Defining Graph Aesthetics

By Ian Williams

An undirected graph is a set of V vertices and a set E of edges where each edge is undirected and associated with a pair of vertices. The number of edges incident to a vertex (v) is defined as the degree of (v). If the graph can be drawn in a plane without any edges intersecting each other than it is known as a planar graph. A face is an area enclosed by edges. Fullerene graphs are 3-regular planar graphs with face sizes equal to 5 or 6. A concave polygon is a polygon with an angle measuring more than 180°. A convex polygon is a polygon that is not concave (no angle in it measures more than 180°.

My research project is to explore what defines a nice picture of a graph. The aim of this paper is to mathematically define the ascetics of an image of a fullerene graph and create an algorithm for consistently judging which pictures are good looking and which ones are not. To accomplish this goal I will determine what traits make an image "appealing". Also important to establishing this is finding what traits make an image "unappealing". The intended outcome of this research is to compare effectiveness of drawing algorithms for fullerene graphs and to help identify flaws in them. It will allow a human user to evaluate larger quantities of images more consistently than they could do themselves by hand. In addition, it could just help a user select a nice looking picture of a graph. I chose to do this particular project out of personal interest. Since I am a combined major in computer science and visual arts the idea of mathematically defining aesthetics seemed like a great way to combine 2 different areas of study in an exciting way.

For my research I will be using a computer program called FUIGUI that generates images of fullerene graphs based on the number of vertices. The version I used was slightly modified by Wendy Myrvold to make my research process easier. FUIGUI consist of five folders and as five steps to compile and run it. The first folder generates the graph information and outputs it to text files based on the number of vertices, which are then moved into the next folder. Folder two contains a program including the code I wrote in C based on my research. It assigns a rating and writes it to a text file that is then moved to folder three. At step three the program combines all the information for the graphs and sorts it by the score from lowest to highest. The fourth folder separates the information into different files and tells the program what variables are stored in which file. The last step runs the program FUIGUI allowing the pictures to be displayed. (Myrvold, Bultena, Daugherty, Debroni, Gim, Minchenko, Woodcock, Fowler, 2007)

A number of studies have been conducted on the aesthetic qualities in graphs. They were conducted with similar methods but different focuses, some focusing on human comprehension and some on pure aesthetics. One study conducted asked subjects to view graphs and were given questions to answer in order to show their understanding of the graph. Questions ranged from shortest path between two nodes, or minimum number of edges removed to disconnect two nodes. Response time and errors were recorded for data. (Purchase, Carrington, Allder, 2002). Another study provided graph information, similar to the data generated by step 1 of FUIGUI. The participants were asked to draw the graphs maximize human understanding. (Purchase, Pilcher, Christopher, Plimmer, Beryl, 2012). Despite the different methods and different aims of the research, most results came down to similar group of characteristics. The most important ones were minimum number of crossing edges, symmetry of graph structure; maximize size of crossing angles and maximizing edge/node orthogonality. (Purchase, 2002). Some other common factors were even distribution of vertices, uniform edge lengths and a minimum number of curved edges. Curved edges are not supported by FUIGUI so the last one isn't of concern for my research. In their research they found that humans didn't value uniform edge length as much as some of the drawing algorithms did. (Purchase, Pilcher, Christopher, Plimmer, Beryl, 2012). More edge crossings directly lead to more errors. One thing that particularly relates to my research was that aesthetics are mostly mutually exclusive so they cannot all be optimized to the maximum. (Huang, Eades, Hong, Lin, 2012). In order to create an aesthetically pleasing graph a balance must be found between all the different characteristics. Overall the research shows that whether focusing on pure aesthetics or human comprehension, they both lead to the same group of traits. Therefore, generally speaking, graphs that are aesthetically pleasing are better for human comprehension and vice versa. This exemplifies another aspect to investigate when conducting research.

The first objective of my research was to find out what traits made one fullerene image receive a better or worse rating compared to another. In order to get this I had to establish a database of images with their ratings. The ratings were a simple integer between 0 and 100. To achieve this, I wrote a simple program in C that would was a simple loop running until told to quit by the user. The loop output the number of previous inputs to the terminal and then accepted an integer between 0 and 100, writing it to a new line in a regular text file. I then simply ran 2 concurrent terminal windows, one running FUIGUI and the other running the rating program. I would load a header file for all the graphs of a certain number of vertices then simply input a number in my program, press enter and then click next graph in FUIGUI. The result was a text file that was merely a list of numbers. If you put the file into the third folder and recompiled starting at step three out of the five then the program automatically sorts the images by their score from smallest to greatest.

