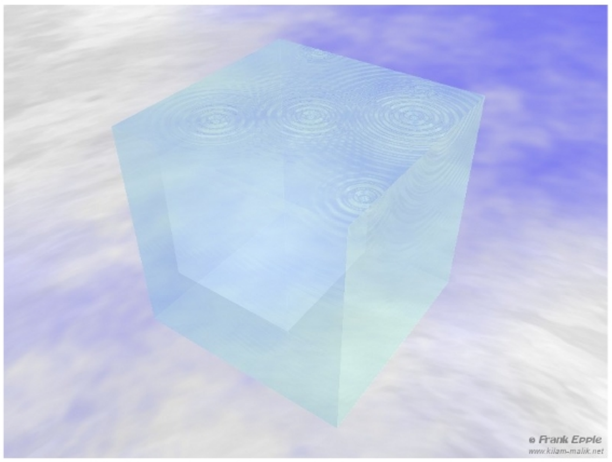
Computer Systems Lab Senior Research Project Posters 2007-2008 2nd Quarter vers.

TJHSST

May 8, 2008

Projects
6th Period Project Posters

Study of the Evolution of Organism Combination
2006-2007
TJHSST
Alexandria, Virginia
Nicholas Brown



If the environment were displayed this is what it would look like without the organisms.

Abstract

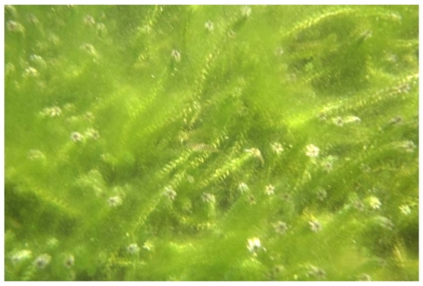
Simple single celled organisms face evolutionary pressures in a similar manner to multi-cellular organisms. In order for multi-cellular life to evolve the combination of cells to form a multi-cellular organisms must be advantagous to the individual cells. The objective of this study is to create a simple fluid environment and populate it with rudamentary predator and photosynthetic organisms.

Introduction

Algae is one of the simplest multicellular life forms that exists in nature. The photosynthetic agents in this simulation are designed to be similar to a single independant Algae cell and will hopefully evolve into a basic Algae like grouping.

Background

One of the most controversial and most studied areas of science to day is evolution. One side is attempting to disprove it and the other side seeks for it to be recognized as the well verified theory that it is. One of the ways to gather evidence in this debate is through the use of simulations like this one. I am seeking to model the evolution of single cell organisms into basic multicellular ones with a single cell type and a self maintaining and regenerating structure.



```
\begin{verbatim}
World::World(int sx, int sy, int sz, int l, bool b, int il, int ip, int miC, int maC, double opc)
{
        int SizeX = sx;
        int SizeY = sy;
        int SizeZ = sz;
        int lignt = l;
        bool loop = b;
        int initPhoto = il;
        int initPred = ip;
        int minCurrentSpeed = miC;
        int maxCurrentSpeed = maC;
        double opacity = opc;
        int[][][] environment = int[SizeX][SizeY][SizeY];
        ...
\end{verbatim}
```

The SizeX, SizeY, and SizeZ variables define the size of the environment. The light variable defines the base light levels and the opacity defines how much of this light is lost with each level. The Current speed variables define the variations in the water current speeds and the init**** vaiables define the initial populations of creatures.

So far there isn't any meaningfull output since I will need to write a short program to sift through the outputs and graph them or otherwise display them.

Defining an Organism Abstractly

The abstract definition of an organism is simply a numerical summary through variables and equations of the characteristics of that organism. For example the color of an organism is modeled through an equation representing the frequencies of light that it reflects and the amounts of these reflections. There are a large number of other variables that represent things such as organelles in a cell which themselves may be bacteria that entered the cell and then mutated. The difficult part of this is determining the correct level of detail to attempt to represent that strikes a balance between simplicity and accuracy.

Defining an Environment Abstractly

The world that the creatures inhabit must be very simple out of nesessity because it is the simgle most resource intensive object. The world is defined by the variables shown below.

A Dynamic Model of Human Populations

By Joshua Choi
TJHSST Computer Systems Lab 2007-2008

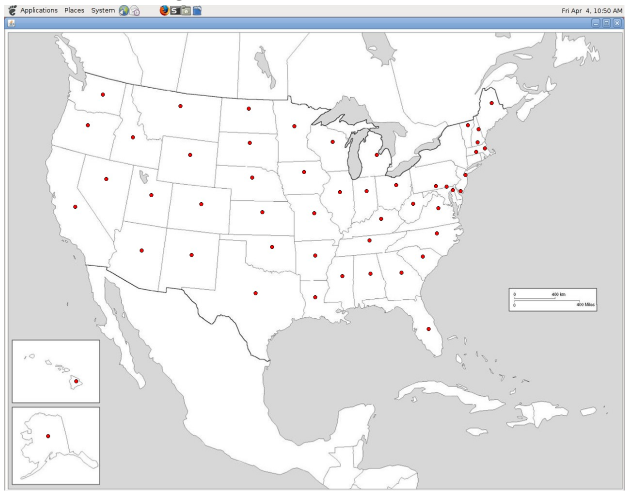
Abstract

The world is becoming better interconnected. As more and more people in developing countries seek to live in economically secure ones, less and less people stay in their own. This constantly changing flux of movement highlights just how important understanding the dynamics of human population is. This project attempts to analyze and understand the growths of a population and the migrations of people across the world. Through understanding how human populations develop, we can predict changes in the future.

Introduction

The human population of the world is now at 6 billion and counting. It is constantly growing, constantly moving. To even try to use human power to analyze all of this data would require thousands of people and thousands of hours of man power to complete. By using computers, we can drastically cut down on the man power needed.

This project can be useful for a great variety of problems. Most prominently, the US takes a census report every ten years. But every decade in between, the census department uses the data gathered to estimate population values. A dynamic model such as the one this project would achieve would be invaluable in assisting their efforts.



Background

"Human Population Dynamics Revisited with the Logistic Model: How Much Can Be Modeled and Predicted?" - The researchers attempt to analyze the problems and reliabilities of logistics curves use to model and predict human populations.
"A Stochastic Population Model Related to Human Populations" - Uses probabilistic factors in order to predict population data. They take into account factors of marriage, age, sex, and migrations.

Procedure

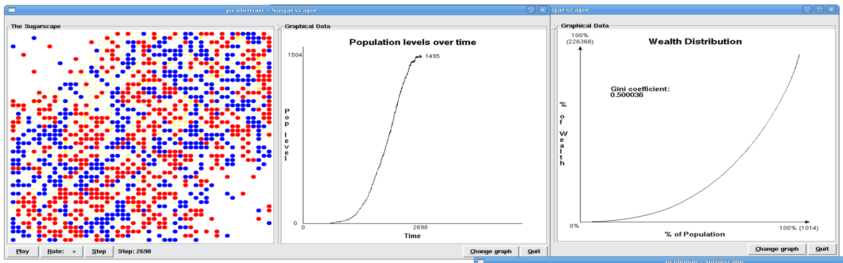
This project works by calculating a population growth rate value using population data for a certain group. First, it starts from the states level. The growth rate is calculated for historically rich states such as Virginia and New York. Then, it moves on to the entire US, in which it obtains the growth rate for each and every state. It takes those rates, displays a graphical representation of the growing population with it, and calculates the growth rate of population for the entire country. A future model of the world will also be created in a similar fashion. I also plan to add a migration display to my models. After the models have been made, we can move on to the testing.

Testing and Analysis

Testing at a basic level is done by comparing my project's predicted values with those of the US census. My predicted values for that year compared with the census' should be relatively close. The predicted values would then be used to craft graphs to compare with past growth rates.

If my model works correctly, I expect my graph to look like a J-shaped graph. I also plan to include in my analysis age pyramids to analyze reasons for a certain growth rate. Demographic transition graphs in conjunction with the age pyramids would be used to show what the change in growth rate means. And a somewhat final, conclusive graph would be used to show the changes in growth rate due to historical implications, such as the advent of the developed world.

