

A Symbolic Approach to Teaching Geometry

Miroslaw Majewski

e-mail: mirek.majewski@mupad.com

College of Arts and Sciences

New York Institute of Technology

Abu Dhabi

United Arab Emirates

Abstract

Teaching geometry with a computer was always based on Dynamic Geometry software, like Cabri, Cinderella or Geometers Sketchpad. Such software allows interactive manipulation of geometric objects and geometric constructions like those done with the help of ruler and compass. This approach is very fruitful. However, it lacks a number of things. One of them is limited ability of producing symbolic or numeric output.

In this paper we will show how we can use symbolic software for teaching and experimenting in analytic as well as in synthetic geometry. In this paper we will concentrate on two tools producing symbolic output—MuPAD and Geometry Expressions. We will show what kind of geometry problems we can solve with these tools and what are the benefits of this approach.

1 Introduction

For years we used a few computer programs that proved itself to be very effective tools for teaching elementary geometry. These are Cabri produced by CABRILOG in France, Geometers Sketchpad produced by KCP Technologies in USA, and Cinderella developed by two scientists Jürgen Richter-Gebert and Ulli Kortenkamp from ETH in Zürich. In this paper for all these tools we use a common name—Dynamic Geometry software (in short DG). The DG approach is very popular. We can find a number of other computer programs with similar functionality. A general feature of all these programs is an ability to mimic geometry constructions that we used to do in high school geometry classes, e.g. drawing a segment or a line, drawing a point or a circle, find intersection points of two or more geometry objects, etc. All elements drawn in such software are usually not attached to their locations and we can drag them across the screen without breaking relations between objects. This way we can experiment, observe some new facts, and draw conclusions. Figure 1 shows an example developed in Geometers Sketchpad. The objective of this example is to show that the circumcenter of a triangle is the center of the circle circumscribed on the triangle. The numerical measurements shown on the figure are not really a proof of that fact. However,

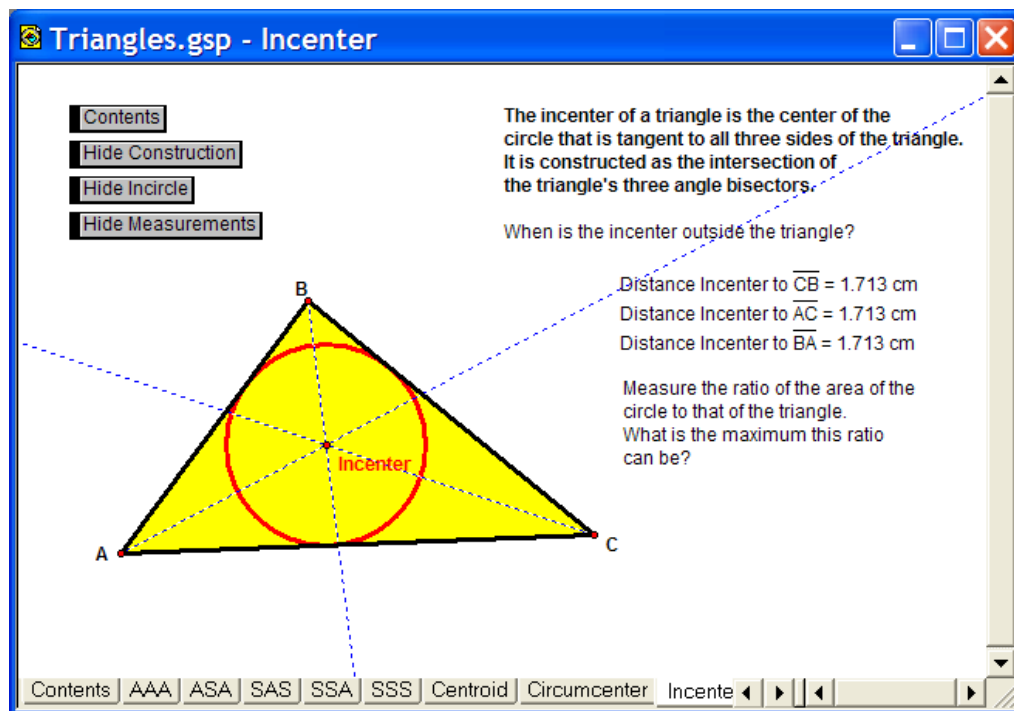


Figure 1: *Example of a geometry construction in GSP*

they are enough to suggest that fact is true and sometimes give a hint on how such fact can be proved.

As we said before, the most important advantage of such software is its ability to mimic simple constructions that we used to do with ruler and compass. Such software is not very intelligent, whatever this means, and usually does not produce a result that goes beyond real numbers. Therefore, numerical values representing distances between points cannot be considered as proofs or even as accurate. In most of cases the real proof should be still done by hand.

Although some of the DG programs may have very limited set of tools, these programs can be used to visualize an incredibly vast spectrum of geometry problems. This makes them favorite tools for undergraduate mathematics. In this group a program worth mentioning is Cinderella. This is the software where we can deal also with some constructions from a non Euclidean geometry, e.g. from hyperbolic or elliptic geometries.

