

Coding experiments with Psychtoolbox

Part 1: Nov 14th, 2017

Alexandra Vlassova

alexandravlassova.com to download sample scripts & slides

Outline

- Help
- Screen
- Drawing
- Images
- Motion
- Sound
- Timing
- Response collection
- Trial setups
- Saving data
- Exporting data
- Coding full experiments
- Testing
- Tricks

Stop script & close window

- **stop script & close screen:**
 - **MAC:** command + period; command + 0; type “sca”
 - **PC/linux:** ctr+alt+del -> terminate or alt+tab to PTB window; ctrl+c; type “sca”

Help

- `help functionname` **for help on high-level functions**
(e.g. `help Psychtoolbox`, `help Screen`)
- `Functionname subfunctionname?` **for help on low-level functions or sub-functions** (e.g. `Screen FillRect?`)

Help output

```
>> Screen FillRect?
```

Usage:

```
Screen('FillRect', windowPtr [,color] [,rect] )
```

Fill "rect". "color" is the clut index (scalar or [r g b] triplet or [r g b a] quadruple) that you want to poke into each pixel; default produces white with the standard CLUT for this window's pixelSize. Default "rect" is entire window, so you can use this function to clear the window. Please note that clearing the entire window will set the background color of the window to the clear color, ie., future Screen('Flip') commands will clear to the new background clear color specified in Screen('FillRect').

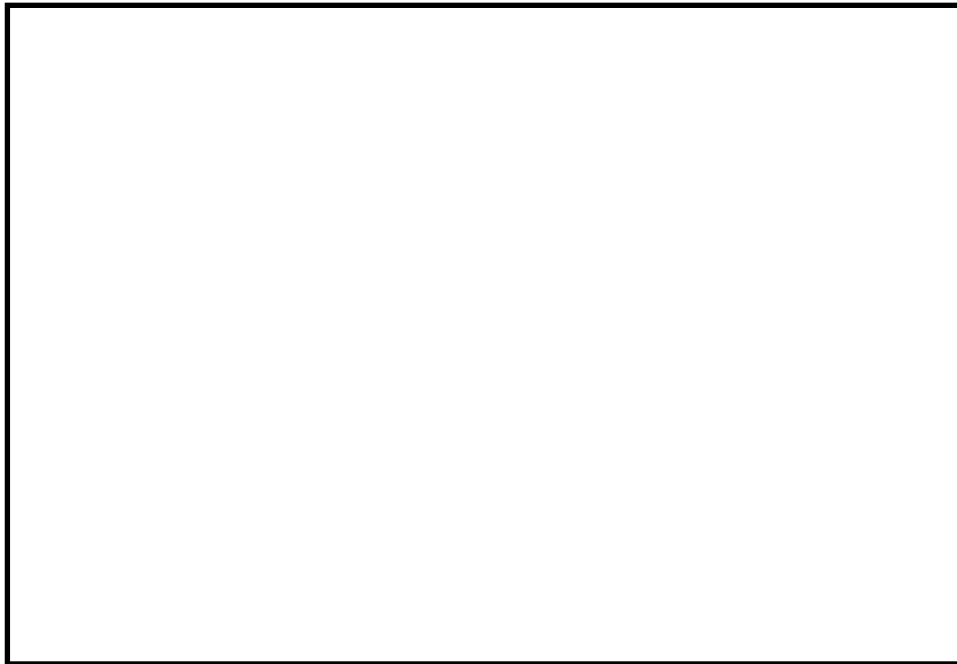
Instead of filling one rectangle, you can also specify a list of multiple rectangles to be filled – this is much faster when you need to draw many rectangles per frame. To fill n rectangles, provide "rect" as a 4 rows by n columns matrix, each column specifying one rectangle, e.g., rect(1,5)=left border of 5th rectange, rect(2,5)=top border of 5th rectangle, rect(3,5)=right border of 5th rectangle, rect(4,5)=bottom border of 5th rectangle. If the rectangles should have different colors, then provide "color" as a 3 or 4 row by n column matrix, the i'th column specifying the color of the i'th rectangle.

See also: FrameRect

Screen()

- **controls graphics and display hardware**
- **performs 2D drawing operations**
- **controls timing**
- **type “Screen” in command window to see a list of all the functions**

Screen



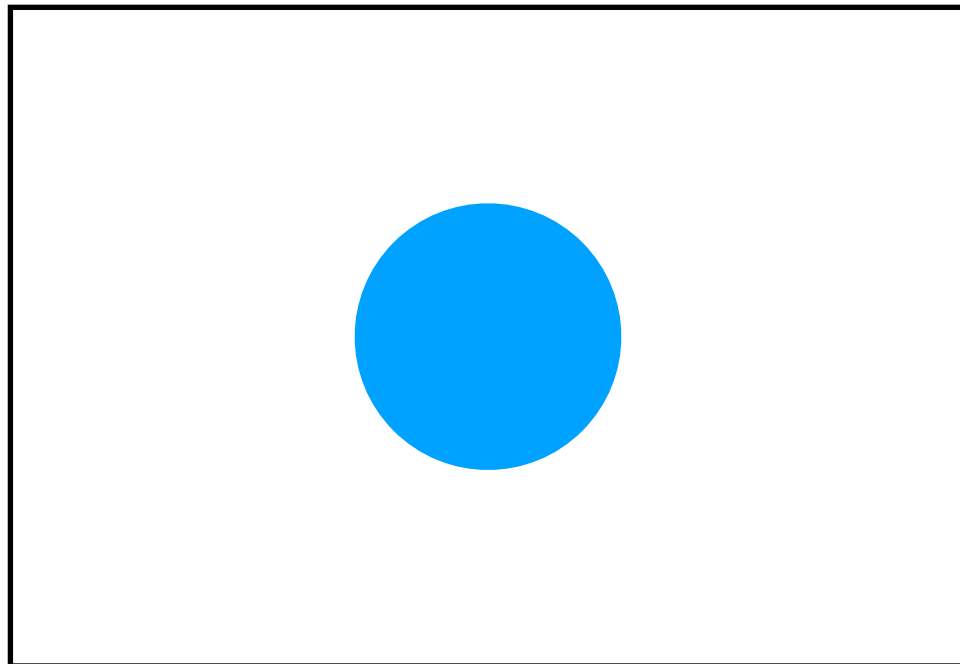
Invisible back-buffer (VRAM)



User-visible front-buffer (display)

Screen

First we draw to the back-buffer



Invisible back-buffer (VRAM)