Programming a New Sugarscape By Patrick Coleman



BACKGROUND

Growing Artificial Societies: Social Sciences from the Bottom Up written by Joshua M. Epstein and Robert Axtell and *Micromotives and Macrobehavior* by Thomas Schelling define Sugarscape and the Schelling segregation model. Tony Bigbee from George Mason University has written the Sugarscape in Java and his code will be used for reference along with the first book primarily. In the book by Axtell and Epstein Schelling's segregation model is mentioned and the Sugarscape is built with two separate groups (tribes) which combat against each other. The results should mirror those of the Sugarscape models in *Growing Artificial Societies*. However, once Schelling segregation is implemented with possibly more than two different colored populations the results will differ. In all likelihood only two groups will survive in the long run. The final results will be presented with screenshots of the running program along with graphs of relationships of variables. It will perform like previous Sugarscape models. *Growing Artificial Societies* and *Micromotives and Macrobehavior* are two books which are used as references to develop this project.

ABSTRACT

This project studies artificial societies, especially the Sugarscape and the Schelling segregation model. To implement the Sugarscape, a display of the sugar-filled environment with agents is outputted. The simulation allows agents to harvest sugar, consume sugar, die of starvation, and die of old age and allows the environment to grow back at a given rate. To implement the Schelling segregation model, two distinct groups of agents are added to the environment with a preference for neighbors of their own kind to determine the effects of the individual preferences on the society at large. The reasons these two projects are being implemented is because while both are often compared, the two models in their original forms have not been combined and analyzed in a single simulation. In addition to displaying the environment, graphs showing the population growth and wealth distribution are displayed. These graphs analyze what is occurring in the simulation. Seasons are implemented to analyze agent migration. The program code is broken up into files: a main file, an environment file, an agent file, a location file, a display file, and a simulation file. The conclusions show that the the model conforms to Axtell and Epstein's models in the areas which were implemented. But more importantly, it shows that the simulation conforms to real world phenomena reasonably well.

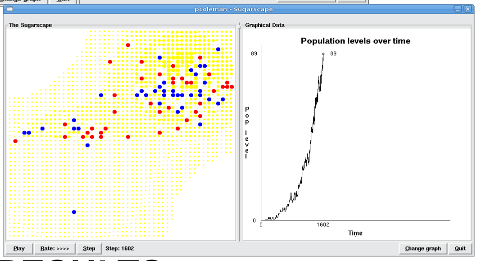
DEVELOPMENT

I. Theory. The algorithm driving the move method of the agents is at the core of the simulation. The agents look out in the four cardinal directions as far as their vision allows and move one square in the direction of the closest location with the most sugar. If more than one location is optimum, a random direction is chosen. To incorporate Schelling segregation, locations in which there would be more agents of the opposite color than of the same color are removed from the possible choices. See the section on agent movement in Appendix A. Agents are added to the environment according to an exponential function which models real life.
The section on population growth in Appendix A shows how the population growth is raphed. The inequality of the population is found using the Gini coefficient.
The Gini coefficient is calculated according to the formula: $1-2*L$ where L is the area under the Lorenz curve, which is calculated using trapezoidal Reiman sums. The section on wealth distribution in Appendix A shows how the Gini coefficient is calculated and how the Lorenz curve is graphed. Lastly, hemispherical winters cause agent migration to other high density locations and cause a drop in population.

II. Design Criteria. The goal of the project is to accurately represent the models it is implementing. It follows the Sugarscape design from *Growing Artificial Societies* by Axtell and Epstein and the Schelling segregation design from Schelling's book *Micromotives and Macrobehavior*. The agents and the environment behave as they should with respect to the aspect implemented so far. The Schelling segregation model will accurately represent Schelling's model as best as possible, but will not be perfect because concessions will need to be made to allow it to run in the Sugarscape. The information shown in the graphs and the display of the environment will be compared to the results found by the authors.

III. Materials. The program code was written in Ruby (see http://ruby-lang.org/). Tk toolkit is used for the GUI representation and graphics in the program. A text file which represents the maximum capacities of sugar in various locations in the environment was used from GMU's Tony Bigbee's files (he wrote a Java version).

IV. Procedures. Currently the program displays the environment, and has the agents move and harvest sugar. The display draws each location in the matrix using a circle whose radius increases based on the amount of sugar at that location. The display draws the agents as a red circle with the same radius as a location with the maximum amount of sugar. The display also shows the current time step. The GUI window has a frame containing the canvas and buttons to play/pause, step the simulation, increase the refresh rate, and to quit the program. The agents themselves choose the closest location with the greatest amount of sugar. If more than one location matches these requirements, one of them is randomly chosen. Locations with more agents of the opposite color are removed from the choices. Then the agent harvests the sugar and consumes from his own supply of sugar. At each time step the sugar in the environment grows back by one. The program begins with a small number of agents and adds to the population using an exponential function so that it reaches carrying capacity. Modifications in the individual agents include an improved move method, a random age limit, and a variable for red or blue color, to allow for segregation. The GUI window has been modified to include buttons to change the graph and change the refresh rate. There is a button to change the refresh rate in the display of the environment and of the graphs. The two graphs which are now displayed are the population growth over time, and the percent of total wealth over the percent of the population (Lorenz curve). To get the population graph, it keeps track of the length of the array of agents at each time step in the simulation file and cycles through the array of population values in the display file. To get the wealth graph, it cycles through the array of agents and stores the wealth of each individual agent. Then it sorts this array and cycles through it keeping a running total to determine percents.



RESULTS

It has been determined that the program meets the design criteria in the areas in which it was implemented. The graphs are what answer many of the experimental questions. Descriptions of the population growth graph refer to the section on population growth in Appendix B. In general it follows the shape of logistic graphs which are proven to be a fairly accurate representation of population growth. Growth is slow when the population is close to zero and close to the carrying capacity, and growth is highest at half of the carrying capacity. The few anomalies reveal certain aspects of the simulation. The initial portion of slow growth is smaller than the final portion because the population begins with three individuals instead of one (but starting with one agent would not completely remedy this). The oscillations near carrying capacity come from the age limit of agents. It takes longer for the population to decrease due to dead agents than it does for it to react to the added agents. The oscillations decrease over time and will eventually disappear. At about half of carrying capacity the line begins to become jagged instead of fairly straight like it was earlier in the simulation. This is a result of the heterogeneous population. In the beginning even agents with low vision and high metabolism (less fit agents) have room to survive in the regions of abundant sugar. As the environment fills up only better fit agents can survive on the fringes, areas with less sugar, so many added agents die quickly. The effects are even more pronounced as population approaches carrying capacity. The graph has even more information to offer when seasons are included. The spikes in the graph represent the winters. The spikes alternate in intensity because the northern winter is more sevr removing more high desnity sugar locations. Descriptions of population inequality refer to the graph in the wealth distribution section of Appendix B. At first a bar graph was used to represent wealth distribution, but it was replaced with the Lorenz curve. Both conform to the graphs in Axtell and Epstein's book. They show that there are very few wealthy agents (agents with a lot of harvested sugar stored) and many poor agents. In this sense the population is pretty unequal. The Gini coefficient is a numerical representation of this phenomenon.
A coefficient of zero represents perfect equality and one represents perfect inequality (one agent has all the wealth). The number is just over .5 showing that the Sugarscape population is closer to perfect inequality than to perfect equality. See Appendix B for a display showing Schelling segregation. There is some segregation at this point. The environment did not split in half as was expected. The most probable explanation is that this is due to the method of adding agents. Random colored agents are added to random empty locations. Asexual reproduction of same-color agents would likely produce better results.

Creation of an Air Traffic Simulation Using Agent-Based Modeling

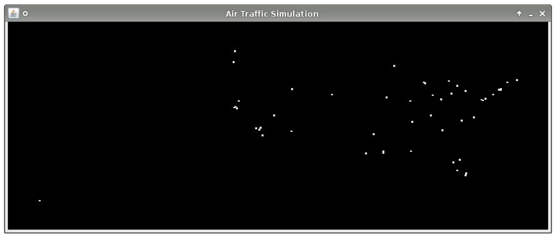
Sam Eberspacher
TJHSST Computer Systems Lab 2007-2008

Abstract:

As the skies over the United States become increasingly crowded, airports in the United States are increasingly stressed to adapt to this increased demand. The goal of this project is to visually represent the strain on airports and passengers as a variety of problems generate record delays. By using agent based modeling, along with real air traffic information, this simulation may accurately predict the proliferation of delays through out airports in the United States.

Background:

The purpose of this project is to visually represent the proliferation of a delay throughout a system of airports. By using techiques such as agent based modelling, the simulation will predict actual delays with decent accuracy. Additionally by repeating the simulation multiple times, the simulation generates increasingly accurate results as the number of trials approached infinity. While a simulation such as this would take a human enormous amounts of time, a computer may be able to run a simulation of 24 hours in a matter of minutes. Due to the scale of the problem, efficiency will be key for the computer to run the simulation in a timely matter.



