IAT 265

Advanced Topics on Image Manipulations & Debugging Tips

SCHOOL OF INTERACTIVE
ARTS + TECHNOLOGY

Outline

- Topics:
  - Review: Loading & displaying images; Accessing pixels
  - Color distance
  - Remove red-eyes from picture
  - Chromachying (blue-screen technique)
  - Debugging Tips

Review: Loading & Displaying images

- Loading Images

```
PImage img = loadImage("<image filename>");
```

- Displaying images

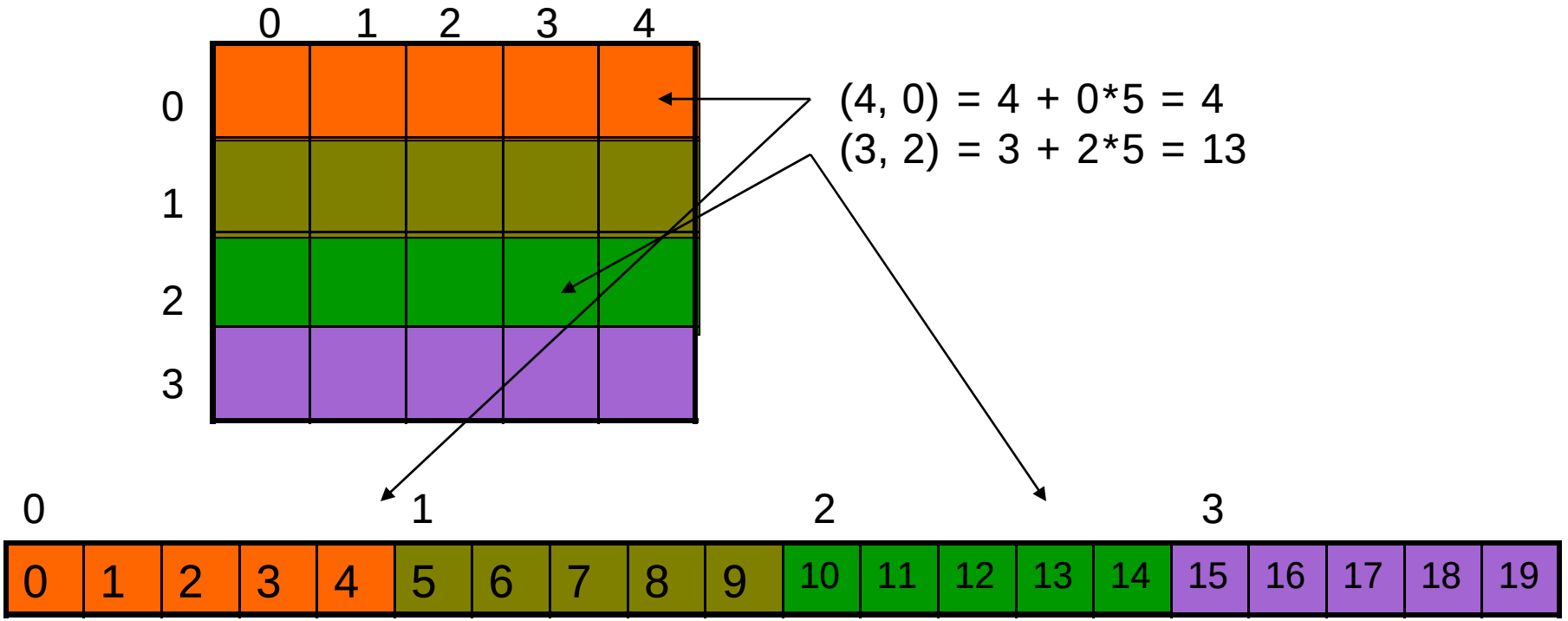
```
image( PImage img, int x, int y);
```

Review: Accessing Pixels

- The `PImage` class allows you to access the RGB values of each individual pixel of the image, with the `pixels[]` array.
- You can get the width and height of the image file using the `width` and `height` fields of `PImage`.

Review: Accessing Pixels

- Array Index
  - x + y*width



Conditional Pixel Change

- How can you remove the red-eyes?

