Keith Schneider, 10 August 2006

Matlab and Psychophysics Toolbox Seminar

Part 4. Animation I

Today we're going to make some animations. This first type of animation we are going to do is simple and interactive. That is, we will be generating each frame of the animation in real time rather than storing them in memory first for playback later. Using double-buffered graphics, there are only two basic steps to making an animation:

1. Draw an object in a new location
2. Flip the buffers (displaying the object and clearing offscreen buffer)
3. Repeat

The trick is of course to figure out where to draw the object each frame. The further away from its old location you draw it, the faster the motion will appear. Also, you will need to compute the trajectory.

Here's a script that draws a moving circle across the screen:

```
[w,rect]=Screen('OpenWindow',0, [0 0 0]);
r = 50; % radius of circle (pixels)
v = 1;  % velocity (pixels per frame)
x = r:v:rect(3)-r;
y = rect(4)/2;
for i=1:length(x)
    % draw circle in new position
    Screen('FillOval', w, ...
        [255 255 255],[x(i)-r,y-r,x(i)+r,y+r]);
    Screen('Flip',w);
end
Screen('Close',w)
```

Try increasing the velocity.

Random motion

Here's a script that does a "random walk". This script will run until you press a key. Notice that I test to make sure that the circle is never drawn off the edge of the screen. This is a boundary condition. (If the circle is moving too slowly you can increase the velocity to verify this).

```
% Random walk animation.  Press any key to exit.
[w,rect]=screen('OpenWindow',0,[0 0 0]);
r=50;   % radius of circle (pixels)
v=10;   % velocity (pixels per frame)
x=rect(3)/2;
y=rect(4)/2;
kdown=0;
while(~kdown)
    % draw circle in new position
    Screen('FillOval',w,[255 255 255],[x-r,y-r,x+r,y+r]);
    Screen('Flip',w);

    % compute new position
    x=x+v*(2*rand-1);
    y=y+v*(2*rand-1);

    % check borders
    x=max(x,r);           % left border
    x=min(x,rect(3)-r);   % right border
    y=max(y,r);           % top border
    y=min(y,rect(4)-r);   % bottom border

    % check whether any keys are depressed
    kdown=KbCheck;
end
Screen('Close',w)
```

Animating in response to user input

Here's a script to test the animation speed and the responsiveness off the mouse. It will draw the circle centered at the current mouse location, exiting when the mouse is clicked.

```
[w,rect]=Screen('OpenWindow',0,[0 0 0]);
r=50;   % radius of circle (pixels)
bdown=0;
while(~any(bdown))
     [x,y,bdown]=GetMouse;
     % check borders
     x=max(x,r);           % left border
     x=min(x,rect(3)-r);   % right border
     y=max(y,r);           % top border
     y=min(y,rect(4)-r);   % bottom border
     % draw circle in new position
     Screen('FillOval',w,[255 255 255],[x-r,y-r,x+r,y+r]);
     Screen('Flip',w);
end
Screen('Close',w)
```

You might want to try this script using `Screen('Flip',w,[],1);` such that the circle isn't erased each time (this may not work on PCs). Or, have the circle drawn only when the mouse button is down (and exit the program some other way). You could also draw the circle in a random color each frame. Note: the buttons on the `GetMouse` function might not work well in the Windows version.

Trajectory

In order to animate an object with a particular trajectory, you need to know the equations of motion for the object. That is, we want to be able to determine how far to move the object each frame in the *x* and *y* directions. Here's a script in which the object moves at a constant velocity, but reflecting off the "walls" of the screen.

```
[w,rect]=Screen('OpenWindow',0,[0 0 0]);
r=50;              % radius of circle (pixels)
v=10;              % velocity (pixels per frame)
xmax=rect(3)-r;    % maximum x position
xmin=r;            % minimum x position
ymax=rect(4)-r;    % maximum y position
ymin=r;            % minimum y position
x=rect(3)/2;       % initial x position
y=rect(4)/2;       % initial y position
th=2*pi*rand;      % initial heading
dx=v*cos(th);      % initial x velocity
dy=v*sin(th);      % initial y velocity
kdown=0;
while(~kdown)
      % draw circle in new position
      Screen('FillOval',w,[255 255 255],[x-r,y-r,x+r,y+r]);
      Screen('Flip',w);

      % reflect off borders if necessary
      if x+dx > xmax | x+dx < xmin
            dx=-dx;
      end
      if y+dy > ymax | y+dy < ymin
            dy=-dy;
      end

      x=x+dx; % compute new position
      y=y+dy;

      % check whether any keys are depressed
      kdown=KbCheck;
end
```

