Art in GeoGebra
or Using GeoGebra as a Spaceship

Preface

When I first wrote the abstract to this workshop, Python was part of GG 4.2 which would be released on 1 September. So during the conference I thought everybody would be running GG 4.2 with Python. Now the situation is a little different and so part 3 of the workshop will be more like a talk on Python. For those wishing to practice Python you need to download GeoGebra 5.0 beta from

http://www.geogebra.org/webstart/5.0/geogebra-50-jogl1.jnlp.

The files used in this workshop can be found on GeoGebraTube

http://www.geogebratube.org/collection/view/id/1432

I hope you can use this either in your teaching of functions or to produce some amazing art yourselves ☺

September 2012

Jonas Hall / The Mad Mathematician
jonas.hall@klartskepp.se

Part 1 – Charting the sky

Hand painting

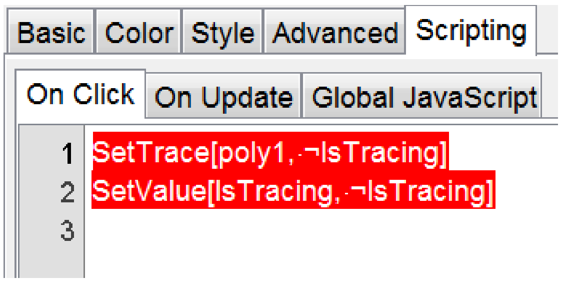
Use the tool **Rigid Polygon** to create a small rigid polygon in the shape of a thin rectangle. In the Formatting bar or the Properties dialog (right click and select Object Properties) decide on a nice color. Also put the Trace on and select Opacity = 40%. Voilà! You have just made a painting tool just like in PhotoShop ☺.

A script to toggle tracing

We can improve it a little bit by making a script that turns (toggles) Tracing on/off if you click the polygon. First create a boolean variable by typing the following in the input bar:

IsTracing = true

Open the Properties dialog, select the Scripting tab, then click the OnClick tab and type in the following (the "not" sign ¬ can be copied from the symbol palette in the input bar).



```
SetTrace[poly1, ¬IsTracing]
SetValue[IsTracing, ¬IsTracing]
```

Then make sure you click OK at the bottom of the dialog.

If you managed to do this you can now turn tracing on/off by clicking on your polynomial "poly1". However, if you plan to do any serious painting like this – get Photoshop!

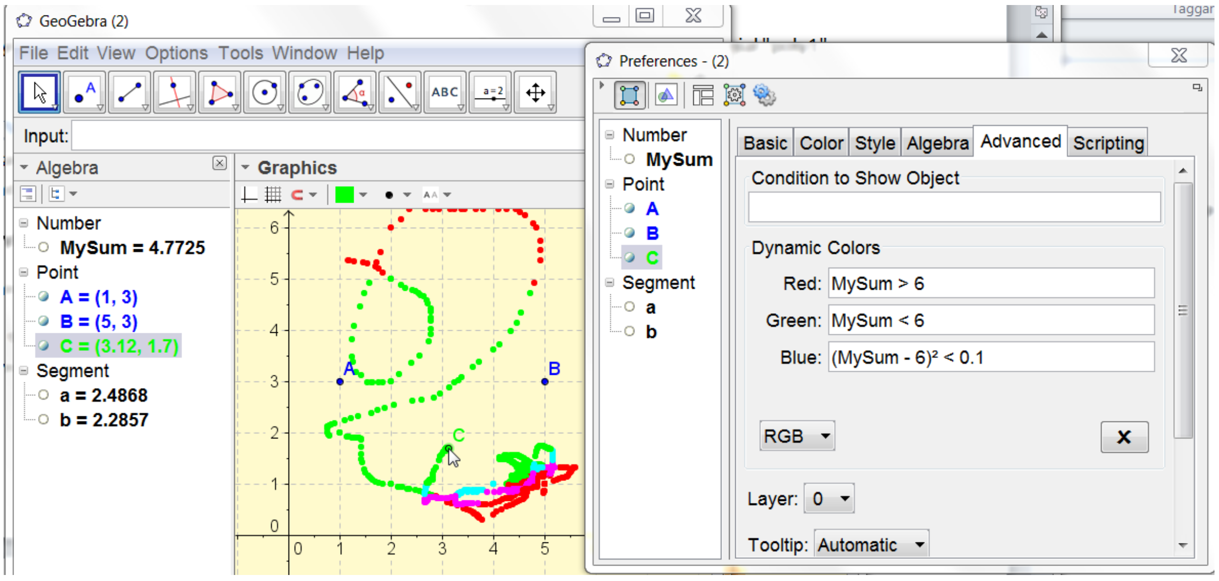
Dynamic color

Start a new window with Ctrl-N. Then create three points A, B and C. Then create segments a = AC and b = BC. You may then hide the segments. Type in the input bar:

MySum = a + b

This is the sum of the lengths of the segments. Now to the interesting stuff.

In the Object properties dialog for point C, on the Advanced tab, fill in the following in the fields for dynamic color:



Algebra:
Number
MySum = 4.7725
Point
A = (1, 3)
B = (5, 3)
C = (3.12, 1.7)
Segment
a = 2.4868
b = 2.2857

Preferences - (2), Advanced tab:
Condition to Show Object:
Dynamic Colors
Red: MySum > 6
Green: MySum < 6
Blue: (MySum - 6)² < 0.1
RGB
Layer: 0
Tooltip: Automatic

GeoGebra (2)
File Edit View Options Tools Window Help
Input:
Graphics
Basic Color Style Algebra Advanced Scripting

Then turn on tracing for point C and drag it around. "Paint" the canvas with the points trace. What shape emerges?

More on Dynamic color

The dynamic colors we used in this example were **conditional statements** with values **true** or **false**, which were interpreted as numbers 1 or 0, respectively. However, the color values can be anything between 0 and 1 for each of the three primary colors red, green and blue. In particular, this value could be the result of a function. Values outside [0, 1] are also accepted but the integer part is dropped (or something).

Colorful functions

Start a new window with Ctrl-N. In the input bar type:

f(x, y) = x^3y – y^3*x* – 10 This is a function of two variables.

This may be drawn by the command

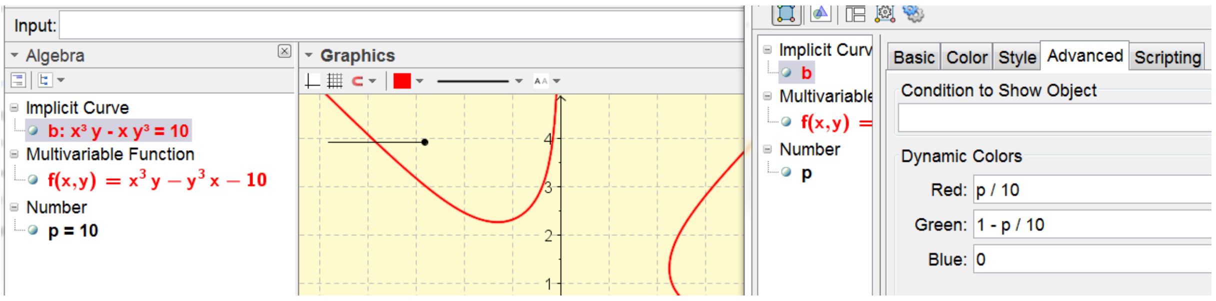
ImplicitCurve[x^3y – y^3*x* – 10] or in our case simpler **ImplicitCurve[f]**

Now introduce a slider called p. Let p = 10 and replace 10 with p in *f(x, y)*. Try changing p. Now add some beauty:

In the dynamic color fields, type in the following:

Red: p/10
Green: 1 – p/10

Then set **Trace on** for the curve and **animate** the slider p. Voilà!



Implicit Curve
b: x^3 y - x y^3 = 10
Multivariable Function
f(x,y) = x^3 y − y^3 x − 10
Number
p = 10

Condition to Show Object

Dynamic Colors
Red: p / 10
Green: 1 - p / 10
Blue: 0

Try changing the line thickness for the curve. Also try to make the slider step smaller.

Experiment (in a state of joy) with the colors for a while and also with the function *f* itself. Who can make the most beautiful image?

Saving the image

In the menu, choose **File – Export – Graphics view as image...**

Part 2 – Launching

This is really important!