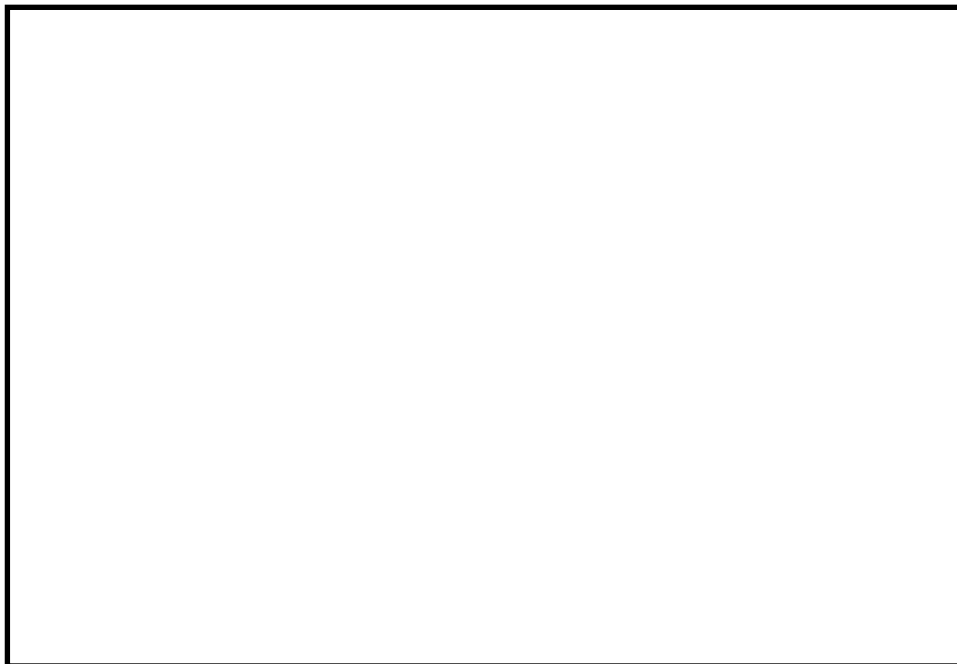


User-visible front-buffer (display)

Screen

Then we call the 'flip' command to flip the back buffer to the front buffer
the back buffer is automatically cleared

```
Screen( 'Flip', frontbuffer)
```



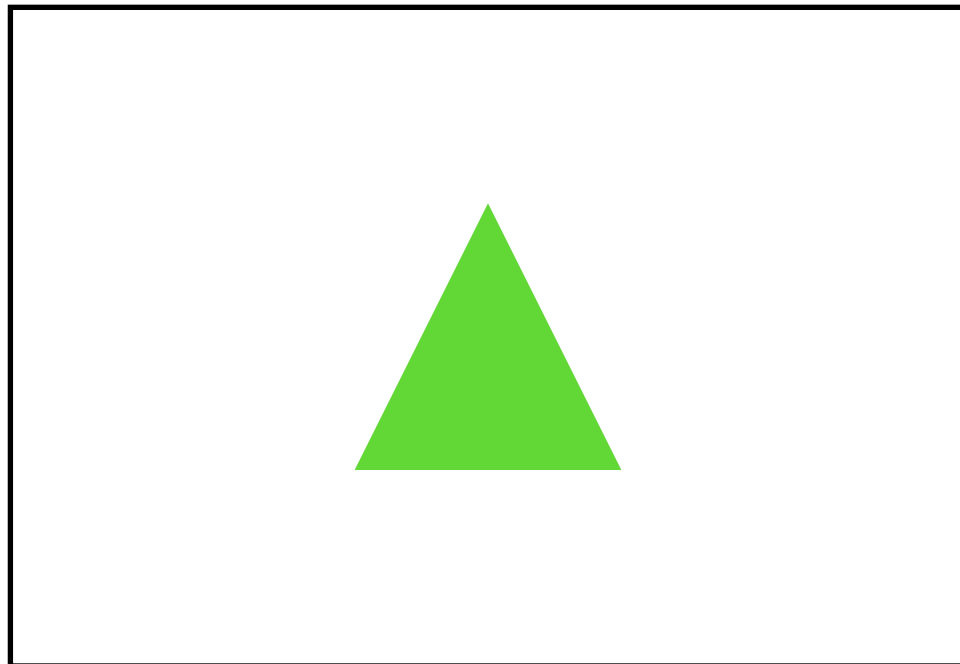
Invisible back-buffer (VRAM)



User-visible front-buffer (display)

Screen

We can then draw other things to the back-buffer



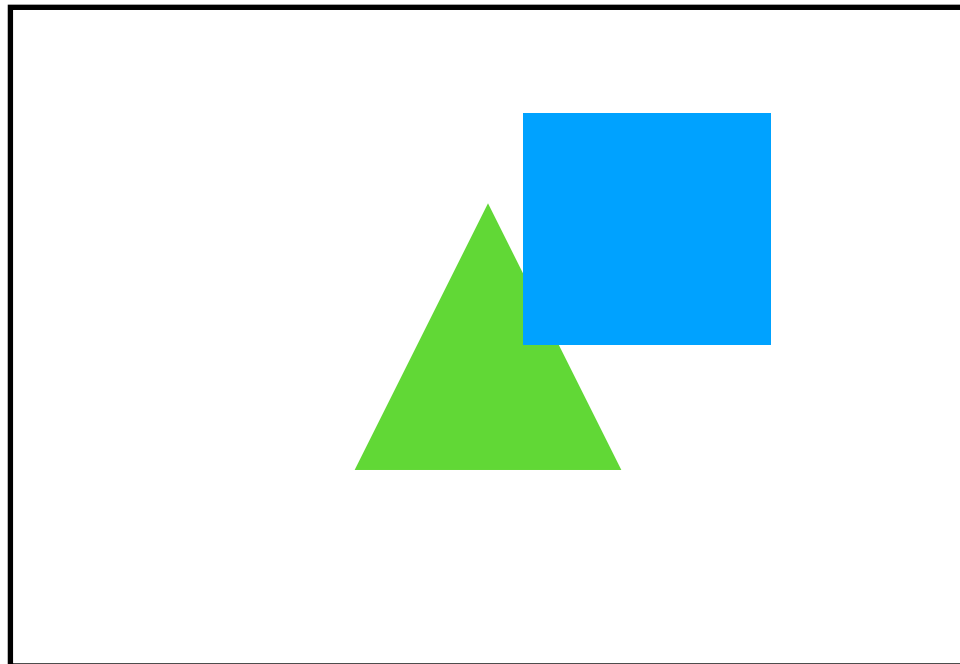
Invisible back-buffer (VRAM)



User-visible front-buffer (display)

Screen

**The front-buffer won't be updated until we 'flip' the screens again.
We can draw more things...**



Invisible back-buffer (VRAM)