After going through hundreds of images of fullerenes with varying number of vertices, the database was complete. The next step was cycling through the images once again but this time sorted by order of scores. In this step one can see what traits become evident at certain score ranges. After analyzing the data there were 5 key traits that became apparent in the distinctions between pictures of higher and lower scores, symmetry, convex faces, even face area distribution, floating points, and crossing edges. I noticed that the traits fell into 2 types, negative and modifier. "Negatives" were ones that lowered the image on an essentially flat rate and "modifiers" affected the score less on varying amounts.

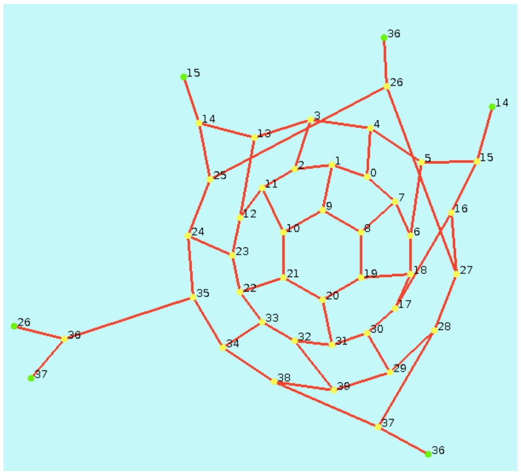

Figure 1: Example of multiple edge crossings

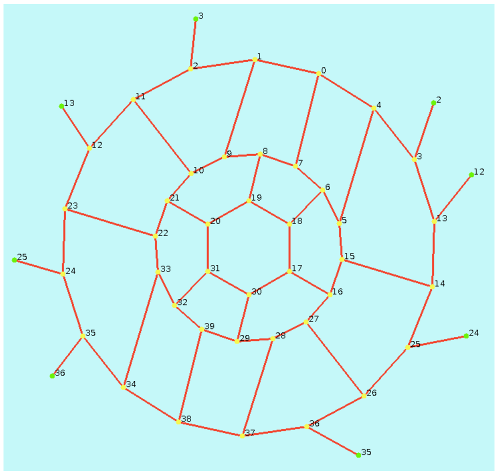

Figure 2: Example of multiple floating points

Crossing edges were a negative trait, they basically ruin an image because when lines are crossing the image becomes very confusing for someone to understand, look poorly and alter other aspects of the image without showing up mathematically. An example of this is a crossing edge cutting a face in half at a strange angle. Mathematically all the program has is the coordinates of the vertices and which vertices are connected to which ones, so if a crossing edge intersect at a small angle you would need to determine which lines cross and then try the angle instead of just checking from the graph coordinates. I decided that this trait was the worst trait of the identifiable ones.

The next trait that became evident was floating points. Floating points are when two nodes are connected by an edge but are not connected visually in the image. In FUIGUI vertices are coloured yellow with red edges, a floating point will have a yellow vertex (u) connected to a green vertex (v). The green point doesn't correspond to an actual vertex in the graph listing. The vertex number of (u) will be the same as a different yellow vertex (w) which is connected to a green vertex (x). Vertex (u) will have the same number as vertex (w) as vertex (v) has the same number as vertex (x). Floating points generally show up on a graph where images would overlap but could be avoided by using a curved edge. They help the image look better than a crossed edge but make it much harder to understand and are not an accurate depiction of the graph. Mathematically floating points aren't visible and thus are treated the same as crossing edges.

The previous 2 examples divide the graphs into categories, those without the negative traits and ones with them, and lower the score accordingly. The other "modifier" traits are used to determine where the graph stands within those groups. They do not affect the score as much as the negative traits. The first example of a modifier characteristic is convex faces (see example below). Generally the faces will have an angle larger than 180 degrees, so I focused mostly on not having faces with small angles. Small angles mean that adjacent faces will have larger angles making the faces become less uniform. Having similarly shaped faces make the whole graph look better.