So we have seen how we can investigate functions of two variables with color. In effect, what we have done is to **completely change the way we look upon functions**. Rather than investigating which *x* and *y* satisfy certain conditions and therefore belong to the function, we accept **all** values for *x* and *y* and assign a color to that point. This reveals a lot more detail than was previously possible, and at the same time, is more pleasing to the eye ☺. We will now attempt to do this a little more systematic.

Combining the spreadsheet with animation and dynamic colors

An efficient tool for handling many items at once is the spreadsheet. Start a new window with Ctrl-N and open the spreadsheet window from the **View** menu or pressing **Ctrl-Shift-S**.

You now face two options. You can either:

- open a ready-made GeoGebra worksheet (**File – Open web page…**) from either http://www.geogebratube.org/student/m17285 (almost exactly like these instructions), or http://www.geogebratube.org/student/m17286 (a little neater)
  and play with it and figure out its construction later, or

- follow the instructions and make your own which will take longer time and will not allow playing for so long but will teach you the techniques more hands-on.

If you choose to open the readymade worksheet, you will find some instructions there. If you chose to make your own, carry on reading.

Now Excel can handle numbers and formulas, but the GeoGebra spreadsheet can handle **objects** ☺. We will soon create 101 animated points but first we need to make a plan...

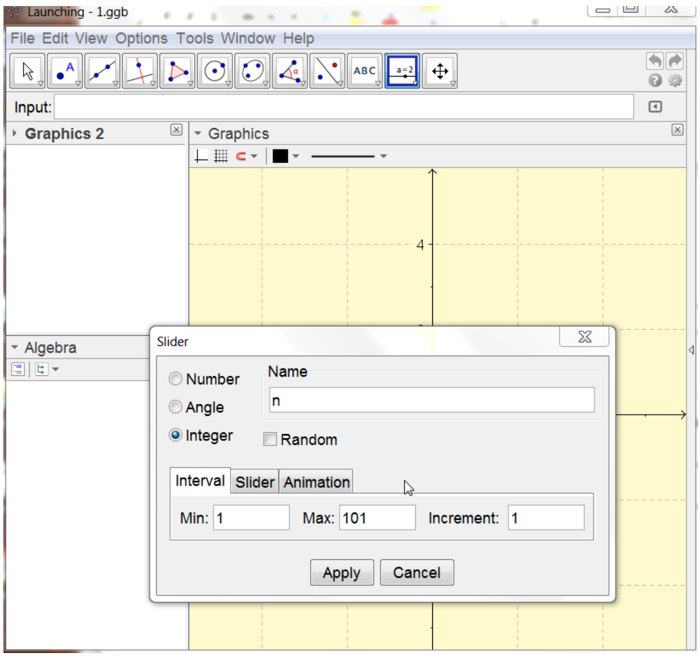
We will create a column of points that will all animate at the same time, all tracing with dynamic color at the same time, sweeping out an image, rather like a slow radar plot, to show us what lies beyond our screens. For the animation to work we first need a **slider** that controls the animation…

The construction

1. Start a new window with Ctrl-N. Make sure you see both axes to a bit more than [-5, 5].

2. Open Graphics View 2 and make it small and out of the way, perhaps above the Algebra View.

3. In Graphics View 2, create a glider called n that goes from 1 to 101 with step 1. This will represent the number of trace positions in the *x*-direction.



Number
Angle
[✓ Integer]
Name: n
Random
Interval | Slider | Animation
Min: 1
Max: 101
Increment: 1

4. Open the Spreadsheet View (Ctrl-Shift-S). In column A, enter numbers from 1 to 101 (enter 1 and 2, select both cells and copy down). This will represent the number of points in the *y*-direction.

5. In cell B1, type **=n/10 – 5.1**. This will represent the *x*-coordinate which is controlled by the slider.

6. In cell C1, type **=A1/10 – 5.1**. This will represent the *y*-coordinate which is controlled by the row number.

7. In cell D1, type **=(B1, C1).** This will create the first point. This point needs to have its properties correctly set before we copy it down.

8. First, we don‘t want labels to show. In the menu, select **Options – Labeling – No new objects**.

9. Set Trace On for the point. Save the file and prepare for the interesting part…

Color Functions

First a word on color functions. Any function has a **domain** and a **range**. The domain is all values of *x* that the function is defined for, and the range is all *y*-values that the function can take. Normally the range is quite large, often] -∞, ∞[. But in the dynamic color fields we need to enter something in the interval [0, 1]. We therefore need to transform (map) the output of our function from a large interval to a small interval. How do we do this? Well, we use **color functions**.

