Computation as an Expressive Medium

Lab 4: Pretty, Pretty Pictures

Joshua Cuneo
For today

- ¼ Checkpoint
- Building Blocks
- Assignment 2
- Images + More Images
- Accessing Pixels
- 2D Arrays
- Project 2
Checkpoint
Organizing these manipulations

Manipulating stored data

Storing the categorized data

Categorizing the data

Data

Letters: a, B, c, &, (, !
Numbers: 1, 2, 3

int, float, char

Variables
Arrays

if
for
Arithmetic

Parsing
Drawing

Functions
Methods
Classes

Hard-coded

User input

mouseX
mouseY
Text boxes

Processing

width
height
second()
random()

Files

Image files
Text files
Web files
Assignment 2!

- A2-01: Using beginShape() and endShape(), create a composition with five or more vertices.
- A2-02: Using beginShape() and endShape(), create a composition with ten or more vertices.
- A2-03: Create an image different from A2-02, but using the same vertex data.
- A2-04: Write a function with one parameter and demonstrate it visually.
- A2-05: Write a function for drawing triangles and visually demonstrate its flexibility.
- A2-06: Write a function with three or more parameters and visually demonstrate its flexibility.
- A2-07: Create a dynamic animation using the cos() function as a generator for motion.
- A2-08: Create a dynamic animation using the cos() and sin() function as a generator for motion.
- A2-09: Move two visual elements across the screen using the random() function as a generator of movement. Give each element a unique nonlinear motion.
- A2-10: Create an event that begins when the mouse is pressed and ends when the mouse is released.
- A2-11: Create a responsive image that behaves differently when the mouse is moving and the mouse is dragging.
- A2-12: Create a button that changes the color of the background when it is clicked.
- A2-13: Program your moving elements from A2-09 but use classes to represent the two visual elements.
- A2-14: Create a subclass of one of the asteroids classes that adds a new capability. Some examples of what you could do: create a subclass of Rocket (or ArmedRocket) that shoots flame when the thrusters are fired and/or plays a sound when thrusters are fired, create a subclass of Asteroid that know when it's been hit (instead of doing this test in draw()), create a subclass of Asteroid that splits into two smaller Asteroids when it's hit.
In fewer words...

- `<sketchbook directory>`
- `sketch_name`
- `sketch_name.pde`
- `data`
- `some.jpg`
1. **Bring the image into Processing**

   ```java
   PImage startrek = loadImage("startrek.jpg");
   ```

2. **Load the image to our canvas**

   ```java
   image(startrek, 0, 0);
   ```
Accessing Pixels

- loadPixels();
- pixels[];
- updatePixels();
Accessing Pixels

0 1 2 3 4

0 1 2 3 4

0 1 2 3 4

0 1 2 3 4
Accessing Pixels
Accessing Pixels

The diagram illustrates a 2D pixel grid with rows and columns labeled from 0 to 4 and 0 to 3, respectively. The grid is partially filled with yellow, indicating certain pixels that are accessed. The arrows point to specific pixels, demonstrating the access pattern.
Accessing Pixels
Accessing Pixels

(4, 0) = 4 + 0*5 = 4
(3, 2) = 3 + 2*5 = 13
Accessing Pixels

Array Index of pixel \((x, y)\)

\[ = x + y \times \text{width} \]

Where width = pixel width of the image
PImage im = loadImage("test1.jpg");

// gets color at (3, 2)
color c1 = im.pixels[3 + 2*im.width];

// set our line color so we can draw with this color.
stroke(c1);
Window vs. Image

// gives us the color at (3, 2) in the window.
color c2 = pixels[3 + 2*width];

VS

PImage im = loadImage("test1.jpg");

// gets color at (3, 2)
color c1 = im.pixels[3 + 2*im.width];
Window vs. Image

- Window’s pixels: manipulate the canvas
- Image’s pixels: manipulate the image before placement
2D Arrays

```java
int[][] bob = new int[3][4];
color[][] pixels2d = new color[200][200];
```
2D to 1D