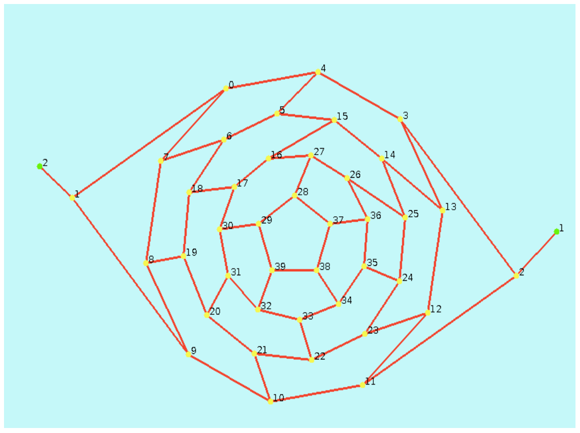
Figure 3: Concave faces with small and large angles

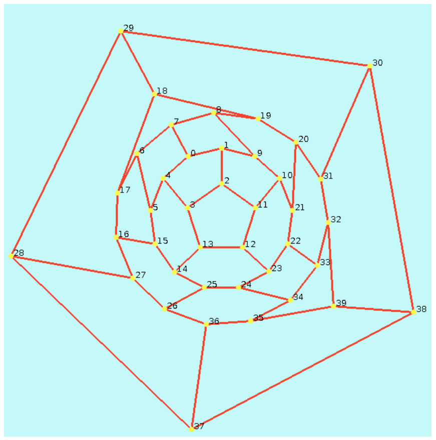
Figure 4: Poor distribution of face area

Another quality whose importance is on par with convex faces is face area distribution. The area of the faces gets bigger as you move outward from the center. The graph image looks a lot better if the faces gradually get larger as opposed to some where the area of the faces closest to the outside are significantly larger than the inner ones.

The last trait that became apparent when looking through my database was symmetry. This trait is the hardest to define mathematically and would affect the score the least. In my research it was symmetry along the Y axis that made a noticeable difference because it is much more noticeable than symmetry across the X-axis. When I was rating the images by hand I usually judged the image based on the previous traits and then symmetry would dictate the hierarchy amongst those common images. Essentially if two images are scored on the first 4 traits and one has a score higher by more than 5 or 10 than adding symmetry to the equation would not change the hierarchy, only if the two images' score difference was less than that.

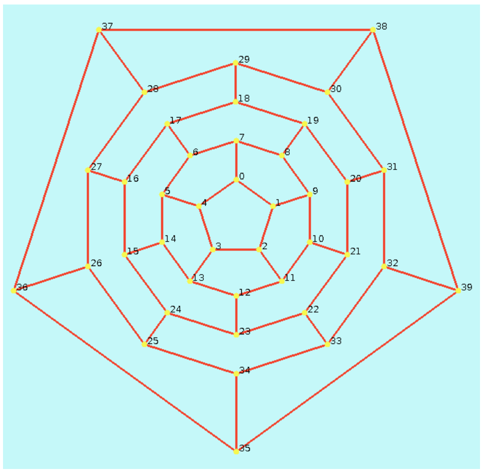
Figure 5: Good example of symmetry along the Y-axis

At this point in the research it was time for the main objective of my project, implementing the algorithm. The purpose is to automatically rate graph images created by FUIGUI that produces the same results as if an actual user was doing it. The scoring function would measure the levels of the predefined characteristics and apply that to the rating. I created a function for either each trait, or multiple functions that can combine their results, as some sort of measurement of a single trait. The code that I wrote is in the file assign_penalty.c, which is the second folder in FUIGUI. The first thing that is modified is the function get_score(). The program reads the information for the graphs generated by the first folder and generates a list of vertices with an array for the X coordinates and an array for the Y coordinates and a 3D array for the vertex number of the 3 adjacent vertices. The get_score() function creates two 2D arrays and fills it so that the 2 lists is a directory of all the edges in the graph, storing coordinates for one vertex in one array and the second vertex of the same edge is stored in the other array at the same index. Once the list of edges is established it calls the other functions, collects the return values and calculates the total score and then returns it, which is put into the score list file.