2 Solving Geometry Problems with CAS

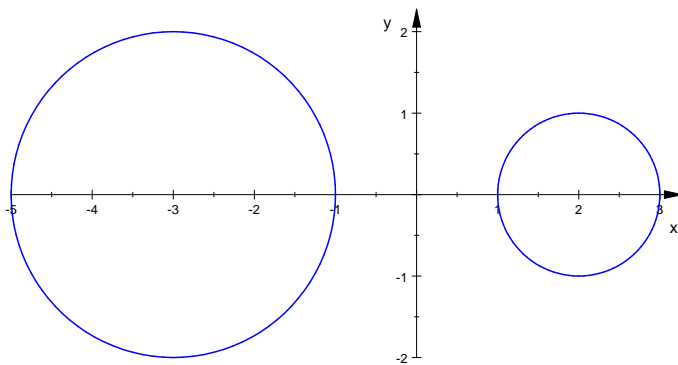
Computer Algebra Systems (in brief CAS), were never popular tools for teaching geometry. The main reason is that all pictures in CAS programs are only static pictures and none of the existing CAS provides interactivity like that in DG software. However, CAS give us something that we do not have in DG software—the ability of using coordinates, equations and getting outputs both in numeric and symbolic form. This is a very serious step forward as we can produce symbolic formulae describing relations between geometry objects.

In CAS creating models of a geometric construction requires typing commands to create geometry primitives, executing commands and finally plotting all objects in a common picture. Below we show one of such examples. Note also, plotting any object in CAS requires its equation with coefficients given in a numeric form. Therefore, code of all plotted objects cannot have symbolic values.

Example 1 *Two circles and their common tangent lines*

We will use MuPAD to create a scene with two given circles and their common tangent lines. Let us suppose that our circles have centers in points (2,0) and (-3,0), and radii 1 and 2 respectively. Developing a plot with these two circles is very easy. Here is the code and the obtained plot.

```
Circle1 := plot::Circle2d(1, [2,0]):
Circle2 := plot::Circle2d(2, [-3,0]):
plot(Circle1, Circle2)
```



Obtaining the remaining parts of the plot is much harder. We expect that we will have four tangent lines, two of them passing between circles and two passing outside.

In order to obtain these four tangent lines we will have to start with a general equation of a line and then use the formula for the distance of a point to a line. This way we will produce a system of equations with two or three variables. Solve such system and obtain equations of all four lines. Below we show only one part of the solution.

In order to simplify our calculations we will use equation of a line in the slope-intercept form $y = mx + b$ or $mx - y + b = 0$. Therefore, the distance of a point (X, Y) from this line will be given by the formula,

$$d = \frac{|mX - Y + b|}{\sqrt{m^2 + 1}}$$

A more convenient for us would be to use a relative distance formula where the absolute value was removed (see [3]). This will allow us to consider each tangent line as a separate case (cases $d > 0$ and $d < 0$).

Here is the MuPAD declaration of the distance formula,

```
distance := (m*x-y+b)/sqrt(m^2+1)
```

$$\frac{b - y + m \cdot x}{\sqrt{m^2 + 1}}$$

Knowing that the distance of a circle center to a tangent line is equal the radius of the circle, we can define two equations:

$$e1 := \text{subs}(\text{distance}, x=2, y=0)=1$$

$$\frac{b + 2 \cdot m}{\sqrt{m^2 + 1}} = 1$$

$$e2 := \text{subs}(\text{distance}, x=-3, y=0)=2$$

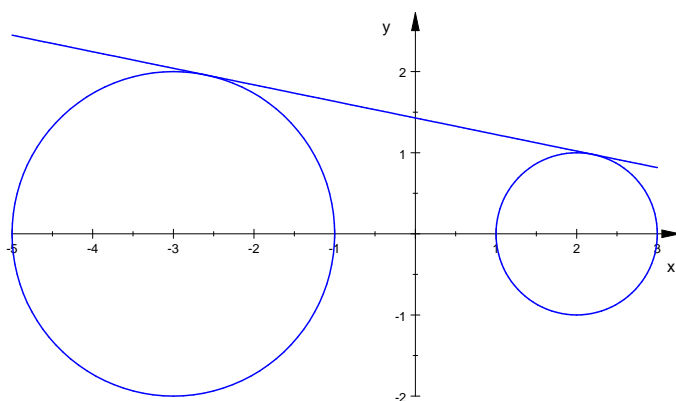
$$\frac{b - 3 \cdot m}{\sqrt{m^2 + 1}} = 2$$

Finally, by solving the system of these two equations we will find m and b . By substituting the obtained results into the equation of the line we will be able to produce a graph of the tangent line. MuPAD calculations may follow like below.

