schnittstelle

Das Paket `geoxml` enthält lediglich die drei Klassen `GeoXMLStrings`, `GeoObjectDescription` und `XMLConstruction` sowie die Schnittstelle `GeoXMLStorable`. Die Klasse `GeoXMLStrings`

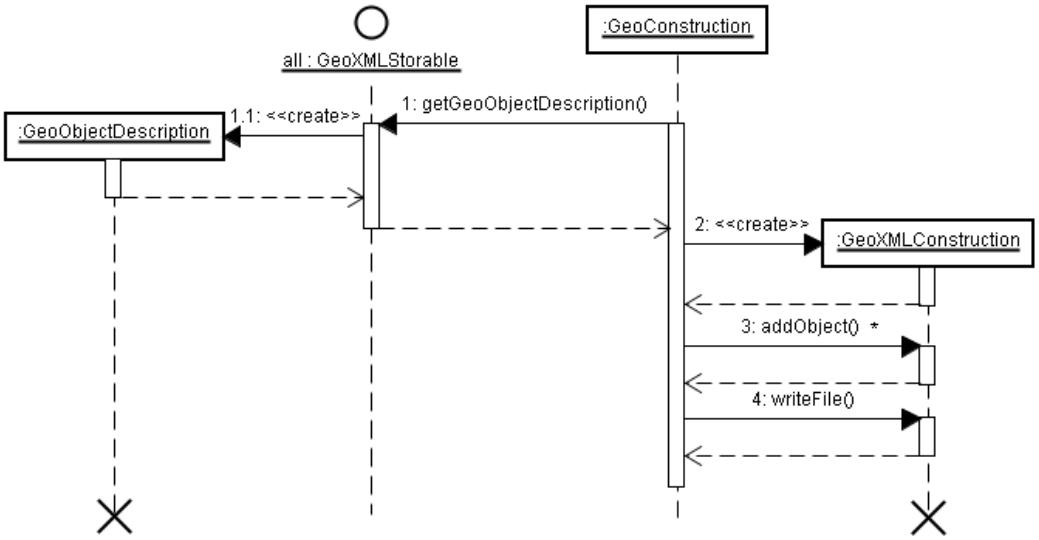

Abbildung 7.1: Speichern einer speziellen Konstruktion als GeoXML-Dokument. Die Instanz von `GeoConstruction` ist eine spezielle Konstruktion.

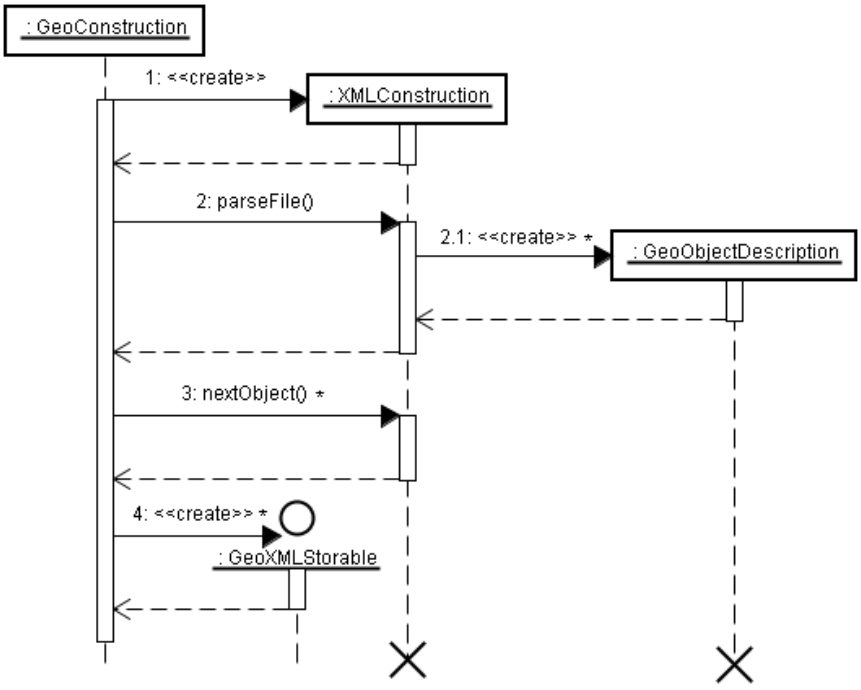

Abbildung 7.2: Laden einer speziellen Konstruktion aus einem GeoXML-Dokument. Die Instanz von `GeoConstruction` ist eine spezielle Konstruktion.

enthält ausschließlich Zeichenketten als konstante Klassenattribute. Diese Zeichenketten werden als Element- und Attributnamen sowie als vordefinierte Elementinhalte und Attributwerte verwendet. Die Klasse `GeoObjectDescription` dient der Kapselung von Informationen über AGOs. Sie besteht lediglich aus privaten Attributen und den zugehörigen Zugriffsmethoden. Jede Instanz dieser Klasse beschreibt die Eigenschaften genau eines SGOs. Die Attribute beschreiben:

- den Namen des SGOs
- die Kennung des SGOs
- den Typ (für Geofuchs Punkt, Gerade oder Kreis) des SGOs
- den Algorithmus des SGOs
- eine Liste der Kennungen der definierenden AGOs
- eine Liste von Zahlen, die die Lage des SGOs beschreiben (für freie und halbfreie SGOs)
- eine weitere Liste von Zahlen, die Farbwerte darstellen
- eine Zahl, die die Größe der grafischen Repräsentation des SGOs beschreibt
- einen Wahrheitswert, der angibt, ob der Name des SGOs angezeigt werden soll oder nicht

Instanzen dieser Klasse werden zum Datenaustausch zwischen Geofuchs und dem GeoXML-Paket verwendet. Die Schnittstelle `GeoXMLStorable` fordert nur eine einzige Methode, die eine Instanz von `GeoObjectDescription` zurückgibt. Da die Attributwerte von Instanzen von `GeoObjectDescription` im durch ihren Typ vorgegebenen Rahmen beliebig gewählt werden und die Listen beliebig viele Elemente enthalten können, ist diese Art der Kommunikation zwischen dem geometrischen Modell eines dynamischen Geometrieprogramms und dem GeoXML-Paket sehr flexibel.

Die Klasse `XMLConstruction` stellt vier öffentliche Methoden zur Verfügung. Sie enthält eine Liste von Instanzen von `GeoObjectDescription`. Zum Speichern einer speziellen Konstruktion (siehe Abbildung 7.1) muss eine Instanz von `XMLConstruction` erzeugt werden. Außerdem wird eine Instanz von `GeoObjectDescription` für jedes in der speziellen Konstruktion enthaltene SGO erzeugt und der Methode `addObject` übergeben. Dann wird die Methode `writeToFile` aufgerufen, die aus den `GeoObjectDescription`-Objekten einen DOM-Baum erstellt und diesen in einem XML-Dokument speichert. Der Methode wird eine Zeichenkette übergeben, die den Pfad und den Namen der Datei bezeichnet. Das gilt auch für die Methode `parseFile`, die die angegebene Datei liest, überprüft, ob es sich um ein GeoXML-Dokument handelt und in eine Liste von Instanzen von `GeoObjectDescription` überführt. Diese können mit der Methode `getNextObject` gelesen werden. Abbildung 7.2 zeigt das Laden eines GeoXML-Dokuments als Sequenzdiagramm.

