# Combining Dynamic Geometry, Automated Geometry Theorem Proving and Diagrammatic Proofs

Sean Wilson [1]

*School of Informatics,*
*The University of Edinburgh,*
*United Kingdom*

Jacques D. Fleuriot [2]

*School of Informatics,*
*The University of Edinburgh,*
*United Kingdom*

**Abstract**

This paper outlines *Geometry Explorer*, a prototype system that allows users to create Euclidean geometry constructions using a dynamic geometry interface, specify conjectures about them and then use a full-angle method prover to automatically produce diagram independent, human-readable proofs to theorems. Our system can then automatically generate novel diagrammatic proofs of the forward-chaining and backward-chaining reasoning used by the geometry theorem prover, as well as visualise multiple proofs to single theorems. We discuss the features of our system, how they were implemented and the issues encountered when trying to create diagrammatic full-angle method proofs.

*Key words:* diagrammatic proofs, dynamic geometry, automated geometry theorem proving, full-angle method.

## 1 Introduction

Dynamic Geometry Software (DGS) allows users to create geometry diagrams using standard ruler and compass constructions and explore them interactively in ways not possible with traditional diagrams drawn on paper. The user creates diagrams by first placing unconstrained geometry constructions

[1] Email: sean.wilson@ed.ac.uk
[2] Email: jacques.fleuriot@ed.ac.uk

(such as free points) and then creating dependent constructions (such as lines through pairs of points or line intersection points). The user can then manipulate the diagram, such as moving the position of free points, and explore its properties in a novel way since dependent constructions dynamically update at interactive rates. Implementations of this concept, such as Cabri [13] and The Geometer's Sketchpad [8], have been well received in schools in the last decade since the educational benefits of DGS have been realised. For instance, these systems can aid geometry teaching by having students interactively explore known geometry theorems.

The field of automated Geometry Theorem Proving (GTP), for its part, aims to find a mechanical means of proving geometry theorems and has become one of the most successful areas of automated reasoning. There exists several powerful so-called polynomial techniques, such as Wu's method [17] and the Gröbner basis [9] method, which have been used to prove a huge number of difficult theorems. Although these represent the most powerful known methods for GTP, a major disadvantage is that they produce proofs that consist of hundreds of polynomial terms and tedious algebraic operations that are not humanly-readable. During the mid 1990s, powerful synthetic techniques were developed which used intuitive geometric quantities such as areas [4] and full-angles [5] to produce proofs that were both short and human-readable. These methods combined simple algebraic operations with traditional tree based searches to prove hundreds of non-trivial theorems [16].

In this paper, we describe *Geometry Explorer*, a prototype system that integrates a dynamic geometry interface with a full-angle method [5] prover, both of which were custom-made. We start with a brief overview of the full-angle method in section 2, followed by a discussion of the system interface and features in section 3. In section 4, we look at how full-angle method proofs can be visualised diagrammatically and the problems encountered when these approaches were implemented. Section 5 presents a basic outline of the system architecture, followed by a short survey of related work in section 6. The final section offers our conclusions and future directions.

## 2 The Full-Angle Method

The full-angle method [5] has been demonstrated to prove hundreds of geometry theorems automatically whilst producing proofs which are both short and human-readable [2,3,16]. These proofs include non-trivial examples featuring in American Mathematics Monthly and International Mathematical Olympiad. The full-angle method relies on a single high-level geometric invariant called the *full-angle* to prove theorems.

### *2.1 Overview*

Throughout this paper, we use the convention of capital letters for point names and lowercase letters for line names.

**Definition 2.1** The full-angle between the ordered pair of lines $u$ and $v$ is written as $\angle[u, v]$ and can be thought of intuitively as the rotation required to make the line $u$ parallel to line $v$. Two full-angles $\angle[m, n]$ and $\angle[u, v]$ are equal if there exists a rotation $R$ such that $R(m) \parallel u$ and $R(n) \parallel v$.

**Definition 2.2** For all $u \parallel v$, $\angle[u, v] = \angle[0]$ is a constant and is said to be a flat full-angle.

**Definition 2.3** For all $u \perp v$, $\angle[u, v] = \angle[1]$ is a constant and is said to be a right full-angle.

**Definition 2.4** For any line $w$, addition of full-angles is defined as $\angle[u, v] = \angle[u, w] + \angle[w, v]$.

The following steps are taken to prove a geometry theorem using backward-chaining:

(i) The hypotheses of the theorem is first inserted into what is called the Geometry Information Basis (GIB). This contains a list of geometry facts in *predicate form* which we know to be true about the construction being reasoned about.

(ii) The conjecture to prove is converted into a conjecture equation of full-angles of the form $\angle[0] = \sum (f_i \cdot n_i)$, where $f_i$ is a full-angle and $n_i$ is its integer coefficient.

(iii) The prover then uses conditional rewrite rules with facts from the GIB to substitute full-angles in the conjecture equation with equal expressions of full-angles. For example, $\angle[u, v]$ can be rewritten to $\angle[1]$ if the GIB contains $u \perp v$. Basic properties of full-angles are then applied to simplify the conjecture equation after each substitution, such as $\angle[u, v] + \angle[v, u] = \angle[u, v] - \angle[u, v] = \angle[0]$ and $\angle[1] + \angle[1] = \angle[0]$ .

(iv) A theorem proof is complete when a path of rewrite rule applications is found that transforms the conjecture equation into the form $\angle[0] = \angle[0]$. This can be found using traditional tree-based search techniques.

### *2.2 Backward-Chaining*

We give examples of rewrite rules that are used during backward-chaining and which will feature later in this paper:

**R1** $\angle[AB, CD] = \angle[AB, EF]$ if $CD \parallel EF$.

**R2** $\angle[AB, CD] = \angle[AB, EF] + \angle[1]$ if $CD \perp EF$.

**R3** $\angle[AB, CD] = \angle[AB, CE]$ if $E$ lies on $CD$.

**R4** $\angle[AB, AC] = \angle[AC, BC]$ if $\overline{AB} = \overline{BC}$, where $\overline{XY}$ denotes the length of the line segment with endpoints $X$ and $Y$.

