Real Time Acrobat Animation: A 3-Phase Power Simulation

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
copyright c2005 as GuruGram #47
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

Adobe's recent addition of JavaScript over Acrobat has dramatically upped the animation possibilities done from **totally within** an **Acrobat .PDF** document.

Obvious immediate uses are the not-previously-possible timed slideshows within a browser window, scroll and marquee effects, or flashing text in a **PowerPoint Simulation**.

I was playing around with some simple **Acrobat** animation using JavaScript scripting and was utterly amazed at how fast and how large Acrobat animation can now be done. All at surprisingly low file sizes and rather simple program constructs. Ferinstance, nearly 20 frames per second seem possible on an older and sedate 850 MHz Pentium XP. While showing 200x200 pixel images. And needing only 8K or so per animation cell for storage.

Before we go into some of the details, let's first click here to...

RUN THE DEMO

Yeah, I coulda built the demo into this **GuruGram**, but let's instead keep things brief and simple. What you are viewing here is a simulation of the electric field surrounding a three-phase power cable as it varies through each of its cycles. This is an expansion of the third example of our new **Rebounding** method of quickly and simply analyzing complex electromagnetic fields.

Getting Started

The key Adobe document needed is the **Acrobat JavaScript Scripting Reference**. This should be built into the help menu of your copy of Acrobat. Otherwise, you can **click here** for a copy of this and other JavaScript support docs.

There are often five steps involved in Acrobat animation...

Design an animated sequence.
Build up the individual Cells.
Assemble a continuing sequence.
Single step the full sequence.
Add user interactive control.

Your animated sequence could be either **PostScript** images or vector graphics and text. Or a mix of the two. Some browsers may be faster and better about glitchlessly displaying pure images, so images may end up better…

Always test your animation WITHIN a browser to avoid any rude surprises over glitches and continuity!

The third field plot example in **REBOUND1.PDF** clearly lent itself to a repeating sequence of **200 x 200** images that trace three phase power over one full cycle.

Twenty-four cells of fifteen degrees of phase shift each seemed about right for the animated display. The more cells, the **smoother** the action, but the **slower** the maximum possible speed. The original field **PostScript** code can be modified for different phases by adding…

```
/phase 195 store       % set phase here

/calca {phase cos 500 mul 500 add} store
/calcb {phase 120 add cos 500 mul 500 add} store
/calcc {phase 240 add cos 500 mul 500 add} store
```

Field values of 1000 represent red and 0 represent blue. Procs **calca**, **calcb**, and **calcc** can be used to find the appropriate boundary value color for each of the three conductors in your current cell as a function of the desired phase angle.

After you have created all of your cells, combine them in one Acrobat document. Then test them by using the front and back arrows. Then add this crucial detail…

When assembling your cells into a complete sequence, DUPLICATE YOUR FIRST CELL to provide a good place to initialize JavaScript variable definitions and such.

And…

When closing your animation loop, always return to the SECOND cell, not the first one. Remember that internal Acrobat page numbering always starts with page ZERO.

Initializing

JavaScript and PostScript complement themselves in many ways. **PostScript** is a far more powerful and more intuitive general purpose computing language, while JavaScript excels at interactive user control.

Two of the many places where JavaScript actions can be added to an Acrobat document are on **page start** and on **mouse action**. Any page start JavaScript command is entered by going to **Document** and then **Set Page Action**. A mouse action for a field such as a button is entered by clicking on the **Form Tool**, right clicking on the button, then followed by **Properties**.

JavaScript variables **must** be initialized before they can be used. If a variable is to be used on **all** the pages of an Acrobat display, it **must** have a name similar to **global.zorch**. Once you figure out all of the arcane rules, the final JavaScript code can be surprisingly compact and simple. Here is some possible JavaScript initialization code for the first document page of our simulation…

```
global.animate = false ;
global.speed = 4 ;
this.pageNum++ ;
```

Our first line defines a flag that determines whether the animation will stop or run. This is important because…

Continuous animation will hang your program!

ALWAYS provide user exit stop buttons or timouts!

The second line is an optional time delay that can be added to each cell. The integer is in **milliseconds** Normally, you are processor limited and want to run as fast as possible. But additional time delay can provide user interaction or reduce the variations with CPU speed. In other uses, a **app.setTimeOut** variable of 1300 might give a 1.3 second display for a chosen slide show cell. Unlike the original full screen slide shows, you can have fractional second values and do your show from **within** a web browser.

The third line moves you unconditionally to the next cell. Thus, on entry or when the user clicks on the "go to beginning" arrow, needed JavaScript values are initialized, followed by a jump to the start of the actual animated sequence.

Cell Sequencing

Here's some page entry code to move you to the next cell in the sequence...

```
bbb = app.setTimeOut
        ("if (global.animate){this.pageNum++}", global.stall) ;
bbb ;
```

This defines a local **bbb** variable that says "Delay for the **global.stall** time, then check to see if the **global.animate** flag will let us animate. If so, move to the next cell at the end of the stall time."

The very last cell in the sequence is slightly different because it has to return to the beginning of the live cells to be displayed...

```
bbb = app.setTimeOut
        ("if (global.animate){this.pageNum = 1}", global.stall) ;
bbb ;
```

Note that **this.pageNum = 1** goes to the **second** cell in the sequence. Because internal page numbering starts at zero. We thus bypass the initialization cell on repeated cycles.

Note also that Adobe uses an **app.setTimeOut** variable that differs from the normal JavaScript **setTimeout** by its case sensitivity. Capital "O", **not** "o".

User Interaction

Three **step**, **stop**, and **run** field buttons are used to provide a user interface. Here is the JavaScript **step** code for an internal cell...

```
this.pageNum++ ;
```

And for the last cell...

```
this.pageNum = 1 ;
```

The **stop** button JavaScript coding is the same for all cells…

```
global.animate = false ;
```

A pair of codings is needed for our **run** buttons. This for internal cells…

```
global.animate = true ;
if (global.animate){this.pageNum++} ;
```

And this one to close the loop for the next cycle…

```
global.animate = true ;
if (global.animate){this.pageNum = 1} ;
```

Some additional tips: Debugging can be eased with temporary alert boxes, perhaps done as…

```
zzz = app.alert ("hi there",0)
zzz
```

Variables can be inserted into the alert message text string and inspected. A similarly done **yyy = app.beep(4) ;** can also be used to place one of four system beeps at any cell during debugging.

For some strange reason, the right and left arrows will be **much slower** (and a lot more glitchy) at moving between your **Acrobat** pages than the **this.pageNum++** or a **this.pageNum = 1** JavaScript commands. Always check "live" in a browser to make sure your page content will appear fast enough for effective, fast, and totally glitch-free animation.

For More Help

Enhancements and improvements on this fast, convenient, and super flexible .PDF animator can be made available to you on a **Custom Consulting** basis. Additional GuruGrams are found **here**, PostScript topics **here**, and Acrobat info **here**.

Further **GuruGrams** await your ongoing support as a **Synergetics Partner**.