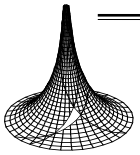

6/2 (2008), 281–288
tmcs@math.klte.hu
http://tmcs.math.klte.hu

Teaching
Mathematics and
Computer Science

On an analogy between spreadsheets and dynamic geometry environments

Reinhard Oldenburg

Abstract. There is a strong analogy between the fundamental way of operation of spreadsheet programs (SP) and dynamics geometry environments (DGE). We explain this analogy, demonstrate it in examples and consider didactical consequences.

Key words and phrases: dynamic geometry, spreadsheet, functional thinking.

ZDM Subject Classification: N80, U70.

1. Introduction

Spreadsheets and DGEs are the two software categories most often used in secondary mathematics education. Interestingly, the history of these two kinds of software is completely different. Spreadsheets were invented in the end of the 1970s on computers without graphical user interface (GUI) and the intended applications were in commercial and, to a lesser extend, scientific fields. The application of this software in schools had never been in the focus of software developers. The case of DGEs is quite opposite: These programs have been developed explicitly as teaching and learning tools and they evolved hand in hand with GUI progress. It is a remarkable fact that up to date the markets for educational geometry programs and professional computer aided design software are completely separated.

Nevertheless, our analysis will show that at a fundamental level SPs and DGE share a common computational model, namely the model of functional programming. Hence both kinds of software support a similar functional mode

of thinking and share a common approach to computer based problem solving. Furthermore, both kinds of programs are rather weak in supporting relational thinking and both kinds provoke similar student errors.

2. The analogy between DGEs and spreadsheets

The central concept that links the two program categories is the concept of function as found in functional programming languages. That is, a function is a computable map from one computer representable set to another. Given an input, it calculates a unique output.

Spreadsheets

Gieding [1] explained in detail that setting up a spreadsheet amounts to composing a functional program. The functions are specified by formulas in the cells. Thus a formula like $= A1 + B1$ in cell $C1$ sets up a function which takes its input values from cells $A1$, $A2$, and puts the result in $C1$. In more common mathematical language this may be written as $C1 = f(A1, B1)$, where f is the addition function. Function composition is realized by taking this output as the input of another function: The formula $= A1^*C1$ in $D1$ may be represented as a function $D1 = g\,(A1, C1)$ which, when unfolded is the composition $D1 = g\,(A1, f\,(A1, B1))$. Deciding alternatives is done using the spreadsheet function 'if' and recursion can be realized by "hand" by filling out a formula down a spreadsheet column.

As the user sets up the spreadsheet, each cell he uses gets one of the following three roles:

- Input cells: The cells where the user enters arbitrary data.
- Intermediate cells: Such cells carry information over to other calculations but are not interesting in them selves.
- Output cells: Here the user reads of the result of the calculation.

The latter two classes of cells update their values when one of the cells of the first class is changed by the user. Values can only be inserted into cells of the first class without destroying the functional program.

In the process of composing a spreadsheet the user has set up a topologically sorted directed graph on the set of all cells as vertices. A direct arc goes from one

cell u to a cell v iff u is used as an input in v's formula. Many SPs offer the option to show this graph as means to understand dependencies in the spreadsheet.

The functional program specified in that way is carried out every time the user changes a value in an input cell. Then the spreadsheet program calculates recursively all dependent cells.

Dynamic geometry

When we look at DGEs we see geometric objects instead of numbers, but internally these objects are stored analytically by their coordinates. Even more important, the geometric objects themselves stand in functional relationships similar to the one described above. Just as in SPs all objects fall into one of three classes:

- Input objects: These are the basic objects that can be dragged with the mouse.
- Intermediate objects: These objects are used in a geometric construction, but are not important to the user, therefore they may be hidden.
- Output objects: These are the objects that the user wants to see.

The latter two classes of objects update their values when one of the objects of the first class is changed by the user. The user can drag only objects of the first class.

Just as the formulas in a spreadsheet define a directed graph so does the construction in an DGE. There is a defined direction of the flow of information. If a new point M is constructed as the midpoint of A and B, then the position of M is calculated as a function $M = f(A, B)$. The flow of information is from the input A, B to the output M.

The functional program thus encoded in the construction is executed each time the user moves one of the basic objects.

