Multiresolution Curve Representations

Jacob Marttinen
UWID: 20313145
User ID: jemartti
Assignment 5
Course: CS488, Section 001
Instructor: Abdallah Rababah
University of Waterloo

April 2, 2012

Abstract

The purpose of this project was to implement a simple multiresolution curve editor (based off Adam Finkelstein's 1996 thesis). This paper describes the theory behind such a representation, the implementation of this project, the project objectives, a manual, a checksum, and a list of references used in the course of development.

1 Implementation

My multiresolution curve model editor was based off the graduate level thesis of Adam Finkelstein (University of Washington, 1996).

The multiresolution curve representation, which is based on wavelets, conveniently supports a variety of operations: smoothing a curve, editing the overall form of a curve while preserving its details, changing its fine details while maintaining its overall sweep, and approximating a curve for scan conversion.

1.1 Theory

Most computer graphics applications represent a function using a discrete set of samples. Images are not, in general, represented exactly. For curves, these samples are typically used to reconstruct a function by treating them as coefficients for a set of basis functions. Given the samples, one can then evaluate the function everywhere, not just at the locations of the samples.

The typical way for arranging these samples for curves consists of deploying them at even intervals. This is called a uniresolution representation: a finite, discrete set of samples spaced uniformly over the domain of the function they represent. Uniresolution representations are easy to understand and implement, usually have a predictable size, and often make use of implicit data structures so there is no extra storage overhead. However, there are many advantages to using a multiresolution representation for smooth, open curves based on wavelets instead of a uniresolution representation.

1.1.1 General Formulation

Consider our curve C^n, expressed as a column vector of control points $[c_1^n, ..., c_m^n]^T$. Suppose we wish to create a low-resolution version C^{n-1} of C^n with a fewer number of samples m'. This process can be expressed as a matrix equation

$$C^{n-1} = A^n C^n$$

where A^n is an $m' \times m$ matrix.

Since C^{m-1} contains fewer samples than C^n, we can see that some data is lost in the process. However, if A^n is chosen appropriately, it is possible to capture the lost detail as another signal D^{n-1} with $m - m'$ samples, computed by:

$$D^{n-1} = B^n C^n$$

where B^n is an $(m - m') \times m$ matrix, which is related to matrix A^n. This process of splitting C^n into C^{n-1} and D^{n-1} is called decomposition.

Note that C^{n-1} and D^{n-1} together have the same amount of information as C^n. If we chose A^n and B^n correctly, we can recover the original set of points by using another pair of matrices P^n and Q^n as follows:

$$C^n = P^n C^{n-1} + Q^n D^{n-1}$$

This process is called reconstruction.

The procedure for splitting C^n into C^{n-1} and D^{n-1} can be applied recursively to the new signal C^{n-1}. Thus, we can express the original signal as a hierarchy of lower-resolution signals $C^0, ..., C^{n-1}$ and details $D^0, ..., D^{n-1}$. This recursive process is known as a filter bank. Since the original curve C^n can be recovered from the sequence $C^0, D^0, D^1, ..., D^{n-1}$, this sequence can be thought of as a transform of the original signal known as a *wavelet transform*. Note that the total size of the transform is the same size as the original signal, so no extra storage is required. Also, if we construct our filter matrices to be sparse, we can calculate our filter bank very quickly (even in $O(m)$ time, where m is the number of points in our curve).

To see how to construct these filters, we associate with each signal C^n a function $f^n(u)$ with $u \in [0, 1]$:

$$f^n(u) = \phi^n(u) C^n,$$

where $\phi^n(u)$ is a row matrix of our basis (scaling) functions. We must require our scaling functions to be *refinable*: for all j in $[i, 1]$ there must exists an $m \times m'$ matrix P^j such that