**R5** $\angle[AO, AB] = \angle[AC, BC] + \angle[1]$ if $O$ is the circumcenter of $\triangle ABC$ and $M$ is the midpoint of $AB$. This is because $\angle[AO, AB] = \angle[AO, OM] + \angle[OM, AB]$ (definition 2.4) and we know that $\angle[AO, OM] = \angle[AC, BC]$ and $\angle[OM, AB] = \angle[1]$ are true from the geometry construction.

**R6** $\angle[AB, BC] = \angle[AD, CD]$ if $A$, $B$, $C$ and $D$ are cyclic.

A geometry theorem proof expressed with full-angles has the property of being *diagram independent*. This is not always the case with traditional angles as different instances of a diagram for one theorem may require various cases to be proved depending on the position of the points and how rewrite rules are expressed.

To guarantee termination and optimise the backward-chaining search tree, the full-angle method takes advantages of the *constructive order* of points. The theorem hypotheses must be supplied with the order in which labelled points are introduced in a way that allows a diagram of the hypotheses to be built constructively. For example, given that $M$ was the midpoint of $AB$ and both $A$ and $B$ were free points, a possible constructive order would be to construct $A$, then $B$ and then $M$. Full-angles are eliminated each step in the search by replacing full-angles with ones which contain lower order points. This works because the location of the lower order points on a diagram, and hence the value of the full-angles expressed with them, cannot be dependent on higher order points by definition of the constructive order. The full-angle method can also generate *multiple proofs* to theorems, which is still the case when this heuristic is used.

### *2.3 Forward-Chaining*

For non-trivial theorems, the backward-chaining approach alone is not powerful enough to find a proof. To remedy this, forward-chaining is applied to all the known geometry facts in the GIB to discover other true geometry facts. These new facts are inserted into the GIB and forward-chaining is applied again. This process happens repeatedly until no new facts can be found. Backward-chaining is then used to look for a proof, although it should be noted that the full-angle method is not complete for the class of constructive geometry statements [16]. We will use the following inference rules in later examples:

**F1** If $A$, $B$ and $C$ are collinear then $AB \parallel BC$.

**F2** If $u \perp v$, $u \parallel w$ then $w \perp v$.

**F3** If $M$ and $N$ are the midpoints of $AB$ and $AC$ respectively then $MN \parallel BC$.

**F4** If $O$ is the circumcenter of $\triangle ABC$ then $\overline{OA} = \overline{OB} = \overline{OC}$.

**F5** If $O$ is the midpoint of $CA$ and $AB \perp BC$ then $O$ is the circumcenter of $\triangle ABC$.

## 3 The Prover Interface

Our full-angle method prover originally used a command-line interface where the program would read a theorem description from a custom file format and produce a proof if one was found. Command-line arguments allowed different search techniques to be selected and the user had a choice of text-based or Latex output for proofs. Due to the highly visual nature of geometry, there is great scope for improving the user interfaces of automated geometry theorem provers with respect to describing theorems and exploring proofs. This was investigated by extending the system to include a dynamic geometry interface which will be described next.

### *3.1 Constructing and Manipulating Diagrams*

The dynamic geometry interface (see Figure 1) for the software provides common geometry construction tools (such as midpoints, perpendicular lines and circumcircles) that let the user construct typical Euclidean geometry diagrams. The simple GUI enables the user to create a new geometry construction by choosing an appropriate tool from a toolbar followed by selecting the constructions in the hypotheses diagram window that the new construction will depend on. Free points have no dependencies on other constructions however and so can be placed arbitrarily. New points are automatically labelled so they can be referred to in proofs. After completing a construction, the user can then switch to a tool which lets them move constructions by clicking and dragging the mouse, where dependent constructions will then dynamically update at interactive rates.

To demonstrate how a user can generate and explore proofs with our system, we will use the following theorem as an example throughout the rest of the paper:

**Example 3.1** (Nine Point Circle Theorem) Let $AD$ be the altitude on $BC$ and let the midpoints of the sides $AB$, $BC$ and $CA$ of $\triangle ABC$ be $E$, $F$ and $G$ respectively. Show that $D$, $E$, $F$ and $G$ are on the same circle.

Figure 1 shows an instance of this theorem that has been constructed with our interface. Figure 2 shows two other instances of this diagram which were created by changing the positions of the free points in the original diagram. The theorem holds for each instance and these diagram manipulations allow the user to explore this in a novel way.

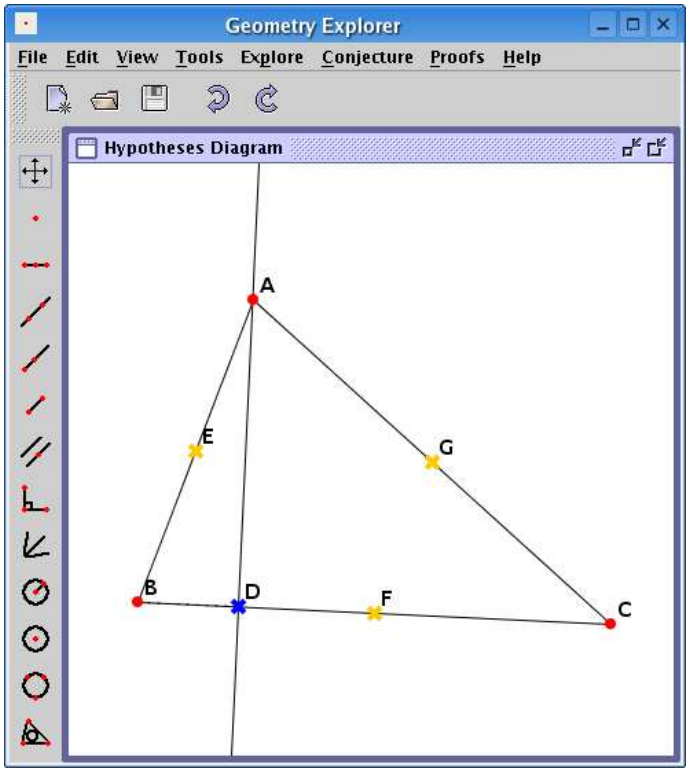


Fig. 1. A screenshot of the Geometry Explorer GUI. The construction tools are shown on the left toolbar and the hypotheses diagram window contains a construction representing the Nine Point Circle theorem.