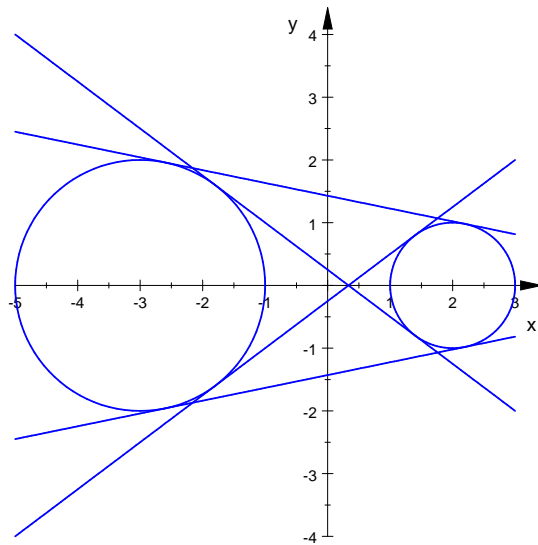
$$\text{solutions} := \text{solve}([e1, e2], [m, b], \text{VectorFormat}=\text{TRUE})$$

$$\left\{ \left(\begin{array}{c} -\frac{\sqrt{6}}{12} \\ \frac{7 \cdot \sqrt{6}}{12} \end{array} \right) \right\}$$

```
m := solutions[1][1]:
b := solutions[1][2]:
line1 := plot::Function2d(m*x+b, x=-5..3):
plot(line1, Circle1, Circle2);
delete m, b
```



Finally using different combinations of signs for the right value of equations $e1$ and $e2$, i.e. $+1, -2$; $-1, +2$ and $-1, -2$ we will be able to obtain the three remaining tangent lines. The plot will look like the one below.



The obtained image can be modified by changing colors of particular lines, fill circles, adding labels and a legend, etc. Finally, the whole process can be automatized by developing a procedure that for two given circles will automatically produce all tangent lines or some of them.


For a teacher trying to use a Computer Algebra System for teaching geometry there is one important conclusion—most of geometry constructions can be developed in CAS, but this task will require from a teacher a lot of preparations and perhaps developing templates for each type of construction. However, the whole process of developing such graphs in a CAS, in our case in MuPAD, has serious didactical advantages—it teaches a lot about the whole process of finding the right solution, in our case finding the tangent lines. Students must know the right formulae and properties of objects before they will be able to develop geometry constructions in CAS.







3 A Brief Overview of Geometry Expressions

Geometry Expressions is a new tool that combines DG approach with features of a Computer Algebra System. The software has an interface that allows creating constructions similar to those known from Cabri or GSP, manual manipulation of geometry objects and, at the same time, producing results in the form of a formula with symbolic coefficients or in numeric form. Geometry Expressions can be used in teaching both synthetic geometry where we do not use a coordinate system as well as in analytic geometry where coordinates are vital elements of solutions. Let us examine solution of the example with two circles and their common tangent lines in Geometry Expressions.

Example 2 *Two circles and their common tangent lines (revisited)*

Before we will jump into solving anything with Geometry Expressions it is important to introduce some of the Geometry Expressions tools that might be necessary for our example. A more detailed information about Geometry Expressions tools and toolboxes can be found in [1].

1. The selection tool, we use it to select objects, drag them on the screen, etc. 

2. A tool to draw a circle 
3. A tool to declare radius of a circle 
4. A tool to declare coordinates of a point 
5. A tool to draw a line 
6. A tool to declare two objects as tangent 
7. A tool to obtain an implicit equation of a line or curve 

All Geometry Expressions tools are very intuitive and in order to use any of them we have to select an object or objects on the screen, click the right tool and in some cases type the required parameters. For instance, in order to declare a circle and a line as tangent we have to click on the circle, press the [Ctrl] key, click on the line and then click the tangency button.

Here is the sequence of steps required to produce a graph of two circles from our example and their tangent lines.

1. Click on the circle tool, drag mouse pointer on the screen and then click the selection tool.
2. Select the center of the circle, click the tool to declare coordinates of the point, and type numbers $-3, 0$. Click again the selection tool.
3. Click somewhere on the circle line, click the tool to declare the radius of a circle and type 2. Click back the selection tool.
4. Repeat steps 1, 2 and 3 to produce the second circle with center in $(2, 0)$ and radius equal to 1.
5. Use the line tool to draw four lines somewhere in the area where the tangent lines should appear. The lines may not even touch any of the circles.
6. Select one of the lines and select one of the circles, click on the tool declaring two objects as tangent. This will move the line in such a way that it will become tangent to the circle.
7. Repeat step 6 for each pair of circles and lines. After this step each line should be tangent to each circle.
8. Select a line and click the implicit equation tool. An implicit formula of the equation will appear next to the line.
9. Repeat step 8 for each of the three remaining lines. The obtained graph may look like the one shown below.