$$\phi^{j-1} = \phi^j P^j$$

Next, let V^j be the linear space spanned by the set of scaling functions. Choosing an inner product for ϕ^j allows us to define W^j as the orthogonal complement of V^j in V^{j+1}. This is, the space W^j whose basis functions $\psi^j = [\psi^j_1(u), ..., \psi^j_{m-m'}(u)]$ are such that ϕ^j and ψ^j together form a basis for V^{j+1}. The basis functions in the space W^j are called wavelets. We can now construct Q^j as the matrix that satisfies:

$$\phi^{j-1} = \psi^j Q^j$$

We can concatenate these matrices together to find:

$$[\phi^{j-1} | \psi^{j-1}] = \phi^j [P^j | Q^j]$$

Finally, A^j and B^j are formed by the inverse relation:

$$[\phi^{j-1} | \psi^{j-1}] \left[\frac{A^j}{B^j}\right] = \phi^j$$

We can also calculate A^j and B^j (albeit inefficiently) by the identity:

$$\left[\frac{A^j}{B^j}\right] = [P^j | Q^j]^{-1}$$

We need to make several decisions before we can construct the wavelets for our specific basis:

1. Choose the scaling functions $\phi^j(u)$ for all levels j. This choice determines the synthesis filters P^j.

2. Select an inner product for any two functions f and g in V^j. This choice determines the orthogonal complement spaces W^j.

3. Select a set of wavelets $\psi^j(u)$ that span W^j. This choice determines the synthesis filters Q^j.

1.1.2 Multiresolution endpoint-interpolation B-splines

We use the Mansfield, de Boor, Cox recursion for determining our B-spline basis functions. The i-th B-spline basis function ${}^dB_i$ of degree d over knot sequence $u_1, u_2, ..., u_n$ may be found by the following recursion:

$$ {}^dB_i(u) = \frac{u - u_i}{u_{i+d} - u_i}^{d-1} B_i(u) + \frac{u_{i+d+1} - u}{u_{i+d+1} - u_{i+1}}^{d-1} B_{i+1}(u) $$

As every recursion needs a base case, our 0-degree B-splines are:

$$ {}^0B_i(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} $$

We will now determine the multiresolution representation for endpoint-interpolating uniform cubic B-splines.

Step 1: Choose the scaling functions

Our scaling functions are simply our basis functions, found above.

$$ \phi^j(u) = \begin{bmatrix} {}^3B_1(u) & {}^3B_2(u) & {}^3B_3(u) & ... & {}^3B_{2^j+3}(u) \end{bmatrix} $$

Step 2: Select an inner product

We need an inner product for any two functions f and g in V^j. We use the standard form:

$$ \langle f\ g \rangle = \int f(u)\ g(u)\ \mathrm{d}u $$

Because our basis functions can be expressed as simple cubic polynomials, we can compute their inner products symbolically.

Step 3: Select a set of wavelets

Finally, we find a set of wavelets $\psi^j(u)$ that span W^j. We use some new notation here: Given two row vectors of functions X and Y, let $[\langle X\ Y \rangle]$ be the matrix whose kl-th entry is the inner product $\langle X_k\ Y_l \rangle$. Since, by definition, scaling functions and wavelets at the same level j are orthogonal, we have

$$ [\langle \phi^j | \psi^j \rangle] = [\langle \phi^j | \phi^{j+1} \rangle] Q^{j+1} = 0 $$

Thus, the columns of Q^{j+1} span the null space of $[\langle \phi^j | \phi^{j+1} \rangle]$. Since the scaling functions are piecewise-polynomial functions, it is straightforward to compute the entries of the matrix of inner products symbolically. Next, we find a basis that spans the null space of this matrix, also a straightforward task.