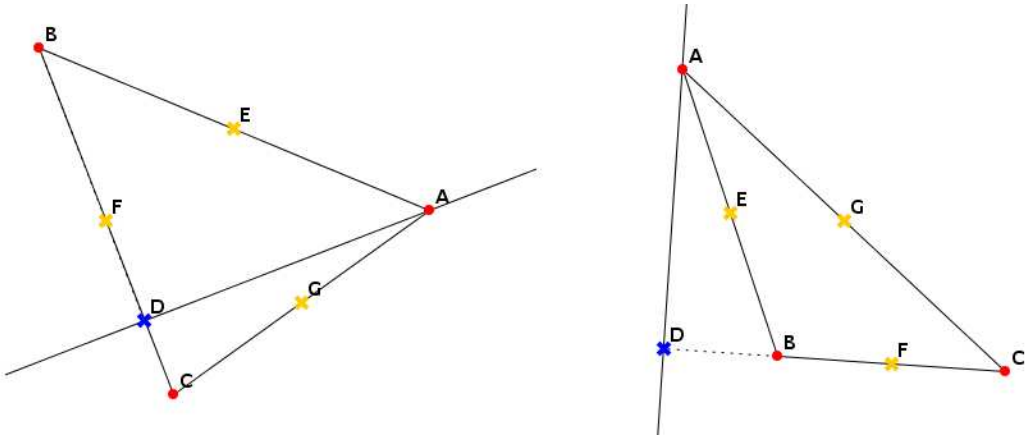


Fig. 2. Screenshots of two instances of the Nine Point Circle Theorem. Notice that on the left diagram, compared to Figure 1, $D$ is now between $F$ and $C$. On the right diagram, $D$ has been moved off the line segment $BC$.

### *3.2 Theorem Proving*

Once a diagram has been constructed, the user can specify a conjecture that they want the system to find a proof for by selecting a conjecture tool from the application menu bar. For instance, they can select the "Prove Four Points are

Cyclic" or the "Prove Three Points are Collinear" tool, followed by clicking the required number of points on the diagram. A dialog box then allows the user to choose which search technique to use, such as depth-first iterative-deepening to find the shortest proof or bounded depth-first search to find solutions with a maximum length. The predicate form of the hypotheses diagram is then supplied to the GTP component along with the stated conjecture, also in predicate form.

If a proof for the conjecture is found, a window will appear showing a Latex document containing a traditional style proof. The user can then access several visualisations of the proof, which will be described in section 4. We note that there is little useful feedback the prover can give about the progress of the proof search except for the current tree depth being examined and the number of nodes explored. The full-angle method is non-interactive, so the user is only given the option of waiting for a proof to be found or to stop the search.

The following Latex proof for the Nine Point Circle Theorem was produced automatically by our system from the diagram in Figure 1 after the conjecture was selected and depth-first iterative-deepening search was chosen. Note that the initial equation comes from a rearrangement of rewrite rule R6 to state the theorem conjecture in terms of full-angles.

**Example 3.2 Proof.**

$-\angle[GE, GD] + \angle[FE, FD]$

$(\angle[GE, GD] = -\angle[GD, DC]$ using R1 ($GE \parallel DC$ (discovered fact)))

$= \angle[GD, DC] + \angle[FE, FD]$

$(\angle[GD, DC] = \angle[DA, CA] + \angle[1]$ using R5 ($G$ is the circumcenter of $DCA$ (discovered fact)))

$= \angle[FE, FD] + \angle[DA, CA] + \angle[1]$

$(\angle[FE, FD] = -\angle[FD, CA]$ using R1 ($FE \parallel CA$ (discovered fact)))

$= -\angle[FD, CA] + \angle[DA, CA] + \angle[1]$

$(\angle[FD, CA] = \angle[DA, CA] + \angle[1]$ using R2 ($FD \perp DA$))

$= \angle[0]$ □

Except for $FD \perp DA$, all geometry facts used in the above proof were discovered in the forward-chaining step of the full-angle method. The system is able to explain these by displaying a list containing every discovered fact, the rule which was used to find each one and the facts with which each rule was instantiated with. We will discuss how this information can be represented diagrammatically in section 4.3.2.

### *3.3 Benefits of a Dynamic Geometry Interface for GTP*

We found that our dynamic geometry interface provided several advantages over the previous command-line interface, which we believe would also apply to GTP systems in general.

#### *3.3.1 Input Error Prevention in the Hypotheses*

It was found that the dynamic geometry interface was useful for detecting and fixing errors in the theorem hypotheses before they were sent to the GTP component. In Figure 1, for instance, an error could have occurred if the user constructed the Nine Point Circle Theorem with $F$ to be the midpoint of $DC$ instead of $BC$. This error would become obvious immediately if the diagram looked wrong or after seeing that $D$, $E$, $F$ and $G$ clearly did not lie on a circle when the diagram was manipulated. This type of feedback would not be available to the user if they had to specify the hypotheses in polynomial or predicate form directly to a GTP program and similar errors would be much easier to make with text-based input. Such basic mistakes could result in the user wasting time either trying to prove a non-theorem or frustrating them as they try to find the error in their theorem file. This also applies to the labelling of points, specifying the construction order and specifying the complete predicate form of a diagram, which the software does automatically, preventing further user mistakes.

#### *3.3.2 Exploring Conjectures*

When the user constructs a diagram, they can easily use the facilities of the software to manipulate the construction to see if one of their conjectures appears to be true in different instances of that construction or experiment to find new conjectures. If a conjecture is false, the user can quickly discover a counter-example. This type of exploration is not possible with text-based theorem entry or with static diagrams.

However, there is a danger that exists where people start to take a dynamic diagram as a proof of a theorem because they cannot find a counter-example whilst manipulating it, as this does not mean one cannot exist. As the user can only explore a finite number of configurations of a dynamic diagram, it can never replace a rigorous mathematical proof which would show that a conjecture will always hold true for the infinity of diagram instances. Combining GTP with DGS allows users to explore conjectures as well as obtain formal reasoning that explains theorems.

