Diploma in Computer Studies Sep2021

Welcome to Creative Computing

DCR2284

Learning Objectives

- At the end of the course, students will be able to:
  - CO1: Describe the creative concepts in mathematics and computing.
  - CO2: Explain the importance origins of geometry to develop motion, images and sound.
  - CO3: Build the Processing application to construct shapes and objects.
  - CO4: Write the coordinate transformations for motions using Processing..

Images

PImage

- We already know the basics of using images in Processing
- We know that we can load them into a variable (either from a file in our `data` folder or a URL)
- And we know that we can display them on the screen like a fancy rectangle

PImage

- We already know the basics of using images in Processing
- We know that we can load them into a variable (either from a file in our `data` folder or a URL)
- And we know that we can display them on the screen like a fancy rectangle

```
PImage catImage;

void setup() {
  size(640,480);
  catImage = loadImage("https://iheartcats.com/wp-content/uploads/2016/08/28858986841_
}

void draw() {
  image(catImage,0,0);
}
```

imageMode()

- Images have `imageMode()` for changing how they're drawn
- `imageMode(CENTER);` means the image will be drawn from its center
- `imageMode(CORNER);` means the image will be drawn from its top-left corner

Imagine: You are the Event Photographer!

```
float theta = 0;
float scaleFactor = 1;
float scaleDirection = 1;
PImage catImage;

void setup() {
  size(640, 480, P3D);
  catImage = loadImage("https://iheartcats.com/wp-content/uploads/2016/08/28858986841_e5
}

void draw() {
  translate(width/2, height/2);
  rotateZ(theta);
  rotateY(theta);
  scale(scaleFactor);
  imageMode(CENTER);
  image(catImage, 0, 0);
  theta += 0.1;
  scaleFactor += (0.01 * scaleDirection);
  if (scaleFactor < 0 || scaleFactor > 2) {
    scaleDirection = -scaleDirection;
  }
}
```

tint()

- There's even a sort of "`fill()`" for images called `tint()`

```
PImage catImage;
color tintColor = color(random(255),random(255),random(255));

void setup() {
  size(640, 480, P3D);
  catImage = loadImage("https://iheartcats.com/wp-content/uploads/2016/08/28858986841_e5
}

void draw() {
  tint(tintColor);
  image(catImage, 0, 0);
}

void mouseClicked() {
  tintColor = color(random(255),random(255),random(255));
}
```

- So far this is not necessarily super mind-blowing?
- Images are pretty much just a picture pasted as a rectangle on the screen
- There's nothing all that dynamic we're doing with them yet
- Beyond messing with the rectangle itself
- So...

Images are BUNCH of Pixels!

Images are just a *grid* of pixels

- Every pixel in an image has an `x` and `y` coordinate in that image, and a `color`
- In fact a single pixel can just be thought of as a `color` at a specific `x,y` position
- The top left pixel is at `0,0`, the next one to the right is at `1,0` and so on
- We already know about that basic idea from drawing shapes in our window
- Because a *window* is just a grid of pixels

`PImage`s are just an *array* of pixels

- A `PImage` in Processing is represented with an array
- The array contains a colour for each pixel in the image
- The weird thing about the array is that it is *one dimensional*
- Just one long sequences of boxes, even though an image is 2D

`.pixels[]`

- We can get access to a `PImage`'s pixels by asking for its `pixels` array
- This `pixels` array is *built into* *`PImage`* so it's there automatically

.pixels[]

- We can get access to a `PImage`'s pixels by asking for its `pixels` array
- This `pixels` array is *built into* *`PImage`* so it's there automatically

```
PImage myImage;

void setup() {
  size(319,319);
  myImage = loadImage("https://www.beatsbydre.com/content/dam/beats/content-blocks/pdp/c
}

void draw() {
  image(myImage,0,0);
  // Get pixel 100
  color pixelOneHundred = myImage.pixels[100];
  // Print the value of pixel 100
  println("Color as integer:" + pixelOneHundred);
```

Changing `pixels`

- More importantly, we can *change* the values of pixels to manipulate what an image looks like!
- But to do that we need to *tell* the image we're going to do that
- So first we use `myImage.loadPixels();` which means "I'm going to change your pixels soon"
- Then we change the values in the pixels array to change the image
- Then we use `myImage.updatePixels();` to actually make our changes take effect

Just one tiny little pixel...

```
PImage myImage;

void setup() {
  size(632,475);
  myImage = loadImage("http://buildingontheword.org/wp-content/uploads/2016/08/cat.jpg")
}

void draw() {
  myImage.loadPixels(); // Get ready!
  myImage.pixels[0] = color(255,0,0); // Edit!
  myImage.updatePixels(); // Change!
  image(myImage,0,0);
}
```

Every damn pixel!