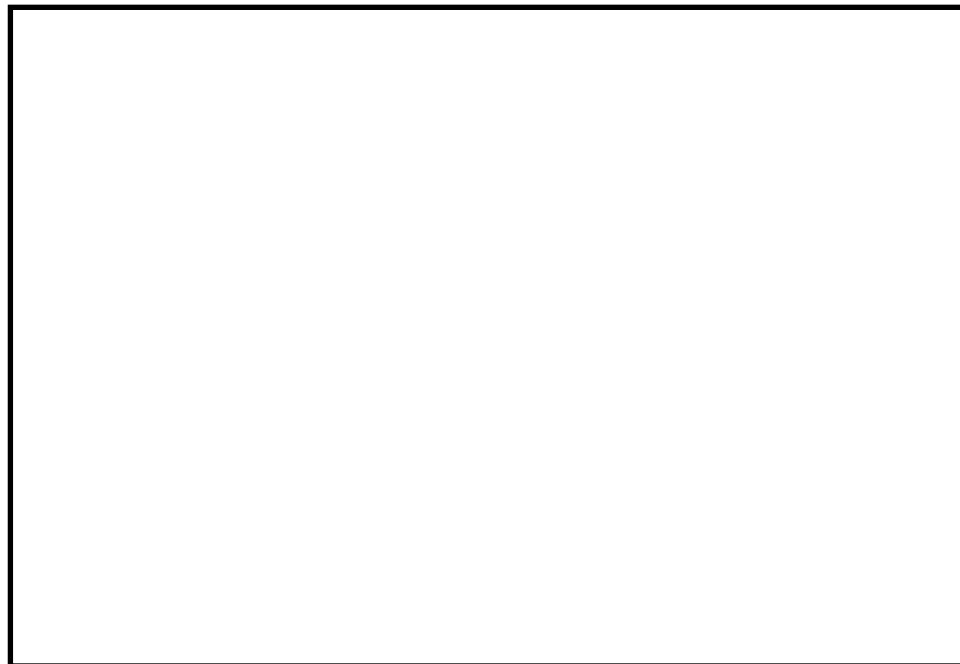


User-visible front-buffer (display)

Screen

Then it will all be flipped once we call the flip command again, wiping the back buffer clear once again.

```
Screen( 'Flip', frontbuffer)
```

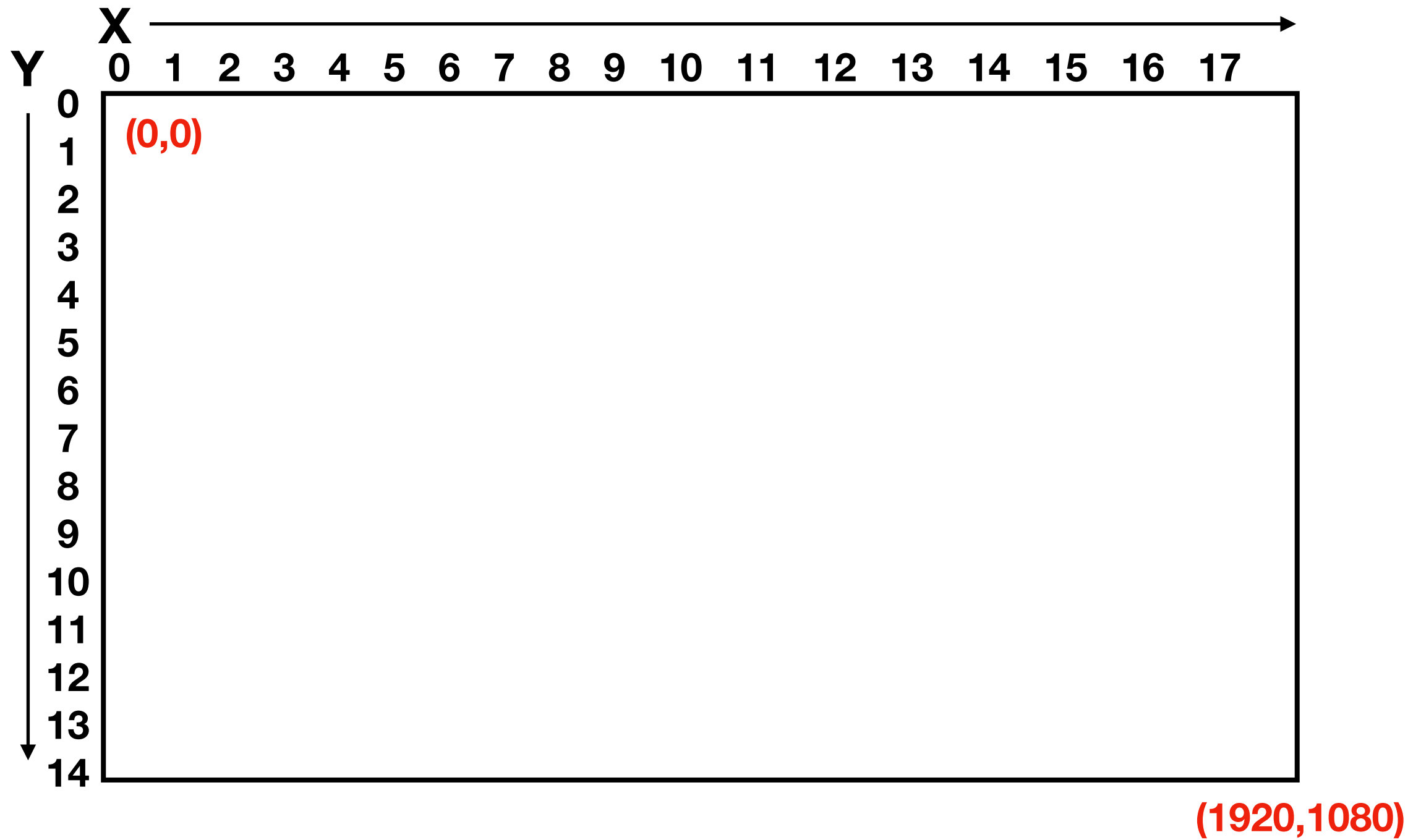


Invisible back-buffer (VRAM)

