Assignment 4: Shape Abstraction

Overview

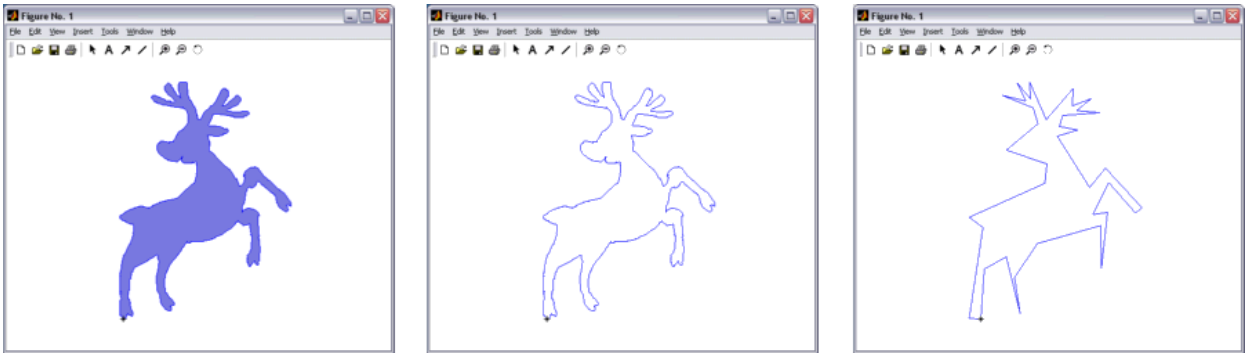
It's getting serious. We will combine data structures with graphics and a real application from computer vision.

Computer vision is the science of image analysis, or short: to teach the computer to understand (the beauty of) the visual world. A basic, yet complicated task is shape recognition. An object's shape can for example be represented by its 2D boundary, see figure below. The computer's task is to recognize this boundary, and act accordingly (a deer: how cute!, a lion => run). The actual shape recognition is a bit too much for this semester, but we will program an important pre-processing step: shape abstraction.

Shape abstraction can be explained quite simply: given an object boundary of connected points, remove those points that don't carry a visual significance, i.e. are not needed to identify the shape. You will most likely remember the "connect the dots" images from your childhood. The bright kid you were, you would have only needed half the dots to identify the resulting object.

But let's first get a bit more into data-structures:

A 2-dimensional shape can be defined by its boundary-polygon, which is a list of all coordinates of its outline points. See the following figure for an example:



The left picture shows the original shape (defined by its AREA), the middle picture the outline (BOUNDARY) of the shape. We want to create a LIST of all points of the outline. In order to do so, we start at some arbitrary point and traverse the outline clockwise. We store the coordinates of each point in a node, and therefore create an ordered list of x,y-pairs of coordinates of all boundary vertices, e.g.
x=12, y=14
x=13, y=14
x=13, y=15
...

Don't panic yet. This list of points will be given to you as a text file, so you don't have to program a boundary tracing algorithm.

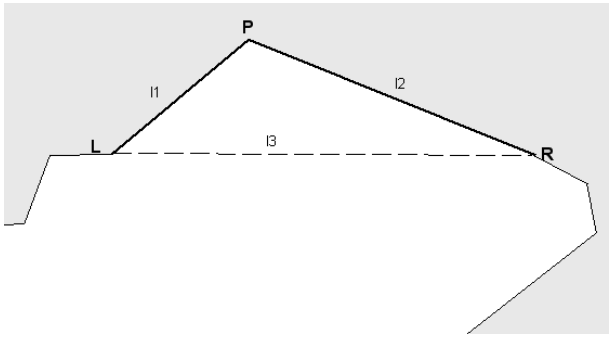
Now take a look at the rightmost picture in the figure above. It surely shows a very similar shape, it seems to be an abstracted version of the original boundary. In fact it is: the rightmost picture shows a certain subset of the original boundary vertices (points), connected by lines.
The remaining subset consists of boundary vertices containing visually significant information (in terms of human visual perception).