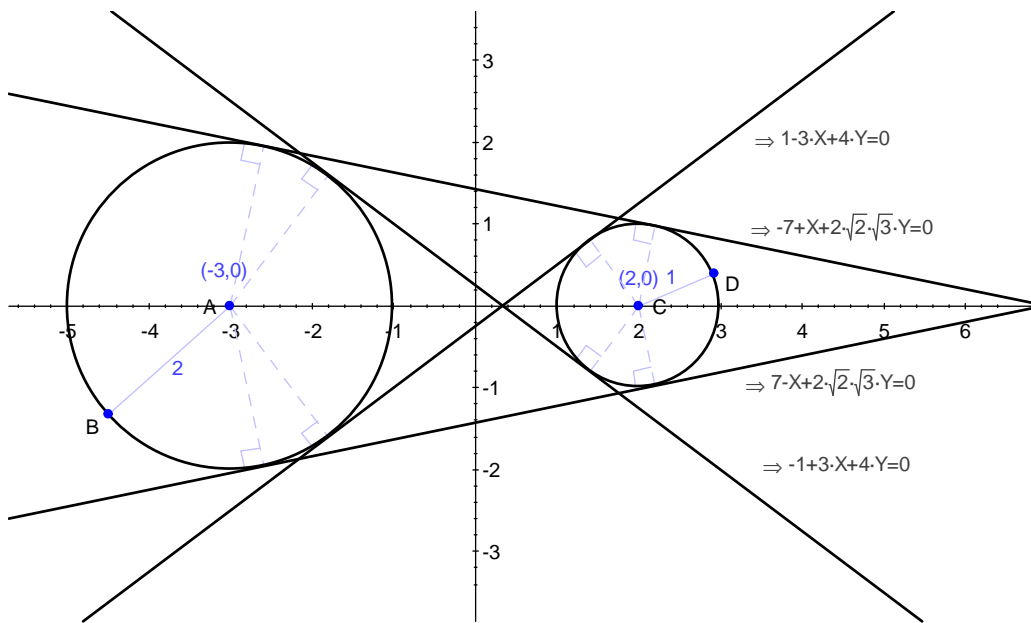


Figure 2: *Two circles and their common tangent lines in Geometry Expressions*

By simple calculations we can easily prove that equations of tangent lines obtained in MuPAD are exactly the same like those obtained in Geometry Expressions.

The above example shows how different is a solution of the problem in CAS and in Geometry Expressions, and how different is philosophy of producing such solution. In CAS we had to mimic all steps that we perform while solving the common tangent lines problem by hand. In Geometry Expressions we had to draw our circles and lines, and apply to them relations of tangency. We do not need to know a precise notion of tangency of two objects or how to calculate a formula of the equation. We use the intelligence embedded into the software to obtain these properties. We certainly lose something in the learning process. This is the same what happens when we use any other highly intelligent software for basic mathematical notions. For example, students are not much learning while applying in MuPAD the solve command to a quadratic equation. In such case the solve command will work like a black box and the way how the solution was obtained will be completely hidden. The same situation occurs while producing equations of lines in Geometry Expressions.

Features of CAS and Geometry Expressions described here make them a bit inconvenient while teaching low level courses. We have to find a good strategy of using these tools in order to not lose anything from the learning process and benefit at the same time from features provided by the software. The situation changes to the complete opposite when teaching mathematics on a more advanced level. Here the easy-to-apply elementary features of the software will allow us to simplify the solving process and concentrate on producing a solution.

4 Geometry Expressions and CAS

From the above example we can conclude that Geometry Expressions is a kind of CAS with DG interface. Indeed we have here a computing engine able to solve simple equations, transform formulae and a few other operations. The engine is not able to deal with more complex mathematical operations like producing an integral or differential, solving a differential equation or even simplify a more complex expression. However, the current engine is strong enough for majority of elementary geometry constructions and basic measurements. If we wish to get more we will have turn back to CAS.

The resemblance of Geometry Expressions to a CAS will be even more obvious if we examine the internal process of solving a problem. Every time when we construct an object, add declaration of a property, or a relation, Geometry Expressions creates a variable or a command representing this step. This way a chain of commands is created. This is almost exactly the same what happened when we created a solution for the two circles and their common tangent lines problem in MuPAD. In Geometry Expressions users are also responsible for creating particular variables and producing solutions in terms of variables defined by them. For example, a formula for calculating area of a triangle can be represented in terms of coordinates of vertices, in terms of lengths of sides of the triangle, or in other combination of chosen terms. The figure 3 shows how it looks in Geometry Expressions. We have there four triangles. Each of the triangles was produced using the polygon tool. In each case we declared the triangle using different parameters. In Geometry Expressions we call them constraints (they are, in fact, declarations of values) or constructions (declarations of relations). For example, for the top right triangle we declared as known three sides a , b , and c . These parameters were used by Geometry Expressions to calculate the area of the triangle.

The chain of declarations produced this way can be modified in some situations. For example, by removing from the picture a formula representing an early declared constraint we remove it from the memory and in place of it we can add another constraints. However, in the current version of Geometry Expressions we are not able to modify or rename constraints added by the program. We are also not able to see or edit the chain of declarations. This makes sometimes interpretation of obtained solution a bit difficult. For example, after drawing a triangle ABC and asking Geometry Expressions to produce its area we obtain the formula:

$$z_0 \Rightarrow \left| \frac{-d_0 \cdot d_1 \cdot \sin(-\theta_0 + \theta_1)}{2} \right|$$

From the above formula we can conclude that d_0 and d_1 are lengths of sides of the triangle and θ_0 , θ_1 are angles in the triangle. However, we still do not know which sides and which angles were used to calculate the result.

