

Test suite for viewing parametric functional transformation matrices

Abstract

Transformation matrices are used for a variety of applications in graphics programming. They provide a convenient way of manipulating graphical objects while maintaining the original geometric properties of the graphical object in question. In this project, I intend to implement a test suite that will allow the user to create transformation matrices that can be defined in terms of a function of a parameter. As the parameter is modified, the matrix changes accordingly and is applied. If the parameter is varied according to time, this can be used to create animations defined in terms of functional matrices.

The idea

A basic example

A rotational matrix is defined as:

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where θ is the angle by which the object is to be rotated. Now, if θ itself were to be mapped to time i.e. if θ ranged from 0 to 2π when time ranged from 0 to 10, then, when applied to a graphical object, it would simulate the animation of an object rotating 360° in 10 seconds.

This mapping of time 't' and the parameter θ need not be linear. Let's say, we had a function that mapped 't' to θ such that the θ increased rapidly initially in both the 5 second time periods in the 10 second animation and changed gradually in the middle 5 seconds or so. To do so, we can use an exponential decay function for θ and the matrix that would be obtained would look something like:

$$\begin{bmatrix} \sin(2\pi e^{(\frac{t-5}{10})^2}) & \cos(2\pi e^{(\frac{t-5}{10})^2}) & 0 \\ -\sin(2\pi e^{(\frac{t-5}{10})^2}) & \cos(2\pi e^{(\frac{t-5}{10})^2}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where 't' is the time. What we have essentially done is mapped θ to time such that:

$$f : [0, 10] \rightarrow [0, 2\pi]$$
$$f(t) = e^{(\frac{t-5}{10})^2}$$

This way we have animated rotation with exponential easing. This can be further extended to bezier easing or pragmatic easing curves. If the user is now allowed to specify the matrix using a scripting language, with a host environment, then a lot is possible.

An advanced example

Assuming a transformation matrix M with dimensions 3×3 , where $M[i][j]$ denotes the element in the i^{th} and j^{th} column. Also, the host environment has the ability to pre-run a script where a user can pre-define any functions that are to be used throughout the cell-specific scripts. Let's take the example of a pre-run script:

```
var aspectEasing = function(t) {
  var aspectRatio = graphics.getHeight()/graphics.getWidth();

  if(aspectRatio < 1) {
    aspectRatio = 1/aspectRatio;
  }

  return (2*Math.PI)*Math.pow(e, Math.pow((t-5)/10*aspectRatio, 2));
}
```

What this does is, it will make the easing curve more rigid for images which have a huge difference between their width and height and smooth for images which have close to equal width and height. Now, to implement the functional matrix:

```
M[1][1] = function(t) {
  return Math.sin(aspectEasing(t));
}

M[1][2] = function(t) {
  return Math.cos(aspectEasing(t));
}

M[1][3] = function(t) {
  return 0;
}

M[2][1] = function(t) {
  return (-1)*Math.sin(aspectEasing(t));
}

M[2][2] = function(t) {
  return Math.cos(aspectEasing(t));
}

M[2][3] = function(t) {
  return 0;
}

M[3][1] = function(t) { return 0; }
M[3][2] = function(t) { return 0; }
M[3][3] = function(t) { return 1; }
```

NOTE: The language used here is ECMAScript (JavaScript). The methods `graphics.getHeight()` and `graphics.getWidth()` are provided by the scripting host environment. Furthermore, if the graphics object is to be scaled as well, the function can be replaced as:

```
M[1][1] = function(t) {
  var scaling;
```

```
if(t < 2) {
    scaling = 1;
} else if(t < 6) {
    scaling = 4/(t-2);
} else {
    scaling = 2;
}

return scaling*Math.sin(aspectEasing(t));
}
```

In this case, the graphics object, while being rotated with an aspect-ratio dependant exponential easing will be scaled along the x-axis as per time. It will remain at its original size for the first 2 seconds, will gradually animate to twice its size in the next 4 seconds and remain at twice its size after that.

Implementation details

This project will be implemented in C++ written using the Qt library. It will be a cross-platform application running on all platforms supported by Qt (Linux, Windows, Max OSX, BSD and major *NIX systems). The scripting environment will be ECMAScript and the engine used will be QtScript for the same, and a custom host environment will be designed for each cell to access the data.

This will be a GUI-driven application and the GUI widgets will be developed using the Qt toolkit. As in the previous example, each cell is defined in terms of a 'code string'. This string when evaluated MUST return a valid ECMAScript function. The function will be called every time the time parameter steps up or down. This is so that functions are evaluated lazily i.e. only when required. Cell scripts can utilize the values of other cells while returning the transformation matrix values and other data that is provided by the host environment that is going to be implemented.