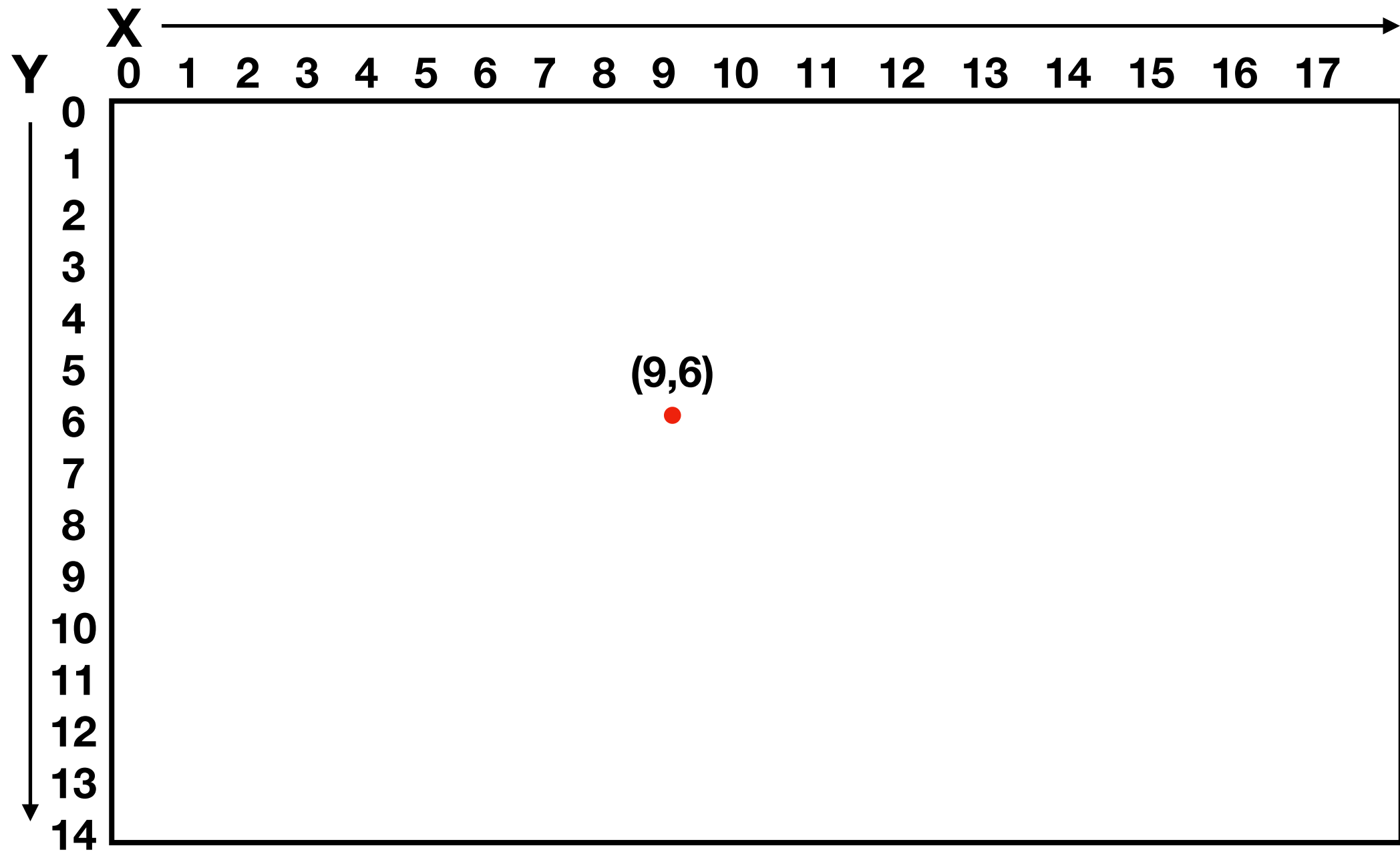


User-visible front-buffer (display)