The first function, called get_crossing(), I programmed was the simplest to define, for crossing edges. The function takes the graph, and the list of all the edges as parameters. Since the edges are in a list, it cycles through the array and compares each edge to all the edges before it. This allows the function to have every edge compared to every other edge but makes it slightly more efficient than N squared time. I programmed it so that it determines if they crosses by doing the typical algorithm for intersecting points in lines. It determines the slope and Y intercept to form the equation of the both lines. If the 2 slopes are the same and the lines aren't the same coordinates than they don't intersect at all. The function finds the point of intersection and then checks to see if it is between the 2 X coordinates and the 2 Y coordinates. If it is in between them then one is added to the count variable and the loop continues on through the list of edges. At the end of get_crossing() the function returns the count variable which is the total number of edge crossings in the graph.

The second function is the one that gets the length of the edges. It uses the list created in the get_score() function as a parameter and calls a separate function that uses the coordinates and simply Pythagorean theory to find the length of each edge and stores it in a list. Since the key aspect of edge lengths is that you want proportional edge lengths, it then sorts the edge lengths in order from greatest to smallest. If the graph has proportional edge lengths then the difference between ordered lengths would be minimized. The difference between one item in the list of ordered lengths minus the next one is then added to all the other differences and finally divided by the number of edges and that number is returned to the get_score() function.

The next trait to be measured is the angles inside the faces. To calculate this value, the program goes through each vertex and checks the 3 adjacent vertices. It uses the main vertex as a center point and calculates the degree of the angle between the usual starting point on the X axis to the line formed by the main vertex and one of the adjacent vertices. The difference between these 3 separate angles is the angles formed by the 3 edges connected at the central vertex. The average of these angles will always be 120 degrees since the 3 of them add to 360 so the program discards the largest of the 3 angles. Even angle distribution would mean an angle of 120, but we also are looking for solid right angles so it determines whether it's closer to 90 or 120 degrees. If it's less than 90 degrees then the difference is 90 – angle or if it is larger than 120 the difference is angle – 120. If it measures between 90 and 120 then the difference is zero. The value returned to the central function for this trait is the average difference of the 2 smallest angle of each vertex for all of the vertices in the fullerene.

Symmetry is the most difficult to mathematically define of the qualities. Since in the analysis of the graph I preferred horizontal symmetry I focused on horizontal symmetric distribution of vertices. I wrote an algorithm that would find the highest and lowest values of both X and Y coordinates in one graph. Then used those to establish the middle of those for the X and Y axes. The graph is then vertically divided into 8 equal sections with each section divided into 2 equal portions along the Y-axis. The algorithm then goes through each vertex determining which section it falls in and adds 1 to the appropriate value stored in a 2D array. The difference in each 8 sections between its left and right portions is recorded and the sum of all these differences is returned to the scoring function as an integer.

All the functions previously mentioned were programmed separately and then placed in, as well as adapted to fit within, assign_penalty.c in FUIGUI. When I started playing with weighting values I thought that something was missing in the entire equation after viewing the results. After comparing my results to the data collected during stage one I decided to factor in the number of straight vertical or horizontal lines as an aesthetic I found pleasing and an extra modifier to diversify the scores even more. I added a function that would calculate the number of straight edges in the graph by traversing the list of edges and comparing to see if the X value of the first vertex matched the X value of the second vertex and the same process with Y values. That function returned an integer showing the number of straight vertical or horizontal lines found in a single image.

After the programming was completed for this stage the next step was to calculate an actual score from the data automatically collected on the graph. Essentially I made the algorithm assign a decimal number to each trait that was measured. To settle on these decimal numbers I would set the program to return the value from the trait measuring function as the total score for the graph. After re compiling FUIGUI I checked the output score files to find the lowest and highest values to discover the range of that quality. Each characteristic integer would go through a series of comparison if statements, which assigned a decimal value, based on where in the range the variable was. If the variable was the best-case scenario then the decimal was 1.0, then get lower as it got worse in the range. The decimal would lower by a constant rate between 3 and 6 depending on the trait it related to. The entire research process was very iterative particularly this part. It consisted of altering the values slightly and then compiling and running the program and then going through fullerene graph images to see the effects. I started out making the modifier traits creating a score out of a hundred and then multiplying it by the decimal for crossing edges. The 2 most important modifying traits were edge length and angles. Using edge length seemed to also be effective for determining face area. The more uniform the edge length, the more uniform the face areas are. The second most important characteristic is the angles for determining convexity of the faces. Originally I made the initial rating composed of edge length and angles, which was then multiplied by symmetry and edge crossings. The results were not ideal, they were decently ordered but there were quite a few instances of a bad image being between the top ones and a good image being lower in the order than it should be. I decided to designate a small portion for symmetry since it is the least important but it was affecting the score more than it should. This is also the point when I added the straight-line function due to a few particular images that I thought always should have been higher despite tweaking values either way. The final values ended up being 55 designated for uniform edge length, 35 for convex faces, and 10 for straight edges and symmetry, then multiplied by edge crossings. After viewing the results for this equation I decided to lower the value of symmetry slightly by multiplying the symmetry modifier decimal by 0.75. The edge-crossing modifier is 1.0 if there are no edge crossings, .5 if some exist but less than 3, 0.4 if there is less than 6 and any more than that means 0.3. Overall the process of weighing the values came down to experimenting until I felt the results generated matched the results I collected by hand.

