Multidimensional Arrays

Eric Roberts
CS 106A
May 9, 2012

Pictures and Pixels

- The focus of today’s lecture is on how to represent images using two-dimensional arrays of tiny dots called pixels.
- The idea of representing an image as a two-dimensional array of dots is much older than modern computing and has several antecedents:
  - Halftone pictures in newspapers
  - Pointillist art
  - Mechanical weaving
- Of these examples, the most amazing connections to modern computing are in weaving, which contributed substantially to early computing.

Arrays of Arrays

- Internally, Java represents multidimensional arrays as arrays of arrays. It is often important to keep this fact in mind when you are creating array structures.
- For example, Java often requires you to initialize the rows of a two-dimensional array individually, as in the following revised code for creating the tic-tac-toe board:

```java
char[][] board = new char[3][3];
for (int i = 0; i < 3; i++) {
    board[i] = new char[3];
}
```

Weavings on the Jacquard Loom

Multidimensional Arrays

- Because the elements of an array can be of any Java type, those elements can themselves be arrays. Arrays of arrays are called multidimensional arrays.
- In Java, you can create a multidimensional array by using multiple brackets in both the type and the initialization parts of the declaration. For example, you can create array space for a 3x3 tic-tac-toe board using the following declaration:

```java
char[][] board = new char[3][3];
```

Arrays of Arrays

- This declaration creates a two-dimensional array of characters that is organized like this:

```
board[0][0]  board[0][1]  board[0][2]  
board[1][0]  board[1][1]  board[1][2]  
```

Static Initialization

- Java makes it easy to initialize the elements of an array as part of a declaration. The syntax is

```java
type[] name = {elements};
```

where `elements` is a list of the elements of the array separated by commas. The length of the array is automatically set to be the number of values in the list.
- For example, the following declaration initializes the variable `powersOfTen` to the values 10^0, 10^1, 10^2, 10^3, and 10^4:

```java
int[] powersOfTen = {1, 10, 100, 1000, 10000};
```

This declaration creates an integer array of length 5 and initializes the elements as specified.
Initializing Multidimensional Arrays

- You can initialize a multidimensional array when you declare it by using nested braces to reflect the levels of array nesting.
- For example, you can declare and initialize a multiplication table for the digits 0 to 9 like this:

```java
private static final int[][] MULTIPLICATION_TABLE = {
    { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
    { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 },
    { 0, 2, 4, 6, 8, 10, 12, 14, 16, 18 },
    { 0, 3, 6, 9, 12, 15, 18, 21, 24, 27 },
    { 0, 4, 8, 12, 16, 20, 24, 28, 32, 36 },
    { 0, 5, 10, 15, 20, 25, 30, 35, 40, 45 },
    { 0, 6, 12, 18, 24, 30, 36, 42, 48, 54 },
    { 0, 7, 14, 21, 28, 35, 42, 49, 56, 63 },
    { 0, 8, 16, 24, 32, 40, 48, 56, 64, 72 },
    { 0, 9, 18, 27, 36, 45, 54, 63, 72, 81 }
};
```

Exercise: Representing a Chess Board

Write a program to number the squares in a crossword grid if they appear at the beginning of a word running either across or down.
- What type might you use to represent the information in this grid?
- How would you represent black squares?
- What rules can you supply to determine if a square is numbered?

Exercise: Crossword Numbering

Pixel Arrays

- If you have a `GImage` object, you can obtain the underlying pixel array by calling the `getPixelArray`, which returns a two-dimensional array of type `int`.
- For example, if you wanted to get the pixels from the image file `JTFLogo.png`, you could do so with the following code:

```java
GImage logo = new GImage("JTFLogo.png");
int[][] pixels = logo.getPixelArray();
```
- The first subscript in a pixel array selects a row in the image, beginning at the top. The height of the image is therefore given by the expression `pixels.length`.
- The second subscript in a pixel array selects an individual pixel within a row, beginning at the left. You can use the expression `pixels[0].length` to determine the width of the image.

Multidimensional Arrays and Images

- One of the best examples of multidimensional arrays is a Java image, which is logically a two-dimensional array of pixels.
- Consider, for example, the logo for the Java Task Force at the top right. That logo is actually an array of pixels as shown in the expanded diagram at the bottom.
- The `GImage` class allows you to convert the data for the image into a two-dimensional array of pixel values. Once you have this array, you can work with the data to change the image.

Pixel Values

- Each individual element in a pixel array is an `int` in which the 32 bits are interpreted as follows:

```
transparency (1)
red
green
blue
```
- The first byte of the pixel value specifies the transparency of the color, which is described in more detail on a later slide.
- The next three bytes indicate the amount of red, green, and blue in the pixel, in which each value varies from 0 to 255. Together, these three bytes form the RGB value of the color, which is typically expressed using six hexadecimal digits, as in the following examples:
Combining Colors of Light

Transparency

- The first byte of the pixel value specifies the transparency of the color, which indicates how much of the background shows through. This value is often denoted using the Greek letter alpha (\(\alpha\)).
- Transparency values vary from 0 to 255. The value 0 is used to indicate a completely transparent color in which only the background appears. The value 255 indicates an opaque color that completely obscures the background. The standard color constants all have alpha values of 255.

Image Manipulation

- You can use the facilities of the `GImage` class to manipulate images by executing the following steps:
  1. Read an existing image from a file into a `GImage` object.
  2. Call `getPixelArray` to get the pixels.
  3. Write the code to manipulate the pixel values in the array.
  4. Call the `GImage` constructor to create a new image.
- The program on the next slide shows how you can apply this technique to flip an image vertically. The general strategy for inverting the image is simply to reverse the elements of the pixel array.

The `FlipVertical` Program

```java
public static GImage flipVertical(GImage image) {
    int[] array = image.getPixelArray();
    int height = array.length;
    for (int p1 = 0; p1 < height / 2; p1++) {
        int p2 = height - p1 - 1;
        int[] temp = array[p1];
        array[p1] = array[p2];
        array[p2] = temp;
    }
    return new GImage(array);
}
```

Selecting Color Components

- If you want to work with the colors of individual pixels inside a pixel array, you can adopt either of two strategies:
  - You can use the bitwise operators described in the text to select or change individual bits in the pixel value.
  - You can use the static methods provided by the `GImage` class to achieve the same purpose.
- Although it is useful to remember that all information is stored as bits, there doesn’t seem to be much point in going into all the details, at least in CS 106A. We will therefore use the second strategy and employ the static methods `getRed`, `getGreen`, `getBlue`, `getAlpha`, and `createRGBPixel`.

Creating a Grayscale Image

- As an illustration of how to use the bitwise operators to manipulate colors in an image, the text implements a method called `createGrayscaleImage` that converts a color image into a black-and-white image, as shown in the sample run at the bottom of this slide.
- The code to implement this method appears on the next slide.
The `createGrayscaleImage` Method

```java
/* Creates a grayscale version of the original image */
private GImage createGrayscaleImage(GImage image) {
    int[][] array = image.getPixelArray();
    int height = array.length;
    int width = array[0].length;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            int pixel = array[i][j];
            int r = GImage.getRed(pixel);
            int g = GImage.getGreen(pixel);
            int b = GImage.getBlue(pixel);
            int xx = computeLuminosity(r, g, b);
            array[i][j] = GImage.createRGBPixel(xx, xx, xx);
        }
    }
    return new GImage(array);
}

/* Calculates the luminosity of a pixel using the NTSC formula */
private int computeLuminosity(int r, int g, int b) {
    return GMath.round(0.299 * r + 0.587 * g + 0.114 * b);
}
```

The ImageShop Application