Algorithms for 3D printing of graphs

Students: Dvir Arad, Jeremie Gabay;

Advisor: Prof. Vadim E. Levit

Summary:

3D printing is rapidly becoming a standard in high scale manufacture. From car parts to prosthetics, this technology is very promising and will without a doubt play a central role in the future of mankind.

Our project is to implement a set of algorithms that will make possible the printing of complex graphs under specific conditions and restrictions. For example, a graph without any edge crossing.

This kind of technology might come in handy in the electric engineering field (printing of full-featured printed circuits under restrictions of size or volume) or in the medical field (printing of nervous implants, blood vessels or even grey matter) and probably in countless other fields, where manufacturing product may be seen as a graph-like objects.

Motivation:

Prof. Vadim E. Levit, our Algorithms teacher and Head of the Department of Computer Science and Mathematics at Ariel University, suggested Graph Theory as a research subject, particularly the 3D printing of graphs.

Since 3D printing is a vast, interesting area of research and discoveries yet to be made, we agreed to board on this research/journey with our well-appreciated teacher.

Needless to say that all the possible applications such a technology might close a little more the gap to are also a very strong source of motivation.

Visit to the 3D-printers unit in Ariel University:

We've been visiting Mrs. Shiri Reizberg, responsible for 3D printers and Mr. Boaz Ya'akobi, responsible for PCB printer. They have been very helpful, supplying us with information and answering our questions. Here's what we learned:

- The Alaris30 printer prints in white. It prints 28 micron-thin layers, UV-hardening them before the next layer. It can also print a removable matter, adding the possibility to print moving parts without having to build them from separate parts. **Its input is a STL file.**
- The CubeX printer prints in any colour. It melts and prints colored threads layer after layer, hardening them before the next layer. Its small-sized printing-head and large printing surface enable the printing of large and complicated models. **Its input is a STL file.**

Our intent is to experiment on both printers.

- Mr. Ya'akobi explained to us how the PCB printer works (scraping out the useless copper from copper-coated insulating plates), and more generally the limitations and capabilities of today's PCB printing technology (number of conductive layers, thickness of the PCB⋯).

Our project's current update:

Our principal advancements are presented further.

Here are the questions that we were required to answer:

1. Can any graph be embedded in 3D-space without any edge-crossing ?
2. In which format should we output the result ?
3. How will our algorithm work (input, output, etc.) ?
4. How many 2D-subspaces do we generate, when we add one extra point, which is not on a 2D-subspace already generated, to the existing set of points ?
5. How many random points do we have to check in order to find the one, which does not belong to any 2D-subspace already generated ?

1. Required proof:

As mentioned in our first report, we must, first and foremost, prove that any given graph can be embedded in 3-dimensional space without any edge-crossing (intersections between edges).

Here is the proof

We also tried to prove this proposition using a sphere but there were too many conditions and limitations (curved edges, detection of intersections needed…). In a more advanced stage of the research will we maybe have to prove the proposition this way.

2. Output format:

After understanding the 3D printers' requirements (i.e. STL files), we searched for a way to, either output an STL file or output another format that can be converted to the STL format.

STL files come in 2 *flavours*: binary and text. Of course we cannot generate the binary file programmatically, so we look into the text version of STL. Its syntax mainly defines full panes of a surface and by connecting all those panes can one get the surface required.

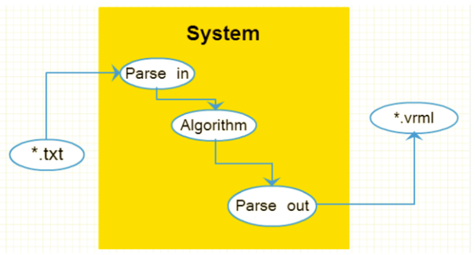
After searching a little more, we found here that there is another 3D modeling format called VRML (mainly used for HTML rendering), which can easily be converted into STL files (using Accutrans3D, according to the author).

Since VRML's format seems more adapted to our graphic needs, we plan to use it.

