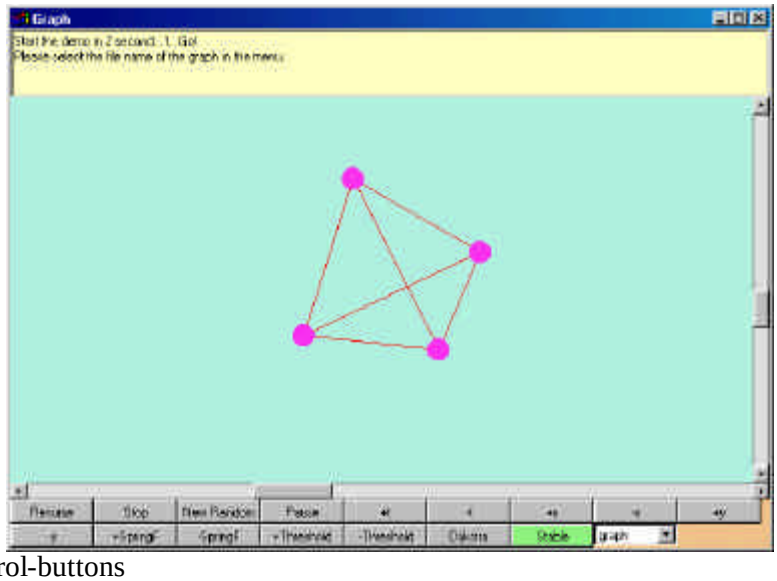
Prapun Suksompong ps92
Supap Kirtsaeng sk158

Design Description

CS 211 Project 6

Demo instruction

- The demo will always start with the graph of a tetrahedron.
- The graph can be changed by selecting the file name in the menu at the bottom right.
- Every time the graph file is loaded, the graph will be created, and the number of vertices, edges and the graph's diameter will be displayed in the text area.
- When all the force is below the threshold value, the graph becomes stable, and the pink "**unstable**" button will change to a green "**stable**" button.
- Sometimes, the graph looks stable but with a little shaking, which doesn't stop for a long time. To fix this, increase the threshold by clicking the "**+Threshold**" button. This will make the graph becomes stable easier.
- Sometimes, the graph becomes stable but actually not, i.e., it becomes stable before it changes to the expected shape. To fix this, lower the threshold by clicking the "**-Threshold**" button. This will make the program takes into account the small forces that are not used before since they are lower than the threshold.
- The **scrollbar**s will rotate the graph about the x axis and y axis.
- **Clicking a vertex** at any time will define that vertex as a base point for the distance calculation. It will use the unweighted-edge calculation and try to calculate the shortest path from the base point to other points. The distances are displayed on the screen along with the vertices' names.
- Clicking the "**dijkstra**" button will change the mode of distance calculation to dijkstra, which will weighted each edge with the actual distance between vertices. It requires selecting the base point. After the base vertex is clicked, the vertices names and the distances from base point will be labeled on the graph. The shortest path to each vertices is printed on the text area. It might be a good idea to "**pause**" the graph before using "dijkstra," because the distances used are the ones at the base-point-click time, which will continually change as the graph changes.
- After "pause," click "resume" button to continue the forces calculation.
- And easy way to see the result of a new graph is to modify the graph.txt file, save it, and then select it again from the menu. No need to restart the program.



More control-buttons

- "**New Random**" button will recreate random positions for the vertices of the current graph.
- "+t" and "-t" buttons will change the time interval between each update of the positions. To make the graph change shape faster, use "-t" button.
- "+x", "-x", "+y", "-y" will shift the center of the graph in that direction.
- Note that the graph is recentered so that it is always at the middle of the screen.
- "+SpringF" and "-SpringF" buttons will change the spring constant. To make the spring force stronger, which will make the graph eventually smaller, use "+SpringF"
- "+Threshold" and "-Threshold" will change the amount of the force that is weak enough and can be disregarded.
- "Stop" button will cease the force calculation and the graph screen update.