Screenshot of Simulation Interface

Results:

RESULTS GO HERE

Also cutting parts of the Geocoder section and including Embedded Statistical Analysis section

Agent-Based Modeling:

In order to simulate such a large system, this project will use a technique known as agent based modelling. The deveopment of a system using agent based modelling is key for the success of the project. Each agent must interact with other agents in the system in the most realistic way possible in order to generate the most accurate results. One benefit of the agent based modelling is that parameters for interaction between agents define the overall behavior of the system. This allows the programmer to work on much smaller problems with the agent in order to alter the overall system.

Embedded Statistical Analysis:

Embedded statistical analysis is a done when real time statistics are needed in a simulation. The program uses data at each time step to readjust statistical values for the desired population. These statistics are useful when determining if the system is able to handle the introduction of new agents or new constraints.

This simulation determines the mean, and standard deviation of delays from all of the agents under the control of an airport. Using properties of these statistics, an overall standard deviation and mean are determined without polling all agents a second time. Tracking the historical values for the mean and standard deviation then allow for regression modeling to determine the speed of propagation through the system. This analysis is particularly useful when changing how agents interact with one another because the statistics inform the user whether of not the change is positive.

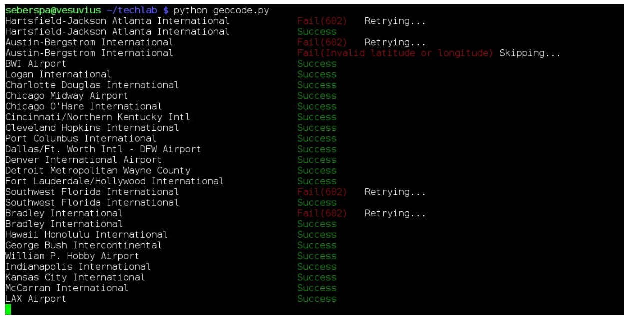
Geocoding:

Geocoding is a process by which a formatted address such as 6560 Braddock Rd. Alexandria, VA 22312 is converted to a longitude and latitude. This process is important when dealing with map information that is displayed on a computer because the computer is unable to relate formatted addresses so longitudes and latitudes are used to generate accurate relationships about location.

I found that Google offered free geocoding with a maximum of 5000 requests per day, which was more than enough for th e project. In order to interact with the Google geocoder, each airport was geocoded through an HTTP request sent to Google servers. These servers then interpret the parameters in the URL of the request and return the ouput specified by the user. The parameters in a request are as follows:

- q - The formatted address to be geocoded
- output - The desired output format (xml, kml, csv, or json)
- key - Google Maps API key

Sample Request (Key removed for privacy reasons):
http://maps.google.com/maps/geo?q=BWIairport&output=csv&key=API_KEY



```
seberspa@vesuvius ~/techlab $ python geocode.py
Hartsfield-Jackson Atlanta International          Fail(602)   Retrying...
Hartsfield-Jackson Atlanta International          Success
Austin-Bergstrom International                    Fail(602)   Retrying...
Austin-Bergstrom International                    Fail(Invalid latitude or longitude) Skipping...
BWI Airport                                       Success
Logan International                               Success
Charlotte Douglas International                   Success
Chicago Midway Airport                            Success
Chicago O'Hare International                      Success
Cincinnati/Northern Kentucky Intl                 Success
Cleveland Hopkins International                   Success
Port Columbus International                       Success
Dallas/Ft. Worth Intl - DFW Airport               Success
Denver International Airport                      Success
Detroit Metropolitan Wayne County                 Success
Fort Lauderdale/Hollywood International           Success
Southwest Florida International                   Fail(602)   Retrying...
Southwest Florida International                   Success
Bradley International                             Fail(602)   Retrying...
Bradley International                             Success
Hawaii Honolulu International                     Success
George Bush Intercontinental                      Success
William P. Hobby Airport                          Success
Indianapolis International                        Success
Kansas City International                         Success
McCarran International                            Success
LAX Airport                                       Success
```

Screnshot of Geocoding Results

Dynamic Image Resizing

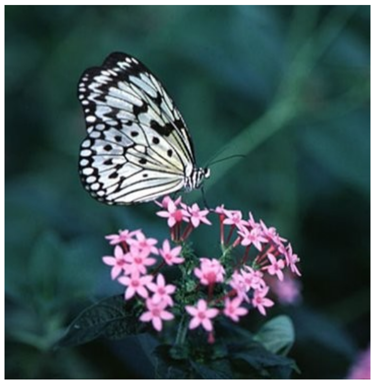
Patrick Elliott

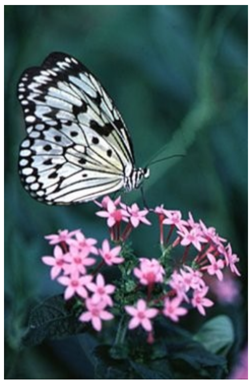
Abstract

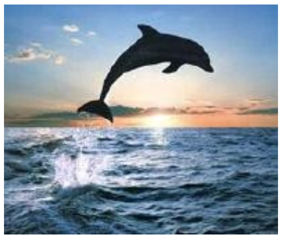
The goal of this project is to be able to resize an image without distorting any important aspects of the image. Commons methods of resizing, including cropping and scaling, remove or distort some of the image and are thus undesirable. By finding the least important pixels and removing them, this dynamic resizing can be possible. These can be found by finding the change of intensity of each pixel to the next and taking away the ones with a very low change. Using this method, humans should be unable to tell if an image has been altered.

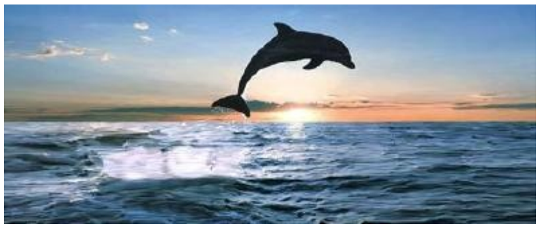
Background

Edge detection is being researched heavily in modern times. Many teams are trying to allow computers to see and identify objects. But there is also much research being conducted about images and modifying them. There is one project called PhotoSynth that is trying to take a large amount of images from the web, and from them, create a 3D model of whatever the images are of. There is also another project that is very similar to what I am trying to do, although I have some ideas for my project that they have not yet implemented.









Methods

Instead of cropping or scaling the image, both of which would ruin certain aspects, there is an algorithm that can find only the least noticeable pixels and remove them so that a human cannot tell the image has been altered. This algorithm finds the gradient magnitude of the image and removes the pixels with the smallest change of intensity.

To expand the image (bottom left), the same method is used, except instead of removing paths, the program adds a path next to it with the average values of the surrounding pixels.

Results

The image above is the original image. The one below it has been modified by the program. You can see both the butterfly and the flowers, both of which look unaltered, whereas scaling would ruin these. The unnecessary portions of the image have been removed.

The Application of Image Processing Techniques to Sign Language Recognition Using a Web Camera

by Byron Hood
March 26, 2008

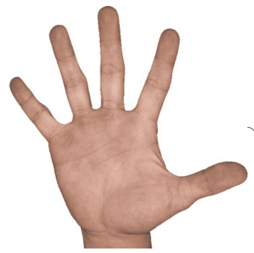
abstract

Sign language recognition is the first step in a long road towards natural language processing, or the ability for a computer to "understand" naturally spoken language. Such an invention would drastically lessen the amount of time require for computer input, maybe even by a factor of two. This project explores using image recognition techniques such as edge detection and line detection to identify sign language in real time, using input from an average web camera ("webcam"). When research is complete, it is expected that the program will be able to identify most, if not all alphanumeric characters with a high degree of accuracy.

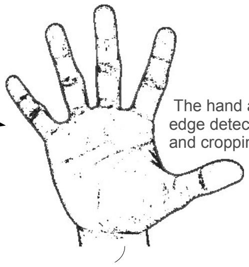
background & introduction

In today's society, people with hearing and speaking disorders communicate using sign language. Through extensive practice and use (as people gain extensive practice speaking their native language), sign speakers are capable of "speaking" as fast as others speak orally, from 200-220 words per minute.
The average computer user types 33 words per minute when transcribing and a mere 19 when composing. If the average sign speaker can communicate using finger spelling at as little as ¼ the pace of regular sign language, they sign 50 words per minute. If they could sign into a computer, this would be a significant speedup in computer input.

program procedures


The image of a hand, to be captured from a webcam. Here we use the sign for "5."