Kapitel 8

Implementierung von Geofuchs

In diesem Kapitel möchte ich auf die Implementierung von Geofuchs eingehen. Die vorgestellte Implementierung hat sich in verschiedener Art und Weise als vorteilhaft erwiesen, ist aber mit Sicherheit nicht die einzige und auch nicht die „beste“ Variante, ein dynamisches Geometrieprogramm zu implementieren. Daher beschränke ich mich darauf, eine grobe Übersicht über die Funktionsweise von Geofuchs zu geben, und vergleiche diese mit den Lösungen, die in anderen dynamischen Geometrieprogrammen zum Einsatz kommen.

8.1 Das geometrische Modell

Die meisten dynamischen Geometrieprogramme einschließlich „Zirkel und Lineal“ basieren auf dem in Abbildung 8.1 dargestellten Ansatz. Diesen werde ich im Folgenden als „konservativen Ansatz“ bezeichnen. Den Ausgangspunkt der Vererbungshierarchie bildet eine normalerweise abstrakte Klasse, die Methoden und Attribute enthält, die für alle AGOs notwendig sind (z. B. Farbe und Beschriftung). Davon abgeleitet sind u. a. die oft ebenfalls abstrakten Klassen zur Repräsentation von Punkten, Geraden und Kreisen[1]. Diese Klassen entsprechen somit weitgehend den in Kapitel 3 vorgestellten Klassen von AGOs. Sie definieren Attribute und Methoden, die allen AGOs des entsprechenden Typs gemein sind. Das sind beispielsweise Koordinaten sowie eine Methode zum Berechnen des quadrierten Abstands zu einem in Weltkoordinaten gegebenen Punkt. Diese Klassen werden zu Mittelpunkten, Schnittpunkten, Parallelen, Loten usw. spezialisiert. Instanzen dieser Subklassen enthalten Verweise auf die definierenden AGOs und überschreiben den Algorithmus des AGOs.

Geofuchs folgt einem anderen Ansatz, auf dem in ähnlicher Form auch Cinderella und GeoGebra basieren (siehe [16] und [19]). AGOs werden dabei nur aufgrund ihres Typs spezialisiert. Die Berechnung der Koordinaten der AGOs aus den Koordinaten ihrer definierenden AGOs wird in eigene Klassen, die sogenannten Algorithmus-Klassen, ausgelagert. Anstelle von Verweisen auf ihre definierenden AGOs und einer Methode zum Berechnen der eigenen Koordinaten haben AGOs in diesem Ansatz genau einen Verweis auf ein Algorithmus-Objekt. Die beiden Schnittpunkte eines Kreises und einer Gerade oder zweier Kreise haben dann ein gemeinsames Algorithmus-Objekt. Ulrich Kortenkamp schreibt dazu in [19]:

> The crucial point is that we will never calculate *one* intersection of a conic and a line, but always *both* intersections, we will never calculate *one* intersection of two

[1] Die Klasse `PointObject` in „Zirkel und Lineal“ ist allerdings nicht abstrakt. Sie repräsentiert freie Punkte und Gleiter.

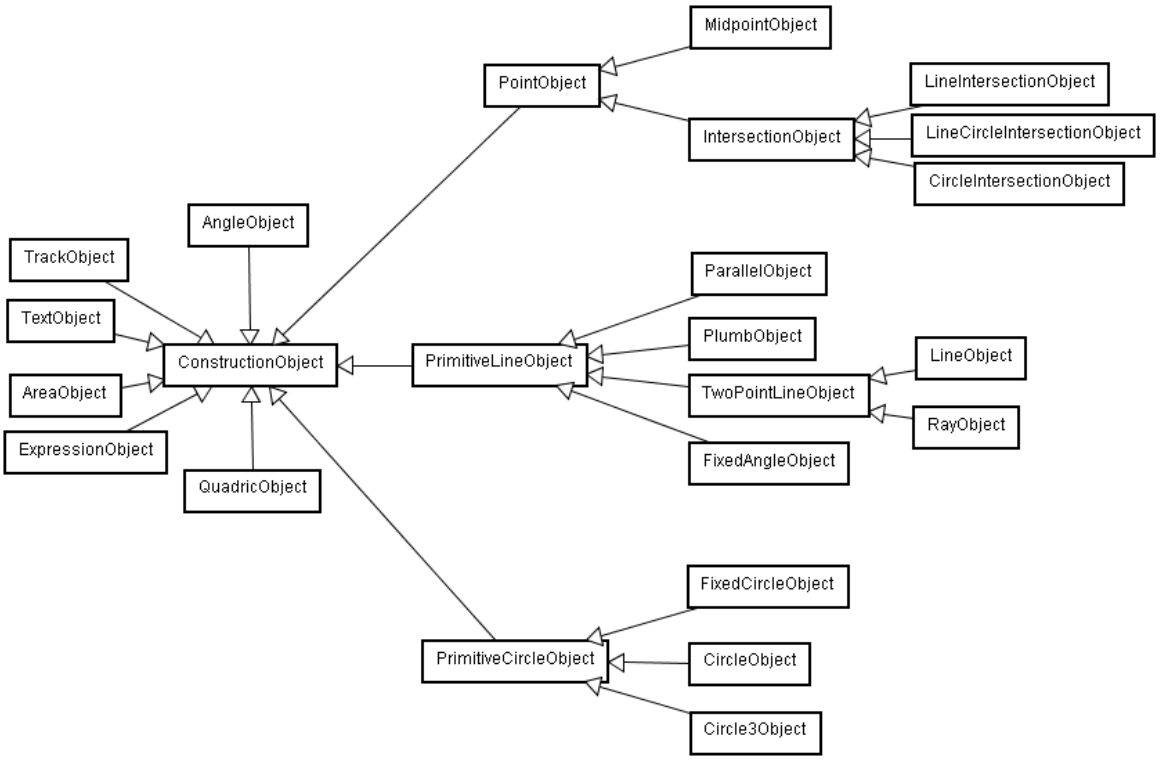

Abbildung 8.1: Die Repräsentation von AGOs in „Zirkel und Lineal“.