Typical color functions are log(*x*), arctan(*x*) and 1/(abs(*x*)+1). They take big values and make them small in one way or another. Ideally, they map] -∞, ∞[onto] 0, 1 [. They may be modified with parameters to allow all primary colors to have different color functions.

In our case, our original function can be the one we used before $f(x, y) = x^3y + y^3x - 10$.

Then we can define color functions like red(*x*, *y*) = atan(*f*(*x*, *y*)) and green(*x*, *y*) = 1 – red(*x*, *y*). This construction will make for a nice red-green blending. You may experiment with your own of course.

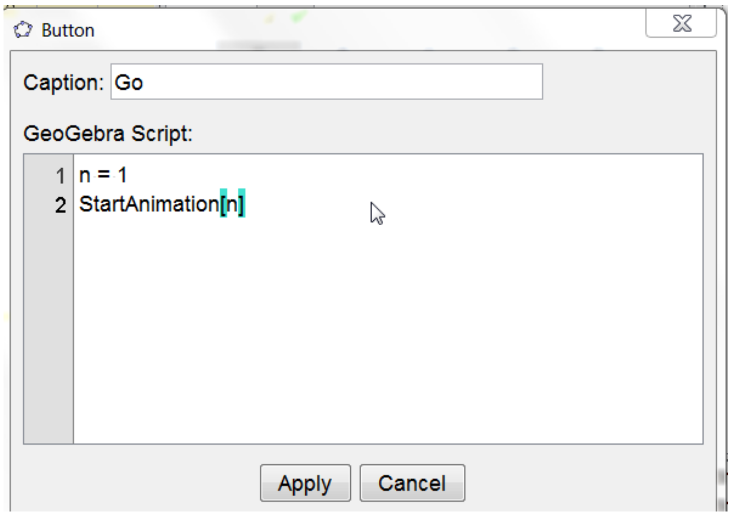
Showing the students functions like these will give them a deeper understanding of what a function is and how it is used. You can also discuss transformations, mappings and modeling.

Back to GeoGebra:

10. In the input bar, type **f(x, y) = x^{3}y + y^{3}x – 10**
11. In the input bar, type **red(x, y) = atan(f(x, y)),** and
    green(x, y) = 1 – red(x, y)
12. In the dynamic color fields for the point in D1, type **red(B1, C1)** and **green(B1, C1).**
    Save your file before the next step!
13. Now we are ready to copy B1, C1 and D1 down to row 101. You do this the ordinary way (like in Excel) by first selecting these three cells and then dragging the little black square in the bottom right corner of cell D1 down all the way to row 101. If successful, you should have created a long column of colorful points from (-5, -5) to (-5, 5). If this worked, you can save your file again.
14. A final setting: Right click on the slider n and select **Object properties**. On the **Slider** tab select **Repeat: Increasing (Once)** and save.
15. Now right click on the slider n, and select **Animation On**. Voilà! You have created a deep-space radar for mathematical spaces ☺
16. Experiment by changing the color functions and the original function. Again: Who makes the most beautiful image...? Don't forget to save some images by **File – Export – Graphics View as Picture...**

Bonus Extras

There are some things you can do to make life easier. Input boxes for the functions make it easier to change them. If you use color functions with parameters these can also be controlled by input boxes. And you can make a command button to re-run the animation. This button can be constructed by using the Command Button tool: and then typing in the following simple script:



Caption: Go

GeoGebra Script:

```
n = 1
StartAnimation[n]
```

Now obviously, with an increased number of points, reduced point size and a smaller step value for the slider you will get better resolution. But it comes at a heavy prize. GeoGebra, with all its internal dependencies, is a slow number cruncher. Increased resolution will result in increased processing time (and a higher risk for a program crash). The answer to this is to take the geometry out from GeoGebra and just program it in Python…

Part 3 – Going to the stars

This is about relatively simple programming, also known as scripting. You have already done a simple script in GeoGebras own scripting language when you made the "Click-me-to-toggle-tracing"-script. Scripting is just about piling a lot of commands on top of each other in such a clever way as to do make the program do what you want it to do.