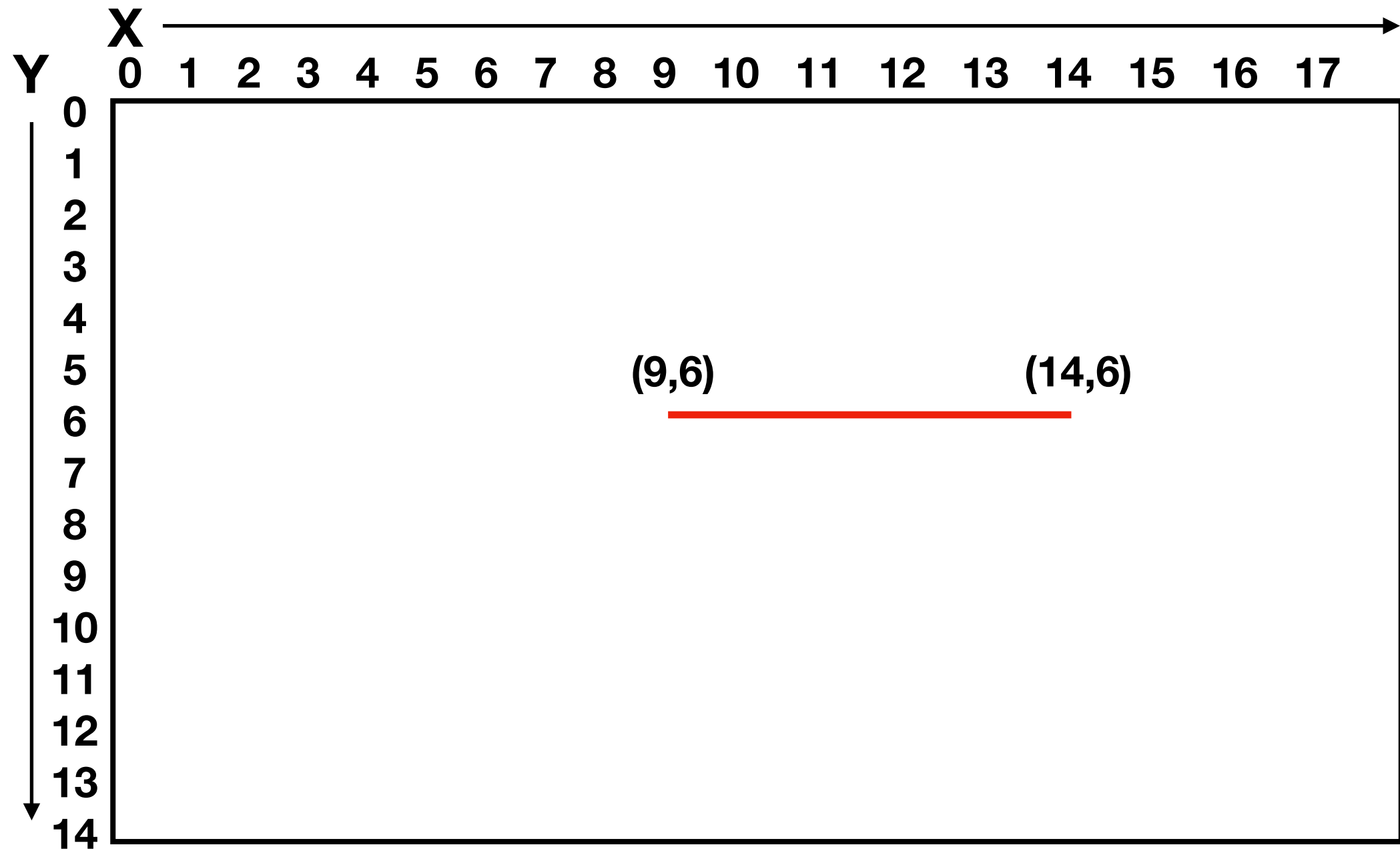
Screen



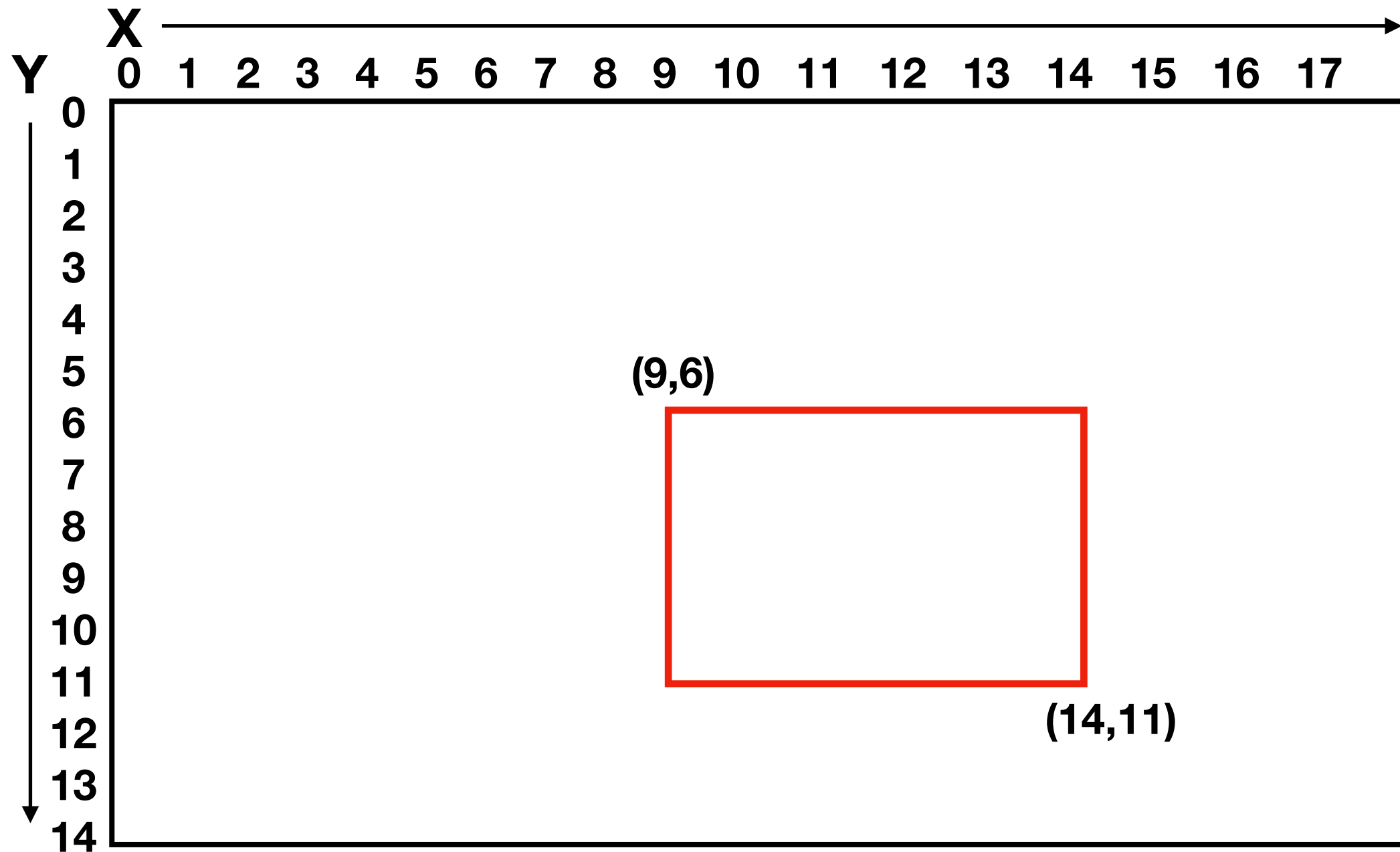
Screen



Screen

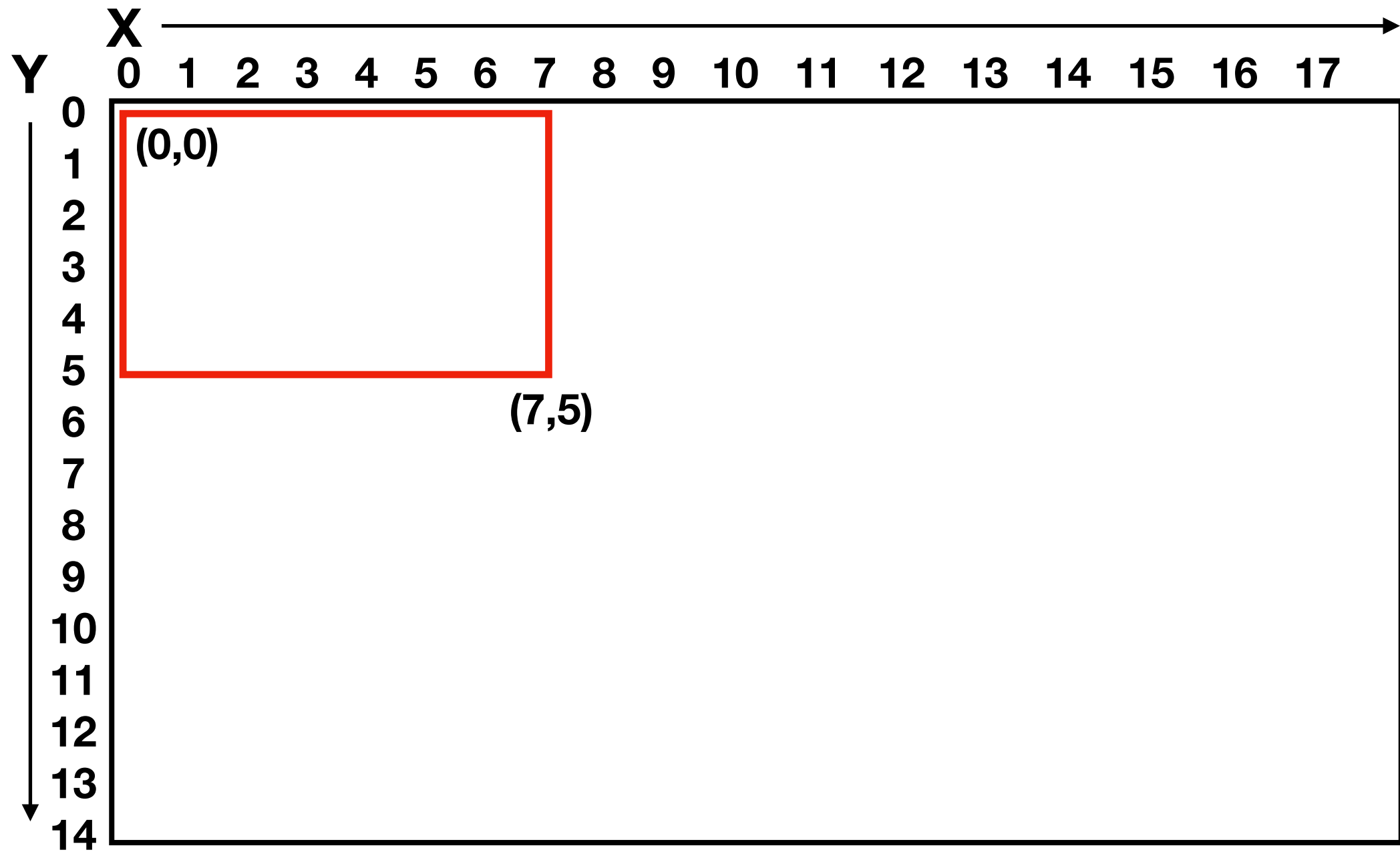


Screen



**always move from top-left-X
to top-Left-Y to bottom-Right-X to bottom-right-Y.**

Screen



Monitors



**CRTs draw one horizontal line at a time.
The current position/line is called the
beamposition.**

**Once it finishes drawing a frame, there is a
short vertical blank interval (VBL) while it
sweeps back to the first line to start