Color Distance and its Applications

- Recall the distance between two points:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- Similarly distance between two colors:

$$\sqrt{(red_1 - red_2)^2 + (green_1 - green_2)^2 + (blue_1 - blue_2)^2}$$

Implementation of Color Distance

```
float calColorDistance (color c1, color c2) {
    float rDist = red(c1) – red(c2);
    float gDist = green(c1) – green(c2);
    float bDist = blue(c1) – blue(c2);

    float distance = sqrt(rDist * rDist + gDist * gDist + bDist * bDist );

    return distance;
}
```

- *float red(color c); float green(color c); float blue(color c)* – extracts the *red, green, blue* value respectively from a *color* object

Case Study: Remove Red-Eyes

```
void removeRedEyes(int startX, int startY, int
    endX, int endY, int distThreshold, color
    newColor) {
      color red = color(255, 0, 0);
     for (int x = startX; x < endX; x++)
          for (int y = startY; y < endY; y++) {
               color color1 = color(img.get(x, y));
               if (calColorDistance(color1, red) <
                  distThreshold) {
                  img.set (x, y, newColor);
               }
          }
}
```

↓

Case Study: Remove Red-Eyes (1)

```
PImage img;           //declare a variable
void setup(){
  img = loadImage("jenny-red.jpg");  //create img object
  size(img.width, img.height);

  //Remove the red-eyes and display
  removeRedEyes(100, 90, 200, 120, 100, color(0, 0, 0));
  image(img, 0, 0);

  //save a copy of the modified image
 save("data\\jenny.jpg");
}
```

Get/Set Pixel at (x, y)

Suppose you have a PImage objcet ***img*** :

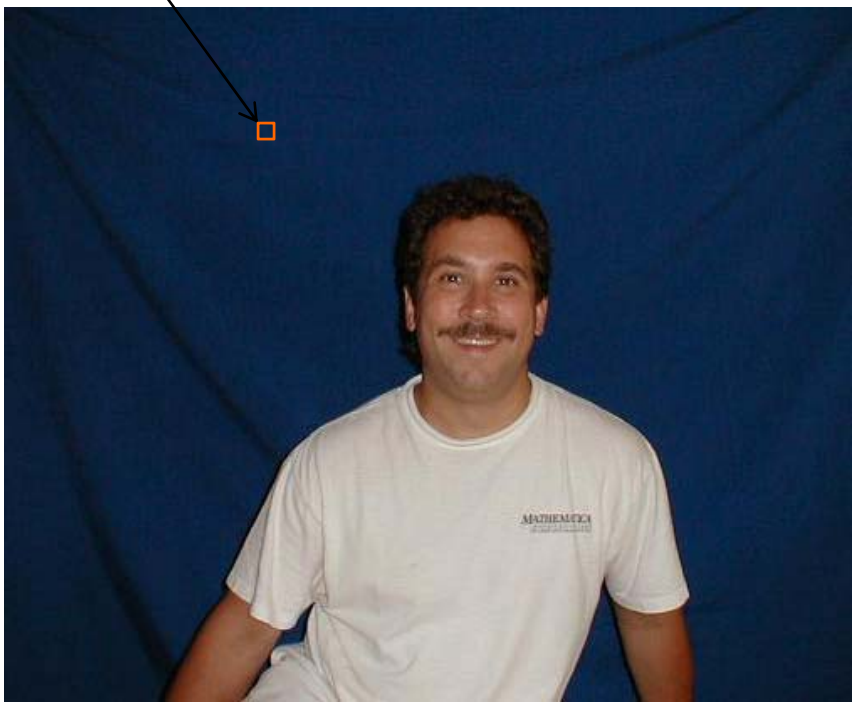
- ***color img.get(x, y)***
  - Reads the color of a single pixel at (x, y)
- ***void img.set(x, y, color c)***
  - Changes the color of a single pixel at (x, y)

Chromakeying

- Chromakeying – the process of superimposing one image on top of another by selective replacement of color
  - Coming from Bluescreen Compositing in film making – invented in 1930s
  - Has evolved into the most commonplace and effective special effect in film

