Table of Contents

Polygon Ratio Problem: Many Proofs, One Degree of Freedom

Part 1. Synthetic geometry (auxiliary lines)

Part 2. Analytic geometry proof

Part 3. One more degree of freedom (ellipse view)

Part 4. Vector analysis

Part 5: Groebner(Gröbner) basis and Wu Method: polynomial elimination methods

Part 6. Modern Geometric Prover

Part 7. Aristotle prover

Polygon Ratio Problem: Many Proofs, One Degree of Freedom

Author: Changchun Shi

Date: First Version Jan 12, 2026 Last update Jan 25, 2026

Introduction

During the Christmas holidays, a basic geometry proof problem showed up in my class alumni chat (Figure 1 is the original screenshot from classmate Lin He). He asked about the second question as the first of the two is almost trivial, and that set the whole discussion in motion. I have always liked Euclidean geometry and have exceled in most of teenage subject competitions (I don't recall failed to proof any of the geometry problems in my competition career). Problems like this rarely stop at one proof. Draw one line and a ratio drops out; draw another and a different argument appears. What follows is a cleaned collection: one quick sine-law proof, several synthetic proofs, an analytic-geometry proof (with help of ChatGPT 5.2-thinking for easy life, edited for clarity), and a one-parameter generalization that reveals deeper structure. I hope it gives younger students a sense of how much structure a single diagram can hide. I also plan to use this pilot problem to introduce the SOTA artificial intelligent ways of solving such problems in seperate notes.

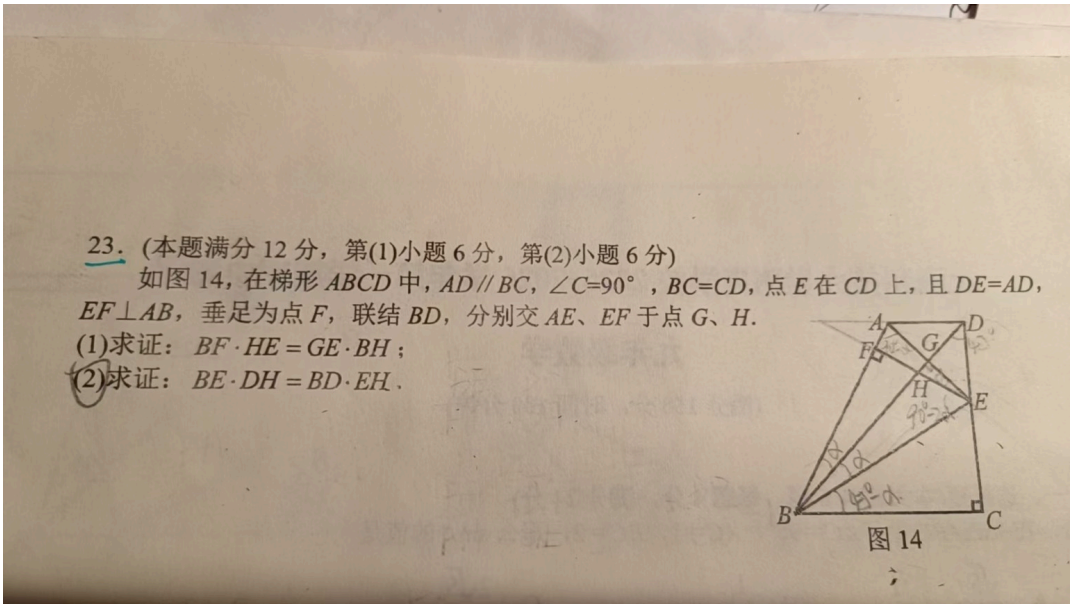

Figure 1: The original problem (photo from the chat).

Let me first restate the problem in words and redraw the diagram more cleanly.

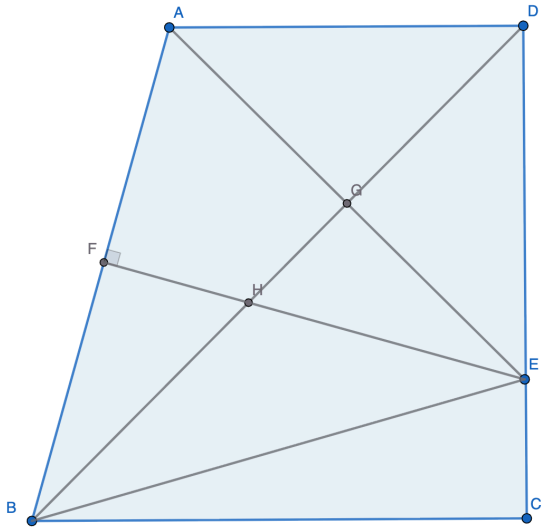

Figure 2: A clean redraw in GeoGebra.

Problem statement (translated). In trapezoid , , °, and . Point lies on , and . Through , draw with as the foot of the perpendicular. Draw ; it intersects and at points and , respectively. Prove:

1. .
2. .

Back to TOC

Part 1. Synthetic geometry (auxiliary lines)

The basic equal angles and lengths are shown in Figure 3.

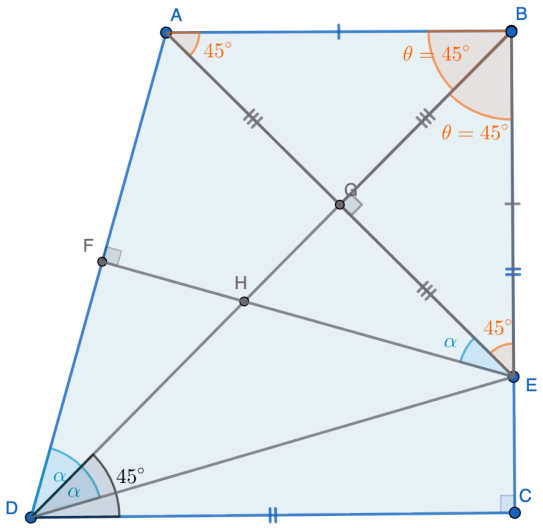

Figure 3: Basic angle and length relations.

Question (1) is a warm-up; it follows from similar triangles visible in Figure 3. For part (2), a quick start is the sine law. For a triangle with side lengths opposite vertices , we have

This gives Proof 0 immediately.

Proof 0 (sine law)

> The ratios and are both ° °, so they are equal. This proof uses no auxiliary lines.

To avoid assuming the sine law, I drew auxiliary lines. Rather than separate diagrams per proof, the next figure collects all extra points. Each added point carries a subscript that matches the proof number (for example, $_1$ is used only in Proof 1).

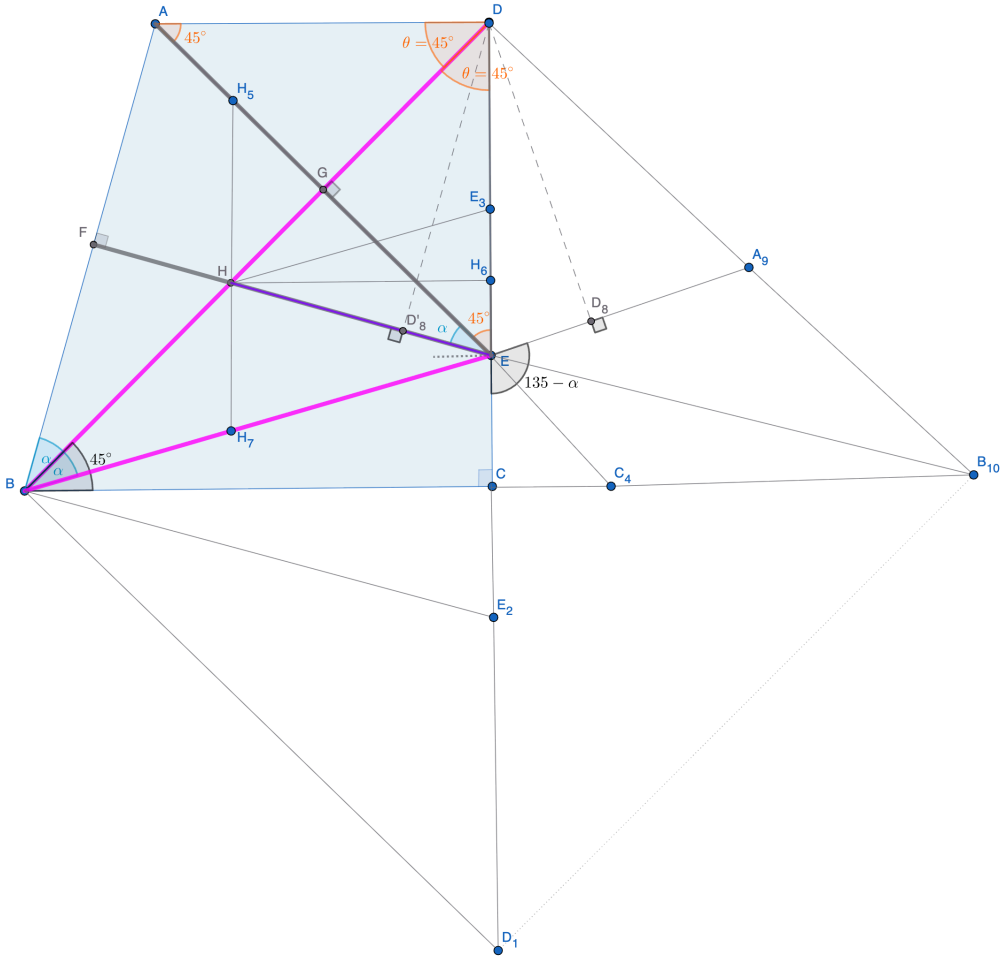

Figure 4: All auxiliary lines used in the synthetic proofs.

Below are brief proof sketches. Each can be completed as a standard middle-school geometry exercise.

Proof 1

> Extend to $_1$ so that $_1$ is symmetric to with respect to . Then $_1$, so the desired ratio follows.

Proof 2

> Extend to $_2$ so that $_2$ is symmetric to with respect to . Then $_2$, so the desired ratio follows.

Proof 3

> From draw a line parallel to and intersect at $'$. Then $'$. The rest follows.

Proof 4

> Extend to meet at $_4$. Then $_4$. Also $_4$, so the ratio follows. (Credit: Prof. Xueheng Lan.)

Proof 4.1

> Use the reflection property: the ray from to reflects across and passes through . This gives the same angle relation at , so the ratio follows.

Proof 5

> Draw a line from parallel to and intersect at $_5$. Then $_5$, and $_5$, which leads to the result.

Proof 6

> Draw a line from parallel to and intersect at $_6$. Then $_6$.

Proof 7

> Draw a line from parallel to and intersect at $_7$. Then $_7$.

Proof 8

> The distances from to and are equal (use $_8$ and $'_8$; these are non-essential since $^\circ$). The distances from to and are also equal. So area to area can be expressed as $_8$ $'_8$, and also as . Hence they are equal.

Proof 9

> Extend to $_9$ where $_9$. Then $_9$, $_9$, and $_9$ $^\circ$. Since bisects $_9$, we get $_9$ $_9$. The rest is trivial.

Proof 10

> Extend to $_{10}$ where $_{10}$ intersects the extension of line . Then $_{10}$ is the reflection of across (since is the reflection path of light at mirror). One can show $_{10}$ $^\circ$, so $_{10}$ $_{10}$.

Proof 11

> This one also needs no auxiliary lines: notice and , and , so the result follows.

A note on degrees of freedom: the only adjustable angle in the original configuration is °. If °, then and overlap and the result is trivial. If °, then lies on the extension of , and lies on the extension of . The roles of and swap, and the same style of arguments still works. Similarly, one can show that a ray from to reflects across and reaches .

Other proofs

By now we should be done. However, if you followed the solutions so far, you likely noticed two patterns: (1) we have essentially drawn a square $_{10}\ _{1}$ in Figure 4, and many proofs are based on various points on this square; (2) polygon is symmetric about , which is why we could leverage and in several steps. It follows that any proof obtained by extending lines in the original diagram has a mirror proof across . These are indicated again by subscripts, and they give Proofs 12, 13, 14, 15, and 16. They form the symmetric rectangle $_{15}\ _{12}$ and are shown in Figure 5.

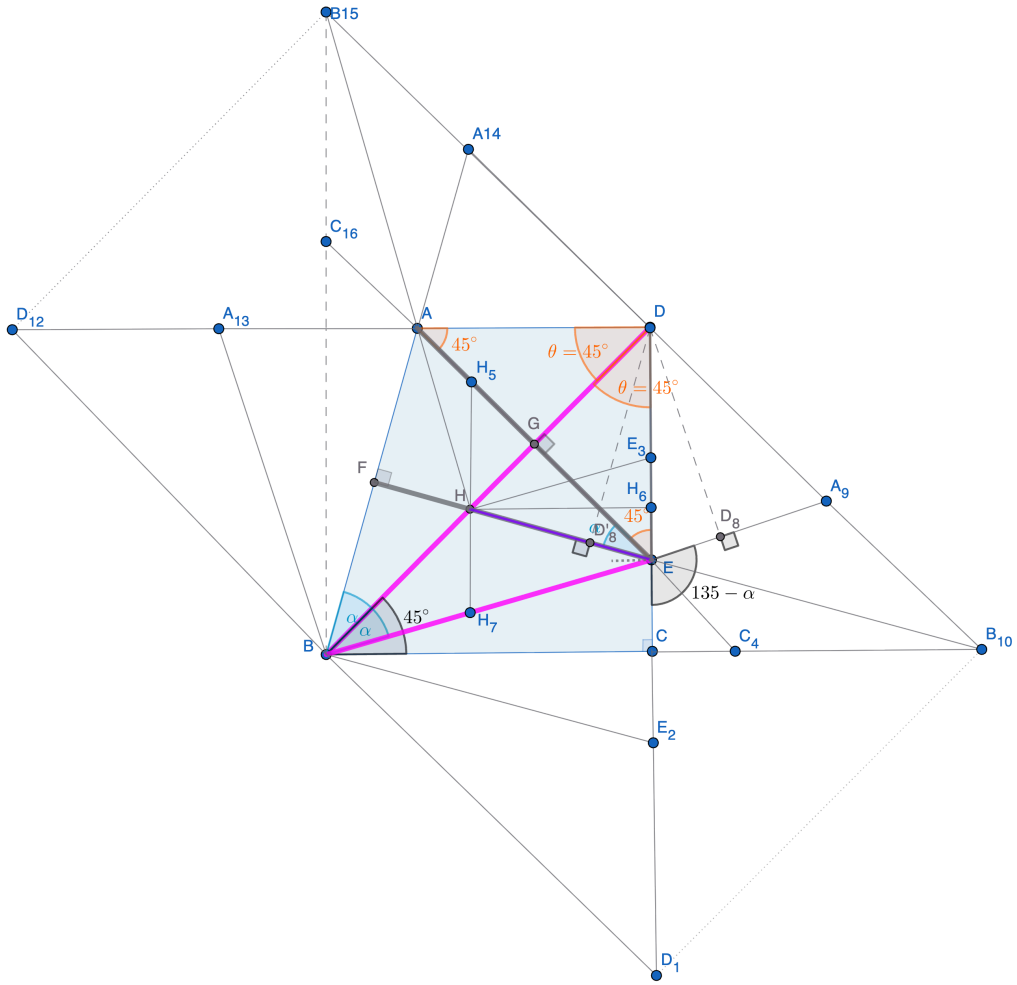

Figure 5: Five additional symmetric proofs.