Edge detection

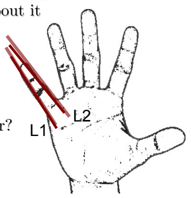

The hand after edge detection and cropping.

Line finding

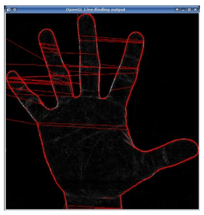
Parameterized lines:
L1: (0, 50) → (30, 95)
L2: (10, 45) → (45, 89)
...

Line-finding is the first step towards identifying complex shapes in an image. You can try this yourself: look at your hand and try to find individual straight lines. Then, imagine what those lines would reveal about the position of your hand if nothing else about it was known.

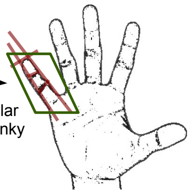
How would *you* recognize your ring finger?



Endpoints of the lines found in the picture. A theoretical version is on the left, and the program output is seen on the right (red lines are detected lines)



Line interpretation:
Analyzes all of the lines in a given region and tries to match them to a finger.


Similar to pinky

Final result:

```
> ./main
 ...
Detected char '5'
```

testing & timing

The program will be tested manually: automated testing would be highly impractical and would require complex image analysis (which is the object of my program). An example test would be running a program five times, and recording the time taken after each iteration, then manually viewing the results after all execution is complete.

Sample timing (done by the program):

```
> ./main
 ...
[DEBUG] Edge detect time: 384ms
```

Agent Based Simulation, Negotiation, and Strategy Optimization of Monopoly
TJHSST Computer Systems Lab 2007 - 2008

Nicholas Loffredo

ABSTRACT

Computers have a difficult time performing common human tasks, such as learning a language well enough to be able to "talk" intelligently with someone/something. Monopoly, one of the most well known and understood board games in the United States, if not the world, provides a good environment to see whether or not a computer can "learn" to negotiate through a number of strategies. It is much simpler than negotiating in the real world, due to the simplified environment, yet complex enough that it may be useful as an example of computer negotiation. By creating a Monopoly simulation with computer agents playing the game, it can be used as a test bed for these computer negotiations. The methods used in this test bed, if it works, could then be applied to more complex computer negotiation. The agents can be given aggressiveness values for different negotiation techniques, such as price "stubbornness" when selling or buying properties from other agents. The results from running the simulation hundreds of times can then be graphed to show which strategies are the optimal strategies for agents.

INTRODUCTION

Computers currently are unable to perform common human tasks such as understanding a language well enough to speak it and effectively communicate. A good example of this is negotiation. Most humans are able to negotiate with one another for various goods. Computers, on the other hand, can't. If computer were able to negotiate effectively, they could be used in many situations that currently require people – such as in diplomacy, selling/buying goods, trading goods, or just negotiating with other people in general. More importantly, it would allow people to instruct a robot/computer to negotiate using certain items and to meet certain goals, instead of hiring people to do it. These computers would be resistant to common human flaws, such as anger or impatience.

Making a computer than can negotiate effectively in a limited environment is a first step towards being able to negotiate in a more complex one. The game of Monopoly is simple enough that negotiation should be able to be implemented within a year, yet complex enough that the method used to achieve the results may be able to be applied towards real negotiation. By making a working simulation of Monopoly, a negotiation capability can be implemented for computer agents that will "play" the game.

Fundamentally, the system must simulate all the rules of Monopoly. Agents must be able to move around the board based on the "dice" roll, be able to buy titles they land on, and buy houses on monopolies they own. Additionally, they should be able to sell houses and mortgage properties. When an agent lands on a Chance or Community Chest square, they should receive the top 'card' from a 'deck' which was randomly sorted before the game, and 'do' whatever the card says. Furthermore, to explore the research areas contained herein, agents should also be able to negotiate with players. In particular, these negotiations will be based on aggressiveness levels. For example, how far an agent is willing to drop/raise his initial price in order to complete a negotiation.

BACKGROUND

Surprisingly, not much research has been done into making agents for Monopoly that can 'learn' the optimal strategy for negotiation. One of the few existing simulations is one that determines the probability of landing on each square in Monopoly. This can be used to see if my Monopoly simulation results correlate to theirs, and determine whether or not the simulation works correctly. Fortunately, there has been research done into reinforcement learning, which is effectively how the agents will learn. Reinforcement learning is when agents take a number of actions over a course of a game, and then are basically 'told' that they did well (when they won) or they did poorly (when they lost). Based on this feedback, each agent will try to change its aggressiveness values (which may involve different strategies) to find the winning values. Also, agents may learn from the strategies other agents used, and whether they won or lost, to determine how their aggressiveness levels should change, which varies from the traditional approach slightly. There is not a state-of-the-art reinforcement learning program yet, however. Everything I Need to Know About Business I Learned from Monopoly, which discusses various strategies for Monopoly, can be used to see if agents develop the strategies the book discusses.

PROCEDURES

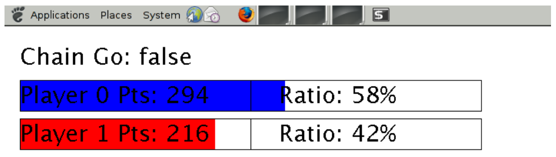
PRELIMINARY TESTING and ANALYSIS

To test my program, an interface was designed to help me spot errors in the game structure. The game works as it should, with agents moving around the board, buying properties, paying income or luxury tax, paying rent on other player's properties, and correctly following the instructions on Chance or Community Chest cards. These were tested by using the interface to watch each step of agents to ensure that they moved the correct number of squares, landed on the correct properties, paid rent when necessary, etc. Agents correctly performed their required actions.

I also tested my program by running it a large number of times and seeing if the learning agent actually has a higher winning rate than the random agent.

EXPECTED RESULTS

I expect to be able to find an optimal strategy for Monopoly based on agent victory rates vs. other agents (and their strategies). I also expect to find that reinforcement learning can be applied to Monopoly effectively.



Chain Go: false

Player 0 Pts: 294 Ratio: 58%

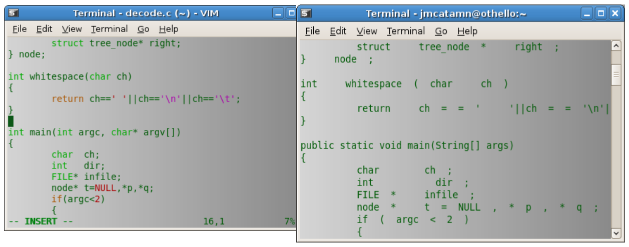
Player 1 Pts: 216 Ratio: 42%

Programming Language Translation

Jamie McAtamney
TJHSST Computer Systems Lab
2007-2008

Abstract

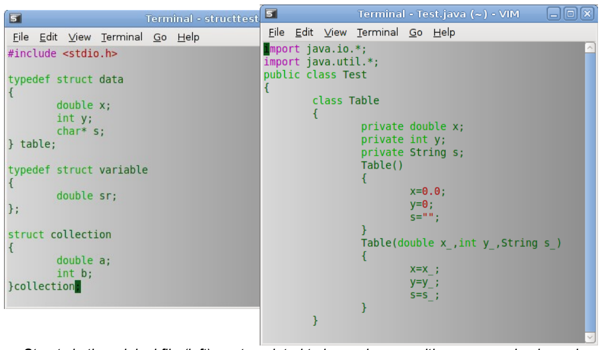
With the modern emphasis on program portability and the new need to run programs on multiple computers in networks or over the Internet, it would be very useful for C programmers to be able to translate either "legacy" C programs or newly written programs into Java to make them more portable; however, currently translation by hand is seen as too tedious and time-consuming, while computer algorithms to do so are not very accurate. A combination of keyword search/replace and algorithms to translate C structs to Java classes and C "include" modules to Java "import" modules can help alleviate or solve the problem of tedious or inaccurate translations specifically between C and Java.