Table 1: Analogy between SP and DGE

SP	**DGE**
strong analogies concerning software functionality	
Cells containing numbers entered by the user	basic objects
Cells containing formulas	dependent (i.e. constructed) objects
change the number in a cell	drag a basic object
recalculation of dependent cells according the functional relationship	recalculation of dependent objects according the functional relationship
slider	slider
if	Boolean points
strong analogies concerning software restrictions	
unable to specify output values (note that the solver tool of Excel does this)	unable to drag dependent object
unable to handle recursive cell reference	unable to identify a constructed point and a basic point
vague analogies	
automatic formula insertion	macros
missing analogies	
missing: way to store the values of a cell while another cell varies	locus
display dependency graph as path to precursor	missing: illustration of the dependency graph
missing: restrictions among several input cells	points on object
random numbers	missing: "random walk" of basic objects
broken analogies	
-	segments of fixed length

Example: Circumcenter of a triangle

A typical task for a DGE is to construct the circumcenter of a triangle. It is the intersection of the orthogonal bisectors of the triangle's sides. As explained, doing such a construction in a DGE is equivalent to setting up a functional program and, of course, the same functional program can be defined in a spreadsheet.

To keep things in this example as simple as possible, we use real x and y coordinates for points and the parameterization $y = mx + b$ for lines, although this excludes vertical lines.

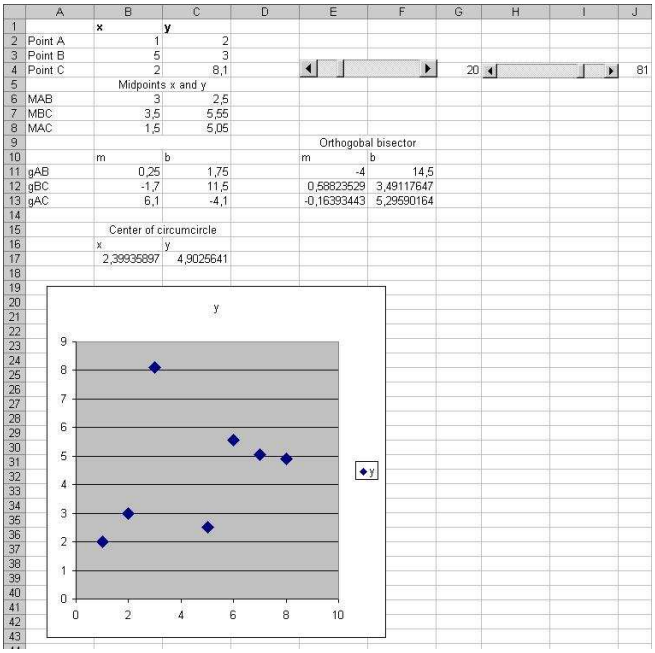

Figure 1. A spreadsheet calculation the circumcenter

The program is organized as follows: Input data are the x- and y-coordinates of the points A, B, C defining the triangle. These coordinates are contained in cells $B2 : C4$. The x and y coordinates of C can be altered by sliders. Of course one could add sliders for the coordinates of A and B as well. The midpoint of A and B, M_{AB}, as well as the other midpoints are calculated easily.

Next, we determine the coefficients m and b of the three lines through AB, BC, and AC respectively. From this, the coordinates of the orthogonal bisectors are calculated and finally the point of intersection is found. All these points are displayed in a diagram. Moving the slider shows how the circumcenter moves in functional response.

Of course, the functional program can as well be carried out in any programming language. Here we use the computer algebra system MuPAD:
Midpoint of two points:

	A	B	C	D	E	F
1		x	y			
2	Point A	1	2			
3	Point B	5	3			
4	Point C	=G4/10	=J4/10			
5		Midpoints x and y				
6	MAB	=(B2+B3)/2	=(C2+C3)/2			
7	MBC	=(B3+B4)/2	=(C3+C4)/2			
8	MAC	=(B2+B4)/2	=(C2+C4)/2			
9					Orthogobal bisector	
10		m	b		m	b
11	gAB	=(C3-C2)/(B3-B2)	=C2-B11*B2		=-1/B11	=C6-E11*B6
12	gBC	=(C4-C3)/(B4-B3)	=C3-B12*B3		=-1/B12	=C7-E12*B7
13	gAC	=(C4-C2)/(B4-B2)	=C2-B13*B2		=-1/B13	=C8-E13*B8
14						
15		Center of circumcircle				
16		x	y			
17		=(F12-F11)/(E11-E12)	=E11*B17+F11			
18						

Figure 2. The formulas in the spreadsheet of Fig. 1

```
MP:=(P,Q)->[(P[1]+Q[1])/2,(P[2]+Q[2])/2]
```

Line through two points. The result is a list of slope and y-axes-intersection:

```
g2P:=(P,Q)->[(Q[2]-P[2])/(Q[1]-P[1]),P[2]-P[1]*(Q[2]-P[2])/(Q[1]-P[1])]
```

Orthogonal line through point P to line g:

```
OL:=(g,P)->[-1/g[1],P[2]-P[1]*(-1/g[1])]
```

Point of intersection of two lines:

```
LLIS:=(g,h)->[(h[2]-g[2])/(g[1]-h[1]),g[1]*(h[2]-g[2])/(g[1]-h[1])+g[2]]
```