Finally, several proofs such as Proofs 3, 5, 6, and 7 focus on what happens inside polygon . Each has a symmetric counterpart (Proofs 17, 18, 19, 20) reflected across . For clarity, these are not displayed.

Back to TOC

Part 2. Analytic geometry proof

This is a short coordinate proof.

Goal

Proof

> Setup. Let , , . Then implies when and with .
>
> Lines. . The slope of is , so has slope and passes through : .
>
> Intersection. Solve to get
>
> Common factor.
>
> Hence
>
> Also and .
>
> Finish.

Conclusion: .

Back to TOC

Part 3. One more degree of freedom (ellipse view)

The ° angle is special. If we relax that angle, the clean right-angle structure disappears, but the problem still has a simple geometric core: reflection on an ellipse. The key insight is to generalize the reflection: the ray from to reflects across and heads to . This reveals an ellipse that reflects light from one focus to the other at any point on it. That reflection property is equivalent to the constant-sum-of-distances definition of an ellipse.

Construction

Let be the ellipse with foci and . Take a point on , with closer to . Draw the tangent line at , and let it intersect at . Take a point on , the reflection of with respect to the major axis. Through draw the line , and choose so that . Draw , and let it intersect at . In general, is not °, but when °, °. The resulting shape is shown in Figure 6.

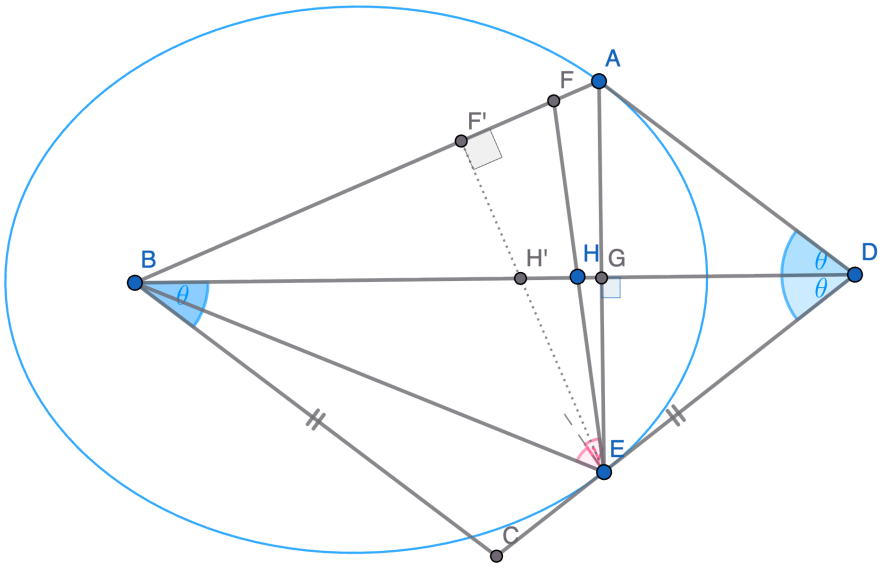

Figure 6: Ellipse construction with foci and .

Proof 3.1 (reflection property)

> By the reflection property of the ellipse (a light ray from to reflects and then passes through), we obtain
>
> ——— ———
>
> Reflect across line to ′. Then

$$\frac{}{} \quad \frac{'}{'} \quad \frac{}{}$$

which is the same proportion.

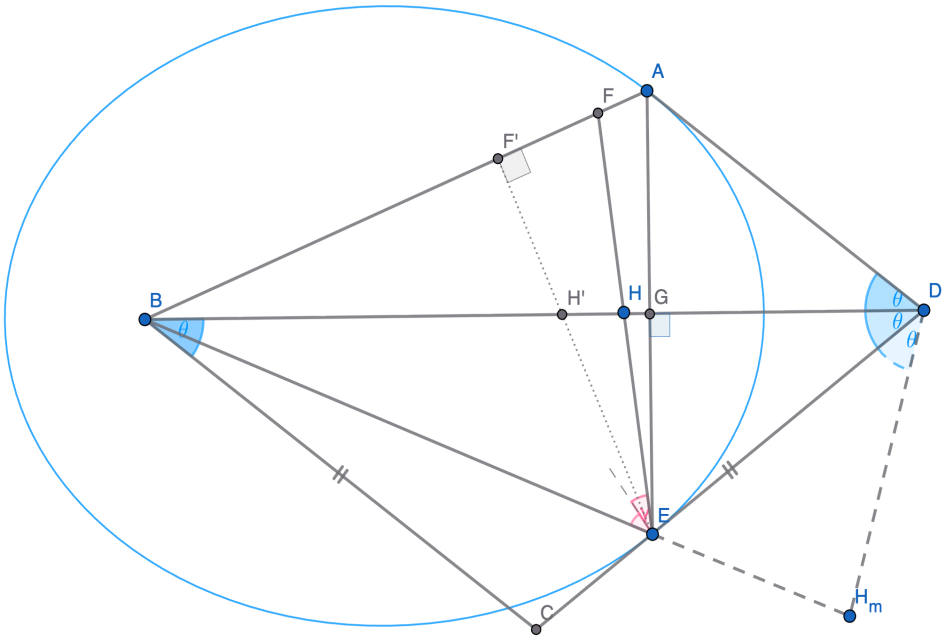

Figure 7: Reflection proof for the ellipse construction.

In this generalized setup, is no longer perpendicular to . In fact, the perpendicular line through intersects and at ′ and ′, as shown in Figure 7. Here ′ differs from and does not satisfy the ratio property. The original problem is special because the ° angles create a surplus of right angles. The single degree of freedom here is essentially the ellipse eccentricity, with and determined by the right angle . When the ellipse degenerates to a circle (foci coincide), the identity becomes obvious.

In this setup, points , , , and , and the lines through them, are not even essential; only the foci and , and the tangent line , are necessary, with on the major axis. This shows the true simplicity of the original problem, as shown in Figure 8 with the pure geometric approach.

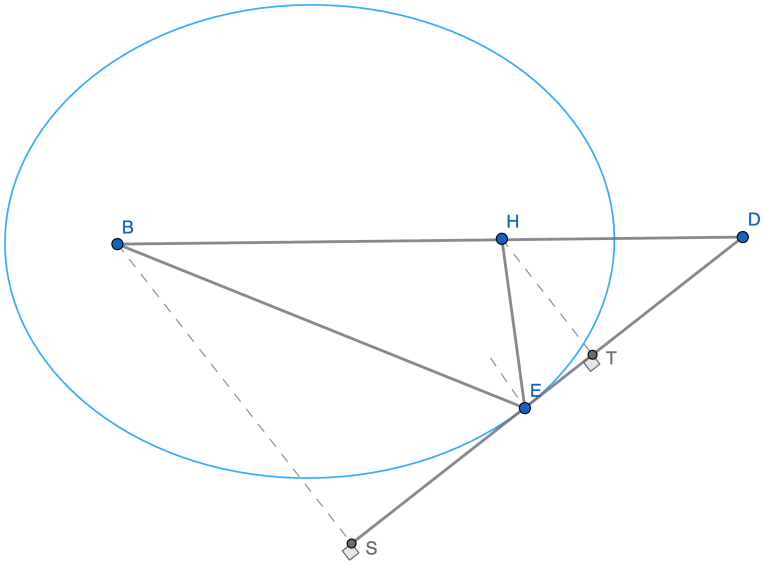

Figure 8: The simplified ellipse setup, which reveals the essence of the original problem.

The diagram also yields a second, similarity-based proof.

Proof 3.2, similarity

> Let and be the feet of the perpendiculars from and to the tangent line . By the reflection property, , so . Likewise, , so . Combining gives .

Interestingly, we can also compute the ratio analytically in the ellipse model. It turns out to be simplest using the de La Hire (eccentric-angle) parameter, revealing the geometry transparent.

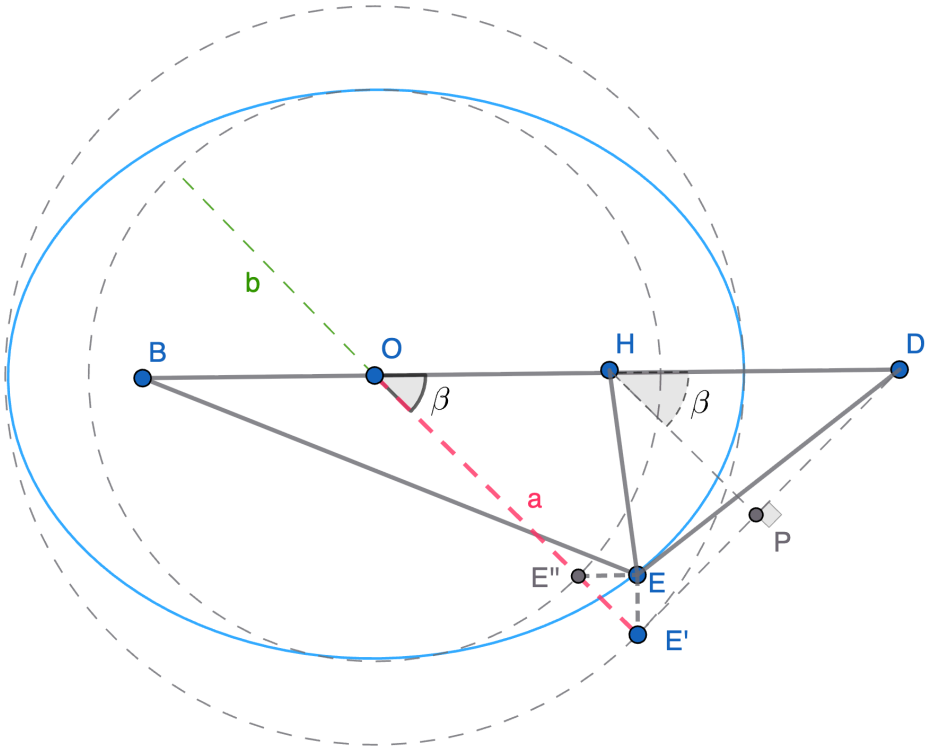

Figure 9: de La Hire parameters and outer-circle view with , ′, and ″.

Proof 3.3 (de La Hire + outer circle)

Let the ellipse be ^{2} ^{2} ^{2} ^{2} with center , foci and , and ^{2} ^{2} ^{2}. Use the de La Hire (eccentric-angle) parameter :

(up to swapping). Let be the tangent at and .

From the tangent formula

$$\frac{^{0}}{^{2}} \quad \frac{^{0}}{^{2}} \qquad \_\_\_\_\_ \quad \_\_\_\_\_$$

setting gives $_{D}$.

Geometrically, the affine map $\frac{a}{b}$ sends the ellipse to the outer circle ^{2} ^{2} ^{2} and sends to ′ . Since only is scaled, and ′ lie on a line perpendicular to and share the same -coordinate. Let ″ be the intersection of the line through parallel to with the outer circle, so and ″ share the same -coordinate (Figure 9). The map preserves tangency, so ′ is right and .

Therefore

$$\_\_\_\_ \qquad \_\_\_\_$$

and

Corollary 1 (axis-focus scaling; inner structure of the ratio)

From $\frac{a}{\cos\beta}$ $\frac{a-c\cos\beta}{\cos\beta}$ and , we get

This shows that the tangent-axis lengths are just the focal lengths scaled by the same factor (see Figure 9). The main identity is exactly the cancellation of that common scale.

Corollary 2 (focus-to-outer-tangent distances)

Corollary 1 gives , and geometrically is exactly the perpendicular component of to ′, i.e., ′ (see Figure 9). Hence

′ ′

The simple consistency check

′ ′ ′

accounts only for the sum. The real content is that each focus-to-tangent distance equals its matching focus-to-point distance; this rigidity is encoded by the de La Hire parameter and the ellipse's affine relation to the circle.

Proof 3.4 Outer circle proof

Once Corollary 2 is in hand, let and ′ be the feet of the perpendiculars from and to ′ (for simplicity Figure 9 only shows). Then ′ (right angles at ′ and a common angle at), so ′ ′ ′, recovering the same ratio.

Finally, there is a fun fact. Note that the feet of the perpendiculars drawn in Proof 3.2, and , are exactly the intersection points of the tangent line with the ellipse's outer circle. A quick proof is to show that (Figure 10). Let ′ be the reflection of across ; then is the midpoint of ′. Since is the midpoint of , we have $\frac{1}{2}$ ′. But ′ , hence . The same reflection with gives , so the fun fact is that and are precisely the two intersections of with the outer circle.

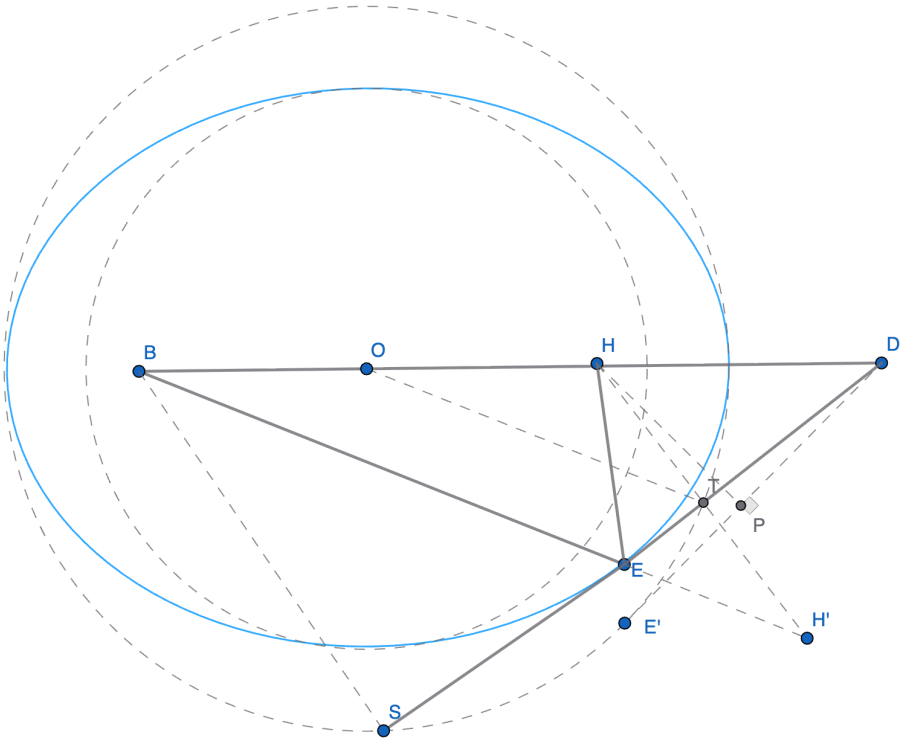

Figure 10: additional properties of T and S

Back to TOC

Part 4. Vector analysis

Once one length is fixed, the trapezoid configuration has a single degree of freedom: the angle . In the broader ellipse construction, a second parameter (the tangent point/slope) appears, and the same ratio follows from the reflection property. The special case ° is exactly the original problem; that is why so many proofs exist, and why each auxiliary line exposes the same invariant ratio in a different way. Part 3 frees the dimension while keeping the reflection property, and that is what ultimately shows the simplicity behind the original picture.

