Scaling: A Design Pattern in Introductory Computer Science Courses

Harriet J. Fell, Viera K. Proulx, and Richard Rasala
Northeastern University
College of Computer Science
Boston, MA 02115, USA

fell@ccs.neu.edu, vkp@ccs.neu.edu, rasala@ccs.neu.edu

Abstract

We present a series of programming exercises that use scaling as a theme for teaching design techniques in the introductory computer science course sequence. All exercises are on the level easily mastered in the first year of programming. Additionally, the exercises introduce a rich variety of applications of computer science.

Introduction

Introductory computer science courses typically focused on teaching basic programming language constructs, followed by a study of basic algorithms and data structures. As the field matured, the emphasis on design and decomposition of a problem into smaller independent parts filtered into the introductory curriculum. The framework for teaching design is evolving. We believe that teaching is best done in a framework of basic program patterns and design threads that appear repeatedly in typical programs. Such a setting provides both the motivation and the context for illustrating design decisions and design practice.

In this paper we show how the concept of scaling permeates computing today and present a suite of exercises that explore and illustrate this concept throughout the introductory computer science course sequence. In addition to presenting several interesting applications of computing, the exercises provide a framework for teaching design. They provide a progression from implementing scaling formulas locally, to designing a scaling function, and finally to designing a scaling class.

Partial support for this work has been provided by the National Science Foundation Leadership in Laboratory Development , award #DUE- 9650552

SIGCSE'97.
Copyright 1997 ACM 1-58113-499-1/97/0006…$5.00.

Examples of scaling

Scaling is ubiquitous to computing. It appears in many forms, not all of them immediately identifiable as simple scaling. We give below a sample of the computing activities in which a scaling type of computation plays a role. These activities span a wide variety of applications and provide a context for numerous interesting exercises.

Direct scaling examples

Typical unit conversions (°F to °C, kilos to pounds, inches to centimeters) are the simplest examples of linear transformation scaling. A similar computation is used to convert random numbers supplied by a random number generator (typically in a fixed standard range) into random values within the particular range needed by the program. Plotting a function graph in a rectangular frame also requires linear scaling. The scale factors along the two axes may be the same (shape preserving scaling) or may be different allowing a better fit of the graph into the available space. The scaling of images to fit into a given region is the most obvious application of scaling. Here the images may be bit mapped or be generated from a collection of geometric objects (lines, rectangles, ovals, or polygons). Typically, the scaling transformation parameters (the offset and the scaling factor) are computed from the given boundaries of the original image (a rectangle given in real world coordinates) and the boundaries of the target rectangle in the graphics window area. In simple animations, sound may be generated proportionately to the location of an object - another example of scaling.

A colored contour map of a function of two variables scales the function values into the color spectrum. If we want to distinguish the values within a particular range, we may use nonlinear scaling that will provide larger variation among the values in this range. Examples of the use of these techniques are medical imaging (CAT Scan) and satellite image processing. Scaling of color is also used to create the wonderful images we see of Julia sets and the Mandelbrot set. An additional technique sometimes used to further identify the details of such contour maps is sonification. Here as a part of the image displayed on a computer screen is traversed with a mouse, the pixel color value is mapped into a frequency of a sound. The sound is heard as the picture is traversed.

Subtle variations not identifiable by sight often become prominent (for example small ripples).

Linear interpolation

LERPing [5] or linear interpolation is used to produce "in-betweens" in animations, or "morphs" of Mona Lisa with George Washington. "Given the values v_s and v_e, of some attribute (position, color, size) in the starting and ending frames, the value of v_t at intermediate frames is $v_t = (1-t)v_s + tv_e$; as the values of t range from 0 to 1, the value of v_t varies smoothly from v_s to v_e.." [2] Lerping is just a special case of scaling, from [0, 1] to [v_s , v_e;]. Lerping can also be used to blend a weather satellite image with a Mercator weather map [6], or to smoothly move a sound from one frequency to another or one amplitude to another.

When a lerping or scaling transform is set up to morph two pictures, values of t greater than 1 result in a picture with features similar to those of the end image but even less like the start image. This can be used to produce automatic caricatures of an image.

Changes in the frame of reference.

This type of transformation was be the bread and butter of engineers who used a slide rule to perform multiplications by converting two numbers into their logarithms, adding the logarithms using the slide, and then converting the result back into the original scale. Today, Fourier transforms used in digital signal processing represent another such example of the shift of reference. There are simpler examples that are more suitable for introductory courses. One is the conversion of character's ASCII code to the letter's sequential position in the alphabet - used in simple Ceasar shift coding. Another is the mapping of an array in the range from `1` to `maxindex` to a range from `low` to `high`.