1.1.3 Linear-time filter-bank algorithm

We can in fact compute our filter bank in linear time. Our synthesis filters P^j and Q^j have a banded structure, allowing reconstruction in $O(m)$ time. However, A^j and B^j are dense. To overcome this issue, the author of the paper presents a cool trick. Let I^j and B^j be the inner product matrices $[\langle \phi^j | \phi^j \rangle]$ and $[\langle \psi^j | \psi^j \rangle]$, respectively. We can then write the following equations:

$$ \begin{aligned} I^{j-1} C^{j-1} &= (P^j)^T I^j C^j \\ J^{j-1} D^{j-1} &= (Q^j)^T I^j C^j \end{aligned} $$

Since these are all banded matrices, the right-hand sides can be computed linearly. We then have two band-diagonal systems of equations, which can be solved in linear time using LU decomposition. Thus, if we have our precomputed matrices I^j, J^j, P^j, and Q^j (for all j), we can fully decompose C^j in linear time.

Also note we have the following relations:

$$\begin{aligned} I^j &= (P^{j+1})^T I^{j+1} P^{j+1} \\ J^j &= (Q^{j+1})^T I^{j+1} Q^{j+1} \end{aligned}$$

1.1.4 Smoothing

Our above technique works great for $m = 2^j + 3$ control points (for any nonnegative integer j). For every noninteger j however, we have a problem. The simple solution given in the Finkelstein paper is to define a fractional level curve $f^{j+t}(u)$ for some $0 \leq t \leq 1$ in terms of a linear interpolation between its two nearest integer-level curves $f^j(u)$ and $f^{j+1}(u)$ as follows:

$$\begin{aligned} f^{j+t}(u) &= (1-t)f^j(u) + tf^{j+1}(u) \\ &= (1-t)\phi^j(u)C^j + t\phi^{j+1}(u)C^{j+1} \end{aligned}$$

These fractional-level curves allow for continuous levels of smoothing.

1.1.5 Editing

Multiresolution analysis allows for two very different kinds of curve editing. If we modify some low-resolution version C^j and then add the detail back into get C^n, we will have modified the overall sweep of the curve. On the other hand, if we modify the set of detail functions but leave the low-resolution versions intact, we will have modified the character of the curve without affecting its overall sweep.

Editing the curve at an integer level of the wavelet transform is simple. We edit the points as normal, and then the edited sweep carries through upon reconstruction. However, at levels between integer levels, editing becomes much more difficult. We can get a fractional relation for C^{j+t} as follows:

$$\begin{aligned} C^{j+t} &= (1-t)P^{j+1}C^j + tC^{j+1} \\ &= P^{j+1}C^j + tQ^{j+1}D^j \end{aligned}$$

In order for the portion of the curve edited to depend on t as in the fractional relation above, the system will have to automatically move some of the nearby control points when c_i^{j+t} is modified. The distance that each of these control points is moved is inversely proportional to t.

This fractional-level editing can then be extended in a straightforward fashion to accommodate direct manipulation of the curve (where the user tugs on the smoothed curve directly rather than on its defining control points).

1.1.6 Extensions

One proposed extension to this representation includes textured strokes. Properties such as colour, thickness, texture, and transparency may simply be considered extra dimensions in the data associated with each control point. Then, most of the machinery for multiresolution editing should be applicable to such curves. A few other proposed extensions include handling discontinuities and sparse representations.

1.2 Coded Implementation

This project was implemented using C++ and OpenGL. Eclipse was used as an IDE, with g++ as the compiler.

A few external libraries were used:

- Eigen, a C++ template library for linear algebra (matrices, vectors, numerical solvers, and related algorithms). I used this for basic matrix operations, as well as for LU decomposition. As Eigen is a pure template library defined in the headers, I just included the Eigen folder in my source.

- Simple OpenGL Image Library (SOIL) is a tiny C library primarily used for loading textures into OpenGL. I used this for importing the image to be traced. libSOIL.a is in /lib, SOIL.h is in /includes, and the library is linked in the makefile accordingly.

The very core of the project is based on the skeleton code for Assignment 3. However, very little code was reused from prior assignments.

I heavily use two data structures in my code:

- The first is simply called point, with double members x, y, and z. This is used to represent each control point.

- The other is called line. This contains the active state information for each individual line, as well as the corresponding control points. The main members are a vector of points, the current resolution level information, and a vector of the D^i matrices.