A combined functionality of a dynamic interface and ability to produce symbolic results can be used in a vast spectrum of geometric explorations and experiments, both in synthetic and analytic geometry. Here is a very interesting example of a theorem that usually requires some calculations. In Geometry Expressions we get a proof of it almost without any effort.

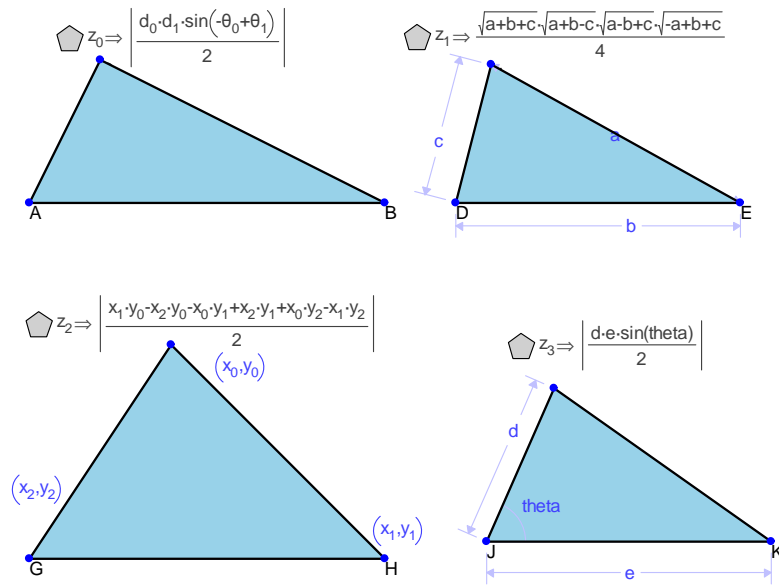


Figure 3: *Different ways of choosing constraints in Geometry Expressions*

Example 3 *Two circles and tangent lines from centers*

There are given two circles $C1$ and $C2$ with radii equal to r and R respectively, and d is the distance between centers of both circles. Let us draw tangent lines to a circle from the center of the other circle. Prove that the length of the chord obtained by connecting points of intersection A, B of two tangent lines with a circle $C1$ is equal the length of the chord obtained by connecting points of intersection $A1, B1$ of the two other tangent lines and circle $C2$ (see figure 4).

Drawing the example in Geometry Expressions and setting the appropriate constraints is a very quick process and does not require any special skills or knowledge. A student needs to know very basic properties of a circle and tangent lines. The figure 4 shows what most of undergraduate students will be able to do.

While changing lengths of radii, distance between circles we still are able to notice that both chords are equal. This will be confirmed by Geometry Expressions if we select each of these chords and ask Geometry Expressions to calculate its length. In both cases we will get the same result $\frac{2Rr}{d}$. For mathematical purists this is not a proof. However, if we treat obtained result as a hint then we can start looking for a real proof using our brain, paper and pencil.

Example 4 *Intersection points of two circles with given centers and radii*

Symbolic software can be extremely useful in many geometry examples where a lot of calculations is needed. Here is one of such examples (taken from [2]).