In this Part of the note, we will focusing on using some other mathematical tools useful in geometry to solve our proof. The problem to proof is almost trivial: DH:DB = EH:EB, given light BE reflects to the EH direction by mirror DE. Part 3 has explained extensively of the solutions. But it is still good to show some of the power of such tools.

Since Euclid (~300 BCE), geometry was developed in a synthetic style: one reasons directly with the figure using congruence, similarity, tangency, and auxiliary lines. That is the mindset behind Part 1. A major second language arrived with Descartes (1637): analytic geometry turns geometric constraints into algebra once you choose coordinates, matching the style of Part 2.

This arc also mirrors the historical split between geometry and algebra in physics and mechanics. Newton, in the *Principia* [NEWTON-PRINCIPIA], favored coordinate-free arguments because they were the clearest language of rigor in 1687, while modern physics often relies on coordinates for universal computation. Analytic geometry existed by Newton's time, but the algebraic language for differential equations matured later; Newton's methods therefore look Euclidean in flavor, mixing auxiliary lines with limiting arguments. Several later authors also gave elementary, coordinate-free proofs (see [KEPLER-TEACH], including Feynman's famous "lost lecture"). It is good for students to appreciate both techniques.

Physics naturally blends geometry with differential ideas. Here we keep that discussion light and focus on the geometric tools that sit just below calculus.

In the 1800s, geometry gained an especially efficient algebraic interface: vector analysis, where the dot and cross products encode the two most basic geometric primitives--projection/angle and oriented area. The modern and notation was introduced by Gibbs (and independently Heaviside) around 1881 and then standardized in physics usage. Later, divergence and curl became standard because they compactly describe fields and their integral laws--but that differential layer is not our focus here.

That same "area backbone" explains the next conceptual step. Grassmann (1844) introduced the wedge product and exterior algebra, where the fundamental geometric object is no longer just a vector but also a bivector (directed area), trivector (directed volume), and so on. Clifford

(1878) then built geometric algebra by unifying these objects into one coherent multiplication system. In our 2D Euclidean setting,

are simply three notations for the same invariant: the area spanned by and the tangent direction . So the proofs are interchangeable across these languages with no change in geometric content.

Finally, modern physics is "geometric" in a deeper sense: it naturally lives on spaces that are not just 2D pictures--3D space, 4D spacetime, and even higher-dimensional configuration/phase spaces. We don't always call those "geometry" in everyday language because they're hard to visualize, but mathematically they are geometry: they carry intrinsic notions of angle, area/flux, invariance, and coordinate-free meaning. In our ellipse problem, we see a toy version of the same idea: at each point on the ellipse, there is a tangent line . Thinking of the tangent line as the attached object that moves with is the simplest concrete example of the bundle/fiber mindset.

A brief timeline of tooling relevant to this note:

- ~300 BCE: Euclid -- synthetic geometry
- 1637: Descartes -- analytic geometry
- 1687: Newton -- Euclidean geometry + limiting arguments
- 1827: Gauss -- intrinsic geometric invariants (surfaces)
- 1844: Grassmann -- wedge / bivectors (area as an object)
- 1878: Clifford -- geometric algebra (bivectors/trivectors as first-class)
- 1881: Gibbs/Heaviside -- standard dot/cross notation
- 1899: Cartan -- exterior derivative/forms (beyond our needs here)

For readers familiar with Maxwell's equations, the table below shows how mathematical language for geometry evolved over time: from coordinate-expanded component equations, to vector analysis, to geometric algebra, and then to differential forms. The tools change, but the physics does not, even as the expressions become more compact.

Formulation	Coordinate status (what's really true)	Maxwell equations written in that language	What you gain / lose	Good reference
Maxwell (1865, component-heavy)	Coordinate-invariant laws, but written in components (explicit x, y, z bookkeeping) and bundled with extra relations (constitutive laws, Ohm,	Not packaged as the modern four; a larger coupled component system (often summarized as ~20 equations/unknowns, plus auxiliaries).	Gain: historically close to derivations. Lose: compactness; structure is obscured by bookkeeping.	Maxwell's 1865 paper [MAXWELL-1865]; Deschamps [FORMS-EM].

Formulation	Coordinate status (what's really true)	Maxwell equations written in that language	What you gain / lose	Good reference
	forces, potentials).			
Vector calculus (Heaviside/Gibbs)	Coordinate-free in content; coordinates enter only when writing component formulas.	(SI, vacuum) $\nabla \cdot \mathrm{E} = \rho/\varepsilon_0$ $\nabla \cdot \mathrm{B} = 0$ $\nabla \times \mathrm{E} = -\partial_t \mathrm{B}$ $\nabla \times \mathrm{B} = \mu_0 \mathrm{J} + \mu_0 \varepsilon_0 \, \partial_t \mathrm{E}$.	Gain: four clean equations; practical in 3D Euclidean space. Caveat: $\nabla\cdot, \nabla\times$ depend on the chosen metric + orientation, though invariantly defined once fixed.	Wilson [VECTOR-ANALYSIS]; cross product [CROSS-PRODUCT]; vector analysis [VECTOR-ANALYSIS-WIKI].
Geometric algebra (Clifford; dot + wedge unified)	Coordinate-free once metric + orientation are fixed. Vectors are bold (e.g., a, b); higher-grade objects (bivectors, trivectors) are not vectors in disguise.	Core algebra: $\mathrm{ab} = \mathrm{a} \cdot \mathrm{b} + \mathrm{a} \wedge \mathrm{b}$. Maxwell can be written compactly (common spacetime GA convention): $\boxed{\nabla \mathrm{F} = \mu_0 \mathrm{J}}$, with the grade split $\nabla \mathrm{F} = \nabla \cdot \mathrm{F} + \nabla \wedge \mathrm{F}$ giving $\nabla \wedge \mathrm{F} = 0$ (homogeneous) and $\nabla \cdot \mathrm{F} = \mu_0 \mathrm{J}$ (sources).	Gain: cross product becomes derived (dual of $\wedge$); div/curl unify as one operator split by grade; rotations often cleaner. Lose: conventions vary (3D GA vs spacetime GA; units/signatures); less standard in mainstream EM courses.	Peeter Joot [GA-EM].
Differential forms (exterior calculus on spacetime)	Manifestly coordinate-invariant; works naturally on curved manifolds (GR). Metric enters via the Hodge star.	$\boxed{\mathrm{dF} = 0}$, $\boxed{\mathrm{d}{*}\mathrm{F} = \mathrm{J}}$. Here $*$ maps k-forms to $(\mathrm{n} - \mathrm{k})$-forms using the metric + orientation (in 4D, $2 \leftrightarrow 2$).	Gain: topology/relativity are transparent; $\mathrm{d}^2 = 0$ exposes structure. Lose: abstraction up front; computations often unpack to components.	Deschamps [FORMS-EM]; Lindell [LINDELL-DF]; Epstein [EPSTEIN-DG].

Table 1 summarizes the evolution of Maxwell's equations across these languages.

Notes:

- Geometric algebra (Clifford; dot + wedge unified): metric + wedge fused immediately via
  .
- Differential forms (exterior calculus on spacetime): forms are wedge-first, metric-later via .

So the concept of vector analysis has evolved significantly since Newton's time. Maxwell's equations were first written in a coordinate-expanded form, which has the flavor of our solution

in Part 2, along with differential operators that were by then well understood. Only later did divergence and curl become standard, giving a more compact, coordinate-free expression using and . Those operators are coordinate-free in meaning, which parallels the shift from the coordinate solution in Part 2 back to the coordinate-free constructions in Part 1. Vector analysis keeps the geometric insight while still enabling the algebraic computations of the coordinate approach. To explain the power of this toolbox, I include a couple of short proofs below.

Since our problem is 2D, both dot and cross products can be used to solve it naturally. We emphasize how these tools enable coordinate-free proofs while still allowing analytic methods beyond the purely Euclidean geometry used in Part 1, yet more direct than the coordinate approach in Part 2.

Proof 4.1 (reflection on DE)

> Use the reflection property: a ray from to reflects across and passes through . Let , , and define unit vectors , .
>
> Then and , with . The tangent at is perpendicular to , so its direction is parallel to . Hence
>
> for some scalar . Because ,
>
> Using gives
>
> Matching coefficients yields and , so $\frac{p}{p-q}$. Finally,
>
> , so
>
> which is equivalent to $\frac{BE}{BD}$ $\frac{EH}{HD}$.

Proof 4.2 (using the cross product)

> Let the tangent at meet at , and take any nonzero tangent vector .
>
> The reflection law says the tangent bisects the angle between and , so the perpendicular components match:

Now and . Cross with and drop the term:

So

Because are collinear, , hence

Substitute to get

which is equivalent to $\frac{BE}{BD} \quad \frac{EH}{HD}$.

I explored several alternative constructions which give various other proofs that I won't detail here since these two are the simplest and representative. For instance, in the original configuration, the condition immediately yields the dot-product constraint . This orthogonality can be used to reduce the degrees of freedom without appealing to any reflection argument. More generally, one can start from essentially any point in the diagram, parameterize the remaining configuration by a small set of length (or scaling) parameters, and then drive the argument to the target length-ratio identity. There are many such routes, and they do not require introducing any auxiliary lines.

In this 2D Euclidean setting, the geometric-algebra approach does not introduce additional structure compare with the cross product. Both encode the oriented (signed) area of the parallelogram spanned by two vectors. The only makes traditional sense in 3D space, resulting in a vector perpendicular to the plane, the operator gives a bivector that's definited universally for d-D space and the bivector doesn't need to utilize the 3rd dimension. Likewise, the differential-forms formulation is not fundamentally different in this context: a 2-form takes two vectors as input and returns a scalar, and 1-form vector is essentially difine the dot product of any vector with a given vector; so the resulting identities reduce to the same area or determinant computations already used in the cross-product argument. For completeness, Proof 4.3 shows how they are united and replaced in the same proof. Be aware that in the modern math/physics which often utilize dimensionality beyond 3, the dot and cross way become hard, which is way we had to separate time t (another dimension strictly speaking) from

the space in order to write the equation as "coordinate free". And that's why you no longer see any use of t in the unified maxell equation using geometric algebra and differential form.

Proof 4.3: Unified simple solution (area language: cross = wedge = 2-form = 1-form)

> Let be the tangent at and let . For any vector based at (here), the area with the tangent is the same in all languages:
>
> where is an oriented area 2-form.
>
> Fix and define the 1-form $_t$, so and . This is the perpendicular component of to the tangent.
>
> Reflection at gives equality of perpendicular components:
>
> Since ,
>
> Using bilinearity and gives
>
> Hence
>
> Because are collinear, , so $$\frac{BD}{HD}=\frac{BE}{EH},\quad\text{i.e.}\quad \frac{BE}{BD}=\frac{EH}{HD}.

By now I hope this part of the note have conveyed an appreciation for coordinate-free methods in our mathematical toolbox. I also hope it gives a preview of what modern physics use to solve geometry problems that are more than 3-dimensional, admittedly in a weak sense.

So far in this part we discussed the simple case of how the reflection property implies the ratio equality. Below we show why the original problem in Part 1 gives the reflection property of and , using a vector argument.

Proof 4.4 Vector analysis: EF is the reflection of BE across CD in Figure 3

Let and . Then and .

Fix the unit normal ———. Since , , and ,

there exists a scalar such that

Reflection across is equivalent to

Because , the direction of is parallel to , so the right-hand side equals

Now compute the two normalized quantities. First,

so ^{2}. Next, by Lagrange's formula

and ,

But

^{2}

hence ^{2}. Finally, , and

^{2} ^{2} ^{2} ^{2} ^{2} ^{2} ^{2} ^{2}

so . Therefore the normalized cross magnitudes match, and the reflection condition holds:

So reflects across into .

We used Lagrange's formula (the vector triple product identity) in the proof above. There are many known proofs,some classic ones pure algebraic, but this one that I came up is purely geometric and makes the identity feel almost unavoidable. I haven't seen this particular "Thales-circle + chord" intuition presented in such a compact form else where, so it seems worth sharing though it can hardly be original given this is elementary. In a sense, it brings the argument back to a Euclid-style picture: analysis and geometry are not rivals, but two languages for the same structure, constantly translating into each other.

Proof of Lagrange's formula based on only geometric properties

Let be the origin and , , . Write ∥ ⊥ relative to the plane . Then ⊥ (since ⊥) and ⊥ ⊥ , so it suffices to assume are coplanar. By linearity of both and in each argument, scaling by multiplies both sides by ; hence it suffices to prove the unit case . Let .