- Let's randomly change pixels in the image...

Class Activity

- How to change all the pixels (Pixelate!)

Class Activity

- How to change all the pixels (Pixelate!)

- You can get some kind of amazing effects by messing around...

The Window's `pixels`

- Our actual program is basically just a series of images displayed in the window
- And the window actually has its own `pixels` array representing all of its pixels
- Which we can also get access to and change at will
- It's just called `pixels[]` because it's the main pixels array
- So we also just call `loadPixels()` and `updatePixels()` too, no image name needed

```
void setup() {
  size(500, 500);
}

void draw() {
  loadPixels();
  for (int i = 0; i < pixels.length; i++) {
    pixels[i] = color(random(0, 255));
  }
  updatePixels();

  //for (int x = 0; x < width; x++) {
  //  for (int y = 0; y < height; y++) {
  //    stroke(random(255));
  //    point(x,y);
  //  }
  //}
}
```

- Notice how much more efficient that is than using `point()` or `rect()`?

Problem: We usually think of an image as 2D

- Currently we're dealing with this one-dimensional `pixels` array by just doing "something" to every pixel
- But if we wanted to talk about specific coordinates in the image we're currently in trouble
- We only have a 1D reference point into the image, so we need a trick...

Problem: We usually think of an image as 2D

- Currently we're dealing with this one-dimensional `pixels` array by just doing "something" to every pixel
- But if we wanted to talk about specific coordinates in the image we're currently in trouble
- We only have a 1D reference point into the image, so we need a trick...

```
int loc = x + y * width;
```

Problem: We usually think of an image as 2D

- Currently we're dealing with this one-dimensional `pixels` array by just doing "something" to every pixel
- But if we wanted to talk about specific coordinates in the image we're currently in trouble
- We only have a 1D reference point into the image, so we need a trick...

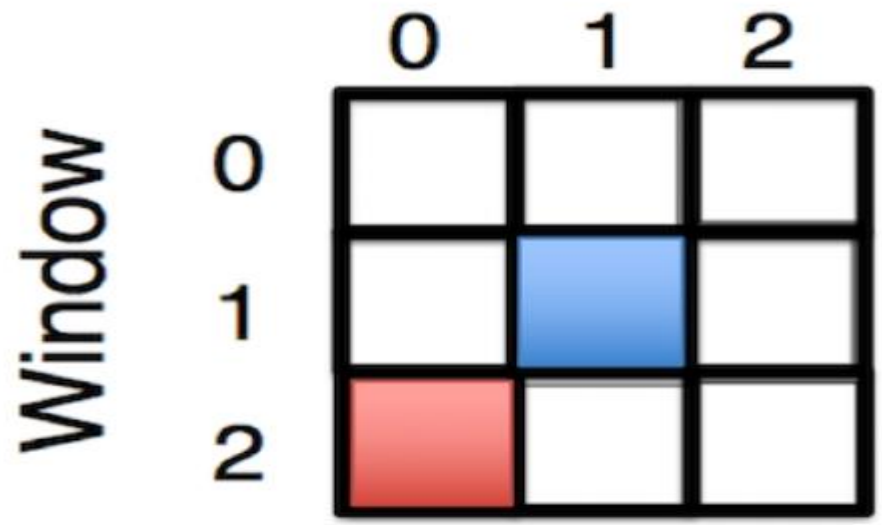
```
int loc = x + y * width;
```

- This calculates a location in the 1D array of pixels based on an `x` and `y` coordinate along with the `width` of the image (or window)

Problem: We usually think of an image as 2D

- Currently we're dealing with this one-dimensional `pixels` array by just doing "something" to every pixel
- But if we wanted to talk about specific coordinates in the image we're currently in trouble
- We only have a 1D reference point into the image, so we need a trick...

```
int loc = x + y * width;
```




```
x = 1; y = 1;
loc = x + y*width;
```

loc = 1 + 1*3 = 4.

```
x = 0; y = 2;
loc = x + y*width;
```

loc = 0 + 2*3 = 6;

