# Journal of Robert Vanyo for Fall Quarter 2008

Robert Vanyo

Fall, 2008

## Week of September 23, 2008

This quarter the Sets group will be making a GUI to accompany the Sets program. We quickly focused our efforts on the dual task of learning the use of the Tkinter package for Python (which provides the tools for building windows and widgets for software), and actually designing what our windows will actually look like and what features they will actually have. One important feature we hope to include is an image viewer, since a central function of the Sets program is to generate images. In the past week, Krishna and I have been trying to figure out how the image viewer aspect of the GUI might work. We have been exploring the possibility of using the Visualization ToolKit, or VTK, which is a software system that handles image processing and 3D graphics. VTK is usable with Python, and there are packages that can bring VTK and the Tkinter windows together, which is exactly what we need. My challenge this past week, and of the coming week, is to get VTK up and running on my computer, and also getting the packages that link VTK and Python installed. I have already attempted to install both VTK and the necessary packages, but I am not at all a computer expert, and for some reason the VTK examples and the Python-VTK examples do not work. I feel like the VTK-Tkinter combination would be perfect for our image viewer needs, so I hope to have the VTK examples working as soon as possible.

## Week of September 30, 2008

This past week, I discovered that some of the examples I was trying to run were outdated and therefore would not work under the current VTK version. When I tried some examples for Python that actually came with VTK, they did work. Therefore, I discovered that I had, in fact, successfully installed VTK. So, I then tried to figure out just how VTK images can be incorporated into windows created in Python using the Tkinter package. After some poking around through some working examples found on the internet, I found that a package already exists and is already installed for Python that allows the integration of VTK displays and interactions into Tkinter windows. The problem now is not only figuring out from these examples how to use such tools to generate such a window with an interactive image, but also figure out how to create the image itself. Many of the examples use stock images already present in the VTK distribution, we need to be able to generate our own interactive image, whatever that may be, and then incorporate it into our window. We also need the ability to save the image and re-access it later, instead of having to recreate it each time we need it. So I'm still trying to learn the use of VTK (which is difficult because I haven't found any good free tutorials yet), and will continue to do so this week.

## Week of October 14, 2008

For the past two weeks I've been reading over a tutorial (actually, more of a textbook) that I found for VTK. It is very long and very detailed, but I feel that I have a basic understanding of the visualization process, and even though the book deals primarily in C++, I can compare examples from there to Python examples that came with VTK and have a basic understanding of what is going on and what the various methods and commands do. This is a great improvement, as before I was trying to work exclusively from examples, and had no idea what was happening.

Now that I generally understand what happens in a VTK code written in Python, I just have to find more examples to learn from and imitate, examples that not only deal with Tkinter, but also with the rendering of 2D images. This is the current problem I am having, because the book I was learning from deals primarily in the visualization of 3D objects, but our goal for now is to work with a 2D image, as that is what our Setmaker program produces. I have searched online for a few examples of a 2D rendering with VTK, and I know that there are several ways to generate a 2D plot. My problem now, specifically, is finding a method of generating a 2D plot that will allow for the sort of user interactivity we seek (not all methods do this), as well as figuring out how to take our list of xy-coordinates and angles from Setmaker, and translate that into something that can be mapped to an actor in VTK that will then be rendered. Basically, the few 2D examples I found that deal with plotting xy-points call on external sources for their data, and not sources generated in the Python program itself. I need to find out what sort of form these sources take so that the xytlist from Setmaker can be modified into this form.

## Week of October 21, 2008

This past week I wrote a script that has the subroutine that will display our Sets data as a VTK visualization. To solve the problems of last week, instead of trying to make the whole set of datapoints from Setmaker into something that VTK can interpret, I map one point at a time to coordinates in the renderer. Basically, instead of having one big file that contains the shapes, locations, and colors of each point, I have a source file that only has the shape of the point. It is called up, colored, and put at a specific location in the render-window itself. So the program will loop and render thousands of points individually, instead of doing them all at once. The disk-shaped source used for the points is really the default VTK cone-source, with a very short height, as suggested by Dr. Martin. Also, basic camera-related interactions are currently working, so the image can be rotated and zoomed. Next, after some tidying up of details, we'll be trying to design some more advanced interactions to boost the overall functionality and usefulness of the program.

## Week of October 28, 2008

Over this past week I've been trying to figure out how to add additional actors (more points) to our visualization without having to close the viewing window and start over. The way the viewing window works now is that once the window is rendered, the program runs through a loop that waits for events, such as mouse clicks and button presses, to respond to. To my knowledge, to end this loop, the window must be closed. So I am trying to find out how to interrupt this loop, add more points, and re-render without actually closing the window. I tried first to make some sense out of Mayavi, a program written in Python that already accomplishes this. However, I am not familiar with the writing of classes in Python, or any language, and do not quite understand it. Still, I next tried to convert the setvis window that the program currently generates into a Tkinter widget, since the whole GUI for the setmaker program will be utilizing Tkinter tools. The default VTK render window does not include the option to attach other widgets such as buttons and text boxes. Once I can render our image into a Tkinter window, we will be able to attach other widgets to the viewing window of our program. But for now, the real concern is that the VTK interactor loop, the loop that waits for events for the VTK viewing window currently in place, may not work once I render into a Tkinter window, since Tkinter windows have their own loop that waits for events. So once I figure out what kind of loop, VTK or Tkinter, will be managing our VTK rendering packed into a Tkinter window, I can then try and find out how to pause the loop and add more points. Unfortunately, I've been stuck. VTK comes with the necessary tools to use it in Python, one of which is a VTK renderer designed to be packed as a widget into a Tkinter window. But in trying to convert the program to render into a Tkinter window, I found that for some reason, Python can't find the vtkTkRenderWindow class that I need in order to do this. I suppose there is some sort of path problem that needs to be solved, but I have thus far been unable to solve it.

## Week of November 14, 2008

Since the last entry, I have figured out how to add actors without breaking the mainloop and successfully built the viewing window as a Tkinter object and packed it into the GUI. Furthermore, I created a button and a function that will clear the image of all actors (points). Also, I re-organized the building of the viewer by making it into a class of its own. This presented a challenge as I was unsure how to pass some of the necessary variables to buttons and functions between classes. Nam and Krishna fixed this issue for me. I also wrote a function that re-orients the camera in the viewer to its original state so that all actors are put in the picture when the function is implemented.

The problem now with the viewer is that since it automatically resets the zoom using a vtk method whenever the plot button or reset view button is pressed, sometimes the automatic zoom setting is inconvenient. This is because when it resets, it makes sure all actors are visibile. However, sometimes there will be a point plotted far from all the rest, and the interesting part of the picture becomes very small and far away because the camera zooms out to accomodate this far-away point. To fix this, but make sure the interesting part of the picture is still conveniently and automatically visible, is a challenge. I thought to create a new button that saves a zoom factor from the current zoom setting, so that the user can revert back to some desired saved zoom level easily, and not be subject to an automatic zoom reset that may be inconvenient. However, I am having difficulty finding vtkCamera methods that will help me do this.