Additionally, if a program is interactively using the mouse to make a selection, be it from a menu or another kind of display, the mouse location needs to be converted to the value that represents the selection. Conversely, to highlight a selection, the selection value needs to be converted to the coordinates of its display on the screen.

Scaling design thread in the first year

In our introductory curriculum scaling forms a design thread that is revisited several times in a different context. We start with direct scaling performed as asimple arithmetic operation every time it is needed. Several other labs use scaling to implement interesting applications. Later, students implement scaling computation as a function. Finally, students learn to build a scaling transform class that encapsulates the scaling factor and the offset, making the scaling calls in the main program look natural. The definition of the scaling transform class can further be modified so that it implements nonlinear scaling (e.g. logarithmic). We then use the scaling class in several exercises to reinforce student's understanding of the concept.

The Pumpkin Lab: Drawing simple scaled images

This laboratory exercise is the first introduction to picture scaling and uses learning by example approach. The students are given a program that draws an image of a pumpkin (jack-o-lantern) using graphics calls to paint rectangles, ovals, and lines. The image is defined using an anchor point that determines its position and a scale factor that determines its size. All of the pumpkin features are programmed relative to the anchor point and the scale. Before doing any programming in the lab, students explore drawing the pumpkin image in various scales and at various locations on the screen. Using the *draw grid* option, students can also determine the relative coordinates of different features.

Students are asked to enhance the pumpkin image with additional features that will scale properly with the original image. After students complete this laboratory, they are asked to design an entire scalable image of their own. The images produced are often quite spectacular and students feel very positive about their success early in the first course.

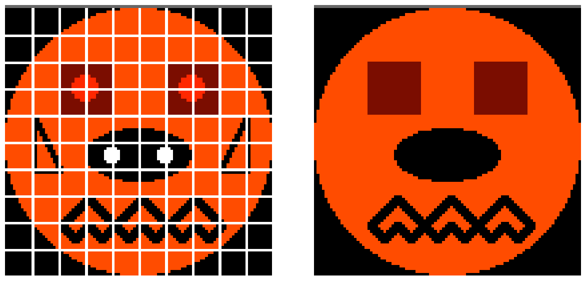
Figure 1. Pumpkin Solution and Student Base Picture

Hidden scaling

In an early Loops Lab, students animate a moving ball whose RGB color values are proportional to its physical coordinates. In addition, as the ball moves, a changing sound tone is produces whose amplitude and frequency is also influenced by its position (Figure 2).. Each of these effects is an instance of scaling.

In the Piano Keyboard Lab, students draw a keyboard on the screen and play the notes by selecting a key with the mouse. Here the mouse location must be scaled to determine which key has been clicked and what note (frequency) should be played.

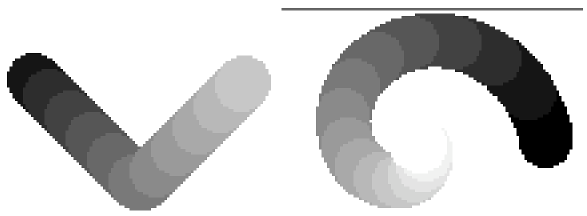

Figure 2. Bounce with shading, Spiral with fading

In almost all applications of random numbers, the numbers supplied by the random number generator must be scaled to fit into a particular range before being used.

Exercises on scaling in one dimension only

To continue the theme of scaling, there are two laboratories which consider scaling in one dimension only. In the Sine Function Lab, students plot the function **`sin(x)`** for **`x`** in degrees between 0 and 360. In a 400 x 400 drawing window, only the vertical component needs to be scaled. In the Simple Array Lab, students are asked to make a bar chart from array data (data selected to be in the range between 0 and 400). Here the focus is on horizontal scaling to define the bars and the spaces between them. The array plot function designed in this lab is then used as a visualization tool in several subsequent labs on sorting. In both laboratories, the emphasis is on direct methods so that students will understand the issues correctly.

Image enhancement: Scaling the color

Several simple image enhancement algorithms are based on scaling the range of color shade values so they would be distributed more evenly across the available spectrum. Our students implement these algorithms to enhance grayscale images from the Viking Mars mission [2]. If all pixel shades fall within the range **`(minx, maxx)`** out of available 256 shades, the linear enhancement just replaces each pixel shade of original shade value **`x`** by a pixel shade **`(x-minx)*256/(maxx-minx)`**. The histogram equalization algorithm is a bit more complex, but again applies a scaling transformation to every pixel value.

In another lab students were asked to display a contour map of a function in two variables representing each range of values with a different color. This method is used in scientific visualization of data such as CAT scan images or satellite photographs of the Earth.