GeoGebra currently handles two different scripting languages. GeoGebra's own language, i.e. the GeoGebra commands, and JavaScript for those situations where you want the power of a proper programming language. Unfortunately Javascript is not the easiest language in the world to start with. Python, on the other hand, is.

Python is a programming language **designed to be simple**, yet powerful. It was included in the beta version of GeoGebra 4.2 but was moved forward to version 5. You can search Khan Academy for instructional videos for Python if you want to learn programming in this language. Unfortunately, the examples aren't adapted for GeoGebra.

Basically, I'm just going to give you some code (se next page) which will produce a simple image. This script is stored in a GeoGebra file called "Art with GeoGebra.ggb"
http://www.geogebratube.org/material/show/id/17339

How to view this in GeoGebra

Open the file in GeoGebra version 5 (Open WebPage…) . Then by opening the Objecgt Properties and selecting the Scripting tab you can view and change the script.

If you open the Python Window (**View – Python**) you will see that there is a global script with the bits that need to initialize as we start GeoGebra, and there is also a Python script connected to a command button which does the rest. (NOTE: At time of writing this is buggy…)

You can use this file to play around with different functions, different color functions different regions etc.

To learn more about Python in GeoGebra look at this: http://dev.geogebra.org/trac/wiki/Jython.

Generic image making script.py

```
# Bring in Java.  These three statements imports commands that we use that aren't in Python.
from java.awt.image import BufferedImage
from javax.imageio import ImageIO
from java.io import File
from math import atan, sin, cos, pi   # Don't forget this if you need it
```

```
# Useful constants. Here we define the file name and where to save it as well as
# the size and resolution of the image.
dir = "C:/GeoGebraoutput/"# Directory to save files
name = "Trial001"         # Prefix to filenames
sizeX = 500               # Number of pixels in x
xmin = -8
xmax = 8                  # x-coordinates of canvas
ymin = -8
ymax = 8                  # y-coordinates of canvas


# Define colour function, in this case 1/(1+x). Colors in Python work a little different from
# colors in GeoGebra so we have to make one (hexadecimal) number of the three values.
def color(value):
  "This function generates a colour integer from value"
  if (value < 0): value = -value
  cr = int( 255.99 / (1.00003457 +value/10) )   # Set red color
  cg = 255 - cr                                 # Set green colour
  cb = 0                                        # Set blue color
  return cr*256*256 + cg*256 + cb


# Define function. This is the function we are interested in.
def myfunction(x, y):
  return y*x**3 - x*y**3 - 10


# Initialize step sizes and SizeY which we need for the loops
deltaX = (xmax - xmin)
deltaY = (ymax - ymin)
aspect = deltaX/deltaY
sizeY = int(sizeX/aspect + 0.001)
stepX = deltaX/sizeX
stepY = deltaY/sizeY


# Create the image object. This is highly technical, but it works :-)
img =  BufferedImage(sizeX, sizeY, BufferedImage.TYPE_INT_RGB)
g = img.getGraphics()


# Step through every pixel
for x in range(sizeX):
  for y in range(sizeY):
    # For each pixel we calculate the interesting stuff
    xF = xmin + x*stepX      # Set coordinates of free point
    yF = ymax - y*stepY      # Minus because image counts rows from top down
    # For each pixel we calculate the interesting stuff
    img.setRGB(x, y, color(myfunction(xF, yF)))  # Set the value using the color function

# After the loops have ended we save to a file.
if not ImageIO.write(img, "png", File(dir + name + ".png" )): print "Save failed!"
```

Epilogue – The Neuberg Conic 4-space

So what function did I use for the art exhibition "A Different World“? Well it’s a rather complicated function. It started with a thought: "What if I could define an **Euler line** for a quadrangle".

The **Euler line** is defined for triangles, being a line through the center of mass, the intersection of the altitudes and the intersection of the perpendicular bisectors of the sides amongst many other points.