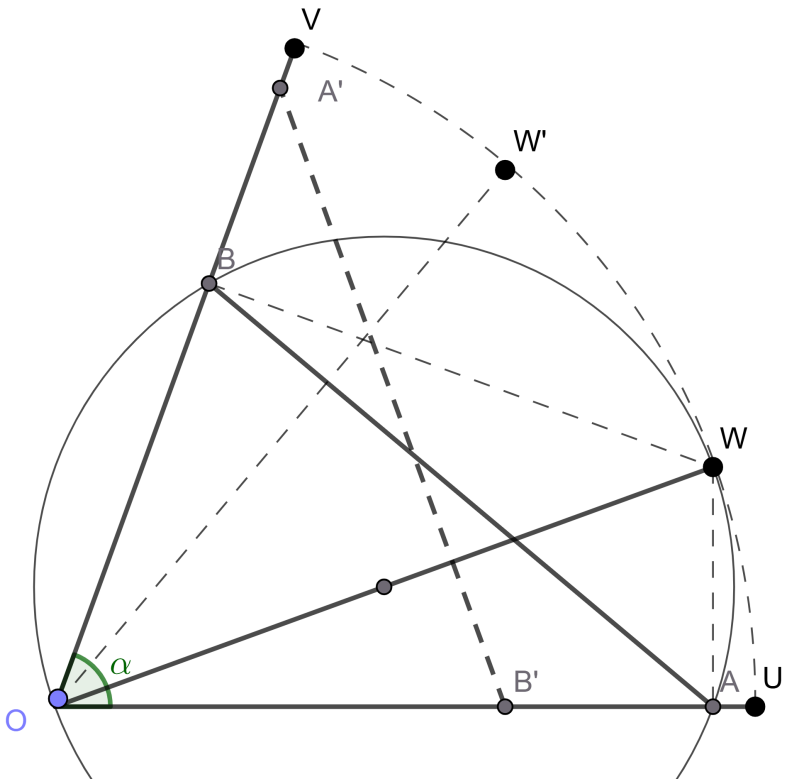

Figure 11: Geometric property of the triple vector formula.

For the plane geometry, refer to Figure 11. Draw the circle with diameter . Let it meet rays again at . By Thales, °, hence and . Define ′ and ′ by ′ and ′ . Then

> , ,
>
> so , is the magnitude of the right-hand side. It is easy to show , ,
> by checking , , but here we want a pure geometric proof. Since
> , , , °, we get , , , matching the left-hand side
> direction. Since , , we have , , . In the diameter-1 circle,
> . But
>
> so both sides are the same vector:
>
> So basically we used the multilinearity of the dot and cross products (it is enough to prove the identity for coplanar unit vectors). We also used that dot and cross magnitudes correspond to cos and sin of the angle. The rest relies on a circle fact: two chords have the same length if and only if their inscribed angles are equal.

Postscript

It can be interesting to see how the following software would approach the problem:

1. Medal-level AI geometry solvers (closest to "gold/silver" performance in the International Math Olympiad)

- AlphaGeometry / AlphaGeometry 2 (research systems that can solve many Olympiad problems)
- Open source software such as Newclid/yuclid by startup company Harmonic

2. Geometry theorem provers (strong automation, not always Olympiad-style proofs)

- GCLC / WinGCLC (area method, Wu's method, Grobner bases)
- JGEX and Sympy (Wu/Grobner/full-angle methods with visualization)
- OpenGeoProver (Wu/Grobner style)
- GeoLogic (interactive Euclidean prover)

3. Proof assistants (formal, but manual)

- Lean 4 + mathlib
- LeanGeo

References

- [CROSS-PRODUCT] Gibbs/Heaviside (1881). "Cross product." Summary: Notation for dot/cross products introduced in 1881. URL: https://en.wikipedia.org/wiki/Cross_product

- [EPSTEIN-DG] Epstein (2013~2014). "Differential Geometry Minicourse." Summary: Minicourse notes associated with a 2013 workshop; book version appeared in 2014. URL: https://marceloepstein.weebly.com/uploads/1/2/8/4/12849112/dg_minicourse.pdf (context: https://www.isbn.de/buch/9783319357140/differential-geometry)
- [EUCLID-VIDEO] Syversen (2025-09-18). "Euclid's ruler-and-compass constructions." YouTube. Summary: Video on the role of straightedge-and-compass constructions in Euclid. URL: https://youtu.be/M-MgQC6z3VU
- [FORMS-EM] Deschamps (1981). "Electromagnetics and Differential Forms." Summary: Proc. IEEE article that sparked broader EM use of differential forms. URL: https://faculty.washington.edu/seattle/physics544/2011-lectures/deschamps.pdf
- [GA-EM] Joot (2018). "Geometric Algebra for Electrical Engineers: Multivector Electromagnetism." Summary: Notes introducing geometric algebra for electromagnetism (version dated April 2018). URL: https://peeterjoot.com/archives/math2018/GAelectrodynamics.V0.1.6.pdf
- [KEPLER-TEACH] van Haandel & Heckman (2007). "Teaching the Kepler laws for freshmen." arXiv:0707.4605. Summary: Pedagogical treatment of Kepler's laws with geometric insights. URL: https://arxiv.org/abs/0707.4605
- [LINDELL-DF] Lindell (2004). "Differential Forms in Electromagnetics." Summary: Lecture notes/book with a 2004 copyright. URL: https://faculty.washington.edu/seattle/physics544/2011-lectures/lindell.pdf
- [MAXWELL-1865] Maxwell (1865). "A Dynamical Theory of the Electromagnetic Field." Summary: Original formulation of Maxwell's equations in component-heavy form. URL: https://archive.org/details/electromagnetic00maxwgoog
- [NEWTON-PRINCIPIA] Newton (1687). *Philosophiae Naturalis Principia Mathematica.* Project Gutenberg. Summary: Foundational work in classical mechanics with Euclidean-style geometric reasoning. URL: https://www.gutenberg.org/ebooks/28233
- [QUATERNION] Hamilton (1843). "Quaternion." Summary: Quaternions were introduced by Hamilton in 1843. URL: https://en.wikipedia.org/wiki/Quaternion
- [VECTOR-ANALYSIS] Wilson (1901). *Vector Analysis* (from Gibbs' lectures). Summary: Classic text that standardizes dot/cross notation and vector calculus. URL: https://archive.org/details/vectoranalysis00wilsrich
- [VECTOR-ANALYSIS-WIKI] Wilson/Gibbs (1901). "Vector Analysis." Summary: Wikipedia entry on the 1901 textbook and its role in standardizing notation. URL: https://en.wikipedia.org/wiki/Vector_Analysis

Back to TOC

Part 5: Groebner(Gröbner) basis and Wu Method: polynomial elimination methods

So far, in Parts 1–4, we've followed a mostly pure-math storyline: from Euclid's synthetic geometry, through analytic and algebraic viewpoints, all the way to the modern language of vectors, forms, and (eventually) geometric algebra—ideas that were already in very solid shape by the early 20th century. A natural next question is: what happened in roughly the last 50 years for solving Euclidean geometry problems? The surprise is that the "next big step" is not a new axiom system, but a new attitude: turn geometry into an algorithm—a replicable, programmable procedure for symbolic reasoning. In Euclidean geometry this becomes especially concrete: translate points, lines, circles, and incidences into variables, and translate geometric constraints into polynomial equations. Once everything becomes polynomial, a proof turns into a computable elimination problem. In this part we introduce two early but seminal algebraic approaches to automated Euclidean theorem proving: Wu's method (first) and the Gröbner basis method (second).

Wu's method (Wen-Tsün / Wenjun Wu, late 1970s). Wu's method—often presented through characteristic sets (triangular-style elimination)—was developed by Wen-Tsün Wu in the late 1970s and became one of the first highly successful, systematic methods for mechanical theorem proving in elementary geometry [WU-ORIG] [WU-CHAR]. The key move is explicitly analytic: choose coordinates for "free" points, convert each geometric relation (collinearity, perpendicularity, equal lengths, parallelism, cyclicity, etc.) into polynomial constraints, then eliminate variables in a structured way until the target statement reduces to an algebraic consequence of the hypotheses [WU-CHAR]. Conceptually it's: geometry → polynomials → elimination, with a procedure that a computer can execute step-by-step.

Gröbner basis method (Buchberger 1965; geometry automation later). Gröbner bases were introduced by Bruno Buchberger (1965), together with the Buchberger algorithm, giving a general-purpose computational engine for polynomial ideal problems [GB-HIST] [GB-THESIS]. In geometry proving, the translation is almost too clean: encode the hypotheses as an ideal I I generated by polynomials, encode the desired conclusion as a polynomial g g, and prove the theorem by checking whether $g \in I$ g∈I (or in a saturation/elimination variant to handle degeneracies). Gröbner bases make this membership test algorithmic via systematic reductions and elimination behavior [GB-HIST].

Same destination, different roads (and "coordinate dependence"). Both methods share the same punchline: a Euclidean proof can be reduced to algebraic elimination, once you choose coordinates and translate relations into polynomials [WU-CHAR] [GB-HIST]. This looks "coordinate-dependent" in execution because we literally pick a coordinate system, but the truth is coordinate-free: any non-degenerate coordinate choice represents the same geometry. Wu emphasized this as a workable philosophy of mechanization—geometry carries enough

intrinsic structure that, after coordinatization, proof becomes a deterministic computation [WU-SURVEY]. That's the deeper shift of the last decades: not just knowing geometry, but compiling geometry into a procedure.So far, in Parts 1–4, we've followed a mostly pure-math storyline: from Euclid's synthetic geometry, through analytic and algebraic viewpoints, all the way to the modern language of vectors, forms, and (eventually) geometric algebra—ideas that were already in very solid shape by the early 20th century. A natural next question is: what happened in roughly the last 50 years for solving Euclidean geometry problems? The surprise is that the "next big step" is not a new axiom system, but a new attitude: turn geometry into an algorithm—a replicable, programmable procedure for symbolic reasoning. In Euclidean geometry this becomes especially concrete: translate points, lines, circles, and incidences into variables, and translate geometric constraints into polynomial equations. Once everything becomes polynomial, a proof turns into a computable elimination problem. In this part we introduce two early but seminal algebraic approaches to automated Euclidean theorem proving: Wu's method (first) and the Gröbner basis method (second).

Wu's method (Wen-Tsün / Wenjun Wu, late 1970s). Wu's method—often presented through characteristic sets (triangular-style elimination)—was developed by Wen-Tsün Wu in the late 1970s and became one of the first highly successful, systematic methods for mechanical theorem proving in elementary geometry [WU-ORIG] [WU-CHAR]. The key move is explicitly analytic: choose coordinates for "free" points, convert each geometric relation (collinearity, perpendicularity, equal lengths, parallelism, cyclicity, etc.) into polynomial constraints, then eliminate variables in a structured way until the target statement reduces to an algebraic consequence of the hypotheses [WU-CHAR]. Conceptually it's: geometry → polynomials → elimination, with a procedure that a computer can execute step-by-step.

Gröbner basis method (Buchberger 1965; geometry automation later). Gröbner bases were introduced by Bruno Buchberger (1965), together with the Buchberger algorithm, giving a general-purpose computational engine for polynomial ideal problems [GB-HIST] [GB-THESIS]. In geometry proving, the translation is almost too clean: encode the hypotheses as an ideal I in I generated by polynomials, encode the desired conclusion as a polynomial g in g, and prove the theorem by checking whether $g \in I$ g∈I (or in a saturation/elimination variant to handle degeneracies). Gröbner bases make this membership test algorithmic via systematic reductions and elimination behavior [GB-HIST].

Same destination, different roads (and "coordinate dependence"). This roughly captures the last 30 years of 20th century's computational contributions to geometric proofs. There are other important development, such as more numerically related--which we will also briefly utilize, for problem verifications. Both methods share the same punchline: a Euclidean proof can be reduced to algebraic elimination, once you choose coordinates and translate relations into polynomials [WU-CHAR] [GB-HIST]. This looks "coordinate-dependent" in execution because we literally pick a coordinate system, but the truth is coordinate-free: any non-degenerate coordinate choice represents the same geometry. Wu emphasized this as a workable philosophy of mechanization—geometry carries enough intrinsic structure that, after

coordinatization, proof becomes a deterministic computation [WU-SURVEY]. That's the deeper shift of the last decades: not just knowing geometry, but compiling geometry into a procedure.

This rest of this part shows two classic algebraic geometry-prover styles on the same configuration:

1. Gröbner basis method: prove the target polynomial lies in the ideal generated by hypothesis polynomials.
2. Wu's method style (characteristic-set / triangular elimination): eliminate variables using a triangular chain (here it becomes simple linear elimination).
3. Wu's method following the algorithm by Wu more religuously to show the difference from 2.

We work entirely in polynomials (no square-roots) by proving the squared identity

$$2 \quad 2 \quad 2 \quad 2$$

which is equivalent to for positive lengths.

Notes on representation

- SymPy is used here for convenience (symbolic polynomials + Groebner/Wu-style elimination). It is not required in principle.
- Matrices are used only as a compact way to write determinants/dot products; this is a modeling choice, not a requirement of Wu's method.
- In later program the geometry is built with `Point` objects in sympy instead of matices, to show a more geometric style; both approaches are equivalent.

5.1 Coordinate encoding of the geometry (bake-in the constraints)

Given:

-
- °
-
- , and
-
-

We can choose coordinates (scale is irrelevant):

so and .

Since , line is horizontal, so:

Point means . Condition gives:

So:

Now let x y be the intersection point we will solve from the constraints:

- are collinear (because)
- (because and)

In [2]:
```
import sympy as sp
sp.init_printing()

def dist2(P, Q):
    v = P - Q
    return sp.expand(v.dot(v))

def sec(title):
    print("\n" + "="*70)
    print(title)
    print("="*70)

a, hx, hy = sp.symbols("a hx hy")

# Fixed points
C = sp.Matrix([0, 0])
B = sp.Matrix([1, 0])
D = sp.Matrix([0, 1])

# Variable point A, and derived point E
A = sp.Matrix([a, 1])
E = sp.Matrix([0, 1 - a])

# Variable point H
H = sp.Matrix([hx, hy])
```

Now we translate hypotheses into polynomials

Collinearity B, D, H

Vectors and must be linearly dependent, so the 2×2 determinant is zero:

$$x \quad x \quad y \quad y$$

$$x \quad x \quad y \quad y$$

Perpendicularity $(H - E) \perp (B - A)$

Dot product is zero:

These give two polynomials $_1$, $_2$.

In [3]:
```
sec("Hypothesis polynomials")

# (1) Collinear B, D, H
f1 = sp.Matrix([[B[0]-D[0], B[1]-D[1]],
                [H[0]-D[0], H[1]-D[1]]]).det()
f1 = sp.expand(f1)

# (2) (H-E) ⊥ (B-A)
f2 = sp.expand((H - E).dot(B - A))

print("f1 (collinear B,D,H) =", f1)
print("f2 ((H-E) ⊥ (B-A))  =", f2)

# For reference, simplify them:
print("\nSimplified forms:")
print("f1 =", sp.factor(f1))
print("f2 =", sp.factor(f2))
```

```
======================================================================
Hypothesis polynomials
======================================================================
f1 (collinear B,D,H) = hx + hy - 1
f2 ((H-E) ⊥ (B-A))  = -a*hx - a + hx - hy + 1

Simplified forms:
f1 = hx + hy - 1
f2 = -a*hx - a + hx - hy + 1
```

Target polynomial is the following identity to prove:

$$2 \quad 2 \quad 2 \quad 2$$

This is equivalent to because lengths are nonnegative.