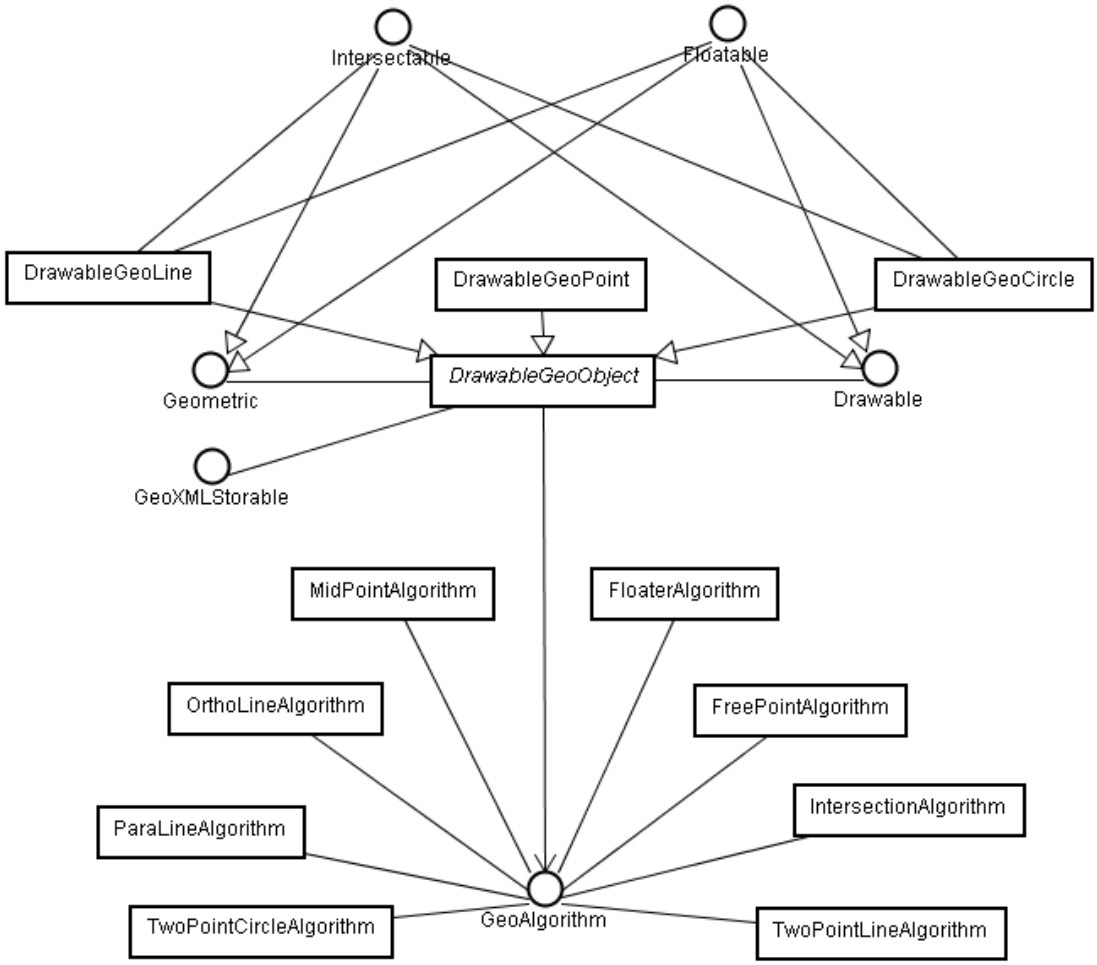

Abbildung 8.2: Die Repräsentation von AGOs in Geofuchs.

> conics, but all *four* of them. This means that in the case where we did define, say, a perpendicular bisector using two circles, we will have to find both intersections and select the right one of these *twice*... The `Algorithm` approach of *Cinderella* is a convenient way to reuse the information we have already found, since the same algorithm instance is plugged into both `PGPoint`-objects. It also increases the stability of the selection process: It can never happen that two intersections collide suddenly...

Außerdem hat sich die Trennung von AGOs und der Algorithmen zur Berechnung ihrer Koordinaten als sehr flexibel erwiesen. Soll der Benutzer im Nachhinein die Beziehungen eines AGOs zu anderen AGOs ändern, z. B. aus einem freien Punkt einen Kreisgleiter machen können, dann muss beim konservativen Ansatz ein neues AGO erzeugt werden. Sind die AGOs dagegen wie beschrieben von ihren Algorithmen getrennt, muss lediglich das zugehörige Algorithmus-Objekt ersetzt werden.

Die abstrakte Klasse `DrawableGeoObject` ist die gemeinsame Oberklasse aller AGOs in Geofuchs. Diese Klasse definiert alle Attribute wie beispielsweise Name, Farbe und Algorithmus, die jedes AGO besitzt. `DrawableGeoObject` fordert die Implementierung der Schnittstellen `Drawable`, `Geometric` und `GeoXMLStorable` (siehe Kapitel 7) von allen nicht abstrakten Subklassen. Die Schnittstelle `Drawable` fordert Methoden zur Beeinflussung der Darstellung (z. B. `setColor`) und zum Zeichnen (Methode `draw`) des AGOs. In Cinderella ist die grafische Repräsentation von AGOs von diesen getrennt in eigenen Klassen implementiert (siehe [19]). Dieser Ansatz bietet aus meiner Sicht mit Ausnahme der klareren Trennung von AGOs und deren grafischer Repräsentation keine Vorteile, die den zusätzlichen (zugegebenermaßen geringen) Programmier-, Speicher- und Laufzeitaufwand rechtfertigen würden. Die Teile des Programms, die der grafischen Repräsentation spezieller Konstruktionen dienen, behandeln die enthaltenen SGOs als Instanzen von `Drawable`. Die Schnittstelle `Geometric` fordert Methoden zur Lösung von grundlegenden Problemen der analytischen Geometrie, beispielsweise die Bestimmung des quadrierten Abstands zu einem in Weltkoordinaten gegebenen Punkt. Nur wenige dieser Methoden werden in `DrawableGeoObject` selbst implementiert. Die Implementierung des größeren Teils findet erst in den Subklassen `DrawableGeoPoint`, `DrawableGeoLine` und `DrawableGeoCircle` statt. Instanzen dieser Klassen sind Punkte, Geraden und Kreise. Geraden und Kreise implementieren die Schnittstellen `Floatable` und `Intersectable`, die beide von `Drawable` und `Geometric` abgeleitet sind. Die Schnittstelle `Floatable` fordert von ihren Instanzen eine Methode zur Berechnung der Koordinaten eines auf das AGO projezierten durch seine Weltkoordinaten gegebenen Punkts. Die Schnittstelle `Intersectable` fordert von ihren Instanzen eine Methode `getIntersectionSet` zum Bestimmen der Koordinaten der Schnittpunkte mit einer anderen Instanz von `Intersectable`.

Um die Änderbarkeit der mathematischen Basis von Geofuchs zu gewährleisten, werden alle Berechnungen durch Methoden der Klasse `Calculator` vorgenommen. Sie stellt ausschließlich statische Methoden zur Verfügung, beispielsweise `getLineLineIntersection` und `getLineCircleIntersection`. Diesen Methoden werden Koordinaten übergeben, und sie geben Koordinaten zurück. Beispielsweise berechnet die `getIntersectionSet`-Methode eines `Intersectable`-Objekts die Koordinaten der Schnittpunkte mit einer anderen Instanz von `Intersectable` nicht selbst. Stattdessen übergibt sie die Koordinaten beider Instanzen von `Intersectable` in Abhängigkeit von ihrem Typ (Gerade oder Kreis) an eine Methode wie `Calculator.getLineLineIntersection`. Diese gibt dann die Koordinaten des Schnittpunkts zurück.

