- Animations Using Shadow-Buffers -

Goal:
Now, our static images learn how to move! Within this exercise we will use shadow buffers to implement simple sprite animations. These are then controlled by mouse events.

Exercise 14.1: Multiple Shapes
Create a new C-project in Eclipse (either as Managed-Make project or Makefile project). Copy your shape program from exercise 12.3 into the new project. Within a for-loop, call your **`draw_shape()`** routine multiple times to draw your shape on the screen along a virtual path

Exercise 14.2: But now with Animation and Shadow Buffer
Create a new C-project in Eclipse (either as Managed-Make project or Makefile project). Define a second memory area to be used as the shadow buffer:

```
      sbp= (unsigned char *) malloc(screensize)
```

Now you implement the functions required to handle the shadow buffer, which are **`restore_shape(), restore_pixel_<color_model>()`** as well as **`save_shape()`** and **`save_pixel_<color_model>().`** Extend the **`for`**-loop from exercise 14.1 to appropriately call the restore and save routines. Then compile your code and execute it on host and target. You should now see a correct animation of your shape without the background being destroyed…

Exercise 14.3: Again, now including Background and Alphablending
Create a new C-project in Eclipse (either as Managed-Make project or Makefile project). Extend your program again to render the gray-scale background. Then you modify the draw and put routines to include an additional alpha value as a parameter, which is processed properly inside.

Exercise 14.4: Interaction using Mouse events
Copy the program **`getevent1`** (x86 and ARM) from the Web directory. Test it on a virtual console, i.e. NOT running in X11, both on your host and on your target:

```
      sudo ./getevent1 –l –q
```

You should see the relative(!) mouse coordinates in real-time. Analyze and understand the output of the program.

Add. note: the **`getevent`** utility is much more powerful (see documentation on the Internet). You can get the source code from the net, study it and also (cross-)compile for yourself.

Exercise 14.5: Integration and Setup of the Demonstration
Now you copy the program **`fb_mouse_mt`** (x86 and ARM) from the Web directory. It expects relative mouse coordinates, which are passed from **`getevent1`** into the program via **`stdin`** by means of the Unix **`pipe`** operator.

Now test this setup on host and target. How smooth does the program react to and process the mouse events? How can this be improved to run more “smoothly”?

Hint: Currently, the output of the **`getevent1`** program to **`stdout`** is buffered. This means, the pipe will hand over only rather large chunks of data to the following program **`fb_mouse_mt`** to process. Learn on http://unix.stackexchange.com/questions/25372/turn-off-buffering-in-pipe, how this behavior can be optimized using utilities such as **`stdbuf`** or **`unbuffer`**. What are the costs? Demonstrate smooth mouse interaction on host and target.

Exercise 14.6: Now do it yourself: complete the Quickstart sources
Create a new C-project in Eclipse called fb_**`mouse`** (either as Managed-Make project or Makefile project). Copy the sources **`*.c, *.h`** from the Web directory into your project. Analyze and understand the course structure of the code. Pls. note the two threads, one to process the input, the other to handle the rendering activities. Also note how the data is transferred between these threads.

These quickstart sources contain the code of a program called fb_touch_mt. It has been developed to be used with a touch screen rather than with a mouse. This means, the program expects absolute coordinates rather than relative ones!

Now modify the input thread of your code to work with relative coordinates as provided by **`getevent1`**. Then verify, that these are correctly handed over into the render thread.

The next task is to flesh out the central **`while`**-loop inside the render thread. Finalize the code and execute it on host and target. It should behave the same way as the **`fb_mouse_mt`** program provided.

Finally you want to extend the program with a clone functionality: Each time you press the <**`C`**> key on the keyboard, the square shape gets cloned. This way, you can render as many squares on the screen as you like.