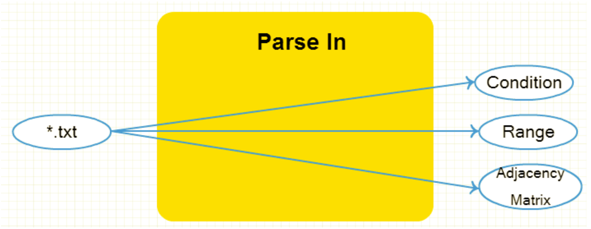
3. Diagrams and general behaviour of the Algorithm

Our Algorithm mainly consists of 3 modules:

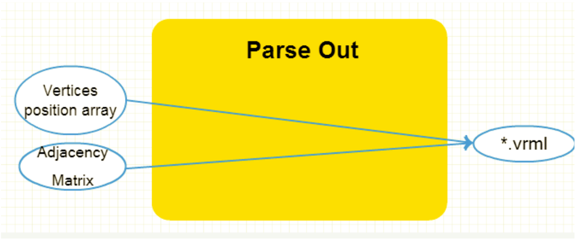
1. Parsing in the .txt file
2. Running the computing algorithm
3. Parsing out to VRML format



```
PARSING IN
a. receives preformatted text file
b. prepares it for the computing main algorithm
c. outputs range, condition & adjacency matrix
END PARSING IN
```

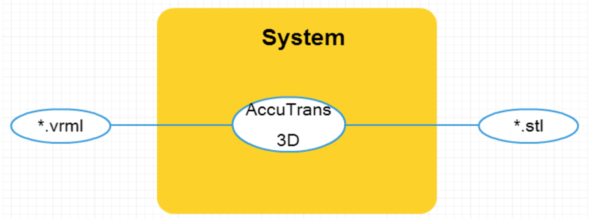


```
MAIN ALGORITHM -> call Get3DEmbedding(...)
METHOD Get3DEmbedding(vSize, condition, range)
       vSize - the amount of vertex
       condition - spacial limits
       range - distance between vertices
       vertices - array of vertex

       numberOfPlanes <- getNumberOfPlanes(vSize)
       surfaces <- arrays of surface equations with size numberOfPlanes
       FOR i <- 0 to vSize
              DO temp <- random vertex according to condition & range
              WHILE (belongsToSurface(temp))

              addPlanes(temp)
              vertices <- temp
       END FOR
END METHOD
METHOD int getNumberOfPlanes(int size)
       numberOfPlanes <- 1
       FOR i <- 3 to size
              n <- n+(size*(size-1))/2
       END FOR

       return numberOfPlanes
END METHOD
METHOD addPlanes(vertex ver)
       IF vertices.size >= 2
       THEN FOR  i <- 0 to vertices.size
                     FOR  j <- i+1 to vertices.size
                     add new surface(vertex.get(i), vertex.get(j), ver)) to surfaces
                     END FOR
              END FOR
       END IF
END METHOD
METHOD boolean belongsToSurface(vertex vertex)
       FOR EACH surface in surfaces
              IF vertex is on surface
              THEN
                     return true
              END IF
       END FOR
       return FALSE
END METHOD
END MAIN ALGORITHM
```



```
PARSING OUT
a. converts vertices & adj. matrix to VRML-formatted text
b. output generated text to a .vrml file
END PARSING OUT
```

VRML To STL →



4. Formula for the number of the generated 2D subspaces

We already know that 3 points are required to define a plane. It's easy to understand that each new extra point added (which does not belong to any plane already generated) will define a new plane in correspondence with each set of 2 already existing points.

For example, adding a 4^{th} *point to 3 already existing points will define* $\binom{3}{2} = 3$ *new planes, resulting in a total of 4 planes defined.*

Generally, for a graph $G(V,E)$ such as $|V| = n$, the total number of planes defined will be :

$$1 + \sum_{i=3}^{n-1} \binom{i}{2}$$

We see that for n big enough, we might encounter memory issues.

5. Number of random points to check

In this test, we aimed to observe how the given condition* and the given number of vertices influenced the number of random points generated during the Main Algorithm per vertice :

Number of vertices	**Number of surfaces**	**Condition - cube's edge size***	**Range**	**Max number of points checked per vertice**
100	161700	4	10	511
100	161700	15	10	3
10	120	1	10	1
18	816	10	1	326
74	64824	4	10	76

**We used a cube for the spatial condition. Its edges' size will limit the size of the usable space.*

More applications suggestions:

Our algorithm can be used to 3D-print all graph-like objects. Here is a list of such objects (not accounting for current limitations of the 3D-printing technology):

- Molecules models	- Blood vessels
- PCBs	- Bars mechanisms
- Nervous systems	- Piping systems
- Neural networks	- etc.

Expected results for the presentation:

We will show a 3D-printed non-planar graph in its 3D intersections-free embedding that our algorithm can generate.

Also, we will demonstrate the actual algorithm efficiency and show different results as 3D navigable models on a screen.