The main function contains a vector of lines (each line in the state of the currently active application), as well as an integer indicating the currently active line.

1.2.1 User Interface

I implemented a very rudimentary gtkmm user interface. Mouse click+drag is used to update colours and widths, mouse selection is used for editing control points, mouse buttons are used to change resolution, and there are toggles for various graphical features.

The only item of note is the slider along the bottom of the screen. This controls the continuous multiresolution curve analysis. As the slider moves, the resolution of the currently active line is adjusted accordingly.

1.2.2 Cubic B-spline curves

The main implementation used for curve approximation is end-point interpolating uniform cubic B-splines. As noted above, we use the Mansfield, de Boor, Cox recursion for determining our B-spline basis functions. The i-th B-spline basis function ${}^dB_i$ of degree d over knot sequence $u_1, u_2, ..., u_n$ may be found by the following recursion:

$$ {}^dB_i(u) = \frac{u - u_i}{u_{i+d} - u_i}^{d-1} B_i(u) + \frac{u_{i+d+1} - u}{u_{i+d+1} - u_{i+1}}^{d-1} B_{i+1}(u) $$

As every recursion needs a base case, our 0-degree B-splines are:

$$ {}^0B_i(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} $$

The main B-spline function prototype is

void bspline(int n, int d, int r, std::vector <point> ipts, point *opts),

where n is the number of knots, d is the degree, r is the resolution of the curve (number of steps to be calculated), ipts are the control points, and opts are the calculated points on the curve. bspline first calculates the knot vector. This is uniform, with even spacing between each knot. However, if endpoint interpolation is on, we set the first/last four knots equal to each other respectively. We then call compute_point for each time step to get the points to be drawn.

compute_point essentially implements the calculation:

$$S(u) = \sum_{i=0}^{n-d-2} P_i^d B_i(u) \text{ for } u \in [u_d, u_{n-d-1}]$$

with each basis function computed by the recursive function blend using the aforementioned recursion relation.

1.2.3 Curve Drawing

The OpenGL drawing code is fairly straightforward: the window is set up, the control points are transformed to the proper coordinate system, the b-spline is calculated, and then the points are drawn. This drawing code loops on all the lines in the stored vector of lines, drawing each one individually as a GL_LINE_STRIP. The control points and the curve hull for each line are also drawn. The drawing of each of these structures (points, curve, hull) can be toggled in the application menu.

This drawing code definitely has its inefficiencies. As the number of control points gets over a few hundred, the application response time lags considerably. This is due to me recalculating the spline points for every single line on every refresh, as well as the spline basis functions on each spline point calculation. This should be fixed, but I did not get the opportunity to resolve this issue.

1.2.4 Point Control

After many hours spent trying to implement an OpenGL version of picking (as in Assignment 3), I realized I had to cut my losses. I could not get the preferred picking algorithm working, so the implementation in the final version is simply a linear search over the points in the active curve. If the point is not found in the active curve, a linear search is performed on every other curve in the application state. If the point is found in one of the other curves, the corresponding curve is set to the active curve and a pointer to the picked point is stored. As a future improvement, more efficiency could certainly be gained here.

Points are input using a left-clink in POINTS mode. The resolution level is first brought up to the highest, we add the point to the end of the active line, and then we update the corresponding size values.

To delete a point, it must be picked. The resolution level is first returned to the highest. Then when we click with the middle mouse button, if we have selected a point, the point is removed from its corresponding line. Finally, we update the corresponding size state values.

To update a point, it also must be picked. When we select with the right mouse button, and if we have selected a point, we then drag the mouse to update the point's position in real-time. Upon release, the point is set with the final point value.

When we are finished with the current line, we can press enter to complete the line. This opens a new (empty) active line that can then be configured as desired. A (reasonably) unlimited number of lines are supported.

1.2.5 Multiresolution Analysis

The bulk of the development in this section lay in getting the proper matrices set up. After calculating the general forms externally, I created functions for getting the specific matrices for our multiresolution calculations: get_pi(int i), get_qi(int i), get_ii(int i), get_ji(int i), get_ai(int i), and get_bi(int i). These functions take in the required integer level (number of points $= 2^j + 3$, where j is the level) and return the required matrix.