In [4]:
```
sec("Target polynomial p")

BE2 = dist2(B, E)
BD2 = dist2(B, D)
DH2 = dist2(D, H)
EH2 = dist2(E, H)

p = sp.expand(BE2*DH2 - BD2*EH2)

print("BE^2 =", sp.factor(BE2))
print("BD^2 =", sp.factor(BD2))
print("DH^2 =", sp.factor(DH2))
print("EH^2 =", sp.factor(EH2))
```

```
print("\np = (BE^2)(DH^2) - (BD^2)(EH^2) =")
print(sp.sstr(sp.factor(p)))
```

```
======================================================================
Target polynomial p
======================================================================
BE^2 = a**2 - 2*a + 2
BD^2 = 2
DH^2 = hx**2 + hy**2 - 2*hy + 1
EH^2 = a**2 + 2*a*hy - 2*a + hx**2 + hy**2 - 2*hy + 1

p = (BE^2)(DH^2) - (BD^2)(EH^2) =
a*(a*hx**2 + a*hy**2 - 2*a*hy - a - 2*hx**2 - 2*hy**2 + 2)
```

5.2 Gröbner basis proof

> We compute a Gröbner basis for the ideal $_1$ $_2$.
> Then reduce modulo . If the remainder is 0, then
>
> $$_1 \quad _2$$
>
> so the hypotheses imply the conclusion.

In [ ]:
```
sec("| Groebner basis reduction")

G = sp.groebner([f1, f2], hx, hy, a, order="lex")

print("| Groebner basis polynomials:")
for g in G.polys:
    print("|   ", g.as_expr())

rem = G.reduce(p)[1]
print("\n| Remainder of p modulo Groebner basis =")
sp.pprint('| ' + sp.sstr(sp.factor(rem)))

assert rem == 0
print("\n| ✅ Groebner proof complete: remainder = 0")
```

```
======================================================================
| Groebner basis reduction
======================================================================
| Groebner basis polynomials:
|    hx + hy - 1
|    a*hy - 2*a - 2*hy + 2

| Remainder of p modulo Groebner basis =
| 0

| ✅ Groebner proof complete: remainder = 0
```

5.3 Wu's method style proof (triangular elimination)

> Wu's method builds a triangular (ascending) chain and repeatedly takes pseudo-remainders to eliminate variables. Here, our constraints are already linear, so the

> “characteristic set” is essentially:
>
> - from collinearity: $_1$ x y
> - from perpendicularity: $_2$ x x y So elimination becomes:
>
> 1. From $_1$:
>
> y x
>
> 2. Substitute into $_2$ to solve $_x$.
> 3. Substitute x y into . If it becomes 0, the theorem is proved. This is exactly the *same spirit* as Wu: reduce the goal with the chain until remainder is 0. See below for the code execution

In [10]:
```
sec("Wu-style triangular elimination")

# Step 1: solve f1 for hy
hy_expr = sp.solve(sp.Eq(f1, 0), hy)[0]
print("From f1=0, hy =", hy_expr)

# Step 2: substitute into f2 and solve for hx
f2_sub = sp.simplify(f2.subs(hy, hy_expr))
print("\nSubstitute hy into f2 ->")
sp.pprint(sp.factor(f2_sub))

hx_expr = sp.solve(sp.Eq(f2_sub, 0), hx)[0]
print("\nSolve for hx -> hx =", hx_expr)

# Then hy
hy_expr2 = sp.simplify(hy_expr.subs(hx, hx_expr))
print("And hy =", hy_expr2)

# Step 3: substitute into p
p_sub = sp.simplify(p.subs({hx: hx_expr, hy: hy_expr2}))
print("\nSubstitute (hx,hy) into p ->")
sp.pprint(sp.factor(p_sub))

assert sp.factor(p_sub) == 0
print("\n✅ Wu-style elimination complete: p reduces to 0")
```

```
================================================================================
Wu-style triangular elimination
================================================================================
From f1=0, hy = 1 - hx

Substitute hy into f2 ->
-a·hx - a + 2·hx

Solve for hx -> hx = -a/(a - 2)
And hy = 2*(a - 1)/(a - 2)

Substitute (hx,hy) into p ->
0
```

✅ Wu-style elimination complete: p reduces to 0

5.4 Extra: numeric sanity check

Since we are in python, we can pick any . (For the original picture you typically have .)

We compute the actual numeric lengths and verify .

In [6]:
```python
sec("Numeric sanity check")

aval = sp.Rational(2, 5)   # a = 0.4
subs_num = {a: aval, hx: hx_expr.subs(a, aval), hy: hy_expr2.subs(a, aval)}

# Evaluate squared lengths
BE2_num = sp.N(BE2.subs(subs_num))
BD2_num = sp.N(BD2.subs(subs_num))
DH2_num = sp.N(DH2.subs(subs_num))
EH2_num = sp.N(EH2.subs(subs_num))

BE = float(sp.sqrt(BE2_num))
BD = float(sp.sqrt(BD2_num))
DH = float(sp.sqrt(DH2_num))
EH = float(sp.sqrt(EH2_num))

print("a =", float(aval))
print("H =", (float(subs_num[hx]), float(subs_num[hy])))
print("BE*DH =", BE*DH)
print("BD*EH =", BD*EH)
print("difference =", (BE*DH) - (BD*EH))
```

```
================================================================================
Numeric sanity check
================================================================================
a = 0.4
H = (0.25, 0.75)
BE*DH = 0.412310562561766
BD*EH = 0.412310562561766
difference = 0.0
```

This is Wu-flavored code, but it does not implement the full, original Wu characteristic-set algorithm.

In the real Wu method, you fix a variable order (a precedence like $y \quad x$), then build a triangular chain (a characteristic set).

You do not "randomly solve" for a variable when it looks convenient. You pick eliminations to make the next polynomial's main variable well-behaved and the chain triangular.

Most importantly: Wu's method avoids fraction of polynomials by using pseudo-division (a.k.a. pseudo-remainder). This guarantees you stay inside the polynomial ring: no denominators ever appear. Internally, pseudo-division multiplies by powers of the leading coefficient, so the "portfolio" always contains pure polynomials.

So we call it "Wu-style" elimination here: same spirit (triangular elimination), but the first version uses solve (which may introduce denominators), while the second version shows the truly programmable "no-denominator" elimination that resembles Wu's mechanics. Below we will show how the problem would be solved had we followed the algo of Wu's method more programmably.

Fully programmable "no-denominator" Wu-like elimination (pseudo-remainder)

This is the key upgrade: we eliminate variables without ever dividing.

Core idea (one line)

To eliminate y, we replace "substitute y " by pseudo-remainder:

$$2 \qquad 2 \quad 1 \quad y$$

which equals $^{k}{}_{2} \quad y$ but with denominators cleared, hence still a polynomial.

What this corresponds to mathematically

We built a triangular chain:

$$1 \quad 1 \qquad y$$

$$2 \qquad 2 \quad 1 \quad y \qquad x$$

Then we reduced the goal:

$$\text{prem}(\cdot, T_1; h_y) \qquad \text{prem}(\cdot, T_2; h_x)$$
$$p$$

That is exactly the "eliminate y, then eliminate x, then the goal collapses" story - but now in a Wu-programmable way.

One important Wu caveat (geometry degeneracy)

Pseudo-division implicitly assumes the leading coefficients used during elimination are not zero (otherwise the "main variable" disappears and you're in a degenerate case). In geometry, this corresponds to "generic position" assumptions (no coincident points, no accidental parallel collapse, etc.).

In [11]:
```
sec("Wu-style triangular elimination (pseudo-remainder / no-denominator)")

from sympy.polys.polytools import prem

# --- Step 1: eliminate hy using pseudo-remainder ---
T1 = sp.factor(f1)                          # main variable: hy
R2 = sp.factor(prem(f2, T1, hy))            # eliminates hy, stays polynomial

print("T1 (main var hy) =")
sp.pprint(T1)
print("\nR2 = prem(f2, T1, hy)  (hy eliminated, polynomial) =")
sp.pprint(R2)

# --- Step 2: eliminate hx using pseudo-remainder ---
T2 = sp.factor(R2)                          # main variable: hx (ideally)
Rp = sp.factor(prem(p, T1, hy))             # first reduce goal by T1 in hy
Rp2 = sp.factor(prem(Rp, T2, hx))           # then reduce by T2 in hx

print("\nT2 (after eliminating hy; main var should be hx) =")
sp.pprint(T2)
print("\nGoal reduction: Rp2 = prem(prem(p,T1,hy), T2, hx) =")
sp.pprint(Rp2)

assert sp.factor(Rp2) == 0
print("\n✅ Wu-style pseudo-remainder elimination: goal reduces to 0 (no denominators
```

```
======================================================================
Wu-style triangular elimination (pseudo-remainder / no-denominator)
======================================================================
T1 (main var hy) =
hx + hy - 1

R2 = prem(f2, T1, hy)  (hy eliminated, polynomial) =
-a·hx - a + 2·hx

T2 (after eliminating hy; main var should be hx) =
-a·hx - a + 2·hx

Goal reduction: Rp2 = prem(prem(p,T1,hy), T2, hx) =
0

✅ Wu-style pseudo-remainder elimination: goal reduces to 0 (no denominators ever use
d)
```

References

- [ATP-GEO] Chou, Gao, Zhang (2001). "Machine Proofs in Geometry." Summary: Monograph on automated geometry theorem proving. URL:

http://www.mmrc.iss.ac.cn/~xgao/paper/book-area.pdf

- [CAS-GEN] "Computer algebra systems: Mathematica, Maple." Summary: General-purpose CAS background for symbolic computation. URLs: https://en.wikipedia.org/wiki/Wolfram_Mathematica ; https://en.wikipedia.org/wiki/Maple_(software)
- [GB-HIST] Buchberger (2005). "An Introduction to Groebner Bases." Summary: Historical and conceptual introduction to Gr?bner bases. URL: https://www3.risc.jku.at/publications/download/risc_3045/2005-07-09-Historic-Intro-to-GB.pdf
- [GB-THESIS] Buchberger (1965/2005 translation). "Bruno Buchberger's PhD thesis (Gr?bner bases)." Summary: Original Gr?bner basis algorithm and theory; later English translation with commentary. URL: https://www.sciencedirect.com/science/article/pii/S0747717105001483
- [WU-CHAR] Wu (late 1970s). "Wu's method of characteristic set." Summary: Overview of characteristic-set elimination developed in the late 1970s. URL: https://en.wikipedia.org/wiki/Wu%27s_method_of_characteristic_set
- [WU-ORIG] Wu (late 1970s). "Wu's method discovery timeline." Summary: Historical timeline for the development of Wu's method. URL: http://www.mmrc.iss.ac.cn/~lzhi/Research/wuritt.html
- [WU-SURVEY] Wu et al. (2007). "Mathematics mechanization" survey. Summary: Survey of algebraic methods and mechanization in geometry proving. URL: http://www.mmrc.iss.ac.cn/~xgao/paper/07-mmsurvey.pdf

Back to TOC

Part 6. Modern Geometric Prover

Wu's method and Gröbner-basis proving are "old school algorithmic" in the best sense: once you translate a diagram into polynomial constraints, the prover can grind deterministically and still surprise you with what it manages to derive. That feels creative because the algorithm is not following a human narrative, yet it discovers nontrivial consequences reliably and at scale. The downside is exactly what you observed: these proofs often become long coordinate eliminations that certify truth, but don't illuminate the geometry. In fact, modern surveys often summarize algebraic methods this way: extremely powerful decision procedures, but the output is typically not a readable, traditional geometry proof—more a yes/no outcome with an algebraic justification. [SURVEY]

A key idea that matured over the last two decades is to keep the computation, but move the "language of thought" closer to geometry itself. Semi-synthetic methods like the area method and the full-angle method do this by replacing raw coordinates with geometric invariants (directed areas, oriented angles, angle expressions) and carefully designed elimination rules. The area method is explicitly praised for producing proofs that are concise and human-readable while still being automated. [AREA] The full-angle line of work went further by building rule systems around angle expressions in a way that can be structured hierarchically, which even enables visually dynamic, step-by-step presentations of proofs—much closer to how a person reasons with a diagram. [VDP]

This "human-facing" direction also pushed geometry provers into interactive and educational software. A milestone was the incorporation and testing of multiple automated geometry provers inside GeoGebra, motivated by the idea that automated deduction had become mature enough to change how people learn and explore geometry. [GGB] Around the same time, community efforts like OpenGeoProver worked on unifying several proof styles (algebraic, semi-synthetic, synthetic) behind a common tool interface, so the user sees geometry objects while the backend can choose the best proving engine.

Then the last two years added a new ingredient that directly targets the "magic of auxiliary constructions": systems that can propose new points/lines/circles in a human-like way. AlphaGeometry (Nature 2024) made this explicit: a neural language model guides a symbolic deduction engine through the huge branching factor of difficult Olympiad problems, and it reaches near gold-medalist performance on a benchmark of 30 problems. [AG1] DeepMind's own description highlights the core leap: the model predicts useful new constructs from an effectively infinite space, while the symbolic engine validates the consequences. [AG1-BLOG] AlphaGeometry2 pushes this further by enriching the language and emphasizing auxiliary constructions even more strongly, and it is built on a more powerful Gemini-based model. [AG2]

What's also interesting is that "auxiliary creativity" is no longer owned by neural models alone. HAGeo shows you can get gold-medal level performance by engineering strong heuristic auxiliary construction strategies wrapped around a symbolic deduction engine, running fully on CPUs without neural inference. [HAG] GenesisGeo attacks the bottleneck from the other side: it open-sources a massive synthetic dataset (including millions of cases with auxiliaries) and accelerates the symbolic engine with C++ components, while also adding a neuro-symbolic prover based on Qwen3-0.6B-Base. [GEN] These results collectively say something profound: the "human-like" part of geometry solving is largely about proposing the right extra objects, and we now have multiple competing ways to do that—LLMs, heuristics, and hybrid search.

Newclid is a nice example of how modern systems separate "frontend geometry" from "backend proof." On the frontend, you describe the construction and goals (JGEX / GeoGebra import); in the backend, a symbolic engine (DDARN) runs deduction and proof search. The important architectural twist is that Newclid exposes an agent interface between them, so the choice of auxiliary construction strategy can be swapped: an LLM agent, a heuristic agent, a BFS-style agent, or even a human-in-the-loop agent. The Newclid paper explicitly frames its CLI as a way to run problems without a Python entry point and even to replace the original LLM with human decisions, while still using the same symbolic core. [NC]

Formal verification languages (Lean/Coq) matter because they make "correctness" non-negotiable. Once you have systems that generate highly non-obvious auxiliaries, you urgently need a small trusted kernel that can check every step. The area method already had a deep formalization path: Narboux implemented it as a decision procedure inside Coq, explicitly valuing short and readable proofs with machine-checkable elimination lemmas. [COQ-AM] LeanGeo extends this spirit to competition-level geometry inside Lean 4, aiming for a unified, rigorously verified framework integrated with Mathlib and released with an open-source library and benchmark. [LG] The big point is: "creative search" can be messy, probabilistic, and exploratory—but the final proof artifact can still be as clean and trustworthy as a textbook, because the checker is unforgiving.