Circumcenter:

```
U:=LLIS(OL(g2P(A,B),MP(A,B)),OL(g2P(B,C),MP(B,C)))
```

Insert sample values and display:

```
AA:=[1,2]:BB:=[5,3]:CC:=[2,8]:float(eval(subs(U,A=AA,B=BB,C=CC)))
plot(plot::PointList2d([AA,BB,[cx,8],
LLIS(OL(g2P(AA,BB),MP(AA,BB)),OL(g2P(BB,[cx,8]),MP(BB,[cx,8])))],
cx=1.5..4.5),Scaling=Constrained)
```

In the resulting graphical representation one can even use a slider to move point C. To move another point defining the triangle one has to modify the plot command but not the construction.

Broken analogy

The analogy explained so far is, as noted in Table 1, not perfect. There are some DGEs that offer the possibility to construct segments of a fixed length. Such constructions break the analogy because they establish a relation, not just a function between the coordinates of the end points. Typically, the software then runs in several situations where such objects are difficult to handle und most DGEs do not support them at all.

The analogy of a segment of fixed length in a SP would consist of four cells for the coordinates of the two endpoints where the user can enter numbers in any of the fields but nevertheless entering a number would change the other numbers.

Such a behavior is difficult to implement (but take a look at the program TKSolver, `www.uts.com`).

3. Didactical consequences

DGEs and SPs are powerful tools because they are domain specific realizations of the idea of functional programming. From a didactical perspective this means that both tools can be used to support the development of functional thinking because they offer examples for and applications of this idea [3].

The fact that both software categories are build on the same principles can be used by establishing in classroom a language that stresses this. One should thus speak about dependent and independent variables resp. objects in both environments. Moreover, problem solutions can in both cases be expressed in verbal form, either as a construction description in the case of DGE or as a calculation description in the case of SP.

When working in with of these software environments the students have to design problem solutions that take into account the directed nature of the flow of information in such systems. To support this, one may use informal graphical planning to sketch the order of calculations resp. constructions. An important step is to refine such sequences until each calculation can be done by elementary calculations or constructions.

When the teacher supplies a spreadsheet or a construction as an electronic worksheet to allow students to explore phenomena they (the students) will operate in the same mode: They study the variation of output objects depending functionally on certain input values. This is quite similar to certain physical experiments, thus the didactics of science education should be consulted to gain insight in the learning process in such cases.

Naturally, the analogy also carries over to student errors. When using DGEs students often fail to come up with proper constructions. They produce a figure which visually seems to have the required property but does not preserve it when dragging. The same behavior has been observed when students compose spreadsheets: E.g. some students calculate certain results in their heads and insert the results rather than the formulas into the cells.

In DGEs students sometimes don't understand that they can't bind a constructed point to a line or a circle [2]. The analog situation with SPs is that

students insert a value into cell that should be calculated by a formula. This does not provoke an error message but it damages the functional program. In both cases the error can be detected because the resulting construction/calculation does not behave properly under variation of the input data.

The common functional basis has some disadvantages: Relational thinking is not supported by these tools. It is not possible to constrain a construction by a constraint. E.g. in a DGEs you can't add a posteriori the condition that two lines should be orthogonal if this has not been arranged from the beginning, and in a SP it is not possible to prescribe certain equations or inequalities between cell values. This restricts the value of the tool for exploration. A partial solution is the solver tool available as a plug-in in SP Excel. For geometry systems a complete redesign is necessary to overcome the functional approach [4].

This last remark hints at the possibility that this analogy may inspire software developers. The table above shows that not all good ideas from one world have been carried over to the other world yet.

Beyond the functional framework the question arises, whether relational thinking is supported well enough by current technology based learning environments.

References

[1] M. Gieding, *Programming by example – Überlegungen zu Grundlagen einer Didaktik der Tabellenkalkulation*, Mathematica Didactica, 2003, 42–72.

[2] R. Hölzl, *"Die konstruierten Punkte noch binden!" – Schülervorstellungen von der Cabri-Geometrie*, in: *H. Krautschitsch und W. Metzler: Anschauliche und Experimentelle Mathematik II. Wien: hpt.*, 1994.

[3] K. Krüger, *Erziehung zum funktionalen Denken*, Berlin, 2000.

[4] R. Oldenburg, Bidirektionale Verknüpfung von Computeralgebra und dynamischer Geometrie, Vol. 26, Journal für Mathematikdidaktik, 2005, 249–273.

REINHARD OLDENBURG
GOETHE UNIVERSITY FRANKFURT
SENCKENBEGANLAGE 9
60325 FRANKFURT
GERMANY

E-mail: oldenbur@math.uni-frankfurt.de

(Received September, 2007)