Wie bereits erwähnt, enthält jedes AGO eine Referenz auf genau ein Algorithmus-Objekt. Algorithmus-Objekte sind Instanzen von Klassen, die die Schnittstelle `GeoAlgorithm` implementieren. Diese fordert Methoden, die Informationen über den Algorithmus (z. B. die Typen der definierenden und des abhängigen AGOs) bereitstellen. Außerdem fordert `GeoAlgorithm` die Methode `createObject`, die ein AGO mit diesem Algorithmus erzeugt und die Methode `calculate`, die aus den Koordinaten der definierenden AGOs die Koordinaten des abhängigen AGOs bestimmt und gegebenenfalls ändert. Der `calculate`-Methode obliegt es auch, degenerierte SGOs zu erkennen und als „ungültig“ zu markieren. Jedes Algorithmus-Objekt hat Referenzen auf ein oder, im Fall allgemeiner Schnittpunkte, mehrere abhängige AGOs. Instanzen von `IntersectionAlgorithm` können für Kreis-Kreis-, Kreis-Gerade- und Gerade-Gerade-Schnittpunkte verwendet werden. Die definierenden AGOs sind in diesem Fall Instanzen von `Intersectable`. Die Koordinaten der Schnittpunkte zweier SGOs werden nicht direkt in der `calculate`-Methode des `IntersectionAlgorithm`-Objekts bestimmt. Stattdessen ruft die `calculate`-Methode die `getIntersectionSet`-Methode eines der definierenden

AGOs auf, die die Koordinaten der Schnittpunkte zurückgibt. Die `calculate`-Methode eines `IntersectionAlgorithm`-Objekts ordnet die Koordinaten der speziellen Schnittpunkte den abhängigen Punkten nach dem in Abschnitt 4.2 beschriebenen Algorithmus zu. Ebenso können Instanzen von `FloaterAlgorithm` als Algorithmus-Objekte für Geraden- und Kreisgleiter genutzt werden. Abbildung 8.2 zeigt die Klassenstruktur zur Repräsentation von AGOs und Algorithmen in Geofuchs.

Die Repräsentation von Konstruktionen erfolgt in Geofuchs durch Instanzen der Klasse `GeoConstruction`. Instanzen von `GeoConstruction` enthalten eine Liste der in der repräsentierten Konstruktion enthaltenen AGOs. Die Klasse `GeoConstruction` stellt eine Vielzahl von Methoden bereit, um einer Konstruktion AGOs hinzuzufügen und die enthaltenen AGOs zu manipulieren. Damit trennt die Klasse `GeoConstruction` den Rest des geometrischen Modells von der grafischen Benutzeroberfläche des Programms. Außerhalb des geometrischen Modells (bestehend aus den Klassen zur Repräsentation von AGOs, Algorithmen und Konstruktionen) werden ausschließlich die Methoden von `GeoConstruction` verwendet, um AGOs zur Konstruktion hinzuzufügen oder zu manipulieren. Beispielsweise stellt `GeoConstruction` eine Methode `addIntersection` bereit, die je eine Instanz von `DrawableGeoPoint` und `IntersectionAlgorithm` erzeugt. Die aufrufende Methode braucht sich somit nicht um Implementierungsdetails des geometrischen Modells zu kümmern. Die Festlegung der definierenden AGOs muss zuvor mit der Methode `selectObject` erfolgt sein. Dieser wird der Name eines in der Konstruktion enthaltenen AGOs übergeben. Die Methode speichert diesen dann in einer Liste, die beim Aufruf von `addIntersection` gelesen wird[2]. Jede Instanz von `GeoConstruction` enthält auch eine Liste der erwarteten Typen des ausgewählten Konstruktionswerkzeugs. Wird ein AGO durch `selectObject` ausgewählt, wird ein dem Typ dieses AGOs entsprechendes Element aus dieser Liste entfernt. Jede Konstruktion enthält zudem zwei Methoden `load` und `save`, die zum Laden und Speichern der speziellen Konstruktion als GeoXML-Dokument (siehe Kapitel 7) dienen. Die Methode `calculate` einer Konstruktion durchläuft alle enthaltenen AGOs und ruft deren `calculate`-Methoden auf. Eine schnellere, aber auch aufwändigere Möglichkeit der Aktualisierung einer Konstruktion wäre es, wie z. B. in Cinderella nur die vom vom Benutzer bewegten AGO abhängigen AGOs zu aktualisieren (siehe [19] für eine Beschreibung dieses Ansatzes und weiterer Optimierungsmöglichkeiten). Weitere Informationen zu den Methoden der Klasse `GeoConstruction` finden sich in Abschnitt 8.3.

8.2 Grafische Benutzeroberfläche

Das Hauptfenster von Geofuchs ist eine Instanz der Klasse `GMainFrame`. Der Konstruktor dieser Klasse erzeugt die Werkzeug-, Status- und Menübalken sowie die Walddarstellung der (zu Beginn leeren) aktiven Konstruktion. Zur Laufzeit gibt es für jede Instanz des Programms Geofuchs genau eine Instanz von `GMainFrame`. Diese ist hauptsächlich für die Verwaltung des Programmzustands (z. B. „Mittelpunkt hinzufügen“) und für die Verwaltung geöffneter Konstruktionen und der damit assoziierten Zeichenfenster (in denen die Konstruktionen dargestellt werden) verantwortlich.

Zeichenfenster sind Instanzen der Klasse `GChildFrame`. Da die meisten Benutzereingaben

[2]Tatsächlich wird nicht der Name des AGOs, sondern der Index des AGOs in der Liste der in der Konstruktion enthaltenen AGOs gespeichert. Zur Vereinfachung vernachlässige ich derartige Details, sofern sie nicht für das Verständnis der grundlegenden Funktionsweise von Geofuchs notwendig sind.

auf einer Komponente des Zeichenfensters, der Zeichenfläche, auftreten, enthält die Klasse `GChildFrame` vergleichsweise wenig Funktionalität. Zeichenflächen sind Instanzen der Klasse `GCoordinateArea`. Diese Klasse stellt Methoden für die Transformation von Zeichen- in Weltkoordinaten bereit und implementiert einige der von Java für die Meldung von Benutzereingaben zur Verfügung gestellten Schnittstellen wie `MouseListener` und `MouseMotionListener`. In den von diesen Schnittstellen geforderten Methoden `mousePressed`, `mouseMoved` usw. werden vom Benutzer auf der Zeichenfläche ausgelöste Mausereignisse verarbeitet und in entsprechende Aufrufe der Methoden der mit der Zeichenfläche assoziierten Konstruktion oder Änderungen der Transformation von Zeichen- in Weltkoordinaten umgesetzt. Ist ein Neuzeichnen der auf der Zeichenfläche dargestellten Konstruktion aufgrund einer Zustandsänderung derselben notwendig, ruft die für das Zeichnen der Zeichenfläche verantwortliche Methode `paintComponent` die Methode `getDrawableObjects` der Konstruktion auf. Diese Methode gibt eine Liste aller in der speziellen Konstruktion enthaltenen SGOs als Instanzen von `Drawable` zurück. Diese Liste enthält unter Umständen zusätzlich ein weiteres SGO als „Vorschau-Objekt“. Die `paintComponent`-Methode durchläuft die Liste und ruft die `draw`-Methoden der SGOs auf.

8.3 Interaktion mit dem Benutzer

Dieser Abschnitt beschreibt, wie Geofuchs Mauseingaben des Benutzers auf einer Zeichenfläche verarbeitet. Jeder Zeichenfläche ist eindeutig eine Konstruktion, also eine Instanz von `GeoConstruction`, zugeordnet. In diesem Abschnitt nehme ich zur Demonstration der grundlegenden Funktionsweise von Geofuchs an, dass es nur eine einzige Zeichenfläche gibt. Damit ist es gerechtfertigt, von „der Zeichenfläche“ und „der Konstruktion“ zu sprechen.