Another piece of the "human-like" story is not just finding a proof, but presenting it in a way that looks like geometry again. Historically, this emerged as a reaction to coordinate/elimination proofs: Wu/Gröbner methods can certify truth, but the output often reads like algebraic elimination rather than geometric insight. Semi-synthetic provers helped because they operate in higher-level invariants (areas, oriented angles, angle expressions), which naturally map back into short Euclid-style steps that a student can follow. The full-angle method is a representative example: it was designed specifically so the proof object is a sequence of geometric statements that can be displayed and checked step by step, rather than a large polynomial certificate [FA].

Once the reasoning steps become "geometric sentences," visualization becomes possible and valuable. A major milestone was the line of work on visually dynamic proof presentation, where the proof is organized hierarchically and the diagram is animated so that objects and relations appear exactly when the text uses them; this turns a proof from a static list of facts into a guided tour that matches human attention [VDP]. In parallel, automated provers began

integrating into interactive geometry environments, so proving becomes part of exploration: you construct a figure, conjecture an invariant, and then ask the prover to justify it inside the same visual workspace, often with a human-readable proof trace rather than an opaque "true/false" result [GGB]. This visualization layer matters because it is where "machine correctness" becomes "human understanding."

Euclid-Omni fits into this contribution as a unification attempt that pushes the interface boundary outward. Compared to systems like Newclid, which cleanly separate a formal geometry frontend (JGEX/diagram constraints) from a symbolic backend (DDARN proof search) [NC], Euclid-Omni aims to connect more of the pipeline end-to-end: it combines formal symbolic proving with LLM/VLM components so that inputs can be more flexible (including natural language or diagram perception), while outputs can remain structured and human readable as logical steps supported by a formal core [EO]. Conceptually, it tries to make the "frontend" not just a parser for JGEX, but also a translator from human modalities into formal predicates, and then back into a readable proof narrative—bridging the last gap between creative construction and explainable reasoning.

Furthermore, Euclid-Omni explicitly frames "open data" as one of the missing pieces in the current gold-medal pipeline ecosystem, and positions itself as a response. It points out that existing IMO-level systems required massive compute to generate training examples, yet did not release their data-generation pipelines or datasets, which blocks broader reproducibility and extension by the community [EO]. In contrast, EO's core contribution is not only a symbolic solver (Euclidea), but also a complete configurable data factory that synthesizes symbolic problems, renders diagrams, and produces aligned natural-language versions, thereby yielding large-scale datasets for training LLMs/VLMs; and it explicitly states "we will release the complete framework," which (by their own definition of the framework) includes the data generation pipeline and the datasets it produces [EO]

Proof 6.1: Newclid

Newclid was created based on AlphaGeometry, by a startup company called Harmonic.

Harmonic built Newclid to make the AlphaGeometry-style approach usable outside a closed research stack: a tool that is easier to run, easier to extend, and easier to integrate into other workflows. The emphasis is on a clean interface between geometry input and the symbolic proof engine, so different construction or search strategies (LLM, heuristics, or human-in-the-loop) can be swapped in without changing the core prover. Newclid use its own DDARN (Deductive Database with Algebraic Reasoning in Newclid) symbolic solver that's derived from the DDAR of AlphaGeometry. In Newclid 3 published in later 2025, the model also included an C++ optimized execution engine of DDARN that is called Yuclid.

In [5]:
```
from pathlib import Path
from newclid import GeometricSolverBuilder
from newclid.problem import PredicateConstruction, ProblemSetup
from newclid.jgex.problem_builder import JGEXProblemBuilder
```

```
from newclid.jgex.formulation import JGEXFormulation
import matplotlib.pyplot as plt

JGEX_PROBLEM = """C B = segment C B;
D = on_tline D C B C, on_circle D C B;
E = on_line E C D;
A = on_pline A D B C, on_circle A D E;
F = intersection_lt F A B E A B;
G = intersection_ll G B D A E;
H = intersection_ll H B D E F
""".strip().replace('\n', ' ')
goals = [PredicateConstruction.from_str("eqratio D H E H D B E B")]


# Build a JGEX formulation into a ProblemSetup
jgex_problem = JGEXFormulation.from_text(JGEX_PROBLEM)
problem_builder = JGEXProblemBuilder(problem=jgex_problem, rng=0)
problem_setup = problem_builder.build()
problem = ProblemSetup(name=problem_setup.name,
                       points=problem_setup.points,
                       assumptions=problem_setup.assumptions,
                       goals=tuple(goals),
                       )
# Build a solver from the ProblemSetup
solver = GeometricSolverBuilder().build(problem)

# Run deduction + algebraic reasoning
success = solver.run()
print("solved: ", success)

if success:
    out = solver.write_all_outputs(Path('out'), jgex_problem=problem_builder.jgex_prob
    plt.close('all')
    # proof = solver.proof()
```

```
solved:  True
```

```
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
```

```
out\html\dependency_graph.html
```

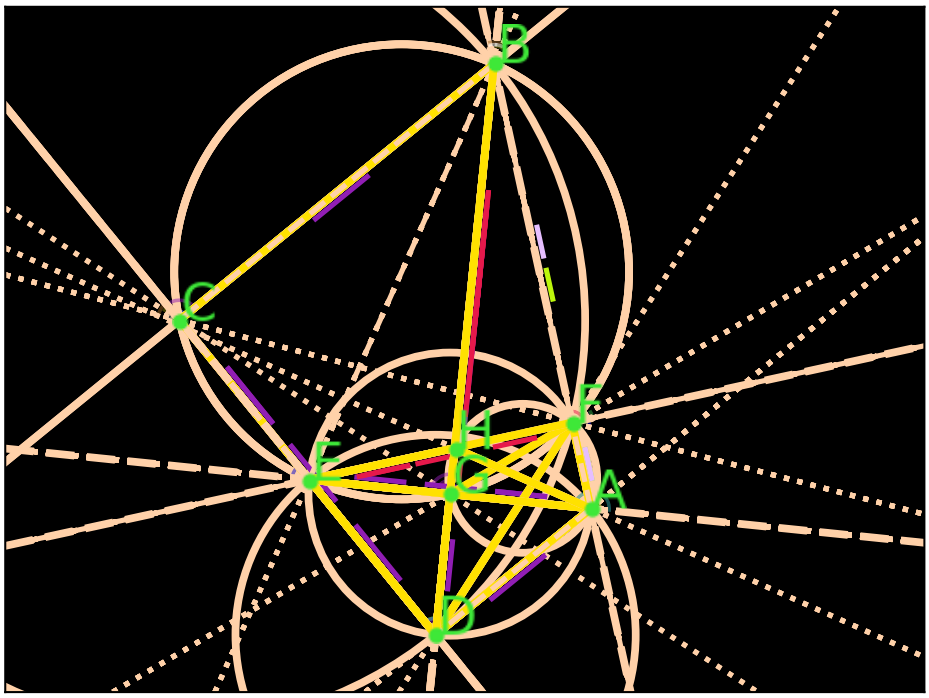

Figure 1: Proof figure generated by Newclid for Proof 6.1 (proof_figure.svg); even though less polished, it does show the geometries fine.

In [2]:
```
from IPython.display import IFrame, display
display(IFrame('out/html/dependency_graph.html', width='100%', height=520))
```

Figure 2: Proof dependency graph for the out run (dependency_graph.html).

Summarize the solution from out/proof.text in human readable steps using ChatGPT:

> Because are concyclic, the angles subtending the same chords are equal; with collinear and collinear, this gives two matching angles between and , so , hence —— ——. Because are concyclic, the same "equal angles on equal arcs" rule gives two matching angles between and , so , hence —— ——. Combining with the previous relation eliminates and yields ————. Next, since are collinear and are collinear, and the right-angle/parallels in the construction align the remaining angles, we get , hence —— ——. Together with the proportionalities already obtained from , this rewrites in the same scale as

> , giving ———————. Now bring in . Because are concyclic,
>
> equal angles in the same segment give , so ———————.
>
> Also, because are concyclic, the same cyclic angle rule gives
>
> , so ———————. Finally,
>
> ——— ——————— ——————— ——————— ———
>
> so .

This proof has 65 steps in its proof.text which is about 500 lines. ChatGPT summarized it well, from it we see this doesn't require auxiliary lines other than some trivial connecting existing points. It is also more complex than some of the other methods listed in part 1. In this sense, this method is innovative and reveals a few other similarities of triangles that are not too obvious.

6.2 auxiliary lines

Newclid supports multiple ways of adding auxiliary lines. In the following session we will explore it. First we need to have the G point defined, without that line, even though it is apparently irrelevant, the solver would fail with error message "ProofBuildError: Goal eqratio B D B E D H E H is numerically false.". This shows the solver first makes sure numerically a generated case can verify the goal. The reason this fails without G is because A and B can sit on the opposite side of CD now. In that case though, EA//BD so they would never intersect. This is why by defining G we eliminated an unexpected case defined by JGEX language. Such trap can be subtle. The GeoGebra's plot support would help with such constraints.

In [3]:
```
import matplotlib.pyplot as plt
JGEX_PROBLEM = """C B = segment C B;
D = on_tline D C B C, on_circle D C B;
D1 = on_tline D1 C B C, on_circle D1 C B;
E = on_line E C D;
A = on_pline A D B C, on_circle A D E;
F = intersection_lt F A B E A B;
G = intersection_ll G B D A E;
H = intersection_ll H B D E F;
""".strip().replace('\n', ' ')
goals = [PredicateConstruction.from_str("eqratio H D H E D B E B")]


jgex_problem = JGEXFormulation.from_text(JGEX_PROBLEM)
problem_builder = JGEXProblemBuilder(problem=jgex_problem, rng=0)
problem_setup = problem_builder.build()
problem = ProblemSetup(name=problem_setup.name,
                       points=problem_setup.points,
                       assumptions=problem_setup.assumptions,
                       goals=tuple(goals)
```

```
                            )
solver = GeometricSolverBuilder().build(problem)

success = solver.run()
print("solved: ", success)
if success:
    out = solver.write_all_outputs(Path('out1'), jgex_problem=problem_builder.jgex_prc
    plt.close('all')
    proof = solver.proof()
```

```
solved:  True
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
out1\html\dependency_graph.html
```

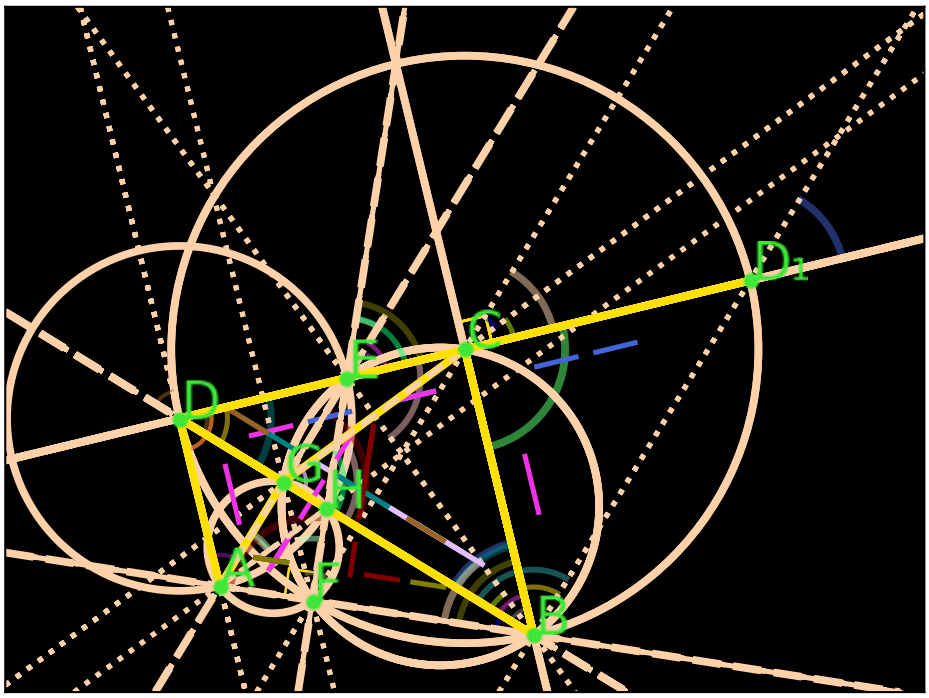

Figure 3: Proof figure generated by Newclid for Proof 6.2 (proof_figure.svg); note the auxiliary point D1 is added compare with Figure 1.

In [4]:
```
from IPython.display import IFrame, display
display(IFrame('out1/html/dependency_graph.html', width='100%', height=520))
```

Figure 4: Proof dependency graph for the out1 run (dependency_graph.html); this is noticeably simpler than the previous Figure 2 which shows the help point D1 has simplified the proof that's also evidential in the proof summary by ChatGPT below.

Proof 6.2 can be summarized based on out1/proof.txt by ChatGPT, we see it becomes significantly simpler than the previous proof

> Introduce $_1$ on the extension of such that $_1$. Then is the midpoint of $_1$, and since we also have $_1$. Hence is the perpendicular bisector of $_1$, so $_1$. Newclid then proves two similarities: $_1$ and . From the first, $_1$; from the second, . Therefore $_1$, and using $_1$ gives , as desired.

Moreover, beyond Newclid's ratio-chasing, the logic can be shortened: we can show that $_1$. That directly yields $_1$, and then $_1$ finishes the proof in one line. This shortened version is exactly what our proof 1 in part 1 did. So Newclid

isn't necessarily putting great efforts to simplify the solution; instead it aims to grow the algebraic reasoning tree till it reach the desired conclusion.

Newclid has a few built-in point-generation heuristics (see https://github.com/Newclid/Newclid/blob/main/notebooks/heuristics_implementation.ipynb). They are a systematic way to propose auxiliary constructions, which is overkill for this small problem, but the exercise of recreating Proof 1 still shows how the tool can help explore plane-geometry ideas. AlphaGeometry2 pushed this further by using LLM-guided proposals for new points, adding another layer of automation to the construction step.

As we have demonstrated on summarizing the complicated proof.text with hundreds of lines into a human readable proof, a separate, complementary trend is using LLMs to translate formal or symbolic proofs into human-readable narratives. That translation layer matters for the human-machine interface; it helps students and researchers see the structure of an argument without wading through raw derivations. For more information, read the Euclid-omni paper [EO].