Terminal - decode.c (~) - VIM
File Edit View Terminal Go Help
Terminal - jmcatamn@othello:~
File Edit View Terminal Go Help

```
        struct tree_node* right;
} node;

int whitespace(char ch)
{
        return ch==' '||ch=='\n'||ch=='\t';
}

int main(int argc, char* argv[])
{
        char  ch;
        int   dir;
        FILE* infile;
        node* t=NULL,*p,*q;
        if(argc<2)
        {
-- INSERT --                        16,1          7%
```

```
        struct     tree_node  *     right  ;
}     node  ;

int     whitespace  (  char     ch  )
{
        return     ch  =  =  '     '||ch  =  =  '\n'|
}

public static void main(String[] args)
{
        char        ch  ;
        int           dir  ;
        FILE  *     infile  ;
        node  *     t  =  NULL  ,  *  p  ,  *  q  ;
        if  (  argc  <  2  )
        {
```

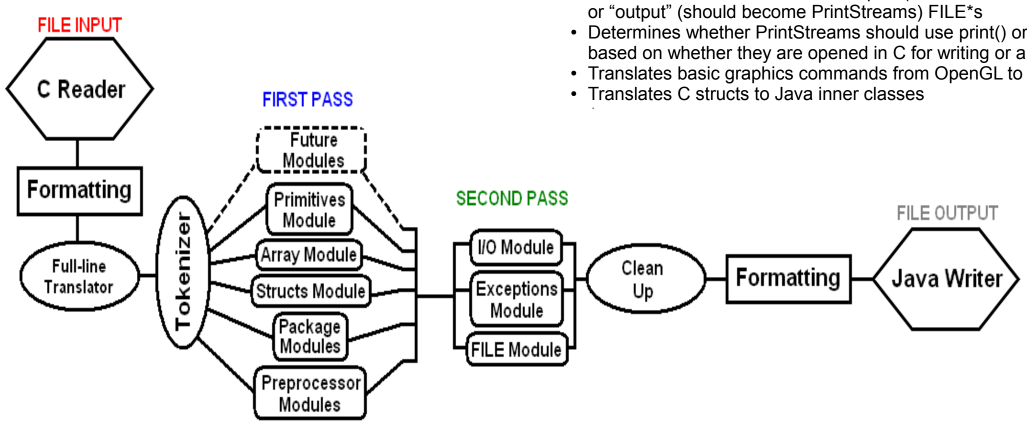
The first step of the translation process: The original file (left) is read in and tokenized, and any syntax-independent lines—such as the main() method—are translated (right).


Terminal - structtest
File Edit View Terminal Go Help
Terminal - Test.java (~) - VIM
File Edit View Terminal Go Help

```
#include <stdio.h>

typedef struct data
{
        double x;
        int y;
        char* s;
} table;

typedef struct variable
{
        double sr;
};

struct collection
{
        double a;
        int b;
}collection
```

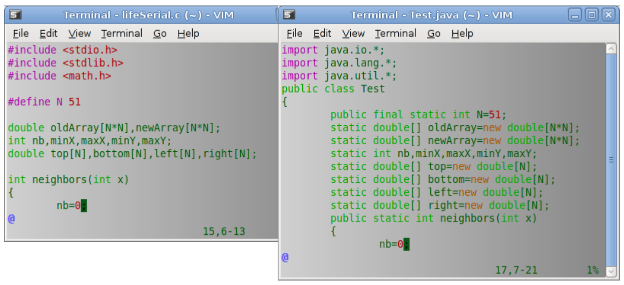
```
import java.io.*;
import java.util.*;
public class Test
{
        class Table
        {
                private double x;
                private int y;
                private String s;
                Table()
                {
                        x=0.0;
                        y=0;
                        s="";
                }
                Table(double x_,int y_,String s_)
                {
                        x=x_;
                        y=y_;
                        s=s_;
                }
        }
```

Structs in the original file (left) are translated to inner classes, with one example given above (right). Note the addition of two constructors for the inner class.



Background

While the differences among programming languages have been studied extensively in comparative languages courses and otherwise, little progress has been made in the area of automated programming language translation. The problems involved with automated translation occur because programming languages are too dissimilar for direct word-for-word translation. Even syntactically similar languages such as C and Java have differences that make simple search-and-replace difficult. For example, while the C char arrays have an analog in Java Strings, because they are two different data structures the methods for accessing them are very different, and this discrepancy must be taken into account. A related difficulty is C's use of pointers: A "string" in C is not simply an array of chars, it is a pointer to an array of chars—expressed as "char*"—which means that one cannot simply copy, compare, or otherwise manipulate strings in the same way one may manipulate ints or chars.


Terminal - lifeSerial.c (~) - VIM
File Edit View Terminal Go Help
Terminal - Test.java (~) - VIM
File Edit View Terminal Go Help

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 51

double oldArray[N*N],newArray[N*N];
int nb,minX,maxX,minY,maxY;
double top[N],bottom[N],left[N],right[N];

int neighbors(int x)
{
        nb=0;
@
                                15,6-13
```