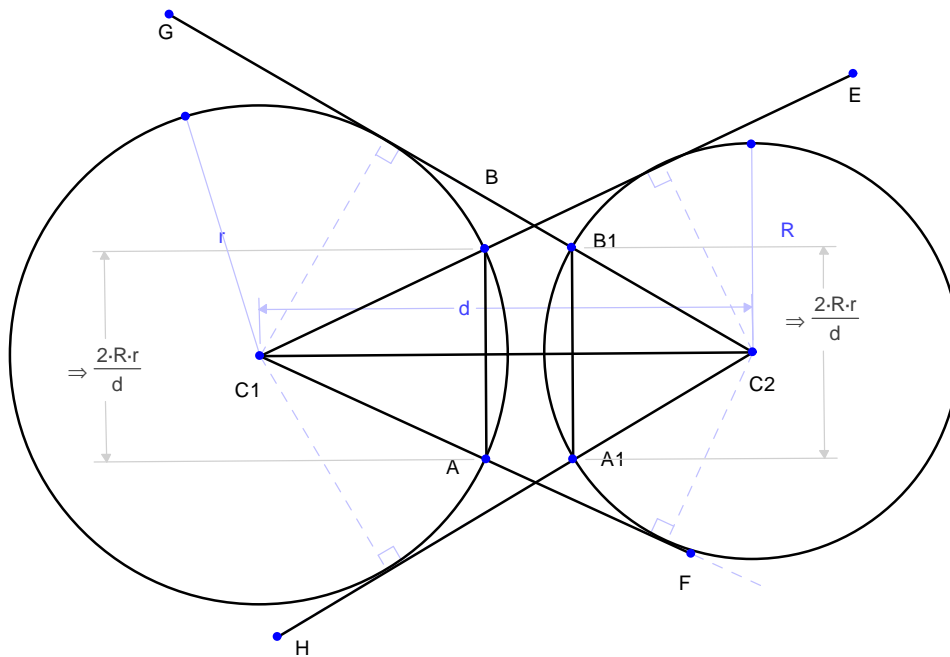


Figure 4: *Two circles and tangent lines from centers in Geometry Expressions*

Let us consider two intersecting circles $C1$ and $C2$ with radii equal to r and R respectively, and coordinates of centers (a, b) and (c, d) (see figure 5). We are going to calculate coordinates of the intersection points.

In Geometry Expressions by selecting one of the intersection points, say Q , and clicking the calculate coordinates tool we will produce a formula representing coordinates of this point. The formula is very long and for this reason it will appear in the output window at the bottom of the screen. In order to simplify this formula we will have to move to a CAS. Like in other parts of this paper we will use MuPAD.

By copying the formula as MuPAD input, and pasting it into MuPAD notebook we will be able to carry further calculations in MuPAD. The formula obtained in Geometry Expressions pasted in MuPAD notebook looks quite complex. Therefore, we will not present here some of the intermediate results obtained from MuPAD calculations. We will show only these formulae that look reasonably simple. Note, in order to carry calculations more efficiently we had to assign the formula obtained in Geometry Expressions to a variable Q . The remaining part of this declaration is the code obtained from Geometry Expressions and roughly formatted to fit it inside of the page area.

```

Q:= // this line was added by hand
[(a+((((a+(c*(-1))))^(2)+((b+(d*(-1))))^(2))))^((-1))*
((a*(-1))+c)*((R^(2)*(-1))+r^(2)+((a+(c*(-1))))^(2)+
((b+(d*(-1))))^(2))*1/2)+((((a+(c*(-1))))^(2)+
((b+(d*(-1))))^(2))))^((-1))*(b+(d*(-1)))*
((R*(-1))+r+((((a+(c*(-1))))^(2)+((b+(d*(-1))))^(2))))^(1/2))^(1/2)*
((R+(r*(-1))+((((a+(c*(-1))))^(2)+((b+(d*(-1))))^(2))))^(1/2))^(1/2)*
((R+r+((((a+(c*(-1))))^(2)+((b+(d*(-1))))^(2))))^(1/2)*(-1))^(1/2)*

```

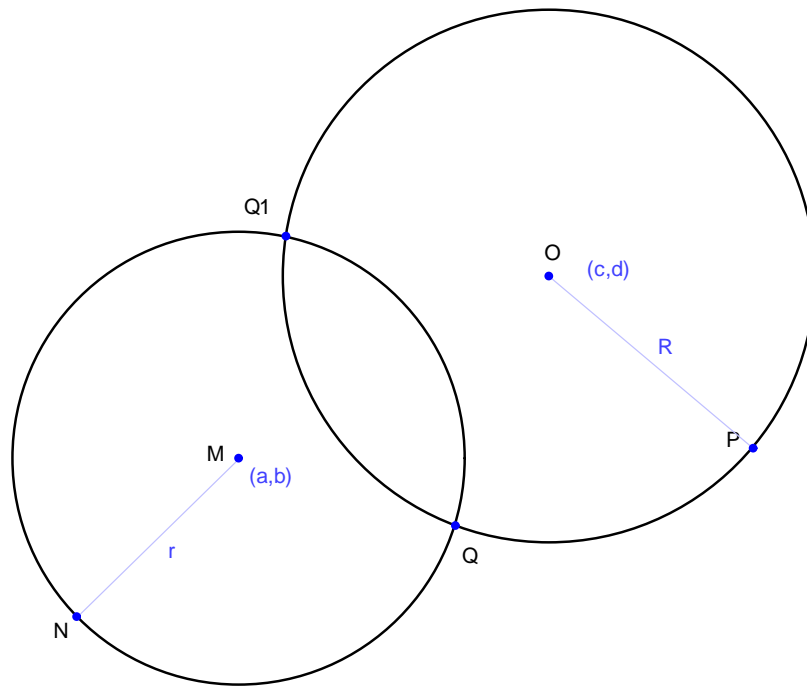


Figure 5: *Two intersecting circles in Geometry Expressions*

$$\begin{aligned} & ((R+r+(((a+(c*(-1))))^2+((b+(d*(-1))))^2))^{1/2})^{1/2}*(-1/2)), \\ & (b+(((a+(c*(-1))))^2+((b+(d*(-1))))^2)^{(-1)}*(a*(-1)+c)* \\ & ((R*(-1))+r+(((a+(c*(-1))))^2+((b+(d*(-1))))^2)^{1/2})^{1/2}* \\ & ((R+(r*(-1))+(((a+(c*(-1))))^2+((b+(d*(-1))))^2)^{1/2})^{1/2}* \\ & ((R+r+(((a+(c*(-1))))^2+((b+(d*(-1))))^2)^{1/2}*(-1)))^{1/2}* \\ & ((R+r+(((a+(c*(-1))))^2+((b+(d*(-1))))^2)^{1/2})^{1/2}*(-1/2))+ \\ & (((a+(c*(-1))))^2+((b+(d*(-1))))^2)^{(-1)}*(b*(-1)+d)* \\ & (((R)^2*(-1)+(r)^2+((a+(c*(-1))))^2+((b+(d*(-1))))^2)*1/2)]: \end{aligned}$$