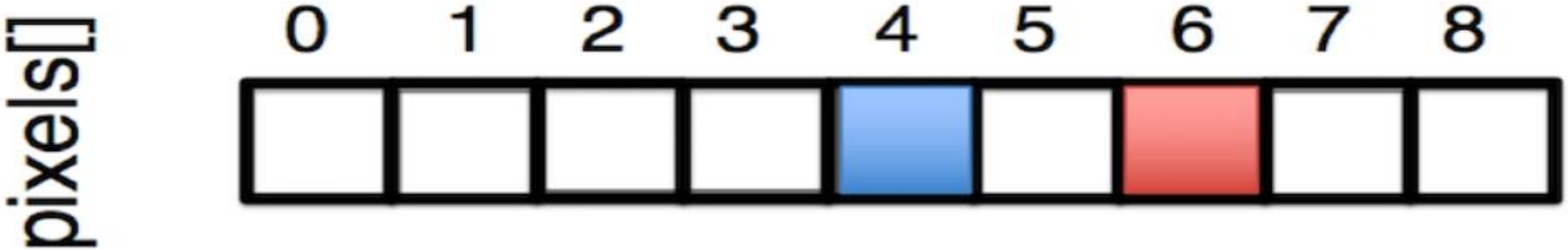


Image Processing

- Now that we know how to convert from coordinates to a specific pixel in the `pixels` array we can:

1. Load an image
2. Load its pixels with `loadPixels()`
3. Go through every pixel based on its `x` and `y` coordinates
4. Either *change it* in an interesting way or *use its information* in an interesting way
5. Update the pixels with `updatePixels()` (if we changed them)

An image processing loop

```
myImage.loadPixels();
loadPixels();
for (int x = 0; x < myImage.width; x++) {
  for (int y = 0; y < myImage.height; y++) {
     int loc = x + y * myImage.width;
    // Do something cool with myImage.pixels[loc]
    // Either change it or use the information it represents
    // (A colour, obviously, but colours mean things!)
  }
}
updatePixels(); // If we changed the pixels array we need to update it
```

Interactive pixels...

```
int loc = x + y*width;
float r = red(myImage.pixels[loc]);
float g = green(myImage.pixels[loc]);
float b = blue(myImage.pixels[loc]);

if (mouseX < width/2) {
  r++;
} else {
  b++;
}
if (mouseY < height/2) {
  g++;
} else {
  r--;
}
myImage.pixels[loc] = color(r, g, b);
```

- `red()`, `green()` and `blue()` give you individual RGB values
- Notice how the image *loses information* over time because we're changing it!

`createImage()`

- Sometimes you might not want to corrupt the original image
- Luckily you can create a *new* image and use *that* to display your whacky tranformations
- We use `createImage(width,height,RGB)` to create a new, blank image that uses RGB information
- So...

```
PImage source;
PImage dest;

void setup() {
  source = loadImage("myNiceImage.png");
  dest = createImage(source.width,source.height,RGB);
}
```

- Now we can *look at* the `pixels` in `source` but *change* the pixels in `dest` based on them
- Note that we have to `loadPixels()` and `updatePixels()` for `dest` if we're going to change them

Class Activity: Image Processing Loop

```
PImage source;
PImage dest;

void setup() {
  size(632, 475);
  source = loadImage("http://buildingontheword.org/wp-content/uploads/2016/08/cat.jpg");
  dest = createImage(source.width,source.height,RGB);
}

void draw() {
  dest.loadPixels();
  for (int x = 0; x < source.width; x++) {
    for (int y = 0; y < source.height; y++) {

      int loc = x + y*width;

      if (loc < dest.pixels.length - 1) {
        dest.pixels[loc] = source.pixels[loc] + source.pixels[loc + 1];
      }
    }
  }
  dest.updatePixels(); // If we changed the pixels array we need to update it
  image(dest,0,0);
}
```

Adjacent pixels tend to be related

- Unless you have a completely random image, adjacent pixels are related
- Because images are usually of some kind of space
- And nearby pixels are therefore representing things that are nearby (or at least related) in space
- We can tune our processing of images toward that idea of relationships between pixels

Edge detection

```
for ( int x = 1; x < source.width; x++ ) {
  for ( int y = 0; y < source.height; y++ ) {
    float threshold = 10;
    int loc = x + y * source.width;
    color pixel = source.pixels[loc];
    int leftLoc = (x - 1) + y * source.width;
    color leftPixel = source.pixels[leftLoc];
    float diff = abs(brightness(pixel) -  brightness(leftPixel));
    if ( diff > threshold ) {
      dest.pixels[loc] = color(255);
    } else {
      dest.pixels[loc] = color(0);
    }
  }
}
```

- `brightness(color)` gives a brightness value for that colour!
- `abs(number)` gives you the *absolute value* of that number
- Why am I starting `x` at `1` instead of `0`?