```
import java.io.*;
import java.lang.*;
import java.util.*;
public class Test
{
        public final static int N=51;
        static double[] oldArray=new double[N*N];
        static double[] newArray=new double[N*N];
        static int nb,minX,maxX,minY,maxY;
        static double[] top=new double[N];
        static double[] bottom=new double[N];
        static double[] left=new double[N];
        static double[] right=new double[N];
        public static int neighbors(int x)
        {
                nb=0;
@
                                17,7-21         1%
```

After the translation process: The original file (left) has been translated (right). Note the C #include statements changing to Java import statements, C array declarations changing to Java array constructors, and the addition of a class declaration.

Progress and Results

At this point, several translation modules have been implemented:

- Translates primitive types, such as char* to String
- Translates C preprocessor directives ("#include" to "import" and "#define" to variable declaration)
- Translates C array declarations to Java array constructors
- Translates input and output methods and structures such as input/output streams and files
- Translates method and module headers
- Translates most packages: math, string, stdlib, stdio, and more
- Handles throwing Exceptions
- Determines whether FILE*s are "input" (should become Scanners) or "output" (should become PrintStreams) FILE*s
- Determines whether PrintStreams should use print() or append() based on whether they are opened in C for writing or appending
- Translates basic graphics commands from OpenGL to JOGL
- Translates C structs to Java inner classes

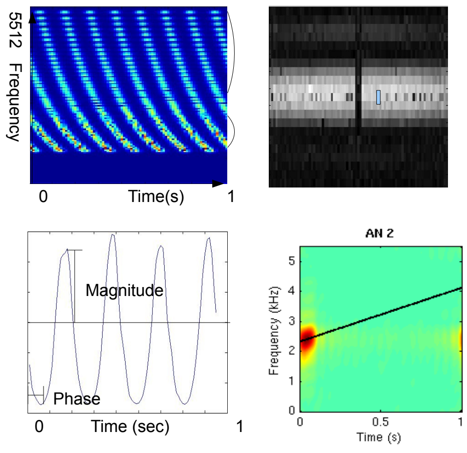
Analysis of spectro-temporal receptive fields in an auditory neural network

Madhav Nandipati, TJHSST Computer Systems Lab

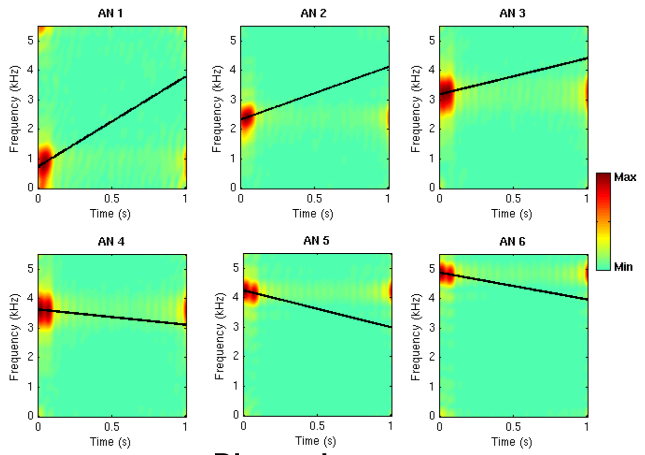
Abstract

Neural networks have been utilized for a vast range of applications, including computational biology. But the validity of these models remains in question. In the models of the auditory cortex, for example, the properties of neuronal populations are hard to fully characterize with traditional methods such as tuning curves. Spectro-temporal receptive fields (STRFs), which describe neurons in both the spectral and temporal domains, have been obtained in a variety of animals, but have not been adequately studied in computational models. The aim of this project is to address these issues by generating the spectro-temporal receptive fields on a basic, neural network model of the early auditory processing stages in the brain. This novel use of STRFs can be used as a means of comparison between a model and its biological counterpart.

Generating STRFs



Results



Discussion

The receptive fields for the six different artificial neurons are plotted in the figure. The abscissa represents time after stimulus onset and the ordinate represents frequency. The bright area of the graph shows where the artificial neuron responds with greatest intensity. The dark area shows where artificial neuron does not respond to, or responds very weakly to. The graphs show that the artificial neurons are responding to distinct frequency ranges. For instance, the STRF for AN 1 shows that the second artificial neuron responds strongly to frequencies up to 1310 Hz. As the artificial neurons are connected to higher frequency input units, the receptive fields show that the artificial neuron also responds to higher frequencies. This result agrees with the hypothesized outcome.

Applications

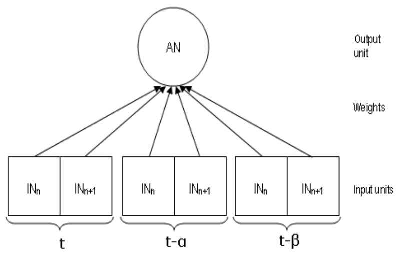
Receptive fields have been extensively analyzed in both animals and neural networks in the visual domain. In the auditory domain, spectro-temporal receptive fields (STRFs) describe both the spectral and temporal aspects of a neuron. The STRFs have been used in many types of animals, but have not been explored in a computational model. This project describes the construction of a neural network of basic auditory processing and the subsequent testing using STRFs. The neural network could also serve as a starting point in investigating new therapies for hearing loss. One possible cortical level treatment for hearing loss is electrode stimulation, where small electrical currents are delivered to parts of the brain. Another type of treatment is transcranial magnetic stimulation (TMS). Both of these therapies can be first tested in neural networks by simulating the effect of brain stimulation. Researchers would be able to quickly validate the use of the therapy by examining the properties of the artificial neurons through STRFs.

Linear Model

The current neural network is a two-layer, forward feeding and linear model of the early auditory processing stages in the brain. The first layer is the input layer. The input to the neural network is a spectrogram, a graph of frequency v. time, which was converted from a waveform of the sound stimulus using a Fourier transform. The first layer receives one timestep of auditory information at a time. A timestep is about 12 ms of auditory information. The second layer is the output layer, a matrix product of the input vector and weight vector. The weights between the two layers were first set at random values, then trained using Oja's rule. After the weights were trained, they were plotted and analyzed.

Temporal Model

This neural network is also a two-layer forward feeding and receives time-delayed inputs from multiple timesteps ago. Each of these timesteps composes the entire first layer, which becomes 387 units long (three timesteps each of 129 units).Using this configuration, the neural network was trained using Oja's rule. As with the linaer model, the weights between the first and second layers of the model were originally set at zero-centered, normally distributed, random values. In addition, the connections between the artificial neurons are be created nor eliminated; they are only modified.



Accurate 3D Modeling of User Inputted Molecules Using a Hill Climbing Algorithm

Ben Parr

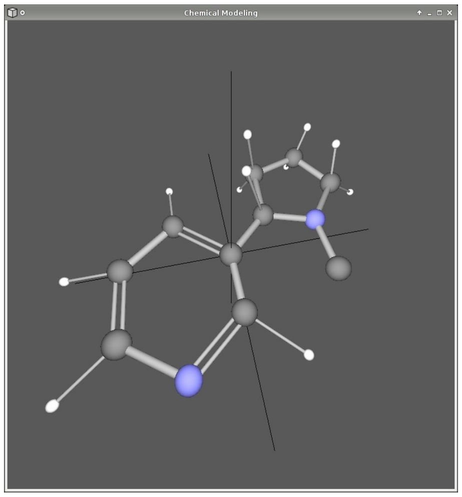
Abstract:

In order to better understand chemistry, chemists create 3 dimensional models of molecules. In a large introductory chemistry class, it is too costly to provide each student with a physical modeling set. Software available online can also be costly. The goal of my research project is to create a program that will allow users to generate 3D models of simple molecules through an intuitive and user friendly program. After the model is created, the program will then position the atoms in the molecule correctly by using a Nelder Mead algorithm. My project will help people, especially students, better understand the geometry of different molecules.

Background:

A lot of research has been done on modeling molecules and the techniques to do so now have become quite advanced. These programs have become increasingly more accurate over the years. However, the cost of these programs has also increased, and now very few people have access to them. For a beginning chemistry student there are not many options to play around with molecules and learn their different geometries.

One of the main features of my project will be the “Auto position atoms” function. This function will use a Nelder mead algorithm to minimize the energy of the model. The Nelder Mead algorithm was created by Nelder and Mead in 1965, and is uses the concept of a simplex in order to minimize a function in a many-dimensional space. A simplex is a polytope of N+1 vertices in N dimensions; therefore, it is a triangle in 2D space and a tetrahedron in 3D space. The algorithm begins with an original simplex. This simplex can not be too small, because it could lead to a local search. A simplex that is too large could a noticeably longer time.

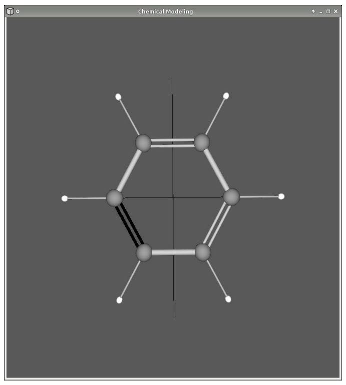


Nicotine ($C_{10}H_{14}N_2$)

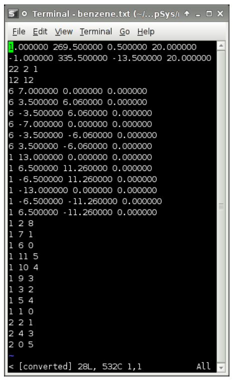
The energy function can be broken up into two parts. The first part is the distance of each atom to every other atom. The program will try to maximize this distance because real atoms do push away from each other. The second part is making the distances of bonded atoms a specific distance apart. This distance is specified in a data file and depends on what two atoms are connected and what type of bond exists between them. Since the Nelder Mead function tries to minimize the energy of the system, the minimum energy will occur when the atoms are as far apart from each other, except for bonded atoms which are the specified distance away from each other.

Methods:

Atoms are represented by spheres; bonds are represented by cylinders. Through inputs from the mouse and keyboard, users can create atoms and bonds, select and delete atoms and bonds, import and export models, draw single, double and triple bonds, choose which element they want to draw, and position the atoms where they want. The Nelder Mead function is used to “Auto position atoms.”



Benzene (C_6H_6)


Terminal - benzene.txt (~/...pSys/
File Edit View Terminal Go Help

