

## Creative Computing II

### Animation

10th November 2009

This lab sheet covers some techniques used in computer animation.

1. The sketch developed in this part simulates a flipbook animation, reproducing the flipbook animation demonstrated in the lecture.
  - (a) Visit the website at <http://www.zuzu.org/printout.html> and download the images on that page.
  - (b) Write a *Processing* function whose argument is an integer  $n$  between 0 and 26 (inclusive) and which returns:
    - "title.gif" if  $n$  is 0;
    - "title2.gif" if  $n$  is 1;
    - the string formed by concatenating "K",  $n - 1$  and ".gif" together otherwise.
  - (c) In your sketch, declare a variable of type `PImage[]` and initialize it to a new 27-element array of `PImages`. In your `setup` routine, set each element of this array to be the result of calling `loadImage` on the name returned by the function defined in part 1b. (For this to work, you will need to place the downloaded images in the `data` folder of your sketch.)
  - (d) Declare a variable in your sketch to store the current frame number. In your `draw` routine, display the image corresponding to that frame number from your `PImage[]` variable, and increment the frame number variable (returning to 0 if necessary).

You should now have a working flipbook animation. Can you think of any ways to improve it?

*All the above parts, put together, should give you a sketch something like the following:*

```
PImage[] frames = new PImage[27];
int frame;

String frameName(int n) {
    switch(n) {
        case 0: return "title.gif";
        case 1: return "title2.gif";
        default: return "K" + (n-1) + ".gif";
    }
}

void setup() {
    frame = 0;
    for(int i = 0; i < 27; i++) {
        frames[i] = loadImage(frameName(i));
    }
}
```

```

    size(frames[0].width, frames[0].height);
    frameRate(9);
}

void draw() {
    image(frames[frame], 0, 0);
    frame = (frame + 1) % 27;
}

```

*This sketch could be improved in several ways: for example, by lingering on the title frames for longer (at present the titles are almost unreadable) or by generalizing it so that it works no matter how many images are in the data folder.*

2. This part constructs an animation based on a simple physical model of a bouncing ball.
  - (a) A single ball, falling vertically, can be described using the equation

$$\ddot{y} = g.$$

Construct a sketch of size  $320 \times 600$  to visualize this behaviour, with the initial conditions  $\dot{y} = 0, y = 100$  for  $g = 3$ , displaying 10 frames per second.

- (b) To model the bouncing, adjust your sketch so that if the  $y$  position goes above 600, it is replaced by  $2 \times 600 - y$  and the velocity  $\dot{y}$  is multiplied by  $-0.9$ . Think about what each of these modifications is intended to achieve in terms of the situation being modelled.
- (c) Allow the simulation to run for a long time, so that the height of the bounces is very small. What do you observe?

How would you extend this sketch to represent a bouncing ball with a horizontal velocity? How would you go about simulating multiple bouncing balls?

*The final sketch should look a bit like the following:*

```

float y = 100;
float ydot = 0;
float g = 3;

void setup() {
    size(320,600);
    frameRate(10);
}

void draw() {
    background(0);
    fill(255);
    ydot = ydot + g;
    y = y + ydot;
    if(y > 600) {
        y = 2*600 - y;
        ydot = -0.9*ydot;
    }
}

```

```
    }  
    ellipse(160,y,10,10);  
}
```

*If you allow the simulation to run for a long time, you should observe that the bouncing becomes a little erratic – sometimes, the ball bounces higher on a subsequent bounce than it did on the previous one. This is an artifact of the numerical simulation of the physical problem; it's clearly unrealistic, but is one of the problems with physical modelling.*

3. This part is about keyframing and linear interpolation. Construct a sketch to meet the following brief:

- At startup, the sketch should do nothing;
- When the user clicks on the sketch window with the mouse, the sketch should begin recording the positions of and time intervals between mouse clicks;
- When the user hits the Return key, the sketch should display an animation of an ellipse moving between the recorded points at the recorded time intervals, interpolating linearly between the positions.

You might find reading the documentation for the *Processing* function `millis()` useful.

*The sketch for this part is available on the course website. You should take the time to study it carefully; it demonstrates a number of useful programming techniques, such as class definition (the `Keyframe` class, holding keyframe information); data structure implementation (storage of an unspecified number of keyframes, doubling the amount of storage reserved when necessary, in the `mousePressed` method); and linear interpolation (in the `draw` method).*

Other resources:

- Lasseter, J. *Principles of traditional animation applied to 3D computer animation*, ACM SIGGRAPH Computer Graphics 21:4 (1987). <http://portal.acm.org/citation.cfm?id=37407>
- Maestri, G. *Digital Character Animation*. New Riders (1996).
- *Animation techniques* Category on Wikipedia. [http://en.wikipedia.org/wiki/Category:Animation\\_techniques](http://en.wikipedia.org/wiki/Category:Animation_techniques)