drawing the next frame.**

**Psychtoolbox tries to perform the flips
during this VBL. If synchronisation to the
VBL fails, then timing may be imprecise and
there may be some visual artifacts.**

Monitors

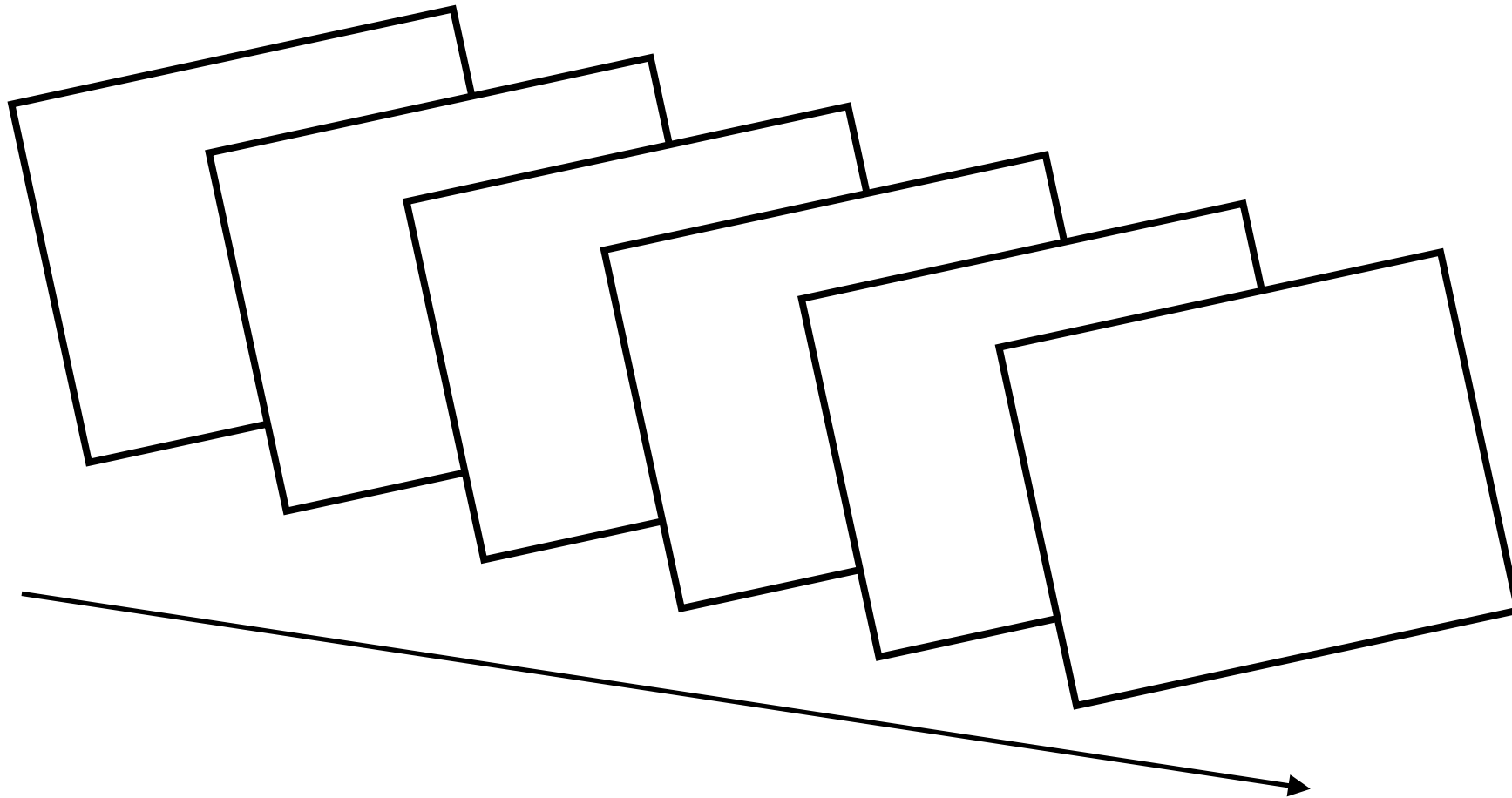


For backwards compatibility, LCD monitors also implement a VBL and report beampositions even though they don't need them.

Timing precision is best on CRTs, but there are some good LCDs available now that do the trick.