### *3.4 Integration Issues*

Once a diagram has been constructed and a conjecture specified, the system is then responsible for converting this information into a format, the aforementioned predicate form, that the GTP component can understand. This presented several integration issues due to the numerous ways that users can

construct a diagram for proving the same theorem.

### *3.4.1 Generating the Predicate Form*

In the implementation, each geometry construction is responsible for producing a predicate form statement that explains how it was constructed. One aspect of generating the full diagram description is knowing which predicate statements are meaningful and being aware of how the same predicate form can come from different constructions.

For example, no useful geometric statement can be made about a circle constructed with point $O$ as its centre and with point $A$ on its circumference since a circle can be constructed with any pair of distinct points. But, if we then construct point $B$ as being a point on this circumference also, we can then state that $\overline{OA} = \overline{OB}$. If we add another point, $C$, to the circumference, we can then state that $\overline{OA} = \overline{OB} = \overline{OC}$ and $O$ is the circumcenter of $\triangle ABC$.

As the supplied construction tools allow the same diagram to be drawn in different ways for usability reasons, care has to be taken that each approach will result in the same final list of predicate statements. For example, a user could give the previous construction in the order given or they could construct the circumcircle of $\triangle ABC$ directly and then add the centre point $O$.

### *3.4.2 Removing Superfluous Hypotheses Points*

The sum of full-angles (definition 2.4) has to be used to find a proof to most theorems in the backward-chaining stage. As one can see, it is not strictly a rewrite rule as it introduces fresh variables on the right-hand-side. Since this rule introduces arbitrary points from those labelled in the hypotheses, the search space size increases rapidly with the number of points available. As a user can place points that are superfluous to the theorem being specified, it is important from a usability point of view to avoid penalising them with slower search times or forcing them to construct a diagram with the minimum number of points. A heuristic to ignore extraneous points is to create the set of all points in the conjecture statement, $C$, and only keep the points in the hypotheses which are members of $C$ or points on which members of $C$ are dependent. This works in all example theorems that we have tried [3], as the last points in the constructive order are always members of $C$, although we have not verified if this will ever prevent a proof from being found in the general case.

## 4 Visualising Full-Angle Method Proofs

This section describes various visualisations of full-angle method proofs that can be generated automatically by our system. We also discuss the issues encountered while attempting to generate diagrammatic proofs.

### *4.1 Visualising Multiple Proofs*

The full-angle method is able to generate multiple proofs for a given theorem. Each of these proofs may use an alternative path of reasoning or give a different insight into the construction being reasoned about. However, viewing multiple proofs of one theorem in the form of a list of separate algebraic proofs does not allow the reader to see the uniqueness of each proof or the similarities between proofs - the reasoning is usually lost in the similar looking rewrite rules and algebraic terms.

To make the exploration of multiple proofs easier, our system allows the user to view all proofs found for one theorem in the form of a graph. Multiple proofs are collected with constraints put on the search space, such as setting a maximum proof length. See Figure 3 for an example of some of the shortest proofs found for the Nine Point Circle Theorem. This visualisation shows a subgraph of the search space that the full-angle method prover navigates and is produced automatically. It displays all the proofs found by the prover by only showing the path of rewrites rules and visited nodes that lead from the initial state node to the goal node. In the example, notice that it is easy to see interesting aspects of the multiple proofs, such as how all proofs share a common rewrite rule, how two paths only differ by the ordering of the rewrite rules used and how one path is unique.

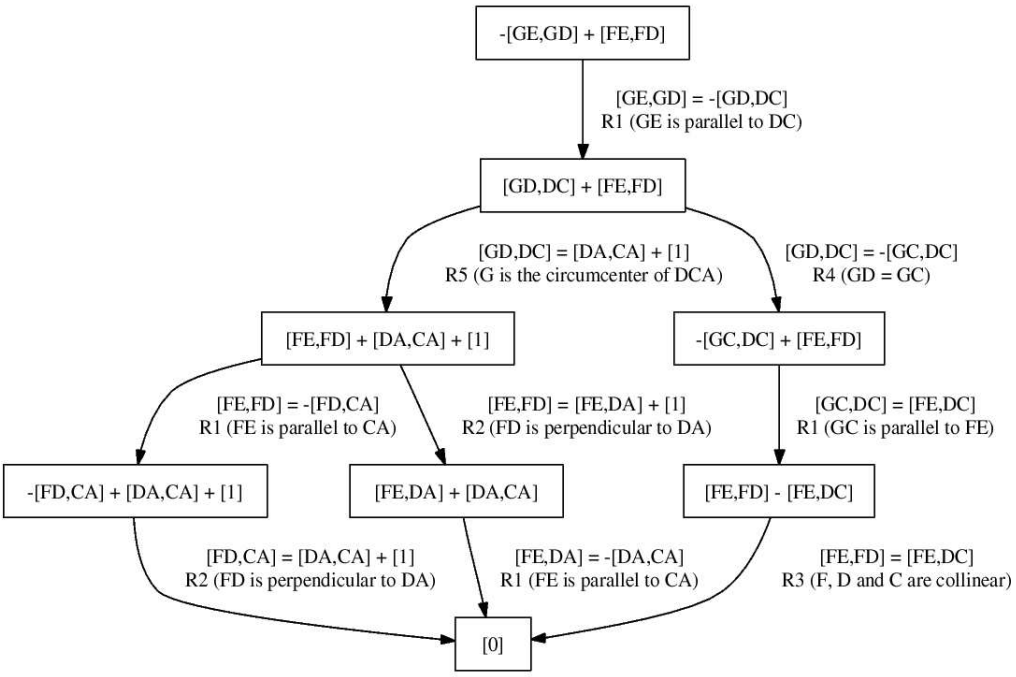


Fig. 3. A screenshot of a visualisation showing some of the multiple proofs found for the Nine Point Circle Theorem.

### *4.2 Visualising Geometric Properties*

Since our system had access to a diagram of the hypotheses, we wanted a means of highlighting geometry statements on it to help explain proofs to the user. This could be used to show the user a non-obvious circle that was discovered or to let them see a full-angle that is being referred to in a proof. This is far more convenient and powerful than trying to locate features on a diagram by inspection. For example, to find a full-angle on a diagram requires the reader to locate the position of the four points of the full-angle, find the intersection of the lines and then mentally rotate the first line into the other to find the angle described.