Example of Chromakeying

Key colors will be replaced by targeted colors



Case Study: Chromakeying

```
PImage chromakey(PImage orgImg, PImage tarBGImg) {
    int w = orgImg.width;
    int h = orgImg.height;
    PImage chromedImg = createImage(w, h, RGB);

    for (int x=0; x<w; x++)
    {
        for (int y=0; y<h; y++)
        {
            color orgColor = orgImg.get(x,y);
            color tarColor = tarBGImg.get(x,y);

            color appliedColor = replaceKeyColor(orgColor, tarColor);

            chromedImg.set(x, y, appliedColor);
        }
    }

    return chromedImg;
}
```

Case Study: Chromakey (1)

```
color replaceKeyColor(color orgc, color tarc) {
    color newColor;
    float r = red(orgc);
    float g = green(orgc);
    float b = blue(orgc);

    if (b > (r + g))                    // If blue color is the dominant color
                                        // replace it with target color
        newColor = tarc;
    else
        newColor = orgc;                //Otherwise keep the original color

  return newColor;
}
```

Case Study: Chromakey (2)

```
void setup(){
  PImage img = loadImage("bluebk.jpg");
  PImage bkImg = loadImage("moon.jpg");
  size(img.width*2, img.height);

  image(img, 0, 0);                    //draw original image

  img = chromakey(img, bkImg);      //do chromakeying

  image(img, img.width, 0);          //draw modified image

}
```

Debugging

How do I know my program is broken?

- Compiler Errors
  - Errors the compiler can catch – easy to fix
- Runtime Exceptions
  - Errors the runtime environment can catch – more difficult to fix
- Your program just doesn't do the right thing
  - The trickies one to fix!!

Compiler Errors

- Syntax errors:
  - Maybe missed a bracket/brace/semicolon or other necessary syntax element – easy to fix
- Logical errors – could be tricky for newbie
  - Method call inconsistent with method's signature
  - Access a variable beyond its scope
  - No return statement in a method with a return type
  - Put a non-boolean method in an *if*-statement

How to fix Syntax Errors?

- Start at the top of the error list
- Use the line number!
- If that line looks OK, check the line above

Match or Count Brackets and Braces

- Mouse click to match braces in pair if IDE has the functionality – such as Processing
- Otherwise – count to match

```
{  qwdkj                    0
   {   dw wqdlk lqwd        1
     {    n,mnwq            2
        }                   2
       }                    1
             }              0
```

Braces match if the last == 0!

Logical Errors

Method call inconsistent with method's signature

- Signature: *Bug(float x, float y, float chgX, float chgY, float sz)*
- Wrong:
  - bugs.add(new Bug());
  - AvatarBug(float x, float y, float chgX, float chgY, float sz) {
    super();
    }
- Right:
  - bugs.add(new Bug(random(width), random(height), random(-1,1),random(-1,1),random(12,36)));
  - AvatarBug(float x, float y, float chgX, float chgY, float sz) {
    super(x, y, chgX, chgY, sz);

Logical Errors (1)

Access a variable beyond its scope

- The "**scope**" of a variable refers to the variable's visibility within a program
  - **Global variables**: accessible from anywhere in the program
  - **Fields** (class member variables): accessible from anywhere in the class that they are declared
  - **Local variables**: declared within a method and accessible only within the method

- A common error: access local variables outside the method that they are declared

Logical Errors (2)

An Example:

- Declare a variable (e.g. *avtBug*) within one method:

```
void setup() {
    AvatarBug avtBug = respawn();
}
```

- then try to access it in another method:

```
void playGame() {
      if (rightKey) avtBug.moveRight();
}
```

- To fix:
  - Declare *avtBug* as a global variable (i.e. declare it outside any method and class)

Logical Errors (3)

No return statement in a method with a return type

- The following method returns boolean but no return statement as the last statement inside it

```
boolean detectCollision(Bug otherBug) {
    if ( abs(bugX-otherBug.bugX)<(bSize+otherBug.bSize)  &&...) {
      return true;
    }
    //??????????????
  }
```

- To fix:

```
boolean detectCollision(Bug otherBug) {
    if ( abs(bugX-otherBug.bugX)<(bSize+otherBug.bSize) && ...) {
      return true;
    }
    return false;
```