8.3.1 Hinzufügen allgemeiner geometrischer Objekte

Auf der Zeichenfläche aufgetretene Mausclicks werden von der Methode `mouseClicked` der Zeichenfläche verarbeitet. Sie überprüft zuerst, ob der Mausclick mit der linken Maustaste erfolgt ist. Mausclicks mit der rechten Maustaste haben keine Wirkung.

Zuerst werden die Zeichenkoordinaten, an denen der Mausclick erfolgt ist, durch die Methode `pointToReal2D` der Zeichenfläche in Weltkoordinaten transformiert. Dann wird die Methode `getMatchingObjectsAt` der speziellen Konstruktion aufgerufen. Dieser Methode werden die Weltkoordinaten, an denen der Mausclick erfolgt ist, übergeben. Sie liefert eine Liste der Namen aller in der speziellen Konstruktion enthaltenen SGOs, die sich hinreichend nah am durch die Weltkoordinaten angegebenen Ort befinden, von einem erwarteten Typ sind und im aktuellen Konstruktionsschritt noch nicht ausgewählt wurden. Ist das aktuell gewählte Konstruktionswerkzeug „Punkt hinzufügen“, werden folgende drei Fälle unterschieden: Ist die von `getMatchingObjectsAt` zurückgegebene Liste leer, wird die Methode `addFreePoint` der Konstruktion aufgerufen. Ihr werden die in Weltkoordinaten transformierten Zeichenkoordinaten, an denen der Mausclick erfolgt ist, übergeben. Enthält die Liste genau eine Zeichenkette, erfolgt stattdessen ein Aufruf der Methode `addFloater`. Dieser Methode werden die einzige Zeichenkette in der Liste und die durch den Mausclick angegebenen Weltkoordinaten übergeben. Enthält die Liste zwei oder mehr Zeichenketten, so erfolgt ein Aufruf der Methode `addIntersection`, der die ersten beiden in der Liste enthaltenen Zeichenketten übergeben werden. Alle drei Methoden fügen der Konstruktion einen Punkt mit entsprechendem Algorithmus-Objekt hinzu.

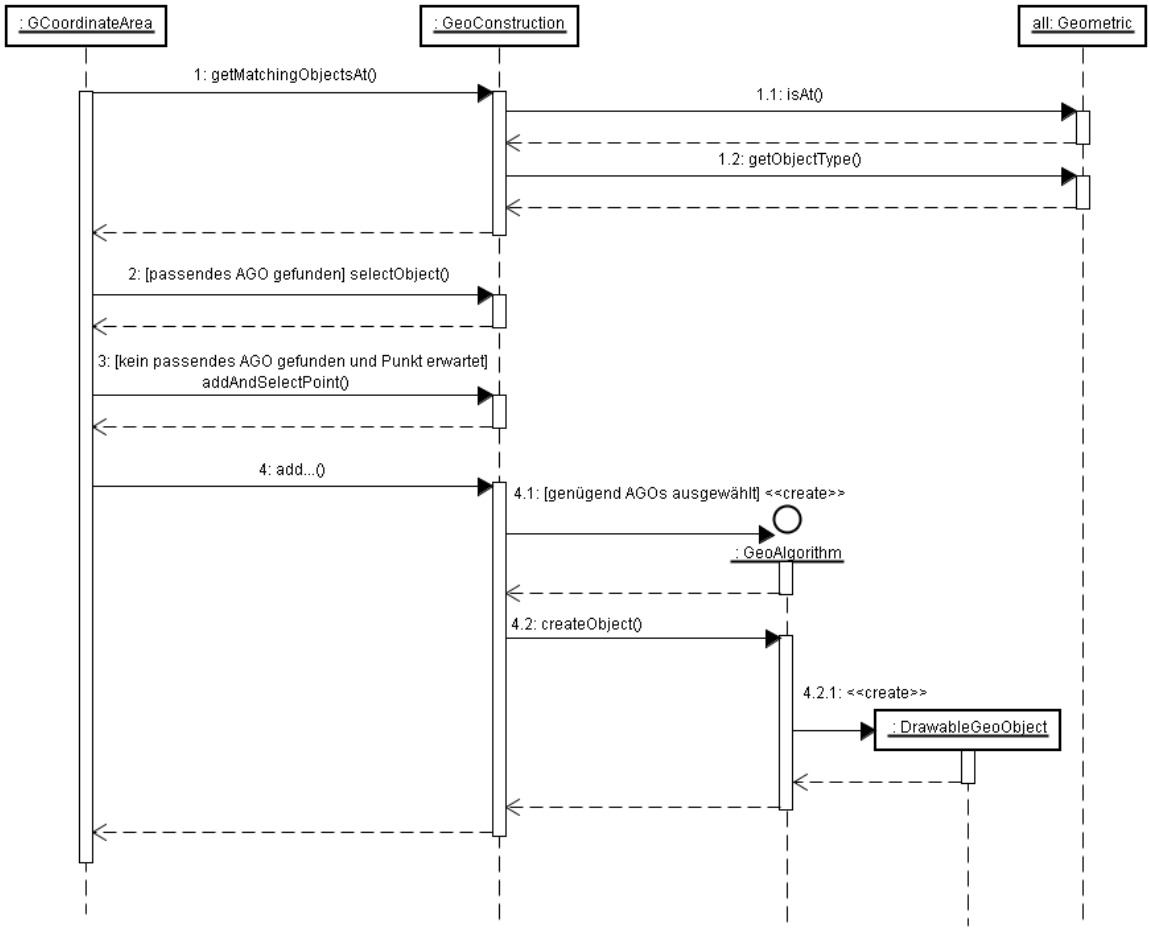

Abbildung 8.3: Verarbeitung eines Mausclicks mit der linken Maustaste auf einer Zeichenfläche (vereinfacht).

Falls ein anderes Konstruktionswerkzeug ausgewählt und die von `getMatchingObjectsAt` zurückgegebene Liste leer ist, wird zunächst mit der Methode `isPointExpected` der Konstruktion ermittelt, ob das aktuelle Konstruktionswerkzeug die Auswahl eines weiteren Punktes erwartet. Falls ja, wird die Methode `addAndSelectPoint` der zur Zeichenfläche gehörigen Instanz von `GeoConstruction` aufgerufen. Diese erzeugt einen neuen Punkt und fügt diesen der Liste der in der Konstruktion enthaltenen AGOs und der Liste ausgewählter AGOs hinzu. Enthält die von `getMatchingObjectsAt` zurückgegebene Liste genau eine Zeichenkette, wird diese Zeichenkette an `selectObject` übergeben. Enthält die Liste mehrere Zeichenketten, werden diese in einem Auswahldialog angezeigt. Der Benutzer wählt eine der Zeichenketten aus. Die vom Benutzer ausgewählte Zeichenkette wird der Methode `selectObject` der Konstruktion übergeben. Letztendlich wird in Abhängigkeit vom ausgewählten Konstruktionswerkzeug eine Methode der Konstruktion, z. B. `addTwoPointLine` oder `addParaLine`, aufgerufen.