Now, we can continue calculations using MuPAD's functionality.

```
Q[1] := subs(Q[1], (a-c)^2+(b-d)^2=U^2):
Q[2] := subs(Q[1], (a-c)^2+(b-d)^2=U^2):
Q[1] := Simplify(Q[1]) assuming U>0;
```

$$a - \frac{(a-c) \cdot (r^2 - R^2 + (a-c)^2 + (b-d)^2)}{2 \cdot U^2} - \frac{(b-d) \cdot \sqrt{R+U-r} \cdot \sqrt{R-U+r} \cdot \sqrt{U-R+r} \cdot \sqrt{R+U+r}}{2 \cdot U^2}$$

```
Q[2] := Simplify(Q[2]) assuming U>0;
```

$$b - \frac{(b-d) \cdot (r^2 - R^2 + (a-c)^2 + (b-d)^2)}{2 \cdot U^2}$$

$$-\frac{(a-c) \cdot \sqrt{R+U-r} \cdot \sqrt{R-U+r} \cdot \sqrt{U-R+r} \cdot \sqrt{R+U+r}}{2 \cdot U^2}$$

The obtained formulae look still quite complex. We will apply them in a more concrete situation. For instance, let us take two circles with radii $R = r = 2$ and centers with coordinates $(1, 1)$ and $(4, 3)$. Below we shown how one can calculate coordinates of one the intersection points in MuPAD.

```
U := sqrt((a-c)^2+(b-d)^2):
u := subs(U, a=1, b=1, c=4, d=3, R=2, r=2);
```

$$\sqrt{13}$$

```
q[1] := subs(Q[1], a=1, b=1, c=4, d=3, R=2, r=2, U=u);
```

$$\frac{\sqrt{13} \cdot \sqrt{4 - \sqrt{13}} \cdot \sqrt{\sqrt{13} + 4}}{13} + \frac{5}{2}$$

```
q[2] := subs(Q[2], a=1, b=1, c=4, d=3, R=2, r=2, U=u);
```

$$2 - \frac{3 \cdot \sqrt{13} \cdot \sqrt{4 - \sqrt{13}} \cdot \sqrt{\sqrt{13} + 4}}{26}$$

Finally we can simplify the obtained coordinates or calculate their floating point values.

```
float(q[1]), float(q[2])
```

2.980384461, 1.279423308

The last two numbers can be inserted back into Geometry Expressions to check if the produced result is correct (see figure 6).

Before finishing this example, let us look on a very interesting opportunity following from using in Geometry Expressions intermediate variables. This time the output obtained from Geometry Expressions was formatted as a MuPAD procedure.

```
Q := proc(a, b, c, d, R, r )
local d_3;
begin
d_3 := (((a+(c*(-1))))^(2)+((b+(d*(-1))))^(2)))^(1/2);
[(a+(((a*(-1))+c)*(((R)^(2)*(-1))+(r)^(2)+(d_3)^(2))*(d_3)^((-2))*1/2)+
((b+(d*(-1)))*(((R*(-1))+r+d_3))^(1/2)*((R+(r*(-1))+d_3))^(1/2)*
((R+r+(d_3*(-1))))^(1/2)*((R+r+d_3))^(1/2)*(d_3)^((-2))*(-1/2))),
(b+(((a*(-1))+c)*(((R*(-1))+r+d_3))^(1/2)*((R+(r*(-1))+d_3))^(1/2)*
((R+r+(d_3*(-1))))^(1/2)*((R+r+d_3))^(1/2)*(d_3)^((-2))*(-1/2))+
(((b*(-1))+d)*(((R)^(2)*(-1))+(r)^(2)+(d_3)^(2))*(d_3)^((-2))*1/2)]];
end_proc;
```

Like before, the Geometry Expressions output after copying it into MuPAD requires some minimal adjustments. Here we assigned the procedure to a name, Q , and we rearranged input parameters of the procedure, so they form a logical sequence. We could also use some local