My research mostly came down to collecting data and writing code. The results matched my original results in the ways that I intended. One thing I noticed when comparing results was that the program was able to accurately rate them in some instances. In stage one I rated hundreds of images. I was pretty consistent in judging them but seeing a graph at the beginning of the process and rating it, then seeing a very similar graph an hour and several hundred images later sometimes lead to a slight discrepancy from human error. Allowing the program to rate them makes the scores that much more consistent. Something I would have liked to add was a function that actually calculates the area of faces based on the coordinates of its corners. I made an algorithm for creating triangles from the corners that worked for the pentagons in the graphs but was unable to complete the hexagon component. The issue was that instead of "pentagons and hexagons" they are merely 5 or 6-sided shapes with all sorts of side lengths and shapes. The algorithm might create a triangle between corners that covers an area outside the shape so you had to be very selective which vertices you used to branch out from to create triangles. I spent several days drawing 5-sided shapes in every possible shape in an attempt to cover every single possible scenario. The pentagon portion used a method of creating a rectangle around the shape of minimal size and then determining which vertices were on the border of that box and calculating which vertex of the inner vertices was in the middle of the others and thus could connect to the other vertices without crossing outside the shape. Once this vertex was chosen you just had to select the other vertices of the triangle and pass them to the function I wrote to calculate the area of a triangle. The hexagons were too many different possible scenarios and I was unable to complete it for this project. Despite the lack of calculating the area of faces, the algorithms I did calculate appear to produce results showing good face area distribution as higher scores by taking traits into consideration such as edge length. I would also like to expand on the traits and scores assigned for them to vary the scores a little more than they currently are.

For future research on this topic I hope to explore the drawing functions of FUIGUI and the programming in general. I would like to work on improving how FUIGUI creates graphs. Another future use is program different known drawing algorithms and using my program to judge the output. This would be an ideal way of not only having a consistent and impartial way of judging the algorithms but a user could rate many pictures at once thus getting a better idea of how effective an algorithm is than a human judging a few outputs of an algorithm.

Overall I concluded that the research I read about from past studies was accurate in general graph aesthetics. I expanded on that research by going mathematically defining the traits and how much they affect a graph image allowing a better understanding of them and writing a program to automatically identify them.

Bibliography

Ware, Colin; Purchase, Helen; Colpoys, Linda; McGill, Matthew

Cognitive Measurements of Graph Aesthetics

Information visualization. 1 (2002) no.2, 103

Huang, Weidong; Eades, Peter; Hong, Seok-Hee;, Lin, Chun-Cheong;

Improving multiple aesthetics produces better graph drawings

Journal of visual languages and computing. (2012) 1 – 11

Purchase, Helen;

Metrics for Graph Drawing Aesthetics

Journal of visual languages and computing. 13 (2012), no. 5, 501

Purchase, Helen; Pilcher, Christopher; Plimmer, Beryl

Graph Drawing Aesthetics – Created by Users, Not Algorithms

Visualization and Computer Graphics. 18 (2012), no. 1, 81 – 92

Purchase, Helen; Carrington, David; Allder, Jo-anne

Empirical Evaluation of Aesthetics-based Graph Layout

Empirical Software Engineering. 7 (2002), no 3, 233 – 255

Myrvold, W; Bultena, B; Daugherty, S; Debroni, B; Girn, S; Minchenko, M; Woodcock, j; Fowler, P.W;

FUIGUI: A graphical user interface for investigating conjectures about fullerenes

Communications in Mathematical and in Computer Chemistry / MATC. 58 (2007) no 2, 403 - 422