Jede dieser `add...`-Methoden (mit Ausnahme von `addFreePoint`) ist wirkungslos, falls nicht zuvor genug AGOs passenden Typs mittels `selectObject` bzw. `addAndSelectPoint` ausgewählt wurden. Andernfalls wird ein Algorithmus-Objekt erzeugt. Dessen Konstruktor werden Referenzen auf die ausgewählten AGOs übergeben. Dann ruft die `add...`-Methode die Methode `createObject` des Algorithmus-Objekts auf. Diese Methode gibt ein AGO zurück, das dann der Liste der in der Konstruktion enthaltenen AGOs hinzugefügt wird. Dann wird die Liste der ausgewählten AGOs „geleert“. Abbildung 8.3 zeigt den groben Ablauf des Hinzufügens eines AGOs als Sequenzdiagramm.

8.3.2 Bewegen der Maus auf der Zeichenfläche

Bewegungen der Maus auf der Zeichenfläche werden von der Methode `mouseMoved` der Zeichenfläche verarbeitet. Abbildung 8.4 zeigt die Verarbeitung von Mausbewegungen als Sequenzdiagramm. In erster Linie wird die Methode `setMousePosition` der speziellen Konstruktion aufgerufen. Dieser Methode werden die der aktuellen Mausposition entsprechenden Weltkoordinaten übergeben. Sie überprüft zunächst für jedes einzelne in der speziellen Konstruktion enthaltene SGO, ob es sich in der Nähe des durch die übergebenen Weltkoordinaten gegebenen Orts befindet, sein Typ einem erwarteten Typ entspricht und es im aktuellen Konstruktionsschritt noch nicht ausgewählt wurde. Dann wird die Methode `setHighlight(boolean)` des SGOs aufgerufen. Ihr wird `true` übergeben, falls das SGO die genannten Bedingungen erfüllt, und sonst `false`. Gibt es genau ein SGO, das die oben genannten Bedingungen erfüllt und muss nur noch genau ein AGO vom Benutzer ausgewählt werden, um den Konstruktionsschritt abzuschließen, dann wird das SGO, das im Falle eines Mausclicks des Benutzers zu diesem Zeitpunkt der Konstruktion hinzugefügt werden würde, erzeugt. Dieses SGO dient als „Vorschau-Objekt“. Das Vorschau-Objekt wird getrennt von den tatsächlich in der speziellen Konstruktion enthaltenen SGOs gespeichert. Es wird bei jedem Aufruf von `mouseMoved` aktualisiert und ausschließlich bei der grafischen Darstellung der speziellen Konstruktion berücksichtigt.

8.3.3 Bewegen von freien Punkten und Gleitern

.

Freie Punkte und Gleiter werden in Geofuchs durch „Ziehen“ der Maus mit gedrückter rechter Maustaste bewegt. Abbildung 8.5 stellt die internen Abläufe beim Bewegen eines Punkts durch den Benutzer vereinfacht als Sequenzdiagramm dar. Beim Drücken der rech-

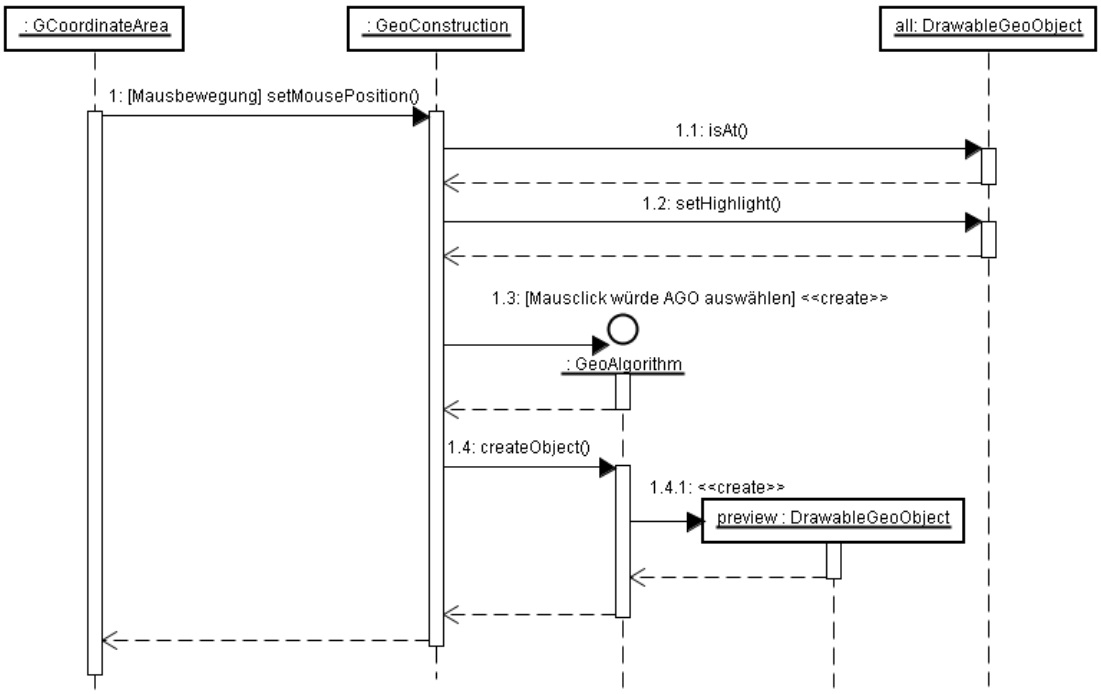

Abbildung 8.4: Verarbeitung von Mausbewegungen in Geofuchs (vereinfacht).

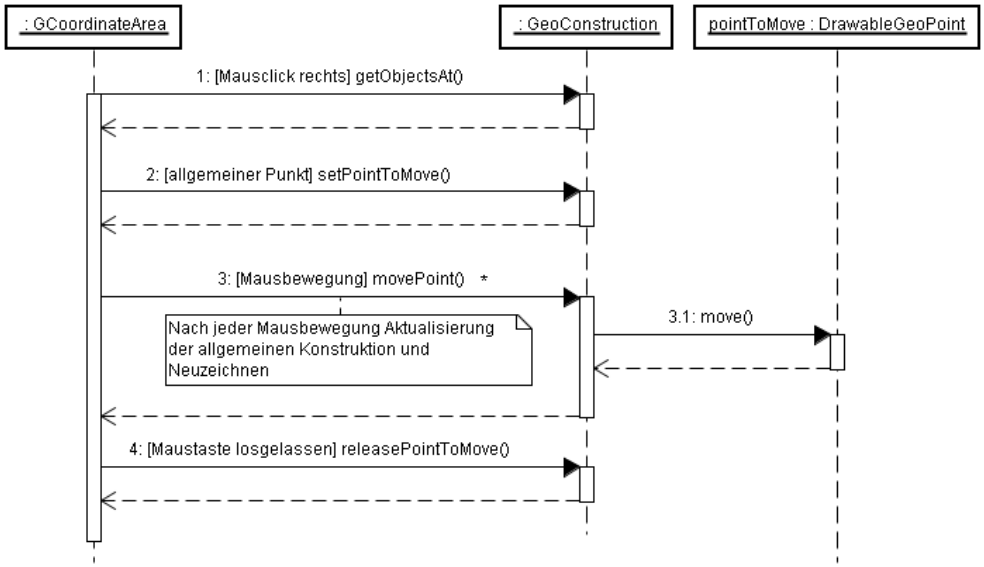

Abbildung 8.5: Bewegen eines Punktes in Geofuchs (vereinfacht).