Morphing images

In a lab on simple array manipulation, students perform simple morphing of one polygonal figure into another. They use lerping on the coordinates of corresponding points in the start and final polygons to produce the "in-between" coordinates. This can be used either to create an animation (transition between two positions of a running person) or to represent morphing of one image into another (fish to frog).

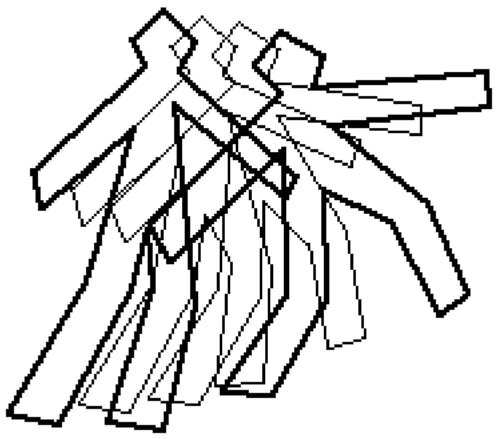

Figure 3. Running Robert

Working with bit map images, we can morph one image into another - first scaling the original pictures to the target rectangle, then lerping the corresponding grayscales or colors. Once students know how to get mouse input and maintain ordered lists of data, they can design and build a moderately sophisticated morphing program. In the example shown (Figure 4) the user matched eight points in the Mona picture with eight points in the Fell picture (e.g. eyes, tip of nose, chin). These points divide each picture into 81 rectangles and corresponding pairs of rectangles were morphed to form the central image. Since rectangles are blended (instead of polygonal or spline bounded regions as in a commercial morph program) the mathematics remains simple but the results are striking. As there are many bells and whistles that can be added or left out, morphing makes a good software design project at the end of CS1.

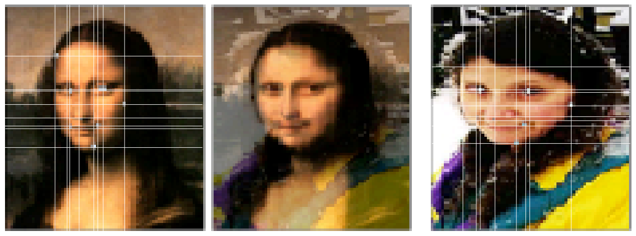

Figure 4. Bitmap piece wise morphing

Heapsort

While all sorting algorithms can easily sort any subrange of an array, the basic format of heapsort expects the indices to range from **`1`** to **`maxindex`**. In our previous labs that included a display of the array values as a bar chart, the driver's call of a sorting function included as arguments the array reference, as well as the low and high index of the subrange to be sorted. This was quite useful for implementing quicksort. In trying to reuse the code for heapsort, we needed to look for an elegant method that will retain the original driver and that will not make the heapsort code messy and unreadable.

Our solution was to use two conversion functions:

`hindex(ai, lower)` returns the heap index of the array item at location **`ai`**

`aindex(hi, lower)` returns the array index corresponding to the heap item at location **`hi`**.

To keep the heapsort code readable, we create three helper functions:

```
long LeftChild(long i, long lower)
long RightChild(long i, long lower)
long Parent(long i, long lower)
```

For example **`LeftChild`** function is implemented as:

```
inline
long LeftChild(long i, long low){
  return aindex(2*hindex(i, low),
low);}
```

The heapsort code then works in the standard way, using indices between **`1`** and **`maxindex`**. Students learn to deal with a shift in the frame of reference in a disciplined way.

Scaling class as a function object

Once students are familiar with scaling we introduce a scaling transform class. Students learn how to define this class and use it in several settings.

Scaling and tiling revisited: scaling transform function object:

This laboratory on scaling images and tiling the graphics window is given about midway through the year after students have had an initial introduction to both objects and templates. We return to the problem of drawing a scaled image, but this time the scaling transform is encapsulated in a function object.

A function object is an instance of a class for which the function call **`operator()`** has been defined. The definition of a linear scaling transfer operator with offset **`offset`** and scale factor **`factor`** is quite simple:

```
   int operator() (double x) const
      {return (factor * x + offset);}
```

The two scaling parameters, **`offset`** and **`factor`** are member variables initialized by the constructor. Once a scaling transform object **`S`** is defined, it can operate on a value **`x`** using function call notation **`S(x)`** which is shorthand for the long-winded call **`S.operator()(x)`**. In other words, the function object **`S`** has the syntax and behavior of an ordinary function. In languages like Java that do not allow overloading of operators, this function can be given a meaningful name (e.g. **`Scale`**). If we then define **`Sx`** to be the scaling transform object for scaling along the x axis, the function call to transform value **`x`** will be **`Sx.Scale(x)`** - still a readable format.

