Final Project
Oral report, code, written report due 29 April (last day of class)
Phase 1: due Monday, 7 April

Either individually or in teams of at most two, propose and carry out a significant coding project using one of the tools we have studied in this course; write a comprehensive report about the project; present it to the class. “Significant” refers not just to the number of lines of code (which should certainly be at least as large as one of the more complicated lab assignments), but also to the level of difficulty and the degree of originality.

For instance, making minor tweaks to a complex program downloaded from openprocessing.org is not considered a valid project, since very little of the code submitted would be “original” to you. Similarly, carrying out a complicated image development process by simply following a step-by-step online Gimp tutorial is not acceptable, since all the steps have been “figured out in advance” for you.

However, using an existing program or tutorial as a foundation for an original project is acceptable, provided you make a significant contribution of your own original code or analysis in order to carry out the project. (And, of course, you must fully document your sources!)

For phase 1: Submit a written project proposal no later than Monday, 7 April.

Here are a few suggestions:

• **Autostereograms.** Section 8.3 of Tanimoto’s book *An Interdisciplinary Approach to Image Processing*, hereafter referred to simply as “Tanimoto”, is about “autostereograms.” These are also called “Magic Eye” pictures—see, e.g., the Magic Eye Image of the Week at [http://www.magic-eye.com/3dfun/stwkdisp.shtml](http://www.magic-eye.com/3dfun/stwkdisp.shtml) Tanimoto described the process of creating a Magic Eye image using PixelMath.

Either carry out one of PixelMath exercises 6, 7, or 8 on pages 187–188, or else implement the autostereogram process in a language such as Processing. In your report, give a detailed description of the process, including a review of the relevant mathematics.

• **Steganography.** Sections 9.1–9.6 of Tanimoto are about steganography, or hiding information inside an image. In a language other than PixelMath (e.g., Processing), implement a two-program system that permits a user to hide a black-and-white image (no shades of gray) inside an arbitrary image and then to retrieve the image. It should be possible for the user to specify an arbitrary black and white image (let’s call it “hiddenimage”) and an arbitrary color image of dimensions greater than or equal to the hidden image (let’s call it “carrierimage”), and to encode hiddenimage into carrierimage in a lossless format such as PNG. The decoding program should take such an encoded PNG and extract the hidden black and white image from it. Use two different methods, e.g., the “least significant bit” or “exclusive or” or
“random mask” method from Tanimoto. (Or use one of the more advanced methods if you prefer.)

In your report, compare the two methods in terms of their effectiveness or efficiency or ease of implementation or ... Consider the limits of your method. For instance, using the least significant bits of the R, G, and B values, we could encode three black-and-white images, or one grayscale image with eight different gray levels. Is there a noticeable degradation in the carrier image? Using the two least-significant bits of each color we can encode six black-and-white images, or three grayscale images with four gray levels each, or one grayscale image with 64 gray levels. Where does the process become infeasible or impractical?

- **Collision Detection.** Read chapter 5, “Physics Libraries,” in Daniel Shiffman’s free online book *The Nature of Code*. Using Shiffman’s libraries of Processing/Java methods, implement an interactive program that enables the user to create different shapes, moving at different speeds (all set interactively), and colliding “realistically.”

In your report, explain the setup process, review the relevant mathematics, and explain how the program is organized and how it works. Note: the fine details are not needed; as Shiffman writes, “We can’t expect to master every detail of physics simulation.” But at least some basics are expected—fundamental rules of force, gravity, and so.

- **Make an Interactive Game or Puzzle.** In Visual Python or Processing, create an interactive game or puzzle. It should use multiple sources of interaction, e.g., mouse and keyboard; it should not be something nearly the same as a game from another source such as processing.org; it should be more visually complex than a variation of “Pong” or “Flappy Bird”. Ideally, it should be original and not a remake of an existing game. Remember that this is a course on visual computing—your game should have a very strong visual component. Don’t get too bogged down in game logic and neglect the graphics!

Your report should detail the organization of your program (please use variables and method names that are intuitive; please comment your code!) and explain in detail how it is played, what things can go wrong, and any limitations.

- **Explore an Advanced Graphics Algorithm.** Look at a problem in image processing. Here are two good ones to consider:

  - The “flood fill” algorithm: the user selects a color, then clicks on the screen. All pixels connected to the clicked pixel except for certain designated “boundary” colors (e.g., black) get changed to the selected color. (This is like the “bucket fill” tool in a lot of drawing programs.)

  - Edge detection algorithms: Do a study of several different algorithms for detecting edges. Tanimoto’s book discusses several simple ones in Chapter 17; implement these and then try to find one more that you can either implement or else explain. Don’t worry about the advanced mathematics, but do try to provide some intuition behind the math.

There will be several stages to the project; subsequent handouts will describe these in more detail.