Frame Rates

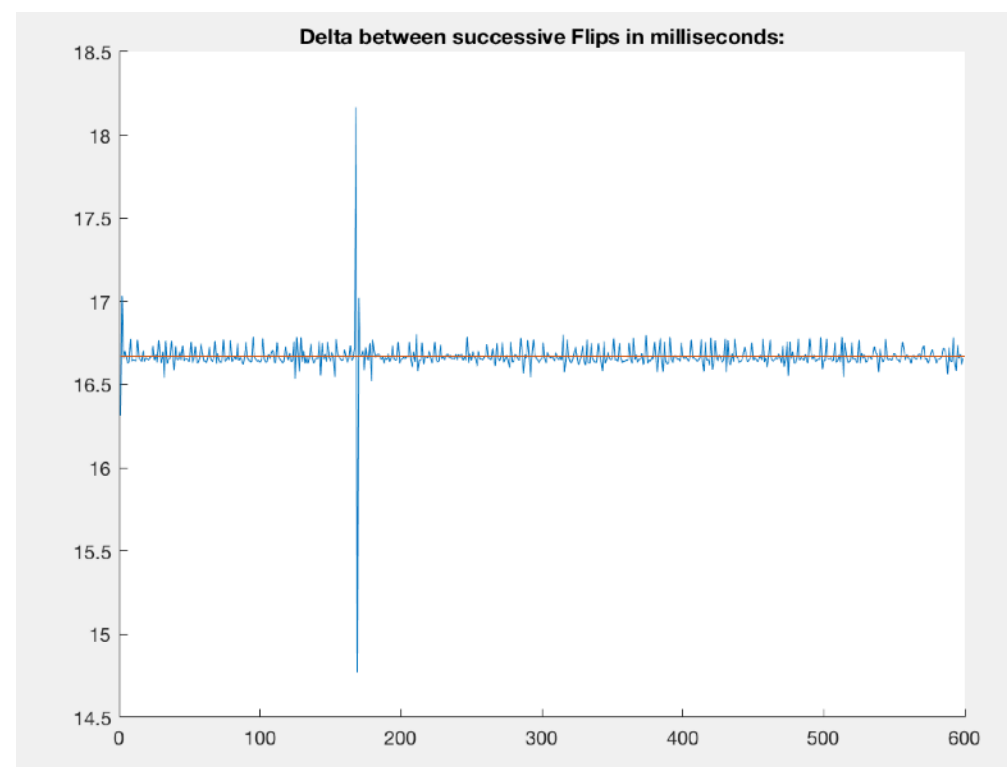
the number of frames drawn per second



If we can draw 60 frames per second, then our frame rate is 60Hz and it will take 16.67 ms to draw each frame - cannot present something shorter than this!

Syncing

- `ScreenTest()` to test hardware/software configuration
- `VBLSyncTest()` to test syncing of PTB to the VBL
- `PerceptualVBLSyncTest()` to test syncing of Screen('Flip') to the VBL



```
PTB missed 0 out of 600 stimulus presentation deadlines.  
One missed deadline is ok and an artifact of the measurement.  
PTB completed 0 stimulus presentations before the requested target time.  
Have a look at the plots for more details...
```

Syncing

- **If using a mac, install** `PsychtoolboxKernelDriver`
- **If using Linux, read** `help SyncTrouble`
- **Don't use Windows if you care about timing**

Syncing

- **For testing/debugging, you can disable the sync tests with `Screen('Preference', 'SkipSyncTests', 1)` but make sure you don't leave this in when you begin testing!**

try/catch

try

% bulk of the code will go here

catch

sca; % closes the screens

ShowCursor; % shows the mouse cursor

psychrethrow(psychlasterror); %prints error

end

Front matter

First we need to set some preferences

```
PsychDefaultSetup(1); % Executes the AssertOpenGL  
command & KbName('UnifyKeyNames')  
  
Screen('Preference', 'SkipSyncTests', 1); % DO NOT  
KEEP THIS IN EXPERIMENTAL SCRIPTS!  
  
Priority(MaxPriority(w)); % Set PTB to have processing  
priority over other system and app processes
```

Front matter

Then we need to chose a display screen

```
getScreens = Screen('Screens'); % Gets the screen numbers
% OSX 0 = primary, 1 = external
% Windows 1 = primary, 2 = external

chosenScreen = max(getScreens); % Chose which screen to
display on

% Get luminance values
white = WhiteIndex(chosenScreen); % 255
black = BlackIndex(chosenScreen); % 0
grey  = white/2;
```

Front matter

Now we will open a PTB window

```
[w, scr_rect] = PsychImaging('OpenWindow',chosenScreen,[0 0 0],[]);  
% Opens a double-buffered full-screen window  
  
% chosenScreen tells it where to open the window (main  
monitor or external)  
  
% the colour of the screen is given by a RGB vector or we can  
pass the "grey" "black" or "white" we previously defined  
  
% the optional argument [] tells it that we want the default  
(fullscreen) setting. Optional arguments are specified in  
order but can be omitted with [] or left out if there are no  
other options after it that you want to specify.  
  
% "w" is the name we have given to the window - we will  
direct our drawing commands to that window  
  
% scr_rect is a rectangle the size of the screen
```

Front matter

Lets get the coordinates for the centre of the screen

```
[centerX, centerY] = RectCenter(scr_rect);  
% get the coordinates of the center of the  
screen using the coordinates for the size of  
the full screen
```

Front matter

Finally, lets check the refresh rate of the monitor

```
ifi = Screen('GetFlipInterval', w); % the inter-  
frame interval (minimum time between two frames)  
  
hertz = FrameRate(w); % check the refresh rate of  
the screen
```

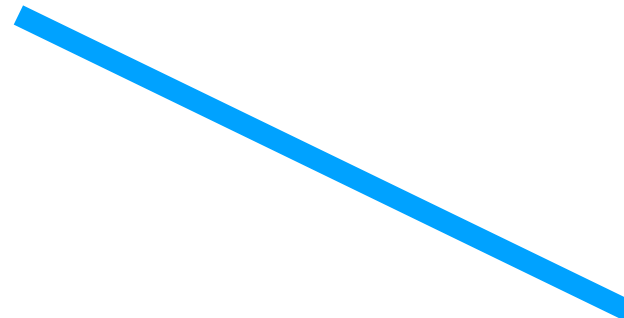
draw a line

the name of our window

[**R** **G** **B**] vector (each
ranging (0-255))

```
Screen( 'DrawLine', w, colour, TopX, TopY,  
BottomX, BottomY, thickness );
```

(TopX, TopY)



(Bottom X, BottomY)

draw a rectangle

```
Screen( 'FillRect', w, colour, coords );
```



where to draw it to?