#### *4.2.1 Implementations Details*

The software can highlight diagram properties that the prover says are true by overlaying extra dynamic constructions onto the diagram (see Figure 4). For instance, visualising the statement $\overline{AB} = \overline{BC}$ is achieved by constructing $AB$ and $BC$ as coloured line segments and constructing short perpendicular bisectors through these. This highlights the statement with traditional notation and makes it stand out on the diagram. As this highlighting updates dynamically with the rest of the diagram, it provides an additional visual tool that lets the user explore geometry properties in various instances of a construction. For example, by manipulating point positions the user can explore how all points on a discovered circle stay on that circle in multiple diagram configurations.

Note that this part of the system must blindly trust the results from the prover. For example, highlighting a statement that four points are cyclic proceeds by constructing a circumcircle through the first three points, where the circumcircle will implicitly pass through the fourth cyclic point.

#### *4.2.2 Issues with Geometry Notation*

To visualise forward-chaining reasoning, we needed an unambiguous representation of each fact in the GIB as a diagram. While trying to highlight various facts contained in the GIB, it became apparent that standard geometric notation was inadequate for visualising many of them. For example, there is no standard way to draw attention to three specific points being collinear on a diagram. This case was resolved by placing a marker circle around each point and drawing a coloured line segment through them. Circles and circumcircles are dealt with in a similar fashion.

To differentiate between congruent lines and parallel line segments, we use the convention of putting a single and double dash through both lines respectively. Indicating midpoints by only marking the congruent line segments is not sufficient with a dynamic diagram as it does not convey the three points involved are collinear, so we combine the notation for collinear points and congruent line segments. Line pairs in perpendicular statements in the GIB

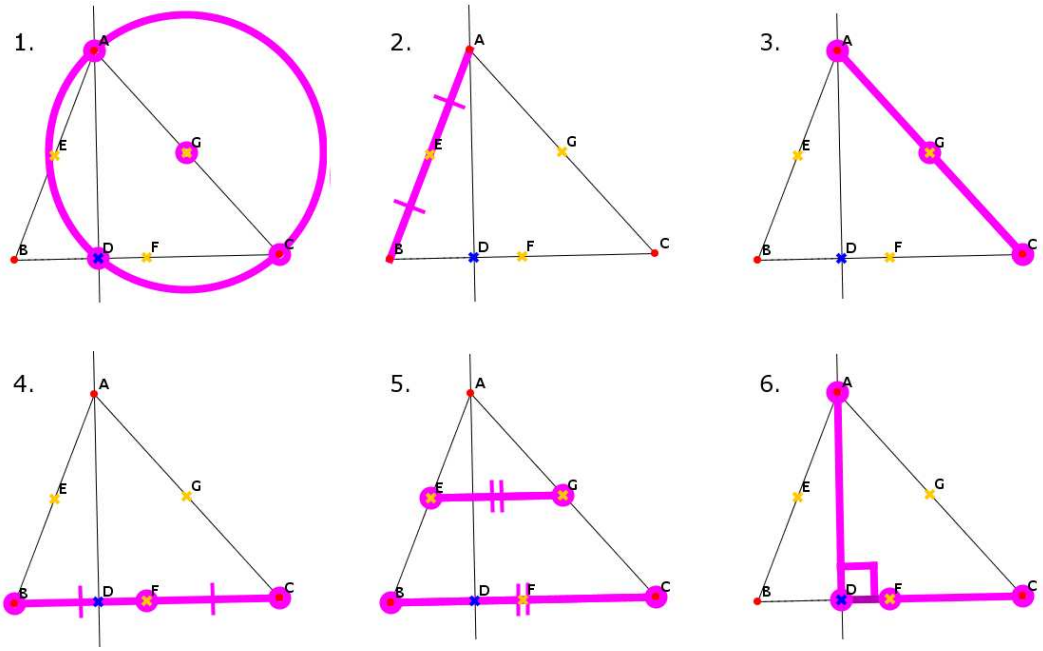


Fig. 4. Each diagram shows a screenshot visualising one specific geometry fact about the Nine Point Circle Theorem: 1. $G$ is the circumcenter of $\triangle ACD$. 2. $\overline{AE} = \overline{BE}$. 3. $A$, $G$ and $C$ are collinear. 4. $F$ is the midpoint of $BC$. 5. $EG \parallel BC$. 6. $AD \perp FC$

commonly do not intersect so it is necessary to extend both lines toward the intersection point before the traditional right-angle symbol is marked. See Figure 4 for examples of the above.

### *4.3 Visualising Geometric Reasoning*

As full-angle method proofs are expressed using high-level geometry invariants, they lend themselves to geometrical visualisation, which cannot be done in an obvious way with polynomial GTP methods. Our system has novel ways of visualising the full-angle method proofs. These take advantage of the diagram independent property of the proofs as each is still valid for different diagram instances.

#### *4.3.1 Visualising Forward-Chaining Reasoning as a Graph*

We first visualised the forward-chaining reasoning using a graph, where each node was labelled with a fact from the GIB and edges were tagged with rule names to indicate inference. To see where a new fact came from, the user can trace how it was derived by following the path of inferences until only hypotheses are reached. This is a useful tool but it is hard to relate textual geometry facts to a separate diagram.

*4.3.2 Diagrammatic Forward-Chaining Reasoning*

The software was altered so each node in the graph would contain a rendering of the hypotheses diagram, with the geometry fact for that node visualised diagrammatically using the method discussed in section 4.2. The aim was to create a purely diagrammatic proof. See Figure 5 for an example, which shows a selection of non-obvious geometry facts found in the Nine Point Circle theorem by forward-chaining.

If the user clicks on a discovered geometry fact on the graph, it is highlighted in the window containing their original hypotheses diagram. They can then manipulate and explore the construction with this new fact explicitly marked. All the diagrams in the forward-chaining graph also update as the hypotheses diagram is changed, allowing them to easily view diagrammatic proofs for different diagram instances. Although not intended for theorem discovery, the forward-chaining used in the full-angle method does find non-obvious facts about constructions. For this reason, the user is given the option to view all facts in the GIB using the above visualisation so that they can explore features of the original diagram they were not aware of.