To modify this script to incorporate multiple objects, we have to introduce collisions between objects. I introduce two different types of collisions in the code below. The first is an unrealistic constant speed model (good for multiple object tracking). The second is a realistic physics or "billiards" model. You can switch between the two using the physics flag near the beginning of the code. Note that if two objects become overlapping for some reason (either their initial positions overlapped, or they were involved in a complicated multiple-object collision), I've introduced a fudge that just temporarily disables collision detection between those two objects until they become disjoint again. If you want the objects to pass through each other, you can turn off collisions altogether by commenting out the appropriate part of the code. You might not want to type this whole program in, so you can cut and paste it from here or ask me for a copy.

```
% bounce_n.m
[w,rect]=Screen('OpenWindow',0,[0 0 0]);
r=50;   % radius of circle (pixels)
v=10;   % initial speed (pixels per frame)
n=10;   % number of circles to draw
physics = 1; % set to 1 for realistic physics, 0 for constant speed

xmax=rect(3)-r;     % maximum x boundary
xmin=r;             % minimum x boundary
ymax=rect(4)-r;     % maximum y boundary
ymin=r;             % minimum y boundary
th=2*pi*rand(1,n); % initial headings
dx=v*cos(th);       % initial x velocity
dy=v*sin(th);       % initial y velocity
x=rand(1,n)*(rect(3)-2*r)+r; % initial x positions
y=rand(1,n)*(rect(4)-2*r)+r; % initial y positions

kdown=0;

% main animation loop
while(~kdown)
    % draw circles
    for i=1:n
        Screen('FillOval',w, [255 255 255], ...
            [x(i)-r,y(i)-r,x(i)+r,y(i)+r]);
    end
    Screen('Flip',w);

    % reflect off borders
    rx = x + dx > xmax | x + dx < xmin;
    dx(rx) = -dx(rx);
    ry = y + dy > ymax | y + dy < ymin;
    dy(ry) = -dy(ry);
```

Continued on next page…

```
% bounce.m continued

    % bounce off each other
    dsq = (repmat(x',[1 n])-repmat(x,[n 1])).^2+ ...
        (repmat(y',[1 n])-repmat(y,[n 1])).^2;  % old distance^2
    % fudge: temporarily disable collisions for overlapping objects
    dfudge = dsq < 4*r^2;
    dsq = (repmat((x+dx)',[1 n])-repmat(x+dx,[n 1])).^2+ ...
        (repmat((y+dy)',[1 n])-repmat(y+dy,[n 1])).^2; % new distance^2
    c = find(triu(ones(n)) & dsq < 4*r^2 & ~dfudge);  % colliding pairs
    if any(c)
        [i j]=ind2sub([n n],c);
        switch physics
            case 0 % constant speed model
                dx([i j]) = dx([j i]);
                dy([i j]) = dy([j i]);
            case 1 % realistic physics model
                dnx = x(i) + dx(i) - x(j) - dx(j);
                dny = y(i) + dy(i) - y(j) - dy(j);
                ddx = dx(i) - dx(j);
                ddy = dy(i) - dy(j);
                p = (dnx .* ddx + dny .* ddy) ./ dsq(c)';
                dx(i) = dx(i) - p .* dnx;
                dx(j) = dx(j) + p .* dnx;
                dy(i) = dy(i) - p .* dny;
                dy(j) = dy(j) + p .* dny;
        end
    end

    x=x+dx; % update positions
    y=y+dy;
    kdown=kbcheck; % check whether any keys are depressed
end
Screen('Close',w)
```

Try increasing the number of objects and their size until you notice some drawing artifacts. You might see some improvement using the `Priority` function.

You can also draw small circles (with a radius less than 32, depending on your graphics card) with the `Screen('DrawDots')` function. To do so, replace the three lines (including the `for` and `end` lines) of the `Screen('FillOval')` loop with:

```
Screen('DrawDots', w, [x; y], 2*r, [255 255 255], [], 1);
```

and also insert the following line after the `Screen('OpenWindow')` command:

```
Screen('BlendFunction', w, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

See the help file for `Screen('DrawDots')` (more on this function next week).

Dynamic objects

The object that is drawn does not need to be static. In the original single-object script, or in the multiple object script above, make the following substitutions. Instead of drawing the object with color `[255 255 255]`, use `255*rand(1,3)` or

```
[255 255 255]*(1+sin(2*GetSecs))/2
```

The latter works better with a screen background color of `[128 128 128]`. You can modify the oscillation rate of the circle by changing the factor multiplying `GetSecs`.

Animating images

It is also easy to animate an image (or pattern you create in Matlab) using the `Screen('DrawTexture')` function. Here's a modified version of the mouse moving script that draws an image instead of a circle. If the image size is too large, you might notice some jitter (if it takes more than a single frame to copy the image each time).

```
img=imread('myimage.jpg');
[yimg,ximg,z]=size(img);
[w,rect]=Screen('OpenWindow',0,[0 0 0]);
sx = 400;           % desired x-size of image (pixels)
sy = yimg*sx/ximg; % desired y-size--keep proportional
t = Screen('MakeTexture',w,img);
bdown=0;
HideCursor % makes mouse pointer disappear
while(~any(bdown))
    [x,y,bdown]=GetMouse;

    % check borders
    x=max(x,sx/2);          % left border
    x=min(x,rect(3)-sx/2); % right border
    y=max(y,sy/2);          % top border
    y=min(y,rect(4)-sy/2); % bottom border

    destrect=[x-sx/2,y-sy/2,x+sx/2,y+sy/2];
    % draw image in new position
    Screen('DrawTexture',w,t,[],destrect);
    Screen('Flip',w);
end
Screen('Close',w)
ShowCursor
```