ten Maustaste ruft die Methode `mousePressed` der Zeichenfläche die Methode `getObjectsAt` der Konstruktion auf. Ihr wird neben Weltkoordinaten auch eine Zeichenkette übergeben. Sie gibt eine Liste der Namen aller SGOs, die sich hinreichend nah am durch die übergebenen Weltkoordinaten gegebenen Ort befinden und vom durch die übergebene Zeichenkette festgelegten Typ sind, zurück. So werden die Namen aller in der Nähe der Mausposition befindlichen Punkte bestimmt. Falls die Liste nicht leer ist, wird ihr erstes Element der Methode `setPointToMove` der Konstruktion übergeben. Den Punkt mit diesem Namen bezeichne ich im Folgenden als den „zu bewegenden Punkt“.

Die Verarbeitung von Bewegungen der Maus, die auftreten, während eine Maustaste gedrückt ist, erfolgt in der Methode `mouseDragged` der Zeichenfläche. Diese ruft die Methode `movePoint` der Konstruktion auf. Die Mauskoordinaten werden in Weltkoordinaten transformiert und der Methode `movePoint` übergeben, die die Koordinaten des zu bewegenden Punkts (falls es einen solchen gibt) mit den übergebenen Weltkoordinaten überschreibt. Gleich darauf erfolgt eine Aktualisierung der Konstruktion durch Aufruf ihrer `calculate`-Methode. Ist der zu bewegende Punkt ein Gleiter, dann wird er von seinen neuen Koordinaten aus auf sein definierendes AGO projeziert. Ist der zu bewegende Punkt fest, dann werden seine Koordinaten sofort wieder auf die durch seine definierenden AGOs vorgegebenen Werte gesetzt. Lässt der Benutzer die rechte Maustaste los, ruft die Methode `mouseReleased` der Zeichenfläche die Methode `releasePoint` auf, um den zu bewegenden Punkt wieder „loszulassen“.

Kapitel 9

Bewertung

In diesem Kapitel möchte ich beschreiben, in welchem Ausmaß Geofuchs den Anforderungen an ein dynamisches Geometrieprogramm gerecht wird, wo die Stärken von Geofuchs liegen und wo aus meiner Sicht Erweiterungen oder auch Änderungen notwendig sind.

Da für die Entwicklung von Geofuchs nur ein eingeschränkter Zeitrahmen zur Verfügung stand, beschränkten sich die zu Beginn des Praktikums an Geofuchs gestellten funktionalen Anforderungen auf die Bereitstellung grundlegender Konstruktionswerkzeuge und die Speicherung von Konstruktionen als XML-Dokument. Obwohl Geofuchs nicht nur diese, sondern auch weitergehende Funktionen (z. B. zur Beeinflussung der grafischen Darstellung von AGOs) bereitstellt, ist der aktuelle Funktionsumfang von Geofuchs in keinster Weise mit dem Funktionsumfang von „Zirkel und Lineal“, Cinderella und GeoGebra zu vergleichen. Da dieser Mangel abzusehen war, sollte Geofuchs möglichst leicht erweiterbar sein. Um dieses Ziel zu erreichen, habe ich nicht nur versucht, die Regeln „guten Programmierens“ wie umfassende Kommentierung des Quellcodes einzuhalten, sondern auch grundsätzlich verallgemeinerte speziellen Lösungen vorgezogen. Beispielsweise können Instanzen der Klasse `IntersectionAlgorithm` nicht nur benutzt werden, um die Schnittpunkte von Geraden und Kreisen, sondern alle Arten von Schnittpunkten zu verwalten. Sollte Geofuchs eines Tages auch Kegelschnitte unterstützen, können deren Schnittpunkte mit anderen AGOs ebenfalls durch `IntersectionAlgorithm`-Objekte verwaltet werden. Ähnlich sieht es aus, wenn Gleiter auf Kegelschnitten implementiert werden sollen. Dazu muss die grafische Benutzeroberfläche nur geringfügig geändert werden, so dass das „Punkt hinzufügen“-Werkzeug auch Kegelschnitte erwartet. Das ist durch einen einzigen zusätzlichen Methodenaufruf möglich. Gleiter auf Kegelschnitten sind dann nach wie vor Instanzen von `DrawableGeoPoint` und haben einen Verweis auf ein `FloaterAlgorithm`-Objekt. Die Klasse, die Kegelschnitte repräsentiert, muss die Schnittstelle `Floatable` implementieren. Diese fordert von ihren Instanzen nur eine einzige Methode, die zur Projektion eines speziellen Punktes dient. Viele weitere der Klassen und Methoden von Geofuchs lassen sich auf ähnliche Art und Weise auch für zukünftige Erweiterungen nutzen. Aus meiner Sicht dringend notwendige Erweiterungen des Funktionsumfangs sind v. a. die Unterstützung von Kegelschnitten und eine Exportmöglichkeit in ein Grafikformat wie EPS. Viele Erweiterungen von Geofuchs werden auch Erweiterungen von GeoXML erfordern. Aus diesem Grund habe ich noch kein Schema für GeoXML-Dokumente erstellt.

Die Klasse `GeoConstruction` bietet eine logische Sicht auf das geometrische Modell von Geofuchs. Da alle Zugriffe auf das geometrische Modell über Methoden dieser Klasse erfolgen, erfordern Änderungen an der Repräsentation von AGOs und Algorithmen keine Änderungen

an Klassen, die nicht zum geometrischen Modell gehören. Abgesehen davon sind in Geofuchs die Aufgabenbereiche einzelner Klassen klar abgegrenzt. Die Aufgaben einer umfangreichen Klasse wie `MidpointObject` in „Zirkel und Lineal“ übernehmen in Geofuchs die Klassen `DrawableGeoPoint`, `MidPointAlgorithm` und `Calculator`. Daher halte ich Geofuchs für vergleichsweise leicht änderbar.

Die mathematische Basis von Geofuchs erreicht nicht die Konsistenz des mathematischen Modells von Cinderella, ist dafür aber einfacher. Es ist mir noch nicht gelungen, in der aktuellen Version von Geofuchs einen Sprung zu beobachten. Überarbeitungswürdig ist das Verhalten von Gleitern. Bei einer Bewegung des definierenden AGOs eines Gleiters wird der Gleiter einfach von seiner Position vor der Bewegung auf das definierende AGO projeziert. Konstruiert man einen Gleiter auf einer durch zwei freie Punkte A und B gegebenen Geraden so, dass der Gleiter zwischen A und B liegt, und bewegt A kreisförmig mehrmals um B, dann bewegt sich der Gleiter immer näher zu B hin und fällt irgenwann mit B zusammen. Rotiert man dann A in entgegengesetztem Drehsinn um B, lässt sich diese Bewegung des Gleiters nicht rückgängig machen. Der Benutzer muss den Gleiter direkt bewegen. Problematisch ist, dass dieses Verhalten von Gleitern zu durch gleiche spezielle Punkte degenerierten speziellen Konstruktionen führen kann. In Cinderella tritt das selbe Problem auf. In „Zirkel und Lineal“ und vor allem GeoGebra ist das Verhalten von Gleitern bei Bewegungen des definierenden AGOs meiner Meinung nach besser gelöst.