I was able to implement the linear time algorithm for integer reconstruction described in Section 1.1.3. However, I ran into issues when attempting to implement a linear time decomposition algorithm. When control points increased past $j = 6$, the curve went wonky. I believe this had to do with the extremely small fractions created for Q^j, but I was not able to resolve the problem. Instead, I implemented the slower algorithm described in Section 1.1.1.

Integer multiresolution level switching was implemented successfully using the algorithm described in Section 1.1.4. As I was not able to get continuous curve editing working successfully, I left in functionality to allow for curve editing on exactly integer levels. This functionality is activated in RESOLUTION mode using left/right mouse clicks.

Continuous multiresolution level switching was also implemented successfully using the linear interpolation algorithm described in Section 1.1.4. There are minor jumps as the level passes integer values but this is expected, as two-point linear interpolation will have corners when approximating a smooth curve. Using the slider bar at the bottom of the window activates this functionality.

1.2.6 Colour/Width

Unfortunately, I did not get to complete this area to my satisfaction. I simply allowed for the setting of the colour and width at the beginning, halfway point, and endpoint of each line, which then get linearly interpolated the values across the length of the curve using the following relation:

$$v(t) = t^2 P_1 + 2(1-t)tP_2 + (1-t)^2 P_3$$

As described in Section 1.1.6 above, there is a much better way to implement colour and width across the multiresolution lines. This is an area for future improvement.

1.2.7 Application

The overall goal of this project was to create an application that supports loading in an image, tracing curves over the image, then saving the traced control points as an approximation of the image.

To load an image, the SOIL library is used. I simply load the image to a GLuint, map the input image to a GL_QUAD, map the quad to the window dimensions, and then allow drawing on top of the image. The visibility of the image can be toggled using the application menus.

The path to the desired (png) image can be passed as a command line argument upon starting the application. If no path is passed to the application, the application loads /data/img.png by default.

A very rudimentary save/load system has also been implemented. When saving, the control points and line state for each line are directly stored in a text file named "points." A set of lines can be loaded in a similar manner from a compatible text file with the same name.

1.3 Code Map

The bulk of the code is in viewer.cpp/hpp:

- The point/line structures are defined at the top of viewer.hpp
- Save/load is implemented by the save()/load() functions
- Add/remove/picking of points are implemented by add_point(int x, int y), remove_point(std::vector<point>::iterator it), and picking(int x, int y)
- B-spline calculations are performed via bspline(...), blend(...), and compute_point(...)
- Resolution changes are performed via level_change(double newl), level_down(), and level_up()
- The matrices for multiresolution calculations are calculated/formed by get_pi(int i), get_qi(int i), get_ii(i), get_ji(i), get_ai(i), and get_bi(i)

Various other pieces are as follows:

- The user interface is implemented in appwindow.cpp/hpp.
- The Eigen library is included in ./Eigen
- The SOIL library is included in ./include and ./lib

2 Objectives

Name:

UserID:

StudentID:

1. Create a user interface for the multiresolution curve editor
2. Implement the drawing of endpoint-interpolation cubic B-spline curves from input data points
3. Allow for dragging the mouse to define a line, and implement functionality to edit the curve by moving control points
4. Implement adaptive curve sampling, ensuring a smooth curve
5. Extend this smoothing to continuous levels (multiresolution) using wavelet analysis
6. Extend the editing of the curves to continuous levels (multiresolution, allowing editing of both the sweep and the details of the curve)
7. Extend the editing of the curves to allow direct manipulation of a curve
8. Allow for the editing of colour and size dimensions
9. Allow for multiple curves to be drawn, allowing for the tracing of an image
10. Demonstrate the drawing functionality by approximating a complicated drawing using at least 20 curves and multiple thicknesses/colours

Declaration: I have read the statements regarding cheating in the CS488/688 course handouts. I affirm with my signature that I have worked out my own solution to this assignment, and the code I am handing in is my own.