To make the diagrammatic proofs more concise, one issue that had to be dealt with is the inference caused by rule F1. For example, given that the GIB initially contains the statement that $A$, $E$ and $B$ are collinear, R1 automatically implies that $AE \parallel AB \parallel BE$. This manifests itself as three parallel facts for each collinear fact in the GIB. Not only does this clutter the graph, as multiple collinear facts feature in virtually all theorems, but (in this example) thinking of $AE$ as being parallel to $AB$ is unnatural and confusing. There is also no obvious way to visualise these overlapping line segments on a diagram in a clean way. However, these inferences are vital as several rules need to use these parallel facts.

These fact nodes were eliminated at the proof presentation stage by merging each collinear fact node with its three corresponding parallel fact nodes into a single node representing the collinear statement. Rules which use any of the derived parallel lines will link to merged node, but they still make sense as the collinear statements are just a special case of parallel lines.

*4.3.3 Visualising Backward-Chaining Reasoning as a Graph*

Presenting a backward-chaining full-angle method proof as a series of rewrite rule applications and simplified equation pairs is sufficient to explain the proof, but this does not emphasise the chains of rewrites applied to the same terms or which terms cancel one another in a clear way. Our system allows a novel view of backward-chaining proofs to visualise these properties.

The backward-chaining proofs were first visualised as graphs, where nodes represented a single full-angle term each and edges were used to show how a full-angle was rewritten or how it was eliminated. If a full-angle $a$ is rewritten to full-angle $b$, we draw an edge from $a$ to $b$ with an "=" node in the middle of it. If a full-angle $a$ is rewritten to $b + c$, we draw a forked edge connecting

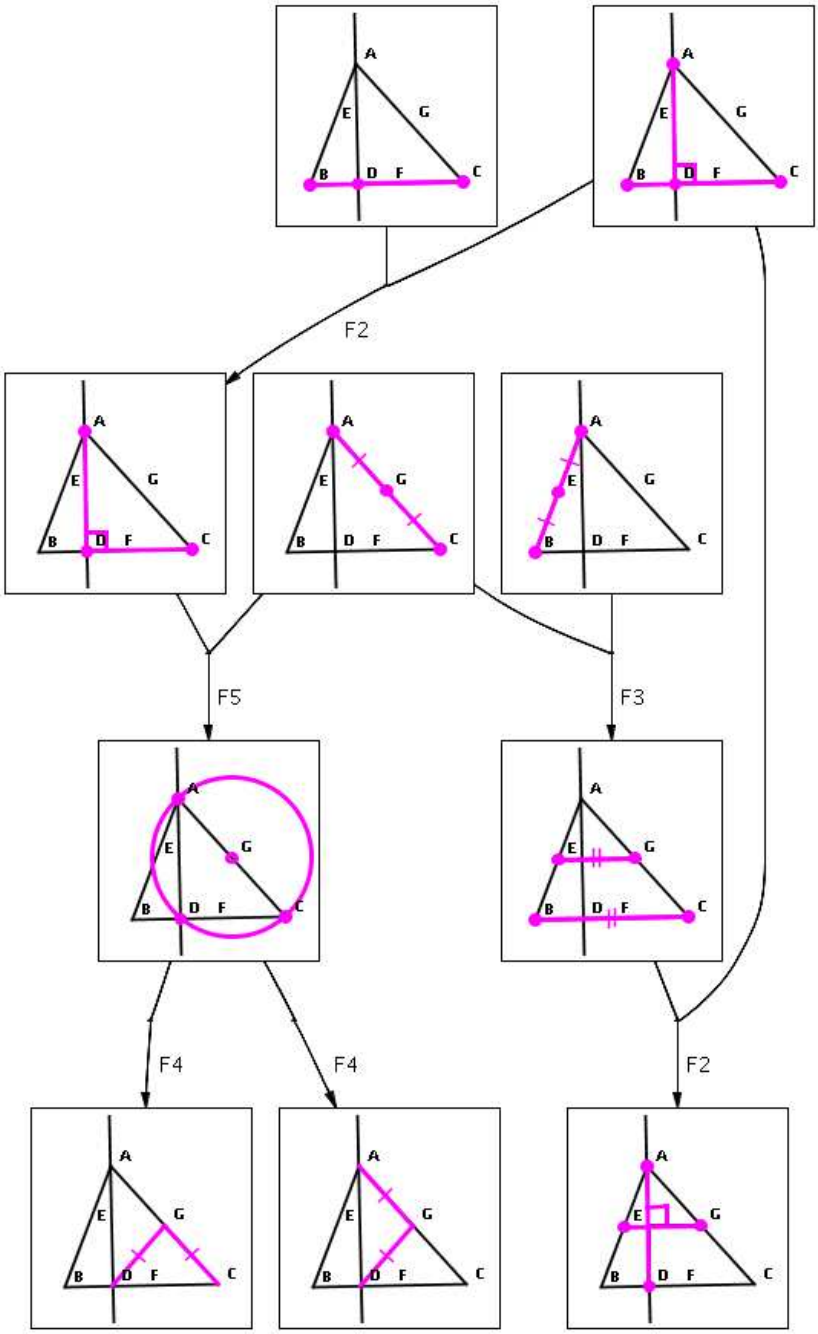


Fig. 5. A screenshot of a diagrammatic proof of several non-obvious properties of the Nine Point Circle theorem which were found with forward-chaining.

$a$ to both $b$ and $c$ with a "+" node in the middle of it. If the full-angles $a + b$ simplify to give $c$, we draw a forked edge connecting both $a$ and $b$ to $c$ with a "+" node in the middle of it. Edges are labelled to describe where terms come from and how they are transformed, which is either from the rewrite rule applications, cancellations, or from the conjecture statement.

The conjecture is shown as a $\angle[0]$ node with edges joining it to the full-

angles in the starting conjecture equation. A valid proof to the conjecture is shown if all directed paths from this $\angle[0]$ node end at another $\angle[0]$ node, meaning that all full-angles were eliminated. This visualisation is sufficient for describing all full-angle method proofs as each equation only uses sums of full-angles.