Logical Errors (4)

- Or a better way to avoid this issue:

```
boolean detectCollision(Bug otherBug) {
    boolean hit = false;
    if ( abs(bugX-otherBug.bugX)<(bSize+otherBug.bSize) && ...) {
      hit = true;
    }
    return hit;
}
```

Logical Errors (5)

Put a non-boolean method in an *if*-statement

- Wrong:

```
void detectBound() {
   if (bugX+bSize > width ) {
    bugX = width-bSize;
   }
   if (bugX-bSize < 0 ) {
     bugX = bSize;
  }}
```

```
void draw() {
   …
 if( bugi.detectBound() ){
     changeX *= -1;
  }

    …
}
```

- To fix: either keep the signature, change the way to call it

```
void detectBound() {
   if (bugX+bSize > width ) {
    bugX = width-bSize;
    changeX *= -1;
   }
   if (bugX-bSize < 0 ) {
     bugX = bSize;
     changeX *= -1;
  }}
```

```
void draw() {
     …
    bugi.detectBound();
     …
 }
```

Logical Errors (6)

- or keep the way of calling, change the signature and the logic inside:

```
boolean detectBound() {
   boolean hit = false;
   if (bugX+bSize > width ) {
    bugX = width-bSize;
    hit = true;
   }
   if (bugX-bSize < 0 ) {
     bugX = bSize;
    hit = true;
  }
  return hit;
}
```

```
void draw() {
   …
 if( bugi.detectBound() ){
     changeX *= -1;
  }

   …
}
```

Runtime Exceptions

Common Runtime Exceptions:

- NullPointerException and ArrayIndexOutOfBoundsException

- Exceptions caused by semantic errors
  - uninitialized variable, bad loop logic, ...

NullPointerException

- Normally because you're calling an object's method when you failed to instantiate it
  - So the variable for the object becomes a pointer pointing to NULL

```
AvatarBug avtBug;
//failed to instantiate it like:
//avtBug = new AvatarBug(width/2, height/2, 4,4,20);
//but try to call its method like:
avtBug.drawBug(); → NullPointerException!!
```

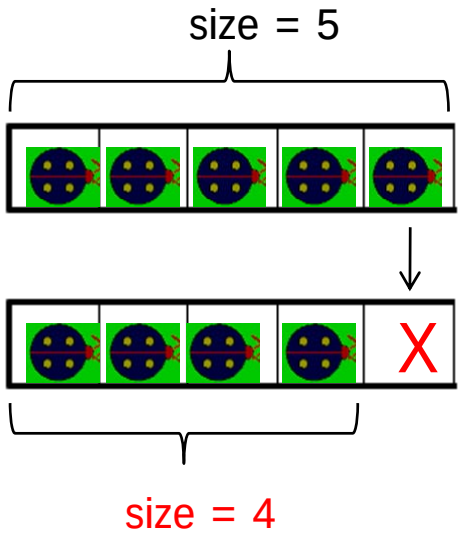
- To fix: Always instantiate after declaring an object!!

ArrayIndexOutOfBoundsException

- Normally because you try to access an array index that goes beyond the max length of an array
  - It becomes tricky esp. when you try to use an constant control value to access a resizable array like ArrayList

```
int count = 5;
for(int i=0; i< count; i++) {
    Bug bugi = (Bug) bugs.get(i);
    ...
```

- To fix: use *bugs.size() as controller*



Your program just doesn't do the right thing

- Use println() when dealing with “Your program just doesn't do the right thing”
  - Print a string literal inside a method or a code block ({...}) to check if it is really gets executed
  - Print variables, array values, array index, objects etc to check if the values are as expected

#1 debugging tip

- **TEST YOUR CODE OFTEN!!**
  - **Code a little bit, Test a little bit, Make it work, Move on**
  - Catching small errors early will help you avoid the big complicated errors later

Wrong and Right

- Build
- Build
- Build
- Build
- Build
- Build
- Test

- Build
- Test
- Build
- Test
- Build
- Test
- Build
- Test

Summary

- Color distance
- Remove red-eyes from picture
- Chromachying (blue-screen technique)
- Debugging tips