In this lab students first define the scaling transform class, then use it to scale several images. In the first part of the completed program user can chose to display one of several images (a pumpkin, a snowman, a black or white oval, and a playing card - front or back). The user also selects the location of the image by specifying its bounding rectangle. In the second part, the program tiles the drawing window with two selected images in a checkerboard fashion. The user selects the number of rows and columns in the tiling. Students can investigate the behavior of a solution by running a compiled application, before they start writing code themselves.

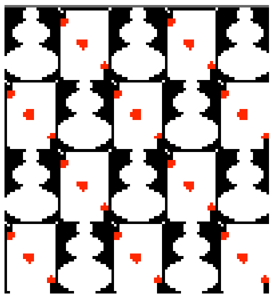

Figure 5. Tiling with snowmen and cards

Students are given the code that implements most of the user interface. Each image drawing is encapsulated in a function. Students are given a complete function that draws a pumpkin, as well as functions for drawing playing cards (bit map images) and colored ovals. Each image drawing is encapsulated in a function. The function signatures are identical, allowing the selection switch statement to pass the function name as a parameter.

Students need to first implement the scaling transform class. To make the scaling transform easier to use, it provides three alternatives for the constructor. The first two are default (**`offset = 0, scale factor = 1`**) and initialization of the **`offset`** and **`scale factor`** directly. The third alternative allow the user to specify the bounds of the real interval and the bounds of the display interval and computes the **`offset`** and **`scale factor`** from these values. When students complete this task, the pumpkin image and the playing cards will be displayed correctly.

Next students use the scaling class in several different contexts. First they add a function DrawGrid that displays a thin line grid with a given number of vertical and horizontal lines across an existing image. Now the scaling transform is used to determine the placement of the lines that are then scaled to fit the picture size. Finally, students implement a function that implements the checkerboard tiling of the graphics window with two images.

The Simple Plot Lab

In this lab students learn how to plot an arbitrary function in any given interval with user selected number of segments. They define a scaling transform object for the horizontal axis, compute the minimum and the maximum value of the function on the given interval, and define the scaling transform object for the vertical axis. Once the scaling in both directions has been defined, the plotting is straightforward.

The Complex Lab: 2D plotting

This lab has a number of pedagogical goals and has been described elsewhere [4]. The important design issue related to scaling is the fact that it uses a toolkit Plot2D that plots any list of (x, y) value pairs (points in a two-dimensional plane) in a window of a given dimensions. Students learn that it is possible and desirable to build small toolkits that perform simple, often repeated tasks. This is a very important design method. Students see the evolution of the design decisions and the benefits of creating a well designed toolkit.

The Fractal Grammar Lab

This lab has also been described in another paper. Here students use the now familiar scaling transform class to make sure the fractal image they create can be displayed within the graphics window. it is necessary to traverse the image twice - first to determine the real world coordinate bounds that are used to define the scaling transform, the second time to actually draw the image.

Conclusion and Acknowledgments

Students need to see examples of good design early in their study of programming. They also need to see how the design methods evolve from using simple direct manipulation to building toolkits. We presented a suite of exercises for introductory computer science courses that use scaling in increasingly sophisticated way and described the design lessons students learn.

We would like to acknowledge Erich Neuwirth, who suggested (and implemented in a spreadsheet) the exercise that generates a caricature.

References

1. Fell, H., and Proulx, V. K., *Exploring Martian Planetary Images: C++ Exercises for CS1*, SIGCSE Bulletin, February 1997, Vol 29(1), 30-34.
2. Foley, J. D., vanDam, A., Feiner, S. K., Hughes, J. F., *Computer Graphics, Principles and Practice, Second Edition in C*, Reading, MA, Addison-Wesley.
3. Proulx, V. K., *Recursion and* Gra*mmars for CS2* , Proceedings, Integrating Technology into Computer Science Education (ITiCSE 97), Uppsala, Sweden, June 1997, (ACM Press), 74-76.
4. Proulx, V. K., Rasala, R., and Fell, H., *Foundations of Computer Science: What Are They and How Do We Teach Them?*, SIGCSE Bulletin, June 1996, Vol 28 Special Issue, 42-48.
5. Raymond, ed., *The New Hacker's Dictionary - 3rd Edition*, Cambridge, MA, MIT Press, 1996. ISBN 0-262-68092-0 or its on-line version:
   http://locke.ccil.org/jargon/, "Jargon File"
6. Russ, J. C., *The Image Processing Handbook, Second Edition*, Boca Raton, FL, CRC Press, 1995.