Stability Leads to Meta-Evolution

1/4/2011

Stable systems become depended upon, forming new stable systems. This process continues indefinitely and has been occurring indefinitely throughout the universe. The history of this phenomenon can be broken down according to scale of time and space:

- Galactic
  - Time: since the Big Bang
  - Space: The entire universe, Galaxies, Solar Systems, Planets
- Geologic time
  - Time: since the birth of Earth
  - The Earth
- Biological evolutionary time
  - Time: since the emergence of life
  - Space: The Earth, Species
- Memetic evolutionary time
  - Time: since the first humans
  - Space: Land, Human activities
- Industrial evolutionary time
  - Time: since the birth of civilization
  - Space: Land, Goods, Services
- Software evolutionary time
  - Time: since the first computer
  - Space: inside computers
- Internet evolutionary time
  - Time: since the Internet
  - Space: on the Internet

I would like to find a suitable visual representation of all time and space, so that anyone in the future can explore the history of time and space for themselves using the tools I develop. This necessarily requires the ability to represent phenomena at many scales, and thus the ability to easily transition between scales. I argue that the most effective way of presenting and navigating multiscale structures is through the use of a mapping between the raw structure and a two dimensional visual representation of the structure. When this is done, two dimensional panning and zooming of a multiscale image corresponds directly with the traversal of a multi-scale data structure.

An Interactive Graphics Framework

¼/2011

Github Repository

To visualize data we'll need to first establish the basics of what "visualization" *is*. To define what visualization is, we'll need to define the things that make it up: data, graphics and the mapping between the two.

2D Graphics is our realm of operation. Pixels on the screen updating at 60 frames per second is our output. Mouse, Keyboard and multi-touch events are our input. Our 2D graphics framework should function well on Android devices, large multi-touch displays and powerwalls.

2D Graphics is essentially dynamic pixels. Above the level of pixels, we have triangles. Above triangles, we have the following graphics primitives:

- circle (x,y,radius)
- line(x1,y1,x2,y2)
- polygon(xs,ys,n)

drawn using the following stateful variables:

- Fill on/off
  - boolean fill
- Fill color
  - int fillR, int fillG, int fillB
- Stroke on/off
  - boolean stroke
- Stroke color
  - int strokeR, int strokeG, int strokeB
- Stroke weight (thickness)
  - double strokeWeight //stroke
- Cap style {BUTT, ROUND, SQUARE}
  - int capStyle

Additionally, there is an image drawing API:

- loadImage(image) //returns an image ID
- drawImage(imageID,x,y,width,height,rotation)

and a text drawing API:

- loadFont(font) //returns a font ID
- text(fontID,x,y, scale, rotation)

We make the assumption that it is cheap for the application to fully redraw itself every frame, and it may often need to to so due to the high interactivity of the application. This makes sense if the application has features such as smooth zooming and panning, brushed selection and global probing. This assumption also greatly simplifies the programming required, as the complicated issues involved with the dogma of "we must only ever redraw those portions of the screen which have changed" disappear entirely. The elegance of Processing programs stems from this way of thinking.

Since the application will redraw itself entirely every frame, the code which executes every frame will really be all about *efficient data structure traversal*. The multiscale data structures required for zooming and panning across time and space are ideally suited to such a graphics paradigm. They typically are trees and have running times of O(log n) for the search and bounded traversal algorithms required for graphical representation of a limited portion of their structure.

Multiscale data structures and their associated algorithms also have the characteristic that they can scale tremendously. For example, a structure for multiscale polygon containment hierarchies (see Quadstream) reaching petabytes of size can live in a distributed database spanning thousands of computers, while retaining the quality that any small region within it is readily accessible and can be then loaded into main memory and rendered as 2D graphics. In addition, zooming and panning can employ prefetching of remote data to give the illusion of having all data locally available. These scalable qualities are sought in all data structures used to visualize portions of the universe.