```
1.000000 269.500000 0.500000 20.000000
-1.000000 335.500000 -13.500000 20.000000
22 2 1
12 12
6 7.000000 0.000000 0.000000
6 3.500000 6.060000 0.000000
6 -3.500000 6.060000 0.000000
6 -7.000000 0.000000 0.000000
6 -3.500000 -6.060000 0.000000
6 3.500000 -6.060000 0.000000
1 13.000000 0.000000 0.000000
1 6.500000 11.260000 0.000000
1 -6.500000 11.260000 0.000000
1 -13.000000 0.000000 0.000000
1 -6.500000 -11.260000 0.000000
1 6.500000 -11.260000 0.000000
1 2 8
1 7 1
1 6 0
1 11 5
1 10 4
1 9 3
1 3 2
1 5 4
1 1 0
2 2 1
2 4 3
2 0 5
~
< [converted] 28L, 532C 1,1             All
```

Exported Benzene file

Results:

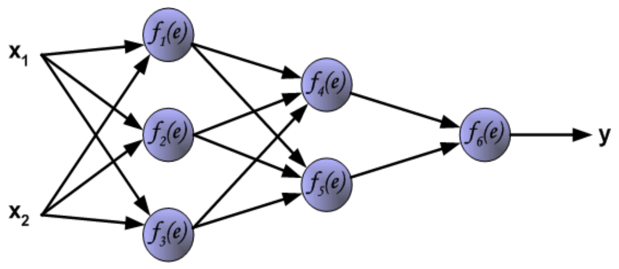
The purpose of my project is to create a free program that will allow users to easily and intuitively create models of molecules. Then, after a user finished creating the molecule, the program will correctly position the atoms. My project currently allows users to create molecules. A user can also rotate the model and zoom it in and out. These functions help users get a better picture of the molecule. I have also finished programming the Nelder Mead algorithm. The focus for fourth quarter will be programming the energy function and the original simplex, and then thoroughly testing the “Auto position atoms” function.

TJHSST Computer Systems Lab 2007 - 2008
Automobile Recognition Through the Use of Image Processing Techniques

by Drew Stebbins

Abstract

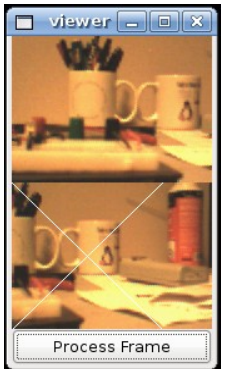
Many law enforcement agencies have recently shown interest in automated automobile recognition and tracking technologies such as license plate reading or GPS tracking. However, some criminals may drive vehicles that have false license plates or are not equipped with GPS tracking devices, making the pursuit of such vehicles difficult. This project aims to create a computer vision system capable of taking real-time input from a static camera and identifying passing cars by make and model in order to assist law enforcement agencies in the tracking of suspect or stolen vehicles.



A simple feed-forward, double-hidden-layer neural net

Background

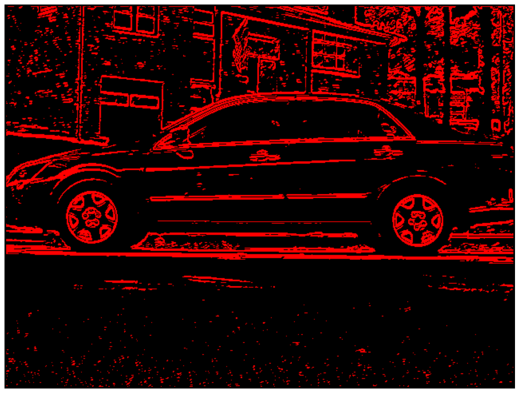
Several computer systems currently exist for the tracking of military and civilian automobiles via License Plate Recognition (LPR) or GPS technology. Such systems are in use by law enforcement entities such as US Customs and Border Protection and the UK police, and have proved very effective in catching criminals. However, these systems fail when an automobile has fake or no plates, and no GPS tracking device, and is able to avoid recognition. My system, on the other hand, will be able to alert law enforcement officers of the presence of any specific shape, color, or size of vehicle regardless of whether or not it is equipped with a GPS receiver or the proper license plates. Some systems already exist that can automatically recognize military vehicles such as tanks, planes, and armored personnel carriers by their shape, size, and color. However, in the course of my preliminary research I found no existing systems capable of automatically recognizing civilian vehicles such as cars or trucks. My system is similar to preexisting systems for the automatic detection of military vehicles in that it defines a certain set of characteristics for comparison, extracts those characteristics from the image of a single vehicle, and searches amongst a list of known vehicles and characteristics for a possible match. The primary difference between previous types of these systems and my own is that mine is much more precise in terms of characteristics such as size and shape, and, when completed, will select possible matches from a much more diverse database.



Webcam used, and streaming input as viewed with GUI

Procedure and Methods

At the moment, my final program exists in the form of many individually compiled component parts. My image processing programs make use of the Canny edge detector, Hough transform line detector, and my own vertex detection algorithm. My streaming video input GUI uses the video4linux API and GTK+ graphical toolkit, and will eventually be integrated with the code in my image processing programs. My single-hidden-layer, feed-forward neural net uses the backpropagation learning algorithm to do simple optical character recognition (I will most likely not be using a neural net in the final version of my program due to the advantages of other methods of object recognition). The language I am doing all of my programming in is C++, which I have selected based on its fast speed and hardware access capabilities.



Output of Canny edge detector

Results

The various components of my program currently perform as expected, accurately detecting lines, recognizing handwritten characters, and displaying video input. The frame rate of my streaming input viewer is somewhat low, a problem which will have to be addressed before accurate motion detection can be attempted. I hope to be able to test an assembled version of my final program on short video segments of cars in motion by the end of the third quarter.

Cayley graphs formed by conjugate generating sets of S_n

Jacob Steinhardt
TJHSST Computer Systems Lab 2007-2008

Poster is on the way

Conformal Mapping Using the Schwarz-Christoffel Transform 2007-2008

Evan Warner

January 28, 2008

Abstract

The Schwarz-Christoffel transform is a conformal mapping from the upper half of the complex plane to a polygonal domain. This transform allows many physical problems posed on two-dimensional, polygonal regions, such as heat flow, fluid flow, and electrostatics, to be solved numerically. This type of problem cannot generally be solved in closed form; the Schwarz-Christoffel transform provides an exceptionally accurate method of solution. This project consists of a software unit that efficiently and accurately calculates Schwarz-Christoffel transforms and inverses. The program incorporates graphical, easy-to-use interfaces and will contain resources to aid in solving physical problems. In addition, research into mathematical extensions to the Schwarz-Christoffel transform, such as the inclusion of simple curves, will be conducted.

Introduction

Many physical problems are expressed as differential or boundary value problems over a surface. Often, these surfaces are or can be approximated by two-dimensional polygons. In this specific case, one method of determining accurate solutions is by taking the polygonal domain to exist in the complex plane and determining a conformal map, which preserves the structure of Laplace's equation, that restates the problem in a simpler domain, most often the upper half-plane. The new problem, now easy to solve analytically or in closed form, is then mapped back to the original domain. For such polygonal domains, a method of determining the specific transform needed is provided by the following formula, known as the Schwarz-Christoffel transform:

$$f(z) = A\int_0^z \prod_{j=1}^{n} (\zeta - x_j)^{-\theta_j/\pi} d\zeta + B. \qquad (1)$$

In this formula, ζ is an independent complex variable in the upper half-plane, the θ_j are the exterior angles of the polygon, the x_j are 'preverticies' of the mapping (given along the real axis), n is the number of verticies of the polygon, and A and B are complex constants that specify the location of the image polygon in the complex plane. The θ_j must satisfy

$$\sum_{j=1}^{n} \theta_j = 2\pi, \qquad (2)$$

which ensures the completeness of the image polygon [2]. Unfortunately, the Schwarz-Christoffel formula is not easy to evaluate, and requires both effective integration algorithms and efficient, convergent nonlinear equation solvers. Implementation of such numerical routines is not a trivial problem, and is the subject of this paper.

Background

The Schwarz-Christoffel transform was first discovered independently in the late 1860s by Elwin Christoffel and Hermann Schwarz. Schwarz used some of the ideas of the transform to provide a more rigorous proof of the Riemann Mapping Theorem, which he had previously shown to be incomplete, but the majority of this work was on a purely theoretical level [5]. The usefulness of the transform was mitigated by the formula's unwieldiness, as the mappings for all but the simplest domains could not be calculated in closed form. Numerical estimates, especially for nonsymmetric polygons with four or more verticies, could not be effectively calculated by hand. Application to physical problems, therefore, was limited at best until the advent of the computer. A computer algorithm to compute the Schwarz-Christoffel transform was first written in the 1960s, and others have been written and modified since then [3].

The first problem in calculating the Schwarz-Christoffel mapping is the evaluation of the integral given by Eq. (1). The integrand contains singularities at each of the endpoints of the image polygon, which tend to render ordinary numerical integration routines either useless or hopelessly slow. In addition, the presence of negative powers in f means that domains of applicability for each of the subfunctions $(\zeta - x_j)^{-\theta_j/\pi}$ must be chosen so that the entire domain in and immediately around the image polygon is meromorphic. Although several quadrature routines have been used for this problem, the method of choice today is Gauss-Jacobi quadrature, which uses a specially-tailored weighting function to choose points of evaluation and weights for the points that maximize efficiency. In practice, the Schwarz-Christoffel formula is altered so that the prevertex x_n is chosen to be both $-\infty$ and $+\infty$ (the values are equivalent for a conformal map, which acts on the Riemann sphere). This can always be done due to the extra degrees of freedom contained in Eq. (1). The integrals that must be evaluated in practice in the course of the Schwarz-Christoffel transform are of the form

$$\int_{x_{i-1}}^{x_i} \prod_{j=1}^{n-1} (\zeta - x_j)^{-\theta_j/\pi} d\zeta. \qquad (3)$$