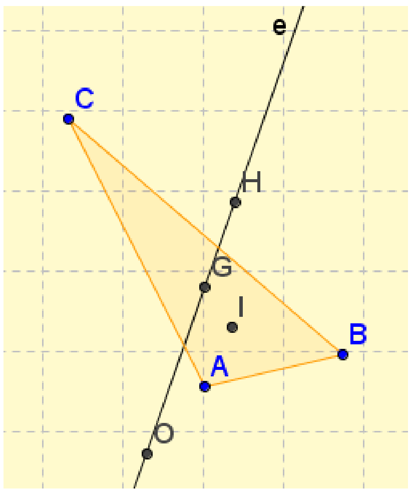
In GeoGebra you can easily make an Euler line by first creating a triangle ABC. Then type in the following commands (don’t forget that arrow up/down allows you scroll through your previous commands):

I = TriangleCenter[A, B, C, 1]	This is the incenter (center of the inscribed circle)
G = TrianglelCenter[A, B, C, 2]	This is the centroid (center of mass)
H = TriangleCenter[A, B, C, 3]	This is circumcenter (center of the circumscribed circle)
O = TrianglelCenter[A, B, C, 4]	This is the orthocenter

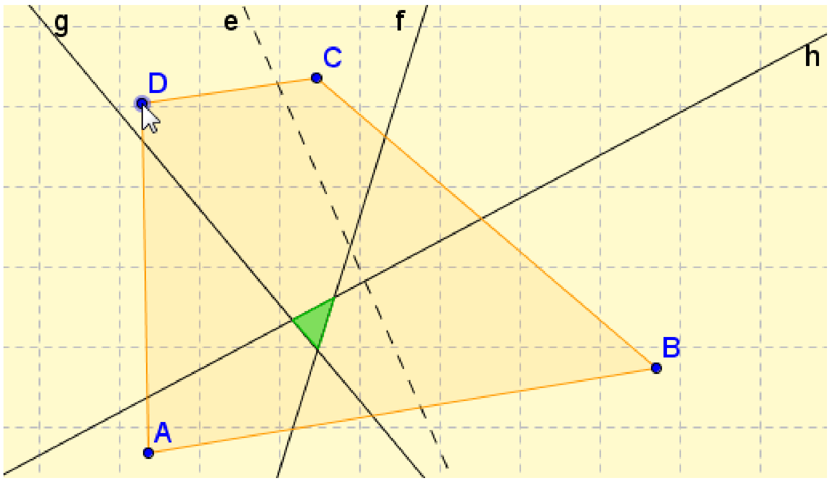
Drag point A and watch them move. Notice that three of them lie on the same line all the time.

e = Line[H, O]	This is the Euler line

Drag the vertices of the triangle and play with it for a while. The Euler line twist and turns which makes it interesting.



I quickly realized that I couldn‘t make Euler lines for quadrangles but I also discovered that since each quadrangle can be dissected into 4 different triangles, I could associate no less than 4 Euler lines to the quadrangle. If the Quadrangle is ABCD, I created the Euler lines for triangles ABC, BCD, CDA and DAC. Playing about with this I also realized that if I moved one of the four corners, say D, around then sometimes all 4 Euler lines would come together in a single point. Try it yourself! You may want to make a new tool for making an Euler line first.



So I started to map these points, looking for ways to describe it. The key moment of discovery was when I started measuring an area enclosed by the three lines that moved when I moved point D (It's easy, just make a new triangle from the points of intersection). This area would be 0 when they coincided but also take other values when they did not coincide. This allowed me to map the area to a color... And there it was in all its beauty. Call it a beast, or a world.

Now, without loss of generality, I can place point A at coordinates (0,1) and points B and C on the unit circle. Their positions can be described by a position angle for each point, so two position angles. Point D is the point sweeping across the canvas, generating the color. This means that for each combination of the two position angles I get a unique 2D-image. 2 parameters and 2 dimensions is the same as a mathematical 4-space. You can traverse this 4-space in some direction and make a film of this object ☺.

It was not until recently I discovered that the points of coincidence in each image form a known curve that is called the Neuberg Conic. This only shows the white curve in the images though. It can easily be viewed in GeoGebra 4.2. Using our triangle ABC, just type **Cubic[A, B, C, 1]** in the input bar!

Though the white part has been known since early this century, I truly believe that I am the first to have witnessed the beautiful dark loops of the Neuberg Conic 4-space. I'd be interested to know if you find any references to this in the literature. If you do, drop me a line ☺