Libraries

- In programming, it would be a huge pain if we always had to write everything from the ground up
- There are lots of tasks that *lots* of people want to be able to do
- So it would be great if they were solved *once* and then *shared*

Libraries

- In programming, it would be a huge pain if we always had to write everything from the ground up
- There are lots of tasks that *lots* of people want to be able to do
- So it would be great if they were solved *once* and then *shared*
- That's what *libraries* are
- A library is a set of code that provides you with code so you don't have to write it
- Like a game engine, or an artificial intelligence algorithm, or 2D physics
- Or using the webcam!

The video library

- There is a library available for Processing that allows us to work with video and the webcam
- It's called "video", which is pretty sensible
- A library is just a set of files that define the code that library provides for us
- So the video library provides a bunch of code for dealing with video and webcams
- Because who wants to write that themselves, after all?
- Nobody!

The video library

- There is a library available for Processing that allows us to work with video and the webcam
- It's called "video", which is pretty sensible
- A library is just a set of files that define the code that library provides for us
- So the video library provides a bunch of code for dealing with video and webcams
- Because who wants to write that themselves, after all?
- Nobody! Except the person who wrote the video library!

Installing the video library

- In Processing if we go to `Sketch > Import Library...` and we can see Video listed, then we know the library is already installed
- If *not*, we go to `Sketch > Import Library... > Add Library...` to bring up Processing's interface for downloading libraries
- From there we can scroll or search for "Video" and click `Install`

Using libraries in general

1. *Install* it as we did with the video library
2. Find its *documentation* online either with a search like "processing video library" *or* by going to the Processing Reference's Libraries section and looking for it there
3. *Read the introduction* to the documentation so you understand what the library is for
4. Click on a *method or object* you're interested in and read about that
5. Copy some *example code* and run it in Processing

`importing` the video library

- To tell Processing we want to use a library, we have to `import` it at the top of our program
- In the case of the video library, we need to write

```
import processing.video.*;
```

- *Or* we can go up to `Sketch > Import Library...` and select "Video" from the list (now that we have installed it)
- That will write the appropriate `import` at the *top* of our code (we always `import` at the top of the main program)

The `Capture` object

- To access the information the video library makes available we use a `Capture` object
- This is a common way that we use libraries - they provide a specific `class` that we can make an object from to access the library's abilities
- Because a `class` is a great way to keep a whole lot of methods and properties together safely
- So we need to declare a `Capture` variable like this

```
Capture video;
```

Making a new `Capture` object

- To actually put an object into our variable, we need to make a `new` one like this in `setup()`

```
video = new Capture(this,640,480,30);
```

- `this` is a bit mysterious but refers to the *this program* (the `Capture` object wants to know about it)
- `640` and `480` are the dimensions we want to capture from the webcam
- `30` is the framerate to capture at

Starting the webcam

- To actually tell the webcam to turn on and start capturing video we need to write

```
video.start();
```

- We don't *have* to do that in `setup()`, we could trigger it elsewhere
- But for now I'll be doing it in `setup()`

Once again

- So to get set up with the webcam we need:

```
import processing.video.*;

Capture video;

void setup() {
  video = new Capture(this,640,480,30);
  video.start();
}
```

- And indeed if we run this, the webcam light should turn on

Size matters

- If we capture at 640x480 that is 307,200 pixels in each frame of video
- If we're capturing at 30 frames per second that is 9,216,000 pixels per second
- That is a lot of pixels, mon frère
- If you're *processing* every single pixel, every single frame, you're performing almost 10 million of those calculations per second
- So don't be surprised if things slow down a bit potentially

Webcam chat with yourself

```
import processing.video.*;

Capture video;

void setup() {
  size(640,480);
  video = new Capture(this,640,480,30);
  video.start();
}

void draw() {
  if (video.available()) {
    video.read();
  }
  image(video,0,0);
}
```

Before we End…

- Assignment-1
- Submission 20/6/2018 before 3:00PM
- Hard Copy: Submission
- Assignment Drop-Box @ 3rd Floor (In front of Office)

- Note: Lab Session:- You have to (individual-group) try to code the task. It is your practice session.
- The solution of the try-out code will be provided to you at the end of the day. ☺

LAB ACTIVITY: (BACK-LoG)

- You may work as your own groups.

- Task:
  - Create a Class Bug()
  - Populate the Bug into a Canvas with Multiple Bug
    - You can use Array[] to achieve the output

- Indicative Solutions will be provided through Moodle

The Game of Life:

- The Game of Life is not your typical computer game. It is a 'cellular automaton', and was invented by Cambridge mathematician John Conway.

- https://bitstorm.org/gameoflife/

Task 2 : Create a Pixel Game: Two gliders

Next Week

- Assignment-#2
- Surgery Session #1
- Group's Progress on Assignment-2.

- Lecture on Processing Algorithms & Deployment

Stretch Break!

Thank you