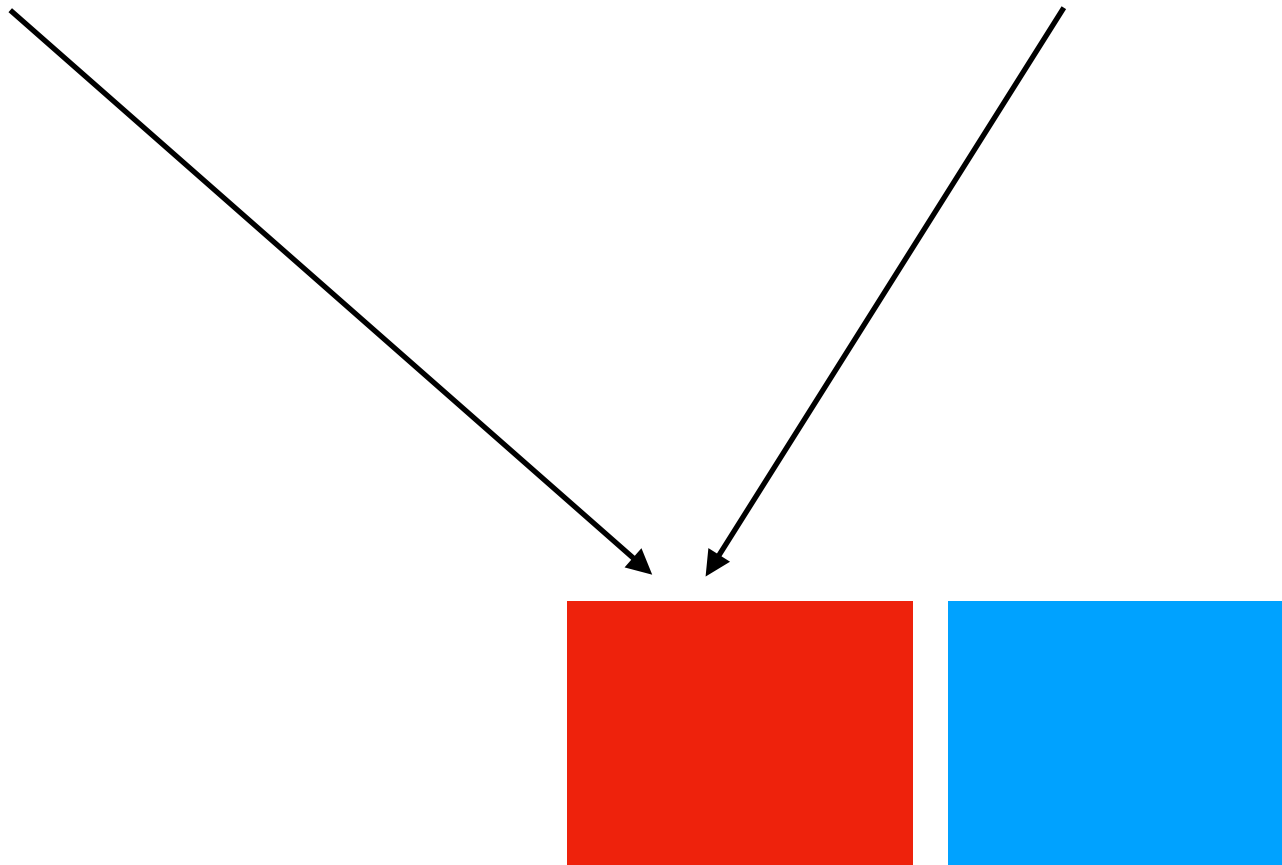
[TopLeftX TopLeftY

BottomRightX BottomRightY]



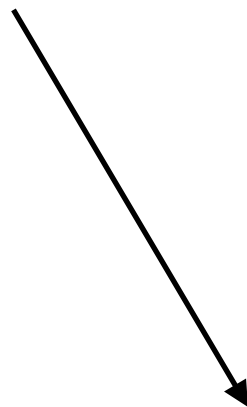
draw two rectangles

```
Screen( 'FillRect', w,  
[colour; colour], [coords; coords] );
```



draw a rectangle outline

```
Screen( 'FrameRect', w, colour, coords,  
thickness );
```

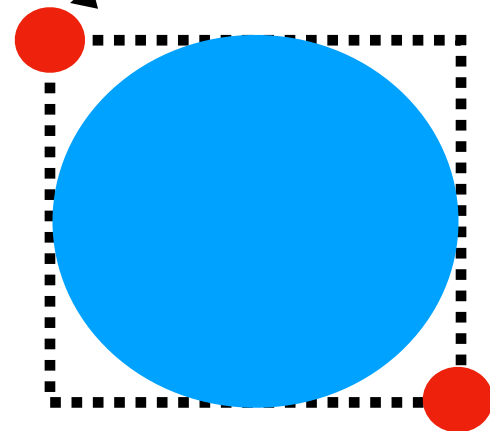


**how thick to we want the
outline to be (in pixels)?**



draw a coloured circle

```
Screen( 'FillOval', w, colour, coords );
```



draw multiple shapes

matrix of xy coords for dot
centers, row 1 = x, row 2 = y,
columns are each dot

can be all the same
size, or give a vector
with the size for each
dot

```
Screen( 'DrawDots', w, xy[, size][, colour]  
[, center][, type]);
```

0=squares, 1=circles,
2= HQ circles

as with size, can be a
single RGB vector, or
a matrix specifying
the colours for each
dot

draw text

%first, what kind of text do we want?

```
Screen( 'TextSize', w, 30 );
```

```
Screen( 'TextFont', w, 'Times' );
```

```
Screen( 'TextStyle', w, type );
```



e.g. 0 = regular, 1 = bold, 2 = italic


type Screen TextStyle? to see the full list

draw text

```
Screen( 'DrawText', w, 'text', x, y, colour );
```



string containing the text
you want to display



where to start drawing the text (might be useful
to know the size of the string in pixels, you can use
`sizeText = Screen('TextBounds', w.text);`

We can also use “DrawFormattedText” which gives us more options

display images

```
imageData = imread( 'image.jpg' );  
imageTex = Screen( 'MakeTexture', w, imageData );  
Screen( 'DrawTexture', w, imageTex );
```

remember that so far all our screen commands have been to the back-buffer - nothing will display on the actual monitor until we call

```
a Screen( 'Flip' , w );
```

timing basics

optionally, we can specify
when we want to flip

```
vb1 = Screen( 'Flip' , w [ ,when ] );
```

timestamp for when the
flip is executed

which screen do we
want to flip to?

timing basics

```
WaitSecs(x); % will pause for x seconds  
before continuing
```

```
time = GetSecs(); % this will give us the  
current time
```

```
tic; %start timer
```

```
toc; %how much time has passed since we  
started timer (in seconds)?
```

lets go through some examples

**you can download the code through links provided on my website
(alexandravlassova.com)**