Die grafische Benutzeroberfläche von Geofuchs halte ich für attraktiv und intuitiv bedienbar. Ebenso ermöglicht sie geübten Benutzern zügiges Arbeiten, da vergleichsweise wenige Werkzeugwechsel notwendig sind. Die Möglichkeit, die Beziehungen zwischen AGOs im Nachhinein zu ändern, erleichtert die nachträgliche Korrektur von Konstruktionsfehlern und das Experimentieren. Als nicht zufriedenstellend empfinde ich lediglich die grafische Darstellung von SGOs auf einer Zeichenfläche. Hierfür verwende ich das in der J2SE enthaltene Abstract Window Toolkit (AWT), das Methoden zum Zeichnen von beispielsweise Geraden und Kreisen (als spezielle Ellipsen) zur Verfügung stellt. Mit diesen Methoden lässt sich leider keine qualitativ hochwertige grafische Ausgabe erreichen. Vor allem Geraden wirken in GeoGebra und Cinderella wesentlich „glatter“ als in Geofuchs und „Zirkel und Lineal“, das ebenfalls das AWT für die grafische Darstellung spezieller Konstruktionen verwendet. Zudem sind die Methoden des AWT recht ungenau (z. B. beim Zeichnen von Kreisen mit sehr großem Radius). Abhilfe könnte die Verwendung einer anderen Grafikbibliothek oder die Verwendung selbst implementierter Methoden unter Einsatz gängiger Techniken zur Verbesserung der Grafikqualität wie Anti-Aliasing schaffen.

Literaturverzeichnis

[1] M. J. Atallah (Hrsg.): *Algorithms and theory of computation handbook*. CRC Press Boca Raton 1999.

[2] H. Balzert: *Lehrbuch der Software-Technik*. Band 1: Software-Entwicklung. 2. Auflage, Spektrum Heidelberg 2000.

[3] T. Bray (Hrsg.) et al: *Extensible Markup Language (XML) 1.0 (Third Edition)*. `http://www.w3.org/TR/2004/REC-xml-20040204`. W3C Recommendation, 2004. Abruf am 03.10.2004.

[4] T. Bray (Hrsg.) et al: *Extensible Markup Language (XML) 1.1*. `http://www.w3.org/TR/2004/REC-xml11-20040204`. W3C Recommendation, 2004. Abruf am 03.10.2004.

[5] H. S. M. Coxeter: *Projective Geometry*. Korrigierte 2. Auflage, Springer New York 1994.

[6] R. Diestel: *Graphentheorie*. 2. Auflage, Springer Berlin 2000.

[7] Document Object Model (DOM) Technical Reports. `http://www.w3.org/DOM/DOMTR`. Abruf am 03.10.2004.

[8] G. Eisenreich: *Lineare Algebra und analytische Geometrie*. Akademie-Verlag Berlin 1980.

[9] G. Fischer: *Lineare Algebra*. Durchgesehene 13. Auflage, Vieweg Braunschweig 2002.

[10] W. Gellert (Hrsg.) u.a.: *Kleine Enzyklopädie Mathematik*. 10. Auflage, Bibliographisches Institut Leipzig 1977.

[11] J. Gosling, H. McGilton: *The Java Language Environment*. `http://java.sun.com/docs/white/langenv`. White Paper, Sun Microsystems 1997. Abruf am 02.10.2004.

[12] H.-G. Gräbe: *Geometriebeweise mit dem Computer*. `http://www.informatik.uni-leipzig.de/~graebe/skripte/geometrie03.ps`. Skript zur Vorlesung an der Universität Leipzig im Sommersemester 2003, Abruf am 20.08.2004.

[13] R. Grothmann: Technical Information. `http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann/java/zirkel/index_de.html`. Internet-Seite mit Informationen zur Arbeitsweise des dynamischen Geometrieprogramms Zirkel und Lineal. Abruf am 31.08.2004.

[14] R. Grothmann: Zirkel und Lineal. `http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann/java/zirkel/index_de.html`. Homepage des dynamischen Geometrieprogramms. Download der Version 1.3 am 31.08.2004.

[15] M. Hohenwarter: GeoGebra. http://www.geogebra.at. Homepage des dynamischen Geometrieprogramms. Download der Version 2.4 am 20.09.2004.

[16] M. Hohenwarter: *GeoGebra - Ein Softwaresystem für dynamische Geometrie und Algebra der Ebene*. http://www.sbg.ac.at/did/mathdid/publications/diplomarbeit_geogebra.pdf. Diplomarbeit an der Universität Salzburg 2002. Abruf am 22.09.2004.

[17] Homepage des JDOM-Projektes. http://www.jdom.org. Abruf am 03.10.2004.

[18] O.-H. Keller: *Analytische Geometrie und lineare Algebra*. Dritte, berichtigte und ergänzte Auflage, Deutscher Verlag der Wissenschaften Berlin 1968.

[19] U. H. Kortenkamp: *Foundations of Dynamic Geometry*. http://www.cinderella.de/papers/diss.pdf. Dissertation am Swiss Federal Institute of Technology Zürich, 1999. Abruf am 23.08.2004.

[20] U. H. Kortenkamp, J. Richter-Gebert: Cinderella. http://www.cinderella.de. Homepage des dynamischen Geometrieprogramms. Download der Universitäts-Version 1.4 am 11.08.2004.

[21] U. H. Kortenkamp, J. Richter-Gebert: *Euklidische und Nicht-Euklidische Geometrie in Cinderella*. http://www.cinderella.de/papers/28_NichtEucklidisch.pdf. Abruf am 23.08.2004.

[22] T. Ottmann, P. Widmayer: *Algorithmen und Datenstrukturen*. 4. Auflage, Spektrum Akademischer Verlag Heidelberg 2002.

[23] J. Richter-Gebert, U. H. Kortenkamp: *A Dynamic Setup for Elementary Geometry*. http://www-m10.ma.tum.de/~richter/Papers/PDF/35_DynamicSetup.pdf. Abruf am 10.10.2004.

[24] J. Richter-Gebert, U. H. Kortenkamp: *Dynamische Geometrie: Grundlagen und Möglichkeiten*. http://www.cinderella.de/papers/34_DynamischeGeometrie.pdf. Abruf am 11.05.2004.

[25] Homepage des SAX-Projektes. http://www.saxproject.org. Abruf am 03.10.2004.

[26] M. A. Weiss: *Data structures and problem solving using Java*. 2. Auflage, Addison-Wesley Boston 2002.

[27] H. Winroth: *Dynamic Projective Geometry*. http://www.lib.kth.se/fulltext/winroth990324.pdf. Dissertation an der KTH Stockholm, 1999. Abruf am 21.08.2004.

[28] XML Core Working Group Public Page. http://www.w3.org/XML/Core. Öffentliche Homepage der XML Core Working Group des W3C. Abruf am 03.10.2004.

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

.....

Andy Stock
Leipzig, am 01.11.2004