I’m going to make life easy for you

```c
int twoD2oneD(int x, int y, int picWidth) {
    return x + y*picWidth;
}
```
beginShape(POLYGON); 
for(int i = 0; i < xvals.length; i++) { 
    vertex(xvals[i], yvals[i]); 
} 
endShape();
Translation

\[ \text{translate}(10, 10); \]
```java
beginShape(POLYGON);
for(int i = 0; i < xvals.length; i++) {
    vertex(xvals[i], yvals[i]);
}
endShape();
```
rect(10, 10, 50, 50);
float angle = radians(45);
rotate(angle);
rect(10, 10, 50, 50);
Rotation & Translation

```
translate(35, 35);
```
Rotation & Translation

rect(-25, -25, 50, 50);

(0, 0)

(35, 35)
float angle = radians(45);
rotate(angle);
rect (-25, -25, 50, 50);
Wait! How do I get back to normal?!

whole crazy mess
started before this

I just want to go back
to the Kansas I just want

(0, 0)

(35, 35)
pushMatrix();
translate(35,35);
rotate( radians(45) );
rect(-25,-25,50,50);
popMatrix();
rect(-25,-25,50,50);
How is this useful?

(You don’t want to have to calculate the angles of every finger, do you?)
Cell

Properties (Variables)
- Membrane
- Reticulum
- Nucleus
- Ribosome
- Mitochondria

Methods
- Replicate
- Process nutrients
- Contract viruses
- Die
Blood Cell

Properties (Variables)
- Everything a cell has

Methods
- Everything a cell can do
- Travels through veins and capillaries
Red Blood Cell

Properties (Variables)
- Everything a blood cell has
- Mechanism to carry oxygen and carbon dioxide gases

Methods
- Everything a blood cell can do
- Exchanges gases with other cells
White Blood Cell

Properties (Variables)
- Everything a blood cell has
- Mechanism to recognize disease

Methods
- Everything a blood cell can do
- Fights off diseases and foreign objects
If we were programming, we would program a **Cell**, then a **Blood Cell**, then a **Red Blood Cell** and a **White Blood Cell**.

We would only program the methods and properties specific to that cell level.
• A red blood cell is a **type of** blood cell, which is a **type of** cell
• The red blood cell *inherits* the blood cell, which *inherits* the cell
• The blood cell is the **superclass** of the red blood cell
• The red blood cell is the **subclass** of the blood cell
Terms

- The red blood cell is an *object*, or template
- Each red blood cell in your veins is an *instance of* a red blood cell object
- Here, we have many instances of red blood cells and one instance of a white blood cell
Why Do We Care About OOP?

- Computationally faster
- Reflects organization in the real world
- Easier to read and code
- Easier to modify the program
Procedural

Object-oriented
Project 2

The contemporary computer scene is dominated by the graphical user interface (GUI). For almost every task, from manipulating text, imagery, sound or video to configuring a computer's operating system (e.g. control panels), from searching for and organizing information (e.g. the web) to the process of programming (e.g. integrated development environments), there are special purpose GUI tools supporting the task through analogies to embodied interaction with physical objects. But no tool is neutral; every tool bears the marks of the historical process of its creation, literally encoding the biases, dreams, and political realities of its creators, offering affordances for some interactions while making other interactions difficult or impossible to perform or even conceive. While the ability to program does not bring absolute freedom (you can never step outside of culture, and of course programming languages are themselves tools embedded in culture), it does open up a region of free play, allowing the artist to climb up and down the dizzying tower of abstraction and encode her own biases, dreams and political realities. What graphical tools would you create? Create your own drawing tool, emphasizing algorithmic generation/modification/manipulation. Explore the balance of control between the tool and the person using the tool. The tool should do something different when moving vs. dragging (moving with the mouse button down). The code for your tool should use at least one class.
Project 2: Translation

- Create an awesome drawing tool
- Give the user some control and the computer some control
- Make it do something different with moving vs. dragging