Signature:

3 Manual

The application can be run by calling ./MultiresolutionCurves from the A5 folder. A command line option specifying the path to the background image can be given at runtime. If none is specified, the application will automatically load img.png from the ./data folder.

The application defaults to POINTS mode. To add a point to the end of the currently active line, use the left mouse button. To delete a point from any line, use the middle mouse button. To edit the position of any point, click and drag with the right mouse button. Note that any point interaction will change the point's parent line to the currently active line. To complete the currently active line and start a new line, press enter.

Switching to RESOLUTION mode allows for the integer level switching of the curve resolution. Click the left mouse button to drop down a level. Click the right mouse button to jump up a level. Click the middle mouse button to drop down to the highest possible level integer level (if the current fractional level is higher than this level).

Switching to any of the COLOUR modes allows for the editing of the colour at three specified points (1 = start of the line, 2 = midpoint of the line, 3 = endpoint of the line). These colours are linearly interpolated across the rest of the line. To adjust the colour, click and drag in the x direction. The (left, middle, right) mouse buttons correspond to changes in the (R, G, B) colour spaces.

Switching to the WIDTH mode allows for the editing of the line width at three specified points. These widths are linearly interpolated across the rest of the line. To adjust the width, click and drag in the x direction. The (left, middle, right) mouse buttons correspond to changes in the point at the (start of the line, midpoint of the line, endpoint of the line).

The OPTIONS menu allows for the toggling of application state settings. These options include the disabling of endpoint interpolation, hiding the display of the control points, curve hull, and the curve itself, enabling the display of the background image, and disabling the effect of the colour/width modifications.

Save/load functionality is implemented under the APPLICATION menu. SAVE simply saves the current application state to a file called "points" in the current directory. LOAD clears the current state and loads the state specified in the "points file" in the current directory (if it exists).

CLEAR (reset all current state, clearing all control points and lines) and QUIT functionality is also implemented in the APPLICATION menu.

Other notes (shortcomings):

- I was not able to get continuous multiresolution curve editing working. Direct curve manipulation also did not make it into this release.
- Colour and width functionality is there, but is not implemented in the most optimal fashion.
- B-spline calculation works fine, but should be more efficient.
- I was not able to get a linear time filter-bank working.
- Picking should be implemented with a more efficient algorithm.
- There are minor jumps in the continuous multiresolution curve smoothing, but this is expected.
- The user interface is rudimentary, and could do with improvement.