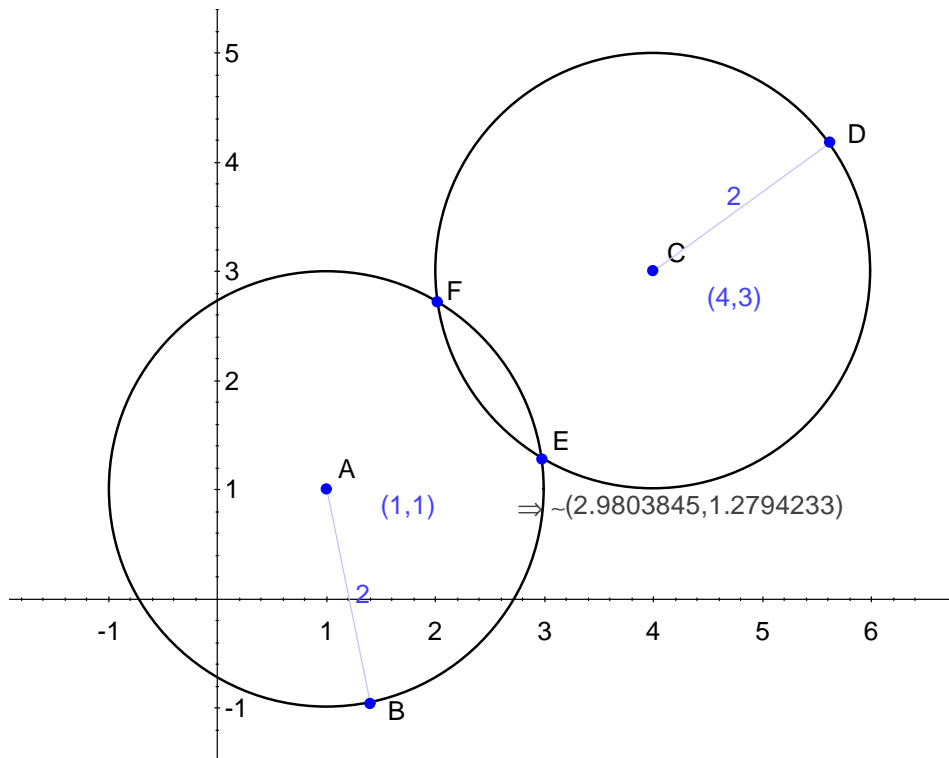


Figure 6: In *GE* the coordinates of the intersection point are the same as in *MuPAD*

variables and a statement to simplify the final result as well as add the code to prevent some input errors (see [4]). However, this is not absolutely necessary.

Applying the procedure to the circles with coordinates of centers $(1, 1)$ and $(4, 3)$, and radii $R = r = 2$, we get the same result like before.

`Q(1,1,4,3,2,2)`

$$\left[\frac{\sqrt{13} \cdot \sqrt{4 - \sqrt{13}} \cdot \sqrt{\sqrt{13} + 4}}{13} + \frac{5}{2}, 2 - \frac{3 \cdot \sqrt{13} \cdot \sqrt{4 - \sqrt{13}} \cdot \sqrt{\sqrt{13} + 4}}{26} \right]$$

`float(Q(1,1,4,3,2,2))`

$$[2.980384461, 1.279423308]$$

5 Final comments and Conclusions

Solving geometry problems in Computer Algebra Systems alone, i.e. without a DG interface, requires more work. However, students are able to follow the solving process step-by-step and go through a very detailed analysis of the problem as well as its solution. Solution obtained this way can be later generalized and programmed in such a way that it will be usable in many similar cases. For instance a process of finding common tangent lines for a given two circles can be wrapped up into a procedure that will produce common tangent lines for any two given circles.

Combination of a CAS and DG interface creates very powerful and attractive set of tools for teaching and working with geometry problems. Such software can be used to explore geometry examples, to do some observations, and draw conclusions. Such conclusions in many examples can be later proved by hand as valid geometry facts.

Symbolic geometry software may not be the best tool for teaching low level topics of geometry where we deal with basic relations and constructions. We have to find a good strategy of using these tools in order to not lose anything from the learning process and benefit at the same time from features provided by the software. The situation changes to a completely opposite while teaching geometry on a more advanced level. Here the easy-to-apply elementary features of the software will allow us to simplify the solving process and concentrate on producing a solution.

In this paper we demonstrated a number of advantages of the symbolic geometry approach. However, this approach still needs a lot of experimenting in a real classroom environment, testing how students will be accepting such approach, and how it will influence the learning geometry on various levels. It will be also necessary to develop teaching resources for teachers—textbooks and notebooks demonstrating in a systematic way how geometry can be taught with such software. At the moment there exists a large library of MuPAD notebooks for teaching undergraduate mathematics. Some of them address analytic geometry topics. However, this library is in German and English version of it may be available in a matter of a few months.

6 References

1. *Geometry Expressions Manual*, Saltire Software, Beaverton, 2006.
2. Coxeter H.S.M., Greitzer S.L., *Geometry Revisited*, The Mathematical Association of America, 1967.
3. Fuller G., *Analytic Geometry*, Addison-Wesley Publishing Company, Massachusetts, 1980.
4. Majewski M., *MuPAD Pro Computing Essentials*, second edition, Springer Verlag, Berlin, 2004.
5. Smart J.R., *Modern Geometries*, Brooks/Cole Publishing Company, Pacific Grove, 1988.