Graph design

All info of a graph is captured inside an object of a Graph type. A Graph object has instance variables:

- Vector vertex – a list of Vertex objects
- Vector edge – a list of Vector, denoting the adjacency list. Each component (also a Vector list) corresponds to a list of adjacent Vertices for the Vertex that owns that list.

Vector vs. other data structures.

Note that using Vector are not always the best solution especially when we have a lot of vertices which we don't know the maximum amount in advance because this required resizing the vector. This problem can be fixed using other kinds of structure, for example, linked list. Using linked list will have fewer problems for the size and also deletion/insertion. However, since we deal with a graph that is not large and create a graph once, then the number of vertices and edges don't change, using Vector is sufficient.

Each Vertex object contains several variables that capture its state. Among these are:

- int dimension,
- double[] coor,
- int dist,
- int via,
- boolean known.

The last 3 variables are used mainly for the shortest path calculation.

The existence of the Point class

From our point of view, a Vertex has all the information of a point namely the coordinate and the dimension. Since there is no places in the program where a point that is not a Vertex is used, we don't code the Point class. Alternatively, we can make the Vertex class extends the Point class. However, because the Point class is never actually used and there is nothing besides Vertex that will extends it, we go straight to the Vertex class.

Keeping integer inside the edge adjacency list vs. keeping links to adjacent Vertices

Using direct links to the vertices inside the adjacency list is useful because we get the adjacent vertices out directly when accessing the list. However, since we expect the vertices to be defined as 0,1,2,3,… not other types of name, and we use Vector to store Vertex, using integer is still easy to refer to any individual Vertex stored in a Vector.

Display Design

The center of the graph is always at the same position (; this position can be changed using the GUI control.) This will make the graph easier to see when rotated.

Change the value of the vertices coordinates as the display changed (rotated, shifted)

vs.

keep the original values and when having to draw, save the state and manipulate a copy.

It might be nice to leave the original values of the coordinates untouched. For example, when doing rotation, pass the values of the coordinates, calculate the new coordinates, use the new one for drawing, modify as need (rotated, shifted) and leave the old one the same. However, since

- the graph have to be changed all the time
- the absolute values of coordinates don't matter. We want to know only the shape and the relative distance, which doesn't change by rotation or shifting.

So, we directly update the coordinates of vertex when rotation occurs, and use them directly when we display. This will make the rotation as the graph changes easier, since we don't need to calculate the after-rotate position again when

update the display. Using other method will have to change the state of the graph by the force and recalculate the rotated position every time the display is updated.

Rotate using the absolute value from the scrollbar vs. using the amount changed from the last position

We choose the later case, because we choose to update the coordinate as the graph's rotated. For example if the value of the scrollbar is at 30 degrees now, and we change it to 60 degrees. Our design will have the graph of 30 degrees rotation in the first place, and to make it 60 degrees, we have to rotate it 30 degrees more. If we choose not to change the coordinates when rotating, we will always have the graph at 0 degree rotation and to make it 60 degrees, we have to use 60.

Values decision

- Vertex.INF – denote the infinite distance. We choose this value to be 1000000 because our display is about 500×700, 1,000,000 should be large enough.
- Vertex.noPath – denote that there is no (shortest) way of coming to this Vertex from others at the current time. We choose this value to be –2, since there is no negative name Vertex. (We use 0, 1, 2, 3, …).
- Charge constants: MVector.CFC = 2500000;
  Spring Constant: MVector.SFC = 1;
  GraphPanel.threshold = 1;
  These values are changed as we test the program in order to make an appropriate size of the displayed graph.
  Note that SFC and threshold have the flexibility of being changed from the GUI control.
- Spring natural length = 100. This number is chosen so that the graph will eventually stable with the edge of length more than 100, making the text label displayed along each vertex easy to read.