Taken together with the previous parts of the note, these developments show how human insight and machine assistance are reshaping plane geometry: from the cleverness of mathematicians to machines that can search, verify, and suggest constructions, while humans interpret, teach, and guide the narrative. Similar shifts are happening across mathematics and science, which makes this a particularly exciting time for human researchers.

References

- [AG1] Trinh et al. (2024). "Solving olympiad geometry without human demonstrations." Nature. Summary: Introduces AlphaGeometry, combining a neural model with a symbolic deduction engine to solve Olympiad geometry at near gold-medal level. URL: https://www.nature.com/articles/s41586-023-06747-5
- [AG1-BLOG] Trinh & Luong (2024). "AlphaGeometry: An Olympiad-level AI system for geometry." DeepMind blog (Jan 17, 2024). Summary: Practitioner-oriented overview of AlphaGeometry's pipeline and results. URL: https://deepmind.google/discover/blog/alphageometry-an-olympiad-level-ai-system-for-geometry/
- [AG2] Chervonyi et al. (2025). "Gold-medalist Performance in Solving Olympiad Geometry with AlphaGeometry2." arXiv:2502.03544. Summary: Extends AlphaGeometry with a richer language and stronger auxiliary-construction capability. URL: https://dblp.org/rec/journals/corr/abs-2502-03544
- [AREA] Janicic, Narboux, Quaresma (2012). "The Area Method: a Recapitulation." Journal of Automated Reasoning. Summary: Surveys the area method and argues for concise, human-readable automated proofs. URL: https://research.matf.bg.ac.rs/handle/123456789/510
- [COQ-AM] Narboux (2009). "Formalization of the Area Method in the Coq proof assistant." Summary: Coq formalization of the area method (user contribution dated 2009). URL: https://www.irif.fr/~narboux/area_method.html

- [EO] Li et al. (2026). "Euclid-Omni: A Unified Neuro-Symbolic Framework for Geometry Problem Solving." OpenReview (ICLR 2026 submission). Summary: Unifies symbolic proving with LLM/VLM components and a data-generation pipeline. URL: https://openreview.net/forum?id=1GQv7jhmtV
- [FA] Baeta & Quaresma (2013). "The full angle method on the OpenGeoProver." ThEdu 2013. Summary: Describes the full-angle method implementation in OpenGeoProver. URL: https://old.cisuc.uc.pt/publication/show/3494
- [GEN] Zhu et al. (2025). "GenesisGeo: Technical Report." arXiv:2509.21896. Summary: Releases a large synthetic geometry dataset and a neuro-symbolic prover to support auxiliary construction research. URL: https://huggingface.co/papers/2509.21896
- [GGB] Botana et al. (2015). "Automated Theorem Proving in GeoGebra: Current Achievements." Journal of Automated Reasoning. Summary: Reports results from integrating provers into GeoGebra. URL: https://research.matf.bg.ac.rs/handle/123456789/503
- [HAG] Duan et al. (2025). "Gold-Medal-Level Olympiad Geometry Solving with Efficient Heuristic Auxiliary Constructions." arXiv:2512.00097. Summary: Achieves gold-medal-level performance using heuristic auxiliary construction strategies without neural inference. URL: https://huggingface.co/papers/2512.00097
- [LG] LeanGeo team (2025-08-20). "LeanGeo benchmark page." Summary: Project page for the LeanGeo benchmark and library (page dated Aug 20, 2025). URL: https://www.alphaxiv.org/benchmarks/university-of-toronto/leangeo
- [NC] Sicca et al. (2024). "Newclid: A User-Friendly Replacement for AlphaGeometry." arXiv:2411.11938. Summary: Presents Newclid's CLI and agent interface for auxiliary construction strategies on top of a symbolic core. URL: https://notesum.ai/share/arxiv_papers/public/2024-11-20/2411.11938v1
- [SURVEY] Chou, Gao, Zhang (1996). "Automated geometric reasoning: Dixon resultants, Grobner bases, and characteristic sets." ADG 1996. Summary: Early survey of algebraic methods in automated geometry. URL: https://link.springer.com/chapter/10.1007/BFb0022716
- [VDP] Ye, Chou, Gao (2010). "Visually Dynamic Presentation of Proofs in Plane Geometry (Part 1)." Journal of Automated Reasoning. Summary: Proposes a hierarchical, animated proof presentation for geometry. URL: https://dblp.org/rec/journals/jar/YeCG10a
- [WU] Chou & Gao (1990). "Ritt-Wu's decomposition algorithm and geometry theorem proving." CADE 1990. Summary: Presents the Ritt-Wu decomposition approach for geometry theorem proving. URL: https://link.springer.com/chapter/10.1007/3-540-52885-7_89

Back to TOC

Part 7. Aristotle prover

Harmonic (the company behind Newclid) also has a Lean-based prover called Aristotle that's their closed source flagship product. The company provides a web interface and API at https://aristotle.harmonic.fun that anyone can register and try (caveat it is very slow with long queue for job requests). This note uses our small, friendly geometry problem to demonstrate the workflow, while the same system is aimed at harder, cutting-edge problems across mathematics (geometry, number theory, analysis, and more) [ARISTOTLE].

How Aristotle differs from Newclid (short version). Newclid is a geometry-first system: you give a diagram-style construction and goals, and its symbolic engine tries to derive a synthetic proof inside that geometry domain. Aristotle is a general Lean prover: it formalizes statements in Lean and then proves them with Lean tactics, so its output is a formal Lean proof rather than a geometry-specific trace. For geometry problems, Aristotle's geometric solver can still use geometric reasoning (which appeared to be Newclid like), but in this demo it chooses a coordinate model, which looks closer to the analytic approach in Part 2. This is why the artifacts here feel more Lean/coordinate-heavy than Newclid's geometric narrative.

Harmonic reports that Aristotle achieved gold medal-level performance on the 2025 International Mathematical Olympiad (IMO) solving 5 out of 6 problems, with formally verified solutions[HN-IMO][BW-IMO]. In the problem solved, problem 2 is about pure plane geometry [IMO2025P2]. It is vague if the solution adopted used Newclid approach, which Aristotle paper hinted that a separate solver is indeed used for geometry problems, following by the paper describing Newclid a month later. The proof also appears to be a geometry deduction instead of the Lean language.

In general those IMO announcements are a useful backdrop for this demo: we stay with our simple trapezoid ratio to show how aristotle solve such simple problems in geometry, but the tooling is designed for much harder proofs in various domain beyond geometry. Without the full knowledge on how Aristotle's geometric engine differs from the open sourced Newclid, we will focus on how the solution to our simple polygon problem differ between the two to give an observation in this part of the notes. In general the aforementioned white papers propose that Aristotle trained a LLM with reinforcement learning to handle highly parallel Monte Carlo Graph Search over Lean proof states, with policy/value network to pick tree nodes and estimate promise. This differs from the Newclid default BSF tree search, which is enhanced by auxiliary point helper when the search logic stuck. This part of the design is beyond the scope of the current discussion. For our simple problem Aristotle indeed gives a distinct solution repying on Lean.

Aristotle has been aimed and used beyond contest problems. In late 2025, it produced a Lean proof of a simplified variant of Erdos Problem 124. Chinese coverage highlighted that the

system completed a variant autonomously in about six hours, while other reports noted that the solved statement was easier than the original conjecture [E124-6H]. Nonetheless this shows its early impact to the math community.

A few days later, Harmonic's tool was credited for helping resolve Erdos Problem 1026. A December 14, 2025 write-up described how the AI tool proved a key step 2 in Lean as part of the eventual solution [E1026-48H]. Harmonic's own news page claims that since Aristotle's public launch in late 2025, mathematicians have used it to formalize proofs across levels and that the company sees this as a new era of discovery. [HN-MSI]

For broader context on Erdos problems, the community-maintained erdosproblems GitHub database tracks problem status, prizes, formalizations, and links to related resources. It is essentially a living index of the current state of the problem list and related metadata, with an auto-generated table based on the repository's data files [ERDOS-SOTA]. Top mathematician Prof. Torrence Tao has been pioneering this efforts and other promotion of using AI for math research. The accompanying wiki page on AI contributions is more careful and nuanced: it catalogs where AI tools have been used, but also stresses that this is not a benchmark or leaderboard, that literature review can be incomplete, and that assessments are provisional and context-dependent. The page is best read as a reference for tracing claims and understanding the role AI has actually played [ERDOS-AI].

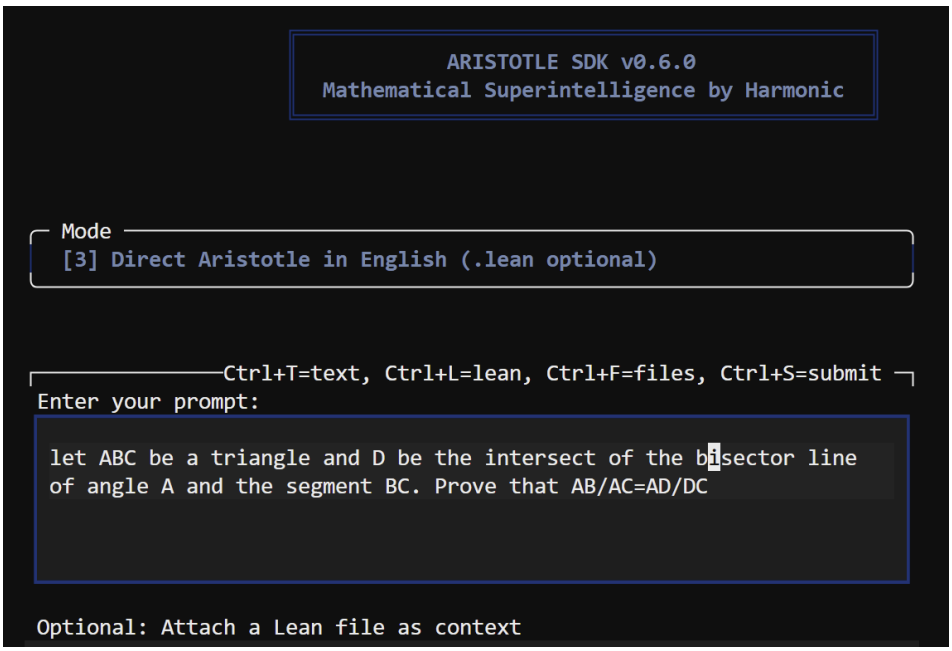

Figure 7.1: Aristotle's prompt interface. It accepts formal Lean input as well as informal text or file uploads.

As Figure 7.1 shows, the command line application of Aristotle can offer an interactive proof interface. Also they offer an API interface in python which is used in the code below to solve our polygon problem, including both questions. Note that the problem text is written in informal plain English, which is accepted by their API as well as the prompt, aside from the alternative lean input format. Under the hood the model clearly utilize strong LLM for understanding such

descriptions. In fact the team has described its solver architecture as a pretrained LLM further trained with reinforcement learning for reasoning [ARISTOTLE].

In [ ]:

```python
import os
import logging
from pathlib import Path
import os
import re

def load_aristotle_api_key():
    key_file = Path.home() / ".aristotle_api_key"
    text = key_file.read_text(encoding="utf-8")

    m = re.search(r'ARISTOTLE_API_KEY="([^"]+)"', text)
    if not m:
        raise RuntimeError("ARISTOTLE_API_KEY not found in ~/.aristotle_api_key")
    return m.group(1)

os.environ["ARISTOTLE_API_KEY"] = load_aristotle_api_key()
print("ARISTOTLE_API_KEY loaded into this kernel")

import aristotlelib
from aristotlelib.project import Project, ProjectInputType

async def prove_polygon_ratio():
    api_key = os.getenv("ARISTOTLE_API_KEY")
    if api_key:
        aristotlelib.set_api_key(api_key)
    else:
        raise RuntimeError("ARISTOTLE_API_KEY environment variable not set")

    logging.basicConfig(level=logging.INFO, format="%(levelname)s - %(message)s")

    problem_text = """
Polygon ratio problem (informal statement).
In trapezoid ABCD, AD ∥ BC, ∠C = 90°, and BC = CD.
Point E lies on CD with DE = AD. Through E draw EF ⊥ AB with F the foot.
Draw BD; it meets AE and EF at G and H, respectively. Prove:
1) BF * HE = GE * BH.
2) BE * DH = BD * EH.
""".strip() + "\n"

    await Project.prove_from_file(
        input_content=problem_text,
        output_file_path="aristotle/polygon_ratio_problem_aristotle.lean",
        validate_lean_project=False,
        project_input_type=ProjectInputType.INFORMAL,
        auto_add_imports=False,
    )

await prove_polygon_ratio()
```

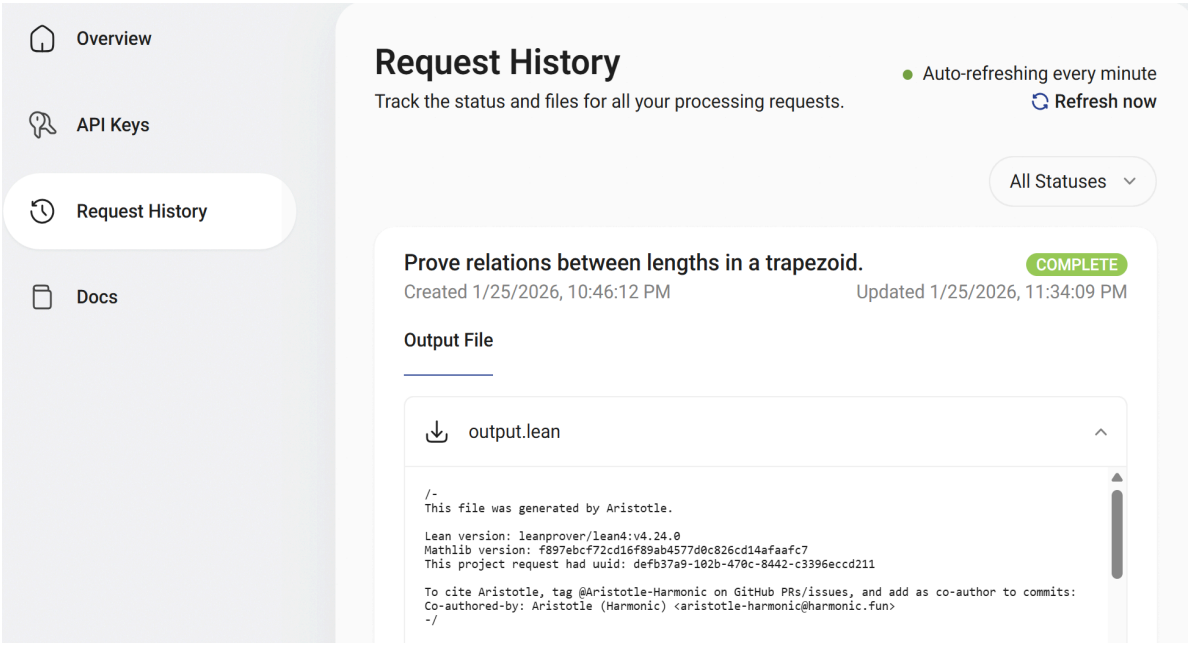

Figure 7.2 The Aristotle web UI at harmonic.fun. The demo job below was queued for a while and then ran for ~20 minutes, suggesting a resource constrained run for the trial account possibly not even on the GPU.