These integrals can always be written as required for Gauss-Jacobi quadrature; that is, in the form

$$\int_a^b (z-a)^\alpha (z-b)^\beta \psi(z) dz, \qquad (4)$$

where α and β are real numbers greater than -1.

The points and weights of a Gauss-Jacobi quadrature are calculated here using a routine from Numerical Recipes [4] which efficiently estimates and solves for the roots of the Jacobi polynomials, which form the sample points just as the roots of the Chebyshev polynomials form the sample points for standard Gaussian quadrature. These points, however, are uniformly calculated in the range $[-1, 1]$, and the integrals must be adjusted slightly to conform to this range. During the calculation of the preverticies, discussed below, the z in Eq. (4) will be restricted to the real axis; however, in direct calculations once the preverticies have been found, the z will generally be fully complex, which must be dealt with by the program.

The second problem is the Schwarz-Christoffel parameter problem, where the x_j in Eq. (1) are calculated. As described in [2], a series of nonlinear, constrained equations can be formed from the requirement that the image polygon and the desired polygon be similar (the constants A and B in Eq. (1) then ensure congruency). Written out, there are $n-3$ linear equations in $n-3$ unknowns, once the extra degrees of freedom have been taken care of by arbitrarily giving three of the x_j precise values. Here, as in the literature, we take $x_1 = -1$ and $x_2 = 0$ in addition to the already-defined $x_n = \pm\infty$. The equations to be solved are then

$$\frac{|\int_{x_{i-1}}^{x_i} \prod_{j=1}^{n} (\zeta - x_j)^{-\theta_j/\pi} d\zeta|}{|\int_{x_1}^{x_2} \prod_{j=1}^{n} (\zeta - x_j)^{-\theta_j/\pi} d\zeta|} - \frac{|w_j - w_{j-1}|}{|w_2 - w_1|} = 0, \qquad (5)$$

where $i = 3, 4, ..., n-1$. However, there is an additional complication, as the order of the preverticies on the real axis matters. The extra constraint can be expressed as

$$0 < x_3 < x_4 < \ldots < x_{n-1} < \infty. \qquad (6)$$

The nonconstrained problem is relatively easy to solve; however, the constraint prevents a naive application of a Newton's Method variant to this problem. To get around this, Trefethen in [3] suggests a simple change of variables that ensures the inequalities of Eq. (6). Take a new series of variables, χ_j, and let

$$\chi_j = \ln(x_j - x_{j-1}). \qquad (7)$$

The resulting χ_j will automatically obey Eq. (6), and the original x_j are found by the simple inverse formula

$$x_j = x_{j-1} + e^{\chi_j}. \qquad (8)$$

This new set of equations in the χ_j is readily solved by a variant of Newton's Method that does not require an explicit calculation of the Jacobian matrix (which would be hopelessly complex), but rather uses progressive estimates.

Development

The software has been written entirely in Java, although certain routines may be later written in C to increase speed if there is a bottleneck at any point in the process. The entire development of the program is designed to be achieved in stages by attacking the subproblems individually. The following is a list of classes, with short descriptions, written up to this point:

- class Complex - this class stores and performs arithmetic on complex numbers, which are not directly supported by Java. Several of the methods, including the multiplication and division algorithms, are designed to run as quickly as possible while avoiding intermediate overflow and floating-point error propagation. The multiplication method, for instance, requires only three real multiplications rather than four.

- class GaussJacobiWeights - this class calculates and stores the sample points and weights for a given Gauss-Jacobi quadrature over the interval $[-1, 1]$. This routine uses Newton's Method to find the roots of the Jacobi polynomials, which are the sample points for the integral, and was taken and translated from [4].

- class SchwarzFunction - this class evaluates the integrand of a given real-valued Schwarz-Christoffel integral, serving as a storage class for data of this kind.

- class GaussQuad - this class accepts as input ψ, a, b, α, and β from Eq. (4). For an arbitrary integral in that form, shifting and scaling the bounds produces the equivalent integral

  $$c^{\alpha+\beta+1} \int_{-1}^{1} (\zeta - 1)^\alpha (\zeta + 1)^\beta \psi(c\zeta + m) d\zeta, \qquad (9)$$

  where $b = \frac{a+b}{2}$ and $c = b - m = m - a$. This integral is then evaluated using the sample points and weights given by the GaussJacobiWeights class and returned. For any GaussQuad object, varying numbers of sample points (and thus varying accuracy) are accepted by its integrate() method.

- class RealNewtonRaphson - this class accepts an array of vertices and calculates the necessary prevertices as well as the constants A and B from Eq. (1). The method employs a standard Newton-Raphson method to solve the Eq. (5). At each step, an approximate Jacobian matrix for the function is calculated using a forward-difference method in each dimension; the step vector is then solved for using an LU factorization on the equation

  $$\mathbf{J}\delta\vec{x} = \vec{f}, \qquad (10)$$

  where $\mathbf{J}$ represents the Jacobian, $\delta\vec{x}$ the step vector, and $\vec{f}$ the current function vector. Note that by employing a forward-difference method to find the Jacobian, the number of function evaluations can be cut in half, as the current function vector can be reused in the Jacobian calculation.

- class ForwardGaussQuad - this class, using already-calculated values for the prevertices, evaluates the Schwarz-Christoffel integral at a given point. To minimize error caused by the presence of singularities near the path of the integral (the singularities at the endpoints are handled by the Gauss-Jacobi quadrature), the path of integration is divided recursively such that no segment is closer to a singularity than one-half its length, a technique employed in [3]. Such recursive subdivision is known as compound Gauss-Jacobi quadrature.

- class SchwarzChristoffel - this class runs the graphical user interface and calls RealNewtonRaphson and ForwardGaussQuad when necessary. The graph itself has the ability to show axes and manually adjust window parameters.

In future iterations of the project, a new set of routines will be implemented to calculate continuous Schwarz-Christoffel problems. Immediately following from Eq. (3) above, we have

$$f'(z) = A\prod_{j=1}^{n-1} (\zeta - x_j)^{-\theta_j/\pi}. \qquad (11)$$

To change this into a continuous problem, we can rewrite this as

$$f'(z) = Ae^{\frac{1}{\pi}\sum_{j=1}^{n-1} -\theta_j \ln(z - x_j)}. \qquad (12)$$

Then, defining the natural logarithm function as single-valued in the upper half-plane, except where $x_i = z$, f' becomes an analytic function in the required domain. To formulate the continuous-boundary problem, we simply replce the sum in Eq. (11) with an integral, and integrate the entire function to find $f(z)$:

$$f(z) = A\int_0^z e^{\frac{1}{\pi}\int_{-\infty}^{\infty} -\theta(x)\ln(\zeta - x_j)dx} d\zeta + B, \qquad (13)$$

where $\theta(x)$ represents the amount of turning per unit length on the real axis, such that

$$\int_{-\infty}^{\infty} \theta(x) dx = 2\pi. \qquad (14)$$

The continuous problem therefore has an extra subproblem to solve, namely, the solution of the integral equation, Eq. (12), to find $\theta(x)$ at every x.

Expected Results

The purpose of this project is to calculate and display Schwarz-Christoffel transforms, which conformally map the upper half-plane to an arbitrary polygon, efficiently and accurately. In addition, additional research into the Schwarz-Christoffel transform itself, including its extension to curved target domains, will be investigated. The evaluation of the Schwarz-Christoffel formula involves several parts, including the efficient calculation of a certain class of integrals as well as a solver of nonlinear systems of equations. Solving the continuous-parameter problem will require numerical solutions to a certain class of integral equations.

The first problem, that of numerical integration, has been solved and refined, and a basic user interface has been designed. The second problem, that of a nonlinear equation solver to calculate the prevertices, has also been completed to satisfaction; current research focuses on correct implementation of the forward transform using given prevertices. Preliminary results indicate the general correctness but inexactitude of the forward transform in the absence of compound quadrature, especially near the boundaries of the given polygon. It is hoped that a full implementation of the compound quadrature will ameliorate these concerns.

The completed program will be useful on several levels: as a teaching aid, and as a tool for researchers solving certain equations on polygonal regions. Once the basic Schwarz-Christoffel problem is numerically solved, the program can form an easy basis for testing research in numerical analysis and mathematics that deals with improving or expanding the Schwarz-Christoffel transform.

References

[1] Howell, L. H. (1990). *Computation of conformal maps by modified Schwarz-Christoffel transformations.* Retrieved September 28, 2007, from http://citeseer.ist.psu.edu/howell90computation.html.

[2] Saff, E. B., & Snider, A. D. (n.d.). *Funamentals of complex analysis with applications to engineering, science, and mathematics.* Prentice-Hall Engineering/Science/Mathematics.

[3] Trefethen, L. (1979). *Numerical computation of the Schwarz-Christoffel transformation.* Retrieved September 28, 2007, from ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/79/710/CS-TR-79-710.pdf.

[4] Press W., Teukolsky, S., Vetterling, W., & Flannery, B. (1992). *Numerical Recipes in C, Second Edition.* Cambridge University Press.

[5] O'Connor, J.J., & Robertson, E.F. (2001). *Hermann Amandus Schwarz.* Retrieved November 3, 2007, from http://www-history.mcs.st-andrews.ac.uk/Biographies/Schwarz.html.