This visualisation emphasises the chain of rewrite rules applied to the same terms, where chains of full-angles linked with only one edge represent equal full-angles. Term cancellation and how each rewrite aids the progress of the proof as terms are eliminated is also much clearer.

#### *4.3.4 Diagrammatic Backward-Chaining Proofs*

Our next step was to take each node and replace them with a rendering of the hypotheses diagram. Full-angles were visualised by marking both lines, marking their start and end points, tracing the two lines to their intersection point, and then drawing an arrow to indicate the anti-clockwise rotation required to make the first line parallel to the second. This creates a novel diagrammatic view of the backward-chaining proofs (see Figure 6).

For each rewrite rule, we originally intended to highlight the geometry facts it used on the full-angle term nodes it created. Unfortunately, this emphasised the problem that most full-angle method rewrite rules are *not* easy to visualise. For example, visualising the sum of full-angles (definition 2.4) is very unintuitive diagrammatically. However, we hold the view that this diagrammatic visualisation of backward-chaining is a valuable tool for investigating full-angle proofs.

## 5 Overview of System Architecture

Our full-angle method theorem prover was implemented using Sicstus Prolog and was the result of a rational reconstruction of the original full-angle method prover by Chou et al [5]. It is able to produce automatic proofs for about 100 theorems from a paper describing 110 sample theorems proved by their system [3].

Our custom-made dynamic geometry component and GUI was implemented using Java with Sisctus's Java/Prolog interface, Jasper, used to interface the Java front-end to the Prolog back-end. The prover outputs traditional style proofs in Latex format and outputs the various graph visualisations in Graphviz [12] format. These graphs, after being laid out automatically by GraphViz, are then rendered in Java using Graphviz's Grappa package.

After finding a proof and generating the graph visualisations, the prover attaches meta-data to nodes and edges about what geometry facts they represent. Using Grappa, this meta-data is retrieved so that geometry statements can be visualised, for example, when viewing the diagrammatic forward-chaining proofs. This is done by associating each Prolog geometry predicate with a suitable set of dynamic geometry constructions that highlight that

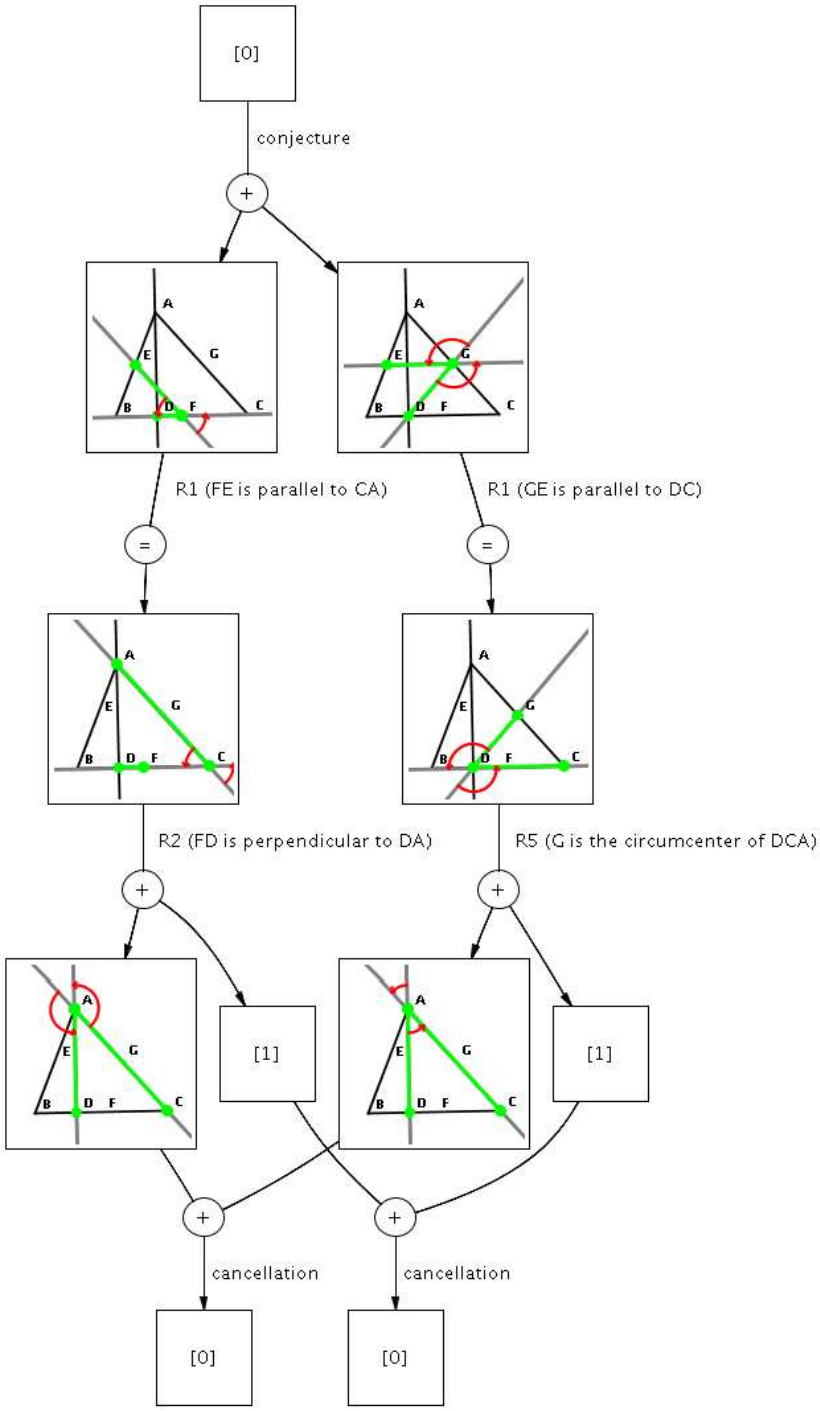


Fig. 6. A screenshot of a diagrammatic visualisation of the backward-chaining proof shown in section 3.2 for the Nine Point Circle Theorem. The top three nodes show the conjecture equation, the nodes at the bottom show term cancellations and the others show rewrite rule applications.

predicate in the dynamic geometry component. After Grappa has drawn the basic graph elements, individual diagrams are superimposed onto the graph using the meta-data to highlight the relevant features.