4 Checksum

sum is: /usr/bin/sum
2012-04-02 09:18 Checksum for A5 for jemartti on gl20.student.cs Page 1
A5:
total 6796
25483469 drwxrwx— 9 jemartti cs488 4096 2012-04-02 09:18 ../
15508788 drwxrwx— 5 jemartti cs488 4096 2012-04-02 09:18 src/
37203323 -rw-r–r– 1 jemartti jemarttinen 30848 2012-04-02 09:14 screenshot02.png
15508781 drwxrwx— 4 jemartti cs488 4096 2012-04-02 09:14 ./
37203322 -rw-r–r– 1 jemartti jemarttinen 34243 2012-04-02 09:12 screenshot01.png
37203313 drwxrwxr-x 2 jemartti jemarttinen 4096 2012-04-02 09:09 data/
53287071 -rwxr-xr-x 1 jemartti jemarttinen 6819864 2012-04-02 09:03 MultiresolutionCurves*
37203320 -rw-r-x— 1 jemartti jemarttinen 2247 2012-04-02 08:56 README*
37203312 -rw-r—— 1 jemartti jemarttinen 2247 2012-04-02 08:45 README
A5/src:
total 104
21714043 drwxrwxr-x 2 jemartti jemarttinen 4096 2012-04-02 09:18 Eigen/
15508788 drwxrwx— 5 jemartti cs488 4096 2012-04-02 09:18 ./
15508781 drwxrwx— 4 jemartti cs488 4096 2012-04-02 09:14 ../
53287072 -rw-r—— 1 jemartti jemarttinen 34621 2012-04-02 09:03 viewer.cpp
53287073 -rw-r—— 1 jemartti jemarttinen 4143 2012-04-02 08:53 viewer.hpp
53287065 -rw-r—— 1 jemartti jemarttinen 4679 2012-04-02 08:53 appwindow.cpp
53287066 -rw-r—— 1 jemartti jemarttinen 861 2012-04-02 08:52 appwindow.hpp
53287067 -rw-r—— 1 jemartti jemarttinen 427 2012-04-02 08:52 main.cpp
53287069 -rw-r–r– 1 jemartti jemarttinen 727 2012-04-01 23:58 Makefile
53287059 drwxrwxr-x 2 jemartti jemarttinen 4096 2012-04-01 23:01 includes/
53287061 drwxrwxr-x 2 jemartti jemarttinen 4096 2012-04-01 23:01 lib/
53287064 -rw-r—— 1 jemartti jemarttinen 9474 2012-03-26 19:44 algebra.hpp
53287063 -rw-r—— 1 jemartti jemarttinen 3308 2012-03-26 19:44 algebra.cpp
A5/src/Eigen:
total 92
21714043 drwxrwxr-x 2 jemartti jemarttinen 4096 2012-04-02 09:18 ./
15508788 drwxrwx— 5 jemartti cs488 4096 2012-04-02 09:18 ../
53014115 -rw-r–r– 1 jemartti jemarttinen 304 2012-02-10 14:52 Array
21714044 -rw-r–r– 1 jemartti jemarttinen 750 2012-02-10 14:52 Cholesky
53014116 -rw-r–r– 1 jemartti jemarttinen 607 2012-02-10 14:52 CMakeLists.txt
53014102 -rw-r–r– 1 jemartti jemarttinen 12105 2012-02-10 14:52 Core
53014106 -rw-r–r– 1 jemartti jemarttinen 122 2012-02-10 14:52 Dense
53014117 -rw-r–r– 1 jemartti jemarttinen 37 2012-02-10 14:52 Eigen
53014118 -rw-r–r– 1 jemartti jemarttinen 2678 2012-02-10 14:52 Eigen2Support
53014119 -rw-r–r– 1 jemartti jemarttinen 1150 2012-02-10 14:52 Eigenvalues
53014110 -rw-r–r– 1 jemartti jemarttinen 1646 2012-02-10 14:52 Geometry
53014112 -rw-r–r– 1 jemartti jemarttinen 621 2012-02-10 14:52 Householder
53014114 -rw-r–r– 1 jemartti jemarttinen 686 2012-02-10 14:52 Jacobi
53014121 -rw-r–r– 1 jemartti jemarttinen 753 2012-02-10 14:52 LeastSquares
53014120 -rw-r–r– 1 jemartti jemarttinen 955 2012-02-10 14:52 LU
17615439 -rw-r–r– 1 jemartti jemarttinen 853 2012-02-10 14:52 QR
20509272 -rw-r–r– 1 jemartti jemarttinen 637 2012-02-10 14:52 QtAlignedMalloc
20509274 -rw-r–r– 1 jemartti jemarttinen 1891 2012-02-10 14:52 Sparse
20509275 -rw-r–r– 1 jemartti jemarttinen 1513 2012-02-10 14:52 StdDeque
20509276 -rw-r–r– 1 jemartti jemarttinen 1446 2012-02-10 14:52 StdList
20509277 -rw-r–r– 1 jemartti jemarttinen 1519 2012-02-10 14:52 StdVector
20509273 -rw-r–r– 1 jemartti jemarttinen 788 2012-02-10 14:52 SVD
A5/src/includes:
total 24
15508788 drwxrwx— 5 jemartti cs488 4096 2012-04-02 09:18 ../
2012-04-02 09:18 Checksum for A5 for jemartti on gl20.student.cs Page 2
53287059 drwxrwxr-x 2 jemartti jemarttinen 4096 2012-04-01 23:01 ./
53287060 -rw-r–r– 1 jemartti jemarttinen 15545 2008-07-07 18:13 SOIL.h
A5/src/lib:
total 96
15508788 drwxrwx— 5 jemartti cs488 4096 2012-04-02 09:18 ../
53287061 drwxrwxr-x 2 jemartti jemarttinen 4096 2012-04-01 23:01 ./
53287062 -rw-r–r– 1 jemartti jemarttinen 84916 2012-04-01 22:30 libSOIL.a
A5/data:
total 264
15508781 drwxrwx— 4 jemartti cs488 4096 2012-04-02 09:14 ../