Visual Significance Measure

A very simple way to achieve this is an algorithm called Discrete Curve Evolution (Latecki/Lakaemper 1999, click

http://knight.temple.edu/~lakamper/courses/cis2168_2012_SPRING/assignments/assig04_folder/CVIU1999.pdf

if you want to read the original paper): It keeps the important points, and dismisses unimportant ones. The importance is measured by the amount of visual information in the following way:



The figure above shows a magnified part of the deer boundary. L,P and R a vertices, the lengths of the segments connecting these vertices are: LP = l1, PR = l2, RL = l3 (Since someone from Greece long ago defined the length between two points as the 'Euclidean distance', it is given by sqrt((x1-x2)^2 + (y1-y2)^2), as you might know).
The line LR shown in the figure is not part of the boundary, but we need its length for the significance measure:

Define the visual significance S of vertex P simply by S(P) = l1 + l2 - l3.
For better understanding, here are some properties of this significance measure S(P) of point P:

- It is zero if L,P,R are collinear (certainly P does not contain any visual information then !)
- It increases with P moving away from the baseline LR
- It is never less than zero

Defining this significance-value allows us to assign the significance to every single vertex of the boundary list.

Now comes the important part: the abstraction. Remember we want to create a subset of the original boundary, i.e. want to keep important vertices, dropping unimportant ones.
A natural way to do this is the following:

1. Assign the significance value S() to every vertex except the first and last one (they don't have 2 neighbors, and you don't want to deal with that, so just define them as 'important')
2. set S for the first and last vertex to INFINITY (that means: define the first and last point as important)
3. LOOPSTART: Find the vertex containing the minimum significance
4. Drop the vertex by removing it from the list
5. compute the new values S for the 2 points, that were the neighbors of the removed vertex
6. loop until the list contains a predefined number of vertices

Some remarks:

- No sorting is necessary in this implementation, we search the minimum by traversing the list (O(n))
- Important: since you only have a single linked list at hand by now, but you need to have access to the previous and next point, you need a workaround. There are two possibilities:
    - you store the previously handled point somewhere
    - you work with the current, current.next and current.next.next to assign the visual importance to current.next, not to current!

Your Task

(if you want to panic, now is the correct time to start)

Please plan your work in this order!

- Read the boundary-list (download-address at the end of this file) , given as a textfile of (x,y)-coordinates of boundary vertices, into a linked list. You must create your own data structure, i.e. your own linked list.
    - Read each line of the text file, extract the 2 points defined in that line, and store them in a new node. The node will be of a class that you have to define.
    - add the node to the list
- Visualize the List, connecting all successive vertices given in the file by lines.
- Implement the abstraction-algorithm. For this, you will need a "remove" method in your double-linked-list.
- Skip the first and last point in the list, they should never be removed. That makes it much easier.
- Simplify the polygon until only 38 points remain.
- Show the simplified polygon

You must at least have classes for

- a Point. A point contains the position x,y, and the importance value.
- a Node. The basic structure for your linked list. It needs to be able to hold a Point, as well as the reference to the next Node.
- a LinkedList. Your main data structure. You will need methods to add a point, to traverse the list. You also must be able to access elements in the list. Here it would make sense to have a method "getNext", along with a method "reset", which resets the reference used in "getNext" to the start.

We will talk in class about the assignment, especially about the data structure, and techniques to implement a "getNext" in a very general way.

Points for partial achievements:

- reading the file & drawing the original deer: 2 points
- implementing the data structure, all methods etc.: 3 points
- implementing the abstraction algorithm: 5 points

Implementation Conditions

- Using a list is mandatory. No arrays this time (zero points for arrays, no discussion).
- The abstraction in the first figure above, rightmost picture, is done by exactly this algorithm, but it uses a different measure for the significance value. This simplified image also contains 38 vertices, but your results might look slightly different.

Earn **3 bonus points** for adding a slider to your application, showing each step of the simplification. Because a slider can go back and forth, you have to store the results of each single simplification step. You could use an Array of Lists for that.

This is a serious assignment. One week is a short time for it. Start immediately.

The text file for the deer boundary list is here:

http://knight.temple.edu/~lakamper/courses/cis2168_2012_SPRING/assignments/assig04_folder/shapelist.txt

Note: the deer you read from the file will appear upside down. You can change that by flipping the y-value of each point and adding an offset: y = -y+700. Create a SimpleGui of size about 800x750. That should be sufficient.