## 6 Related Work

DGS offers major benefits to the field of GTP and previous work has combined the two in a multitude of ways. GeoView [1] is a tool which can take a text-based Pcoq theorem statement and produce a dynamic geometry diagram from it as a visual aid to the user. Our system works in the opposite direction and Bertot et al commented [1] that a user constructed diagram as the input mechanism for the theorem prover would offer usability advantages and be more concise than text input.

MMP/Geometer [7] is a powerful geometry theorem proving tool which implements Wu's method, the area method and the deductive database method [6] and provides several ways for the user to describe theorems. In addition to using diagrams as an input method, it also supports text-based input in polynomial form, predicate form, constructive form and a pseudo-natural language. When a diagram is not supplied, MMP/Geometer can also generate one. It does not, however, use diagrams in combination with any of its GTP methods to visualise proofs.

Cinderella [10] is a dynamic geometry package which is notable for its inclusion of a randomised theorem checker [15]. This can identify geometry theorems about a given construction by searching for instances of counter-examples to conjectures in such a way that the probability of it not finding a counter-example for a non-theorem is very low. The algorithm used is fast and has a guaranteed maximal running time, so Cinderella uses this to discover non-trivial geometry theorems from a diagram as it is constructed, although proofs cannot be provided. It would be interesting to combine this with a human-readable GTP method that could provide proofs to theorems as the theorem checker discovers them. Of note is an extension to Cinderella [14] which takes the last theorem identified by Cinderella's theorem checker and automatically proves it with the Gröbner basis method.

## 7 Conclusions and Future Work

This paper demonstrates that a dynamic geometry interface offers major benefits to GTP systems in regard to ease of use and the invaluable ability for the user to visually explore theorems and proofs. Our system takes advantage of the diagram independent nature of full-angle method proofs and combines this with dynamic geometry to offer an interface that allows geometry theorems to be specified easily and each proof step to be investigated geometrically.

We explored several novel ways to view the forward-chaining and backward-chaining reasoning used in full-angle method proofs, as well as looking at how

to visualise the multiple proofs produced for each theorem. Finally, we tried to use the visual nature of geometry to produce purely diagrammatic proofs with the full-angle method. These worked surprising well for the forward-chaining proofs as the rules used were simple and intuitive, but they are not powerful enough for difficult theorems by design. The rules used in backward-chaining proofs were found to be unintuitive to visualise and our work here must only be viewed as a tool for investigating full-angle proofs. Despite this, all of the visualisations presented offered insights into full-angle method proofs that were not obvious from traditional proof documents.

A GTP method that only used reasoning that is diagrammatically intuitive would be desirable in producing more accessible, easier to understand geometry proofs. An interesting extension would be to look at diagrammatic reasoning with the deductive database [6] approach, which uses similar, natural forward-chaining reasoning to the full-angle method, but with more powerful inference rules as a discovery and proving mechanism.

## 8 Acknowledgements

This research was supported by an EPSRC DTA studentship and EPSRC Grant GR/S01771/01.

## References

[1] Bertot, Y., F. Guilhot and L. Pottier, *Visualizing geometrical statements with Geoview*, in: *UITP*, 2003.

[2] Chou, S.-C., X.-S. Gao and J.-Z. Zhang, *Automated solution of 135 geometry problems from A.M.M.*, Technical Report 94-10, Department of Computer Science, The Wichita State University (1994).

[3] Chou, S.-C., X.-S. Gao and J.-Z. Zhang, *A collection of 110 geometry theorems and their machine proofs based on full-angles*, Technical Report 94-4, Department of Computer Science, The Wichita State University (1994).

[4] Chou, S.-C., X.-S. Gao and J.-Z. Zhang, *Automated generation of readable proofs with geometric invariants, I. Multiple and shortest proof generation*, Journal of Automated Reasoning **17** (1996), pp. 325–347.

[5] Chou, S.-C., X.-S. Gao and J.-Z. Zhang, *Automated generation of readable proofs with geometric invariants, II. Theorem proving with full-angles*, Journal of Automated Reasoning **17** (1996), pp. 349–370.

[6] Chou, S.-C., X.-S. Gao and J.-Z. Zhang, *A deductive database approach to automated geometry theorem proving and discovering*, Journal of Automated Reasoning **25** (2000), pp. 219–246.

[7] Gao, X.-S. and Q. Lin, *MMP/Geometer - a software package for automated geometry reasoning*, in: F. Winkler, editor, *Automated Deduction in Geometry* (2004), pp. 44–66.

[8] Jackiw, N., "The Geometer's Sketchpad," Key Curriculum Press, Berkeley (1990).

[9] Kapur, D., *Using Gröbner Bases to reason about geometry problems*, Journal of Symbolic Computation **2** (1986), pp. 399–408.

[10] Kortenkamp, U., "Foundations of dynamic geometry," Ph.D. thesis, ETH Zürich (1999).

[11] Kortenkamp, U. and J. Richter-Gebert, *Using automatic theorem proving to improve the usability of geometry software*, in: *Mathematical User Interfaces*, 2004.

[12] Koutsofios, E. and S. C. North, "Drawing graphs with dot," AT&T Bell Laboratories, Murray Hill, NJ (1993).

[13] Laborde, J.-M. and F. Bellemain, "Cabri-Geometry II," Texas Instruments, Dallas (1993).

[14] Roozemond, D., *Automatic geometric theorem proving* (2003), Bachelor Project, Eindhoven University of Technology.

[15] Schwartz, J. T., *Probabilistic algorithms for verification of polynomial identities*, in: *EUROSAM '79: Proceedings of the International Symposiumon on Symbolic and Algebraic Computation* (1979), pp. 200–215.

[16] Shang-Ching Chou, J.-Z. Z., Xiao-Shan Gao, "Machine Proofs in Geometry: Automated Production of Readable Proofs for Geometry Theorems," World Scientific, 1994.

[17] Wu, W.-T., *Basic principles of mechanical theorem proving in elementary geometrics*, Journal of Automated Reasoning **2** (1987), pp. 221–252.