37203313 drwxrwxr-x 2 jemartti jemarttinen 4096 2012-04-02 09:09 ./
37203318 -rw-r–r– 1 jemartti jemarttinen 6728 2012-04-02 01:48 pointsatv
37203315 -rw-r–r– 1 jemartti jemarttinen 141917 2012-04-02 01:41 atv.png
37203319 -rw-r–r– 1 jemartti jemarttinen 14450 2012-04-02 01:38 points
37203317 -rw-r–r– 1 jemartti jemarttinen 83274 2012-04-01 23:08 img.png
A5
A5/data
A5/data/atv.png 17887 139
A5/data/img.png 54227 82
A5/data/points 09680 15
A5/data/pointsatv 37940 7
A5/MultiresolutionCurves 62556 6661
A5/README 42874 3
A5/README 42874 3
A5/screenshot01.png 39171 34
A5/screenshot02.png 25150 31
A5/src
A5/src/algebra.cpp 00294 4
A5/src/algebra.hpp 65153 10
A5/src/appwindow.cpp 29703 5
A5/src/appwindow.hpp 17033 1
A5/src/Eigen
A5/src/Eigen/Array 33180 1
A5/src/Eigen/Cholesky 54241 1
A5/src/Eigen/CMakeLists.txt 11256 1
A5/src/Eigen/Core 03117 12
A5/src/Eigen/Dense 53271 1
A5/src/Eigen/Eigen2Support 50449 3
A5/src/Eigen/Eigen 60386 1
A5/src/Eigen/Eigenvalues 17922 2
A5/src/Eigen/Geometry 14611 2
A5/src/Eigen/Householder 32620 1
A5/src/Eigen/Jacobi 44323 1
A5/src/Eigen/LeastSquares 33056 1
A5/src/Eigen/LU 20775 1
A5/src/Eigen/QR 39921 1
A5/src/Eigen/QtAlignedMalloc 34379 1
A5/src/Eigen/Sparse 28669 2
A5/src/Eigen/StdDeque 36311 2
A5/src/Eigen/StdList 46072 2
A5/src/Eigen/StdVector 42139 2
A5/src/Eigen/SVD 14614 1
A5/src/includes

A5/src/includes/SOIL.h 30876 16
A5/src/lib
A5/src/lib/libSOIL.a 12314 83
A5/src/main.cpp 10689 1
A5/src/Makefile 06731 1
A5/src/viewer.cpp 35027 34
A5/src/viewer.hpp 28719 5

References

[1] Multiresolution Curves. Finkelstein and Salesin, University of Washington 1994. *This PhD thesis describes a multiresolution curve representation, based on wavelets, that supports a variety of operations (smoothing, editing preserving details, approximation within any given error tolerance). Most of the project is based off of this paper.*

[2] Multiresolution Wavelets transform for Haar B-spline MR. University of Calgary 2008. *Although this .ppt presentation has no listed author, it was helpful in working through some of the details of the multiresolution algorithm.*

[3] Personal interview. S. Mann, University of Waterloo 2012. *Prof. Mann assisted me in understanding some of the more obscure portions of the Finkelstein paper.*

[4] Wikipedia - B-splines. 2012. *I used the B-spline approximation algorithm from the article.*