Lean quick intro

Lean is an interactive theorem prover: humans (or automated tactics) provide the proof steps, and Lean checks every step for correctness. In typical use, the proof is supplied by the user via tactics or proof terms, though automation can discharge many routine goals.

Before Lean became popular, prominent proof assistants and formal verification systems included Coq, Isabelle/HOL, HOL Light, ACL2, and Mizar. Lean was created at Microsoft Research by Leonardo de Moura and collaborators. It began as a general-purpose theorem prover for formal verification and mathematics, and later grew a large community library (mathlib). Rough timeline (high-level): early 2010s initial Lean releases; mid2010s Lean 3 matured and became the main community platform; early 2020s Lean 4 modernized the system and tooling.

In recent years, researchers and tool builders have started using Lean not just for verification but also for *assisting* proof discovery: automated tactics, proof search, and AI-guided systems propose steps that Lean then verifies. This notebook uses Aristotle to generate a Lean proof automatically, while Lean itself remains the verifier. [LEAN]

Proof 7.1 Aristotle (Lean files + reading guide)

> ### Human-readable summary (abstracted based on Lean)
>
> Aristotle proves the ratio identities by coordinatizing the trapezoid in the complex plane. It sets , , , , and , so the geometric constraints are encoded directly. The auxiliary

points are then defined algebraically (projection and intersections). The two target identities are reduced to equalities of squared distances, denominators are cleared, and the resulting expressions are simplified by algebraic tactics. In short: the geometry is converted to algebra, and Lean verifies the algebraic identities. This translation indicated the step of converting the informal input description to a format one saving some work from the human.

Human-readable analytical proof (abstracted based on Lean)

1. **Coordinate model.** Set , , , , . This makes , °, , and automatic.

2. **How is found.** The diagonal has direction and can be parametrized as . The line through perpendicular to has direction and can be parametrized as . Solving and gives $\frac{x}{2a-x}$, hence

   ———— ————

3. **How is found.** is the orthogonal projection of onto the line . In coordinates this uses the standard projection formula

   ————————

   This yields explicit coordinates for , which Lean uses algebraically rather than geometrically.

4. **Second identity.** Compute squared lengths: ^{2} ^{2} ^{2}, ^{2} ^{2}, ^{2} $\frac{2a^2x^2}{(2a-x)^2}$, and ^{2} $\frac{x^2(a^2+(a-x)^2)}{(2a-x)^2}$. Thus ^{2} ^{2} ^{2} ^{2}, implying .

5. **First identity.** The proof is analogous: substitute the explicit coordinates, expand squared distances, clear denominators, and simplify. The Lean proof follows the same algebraic pattern as the second identity, so we omit the details.

Input file (informal statement) - `@polygon_ratio_problem.lean`

This is the informal problem text sent to Aristotle (comments start with `--`):

```
-- Polygon ratio problem (informal statement for Aristotle)
-- In trapezoid ABCD, AD ∥ BC, ∠C = 90°, and BC = CD.
-- Point E lies on CD with DE = AD. Through E draw EF ⊥ AB with F the foot.
-- Draw BD; it meets AE and EF at G and H, respectively. Prove:
```

```
-- 1) BF * HE = GE * BH
-- 2) BE * DH = BD * EH
```

Aristotle output (formal Lean proof) - `polygon_ratio_problem_aristotle.lean`

Below are key parts of the generated Lean file with **line numbers**, each followed by extended commentary that explains the Lean keywords and how the proof progresses.

Lean output (lines 1-53):

```
 1: /-
 2: This file was generated by Aristotle.
 3:
 4: Lean version: leanprover/lean4:v4.24.0
 5: Mathlib version: f897ebcf72cd16f89ab4577d0c826cd14afaafc7
 6: This project request had uuid: defb37a9-102b-470c-8442-c3396eccd211
 7:
 8: To cite Aristotle, tag @Aristotle-Harmonic on GitHub PRs/issues, and add
as co-author to commits:
 9: Co-authored-by: Aristotle (Harmonic) <aristotle-harmonic@harmonic.fun>
10: -/
11:
12: /-
13: We define the points of the trapezoid in the complex plane and prove the
two required geometric identities using algebraic simplification of the
distances.
14: The points are defined as follows:
15: - C is the origin (0,0).
16: - B is at (a,0).
17: - D is at (0,a).
18: - A is at (x,a).
19: - E is at (0, a-x).
20: - F is the projection of E onto AB.
21: - G is the intersection of AE and BD.
22: - H is the intersection of EF and BD.
23:
24: We prove:
25: 1. BF * HE = GE * BH (`polygon_ratio_1`)
26: 2. BE * DH = BD * EH (`polygon_ratio_2`)
27: -/
28:
29: import Mathlib
30:
31: set_option linter.mathlibStandardSet false
32:
33: open scoped BigOperators
34: open scoped Real
35: open scoped Nat
36: open scoped Classical
37: open scoped Pointwise
38:
```

```
39: set_option maxHeartbeats 0
40: set_option maxRecDepth 4000
41: set_option synthInstance.maxHeartbeats 20000
42: set_option synthInstance.maxSize 128
43:
44: set_option relaxedAutoImplicit false
45: set_option autoImplicit false
46:
47: noncomputable section
48:
49: /-
50: Definitions of points A, B, C, D, E, F, G, H in the complex plane for the
trapezoid problem.
51: -/
52: open Complex
53:
```

Commentary (lines 1-53): This block is metadata and explanatory prose rather than executable code. The multi-line comment delimiters `/- ... -/` mark documentation. The header records the Lean and mathlib versions, which matter because tactics and lemmas can change across releases. The project UUID identifies the Aristotle run. The next comment describes the coordinate model and names the two target identities, giving the reader a roadmap before formal definitions begin. The `import Mathlib` and `set_option` lines also live in this block: they load the library and tune the proof engine (heartbeats, recursion depth, instance search limits) so algebraic automation can complete. This section moves the proof from narrative intent to a precise plan.

Lean output (lines 54-76):

```
54: noncomputable def pC : ℂ := 0
55: noncomputable def pD (a : ℝ) : ℂ := I * a
56: noncomputable def pB (a : ℝ) : ℂ := a
57: noncomputable def pA (a x : ℝ) : ℂ := x + I * a
58: noncomputable def pE (a x : ℝ) : ℂ := I * (a - x)
59:
60: noncomputable def pF (a x : ℝ) : ℂ :=
61:   let AB := pA a x - pB a
62:   let EB := pE a x - pB a
63:   let t := (EB * star AB).re / (AB * star AB).re
64:   pB a + t * AB
65:
66: noncomputable def pG (a x : ℝ) : ℂ := (x / 2) + I * (a - x / 2)
67:
68: noncomputable def pH (a x : ℝ) : ℂ :=
69:   let den := 2 * a - x
70:   (a * x / den) + I * (2 * a * (a - x) / den)
71:
72: /-
73: Prove that BF * HE = GE * BH in the configured trapezoid.
74: -/
```

```
75: open Complex Metric
76:
```

Commentary (lines 54-76): This block is the fully translated Lean version of the geometric setup. `noncomputable` and `open Complex` make complex-number geometry available. The `def` keyword introduces named definitions; here `pA`, `pB`, `pC`, `pD`, `pE`, `pF`, `pG`, `pH` are points encoded as complex numbers, with `I` the imaginary unit. `pF` encodes orthogonal projection using complex conjugation (`star`) and real parts (`.re`) as dot products. `pG` and `pH` are explicit intersection formulas. This is the precise coordinate model that the later algebraic proofs use.

Lean output (lines 77-92):

```
77: theorem polygon_ratio_1 (a x : ℝ) (ha : a ≠ 0) (hx : x ≠ 0) (hden : 2 *
a ≠ x) :
78:   dist (pF a x) (pB a) * dist (pE a x) (pH a x) = dist (pG a x) (pE a x)
* dist (pH a x) (pB a) := by
79:   unfold pF pE pG pH at *;
80:   unfold pA pB at *;
81:   norm_num [ Complex.dist_eq, Complex.normSq, Complex.norm_def ];
82:   norm_num [ Complex.normSq, Complex.div_re, Complex.div_im ] at *;
83:   field_simp;
84:   rw [ ← Real.sqrt_mul <| by positivity, ← Real.sqrt_mul <| by positivity
];
85:   rw [ ← Real.sqrt_mul <| by positivity, ← Real.sqrt_mul <| by positivity
] ; ring;
86:   grind
87:
88: /-
89: Prove that BE * DH = BD * EH in the configured trapezoid.
90: -/
91: open Complex Metric
92:
```

Commentary (lines 77-92): `theorem` introduces a proposition to prove. The objective statement is on line 77: it asserts the first ratio identity as an equality of distances. The hypotheses `ha`, `hx`, and `hden` assert non-degeneracy so denominators are safe (e.g., `2a-x ≠ 0`). The proof enters tactic mode with `by`. `unfold` expands definitions, turning geometric objects into explicit algebra. `norm_num` expands norms and distances into algebraic expressions. `field_simp` clears denominators. `rw` is the rewrite tactic: it replaces the current goal using a known equality (here, square-root identities), so expressions are put into a form that `ring` and `grind` can finish. This block advances the proof by reducing geometry to a polynomial identity and closing it mechanically.

Lean output (lines 93-101):

```
 93: theorem polygon_ratio_2 (a x : ℝ) (ha : a ≠ 0) (hx : x ≠ 0) (hden : 2 *
a ≠ x) :
 94:   dist (pE a x) (pB a) * dist (pD a) (pH a x) = dist (pD a) (pB a) *
dist (pE a x) (pH a x) := by
 95:   unfold pB pD pE pH;
```

```
 96:   norm_num [ Complex.normSq, Complex.norm_def, Complex.dist_eq ];
 97:   norm_cast; norm_num [ Complex.normSq, Complex.div_re, Complex.div_im,
ha, hx, hden ] ; ring;
 98:   rw [ ← Real.sqrt_mul, ← Real.sqrt_mul ];
 99:   · grind;
100:   · positivity;
101:   · nlinarith [ sq_nonneg ( a - x ) ]
```

Commentary (lines 93-101): `polygon_ratio_2` mirrors the first proof. `norm_cast` resolves coercions between numeric types. `positivity` discharges non-negativity side conditions required for square roots. `nlinarith` solves remaining nonlinear arithmetic constraints. Conceptually, this block repeats the same algebraic pipeline: expand distances, clear denominators, and verify the polynomial equality. It completes the proof of the second identity under the same non-degeneracy assumptions.

Note on proof style. In this problem Aristotle proceeds by a coordinate model, so the proof is essentially algebraic and very similar in spirit to Part 2. That choice is not a rule of Aristotle; it is just the strategy selected for this run. In contrast, Newclid (Part 6) is designed to stay in the geometry domain and produce a synthetic-style trace from diagram constraints. So the key difference to keep in mind is not *correctness* but *style of reasoning and output*: Aristotle outputs a Lean proof (general formal system), while Newclid outputs a geometry-specific derivation. For other problems, Aristotle can also use geometric/angle-chasing reasoning, as seen in the IMO 2025 P2 formalization [IMO2025P2], but here it stayed with coordinates for simplicity.

All in all tools like lean MathLib (lean language and thereom database) + Aristotle (LLM with RL training in lean) + ChatGPT 5.2 Pro (and its competitor frontier LLM) have become the most cutting edge AI tools for mathematicians, which has revealed a frontier of realization of mathematical AGI.

References

- [ARISTOTLE] Harmonic. "Aristotle" (web interface). Summary: Web interface and API access for the Aristotle prover. URL: https://aristotle.harmonic.fun; arxiv link of the paper at https://arxiv.org/pdf/2510.01346
- [E124-6H] 36Kr (2025-11-). "AI solved a 30-year math problem in 6 hours. Terence Tao: ChatGPT and others failed." Summary: Coverage of a simplified Erdős #124 variant solved with Aristotle. URL: https://eu.36kr.com/en/p/3576638922980231
- [E1026-48H] 36Kr (2025-12-14). "Terence Tao is shocked: The “pit” dug by a mathematician in 1975 was filled by AI and netizens around the world in just 48 hours." Summary: Coverage of Erdős #1026 and an Aristotle-assisted Lean step. URL: https://eu.36kr.com/en/p/3596176018898947
- [HN-MSI] Harmonic News (2025). "Harmonic announces mathematician sponsorships to accelerate mathematical superintelligence." Summary: Harmonic’s announcement and claims about Aristotle usage. URL: https://harmonic.fun/news

- [HN-IMO] Harmonic News (2025-07-28). "Aristotle Achieves Gold Medal-Level Performance at the International Mathematical Olympiad". Summary: Harmonic announcement of IMO 2025 results and formal verification. URL: https://harmonic.fun/news
- [BW-IMO] Business Wire (2025-07-28). "Harmonic Announces IMO Gold Medal-Level Performance & Launch of First Mathematical Superintelligence (MSI) AI App." Summary: Press release on IMO 2025 result and Aristotle launch. URL: https://www.businesswire.com/news/home/20250728394917/en/Harmonic-Announces-IMO-Gold-Medal-Level-Performance-Launch-of-First-Mathematical-Superintelligence-MSI-AI-App
- [IMO2025P2] Harmonic AI (2025). "IMO 2025 P2 formalization (Lean)." Summary: Lean formalization file for IMO 2025 Problem 2. URL: https://github.com/harmonic-ai/IMO2025/blob/main/HarmonicLean/IMO2025P2.txt
- [ERDOS-SOTA] Terence Tao et al. (2025). "Erdos Problems" (overview). Summary: Community-maintained status tracker for Erdos problems. URL: https://github.com/teorth/erdosproblemstab=readme-ov-file
- [ERDOS-AI] Terence Tao et al. (2025). "AI contributions to Erdos problems" (wiki). Summary: Notes on AI involvement in Erdos problem progress. URL: https://github.com/teorth/erdosproblems/wiki/AI-contributions-to-Erd%C5%91s-problems
- [LEAN] Lean Prover (2025). "Lean Theorem Prover" (official site). Summary: Official documentation and downloads. URL: https://leanprover.github.io/