The Manipulation of Vertices in a 3D Space to Generate and 3D Print an Object using a Genetic Algorithm

By: Patrick Freel

The purpose of this project is to develop a genetic algorithm that is capable of generating a 3D printable DX dice, where X is the number of sides. Upon finding the fittest chromosome containing the desired size dice it would be sliced and 3D printed. The program begins by generating a number for the desired amount of sides that should be attributed to this chromosome of dice and the total population size. The population size can be from one hundred to five hundred in size while the dice can have a size between a D4 and a D120. Next vertices are randomly generated around the 3D plane, the amount of vertices generated is always twice the number of faces. Generation is limited to plus or minus five in the x, y, and z directions and each point in added to a list. Edges are then generated by going through each vertex in the list and generating a random partner. Each vertex should have between three and five edges, but will usually have more. To help facilitate the next step, at the end of edge generation for each vertex a connection is made between the first and last partner. This way it is guaranteed to have more complete connections. After face generation edges are not used again unless calculating a visual or 3D generation. Finally faces are generated. Each item in the list of faces is made up of three vertices that form a complete connection, thus making every face a triangle when viewed from above.

The two parent selection methods implemented are tournament selection and another tournament selection with a set tournament size of ten. A coin flip is used to decide which method is used each generation. Both begin by calling the fitness function and passing the population list. The fitness function evaluates the surface area of each face in a chromosome, appends each value to the corresponding index, then evaluates an average by taking the maximum value and subtracting the minimum value, appends it to a list and returns the list. Fitness also picks out the “elite” for each chromosome and appends it to the beginning of the parent list. It selects this “elite” by finding the closest fitness value to zero. Then selection for a tournament begins by generating a random tournament size. It then iterates through randomly generated picks and compares it to the best, previously selected, pick. Upon completion the new parent population is returned.

The parent population is passed to the crossover method which again flips a coin to decide between single point crossover and two point crossover. Single point crossover generates a random starting point. It then grabs the vertices from the faces index values from the chromosome at index and the next index. Since each face has three positions the decision was made to randomize which vertices would be switched, however they always switch with their corresponding axis. Since one vertex isn’t locked to a single face there is a theoretical chance to cross values between multiple vertices. For instance if index 0 is in faces (0, 5, 3), (2, 0, 7), and (21, 9, 0) there is a chance that 0 will swap vertices with 2 and 21. It then returns an updated version of the list it initially received. Two point crossover works exactly the same way except it generates a random starting and ending location.

The final stage in a complete iteration is the mutation. Mutation generates a testing threshold of 1/L where L is the length of the population. For each set of vertices in a chromosome a score is assigned and tested against the threshold. If the score is lower than the threshold that set of vertices is assigned a new set of randomized coordinates following the same generation guidelines outlined earlier. Upon completion the process begins again with the mutated chromosome in the parent selection method, best fitness results are displayed every fifteen generations. The maximum number of generations allowed is 250 or the process will break when the best chromosome value equals one.

The non-genetic solution utilizes the same generation code as the genetic solution to create a random size population and chromosome. This method is basically performing multiple hill climbing methods per population. The population is passed through the fitness function which returns a list of each chromosomes fitness again based on surface area. The population is then passed to the mutation function which returns a new mutated population. This new population is then passed through the fitness function as well. Once the fitness of each chromosome in both populations is calculated we run through a comparison of the two populations. If the fitness of a chromosome in the mutated population is better that chromosome replaces its lesser in the original list. The best fitness results are displayed every fifteen generations or when the best fitness equals one, at which the program ends.

Future additions to the project would include refinements to the fitness function, a built in compatible 3D printing file formatter, and threading. In its current state the project, hopefully, displays an adequate use of genetic algorithm processing and generation. However its original purpose to generate various faced die is not reached. An addition to the fitness function to monitor and promote better distancing between vertices and form a complete object, versus its current messy state, would be needed. With the ability to produce a complete 3D object implemented, the ability to automatically generate an STL file containing the fittest object would be optimal. Without this step human data entry is required to copy the fittest vertices, edges, and faces into an object editor such as Blender or OpenSCAD for exportation to a compatible file format. Due to the copious amounts of calculations required it would be beneficial to implement threading capabilities for parent selection, fitness calculation, and crossover. This would increase the speed and efficiency greatly, especially since a core on a Ryzen 7 1700 will hit peak when running.

The two background papers are the same ones I presented and are reattached in the submission folder.

GA Results with Population size of 50 and Chromosome size of 4 and 100 iterations:

|  |  |  |  |
| --- | --- | --- | --- |
| 10 Tournament  Single Point | 10 Tournament  Double Point | N Tournament  Single Point | N Tournament  Double Point |
| 2.202 | 0.959 | 0.935 | 2.478 |
| 1.366 | 1.659 | 1.169 | 1.396 |
| 1.465 | 1.701 | 1.617 | 1.707 |
| 1.574 | 2.518 | 2.116 | 0.875 |
| 1.435 | 0.992 | 1.549 | 1.157 |
| 2.653 | 1.159 | 1.506 | 1.032 |
| 1.075 | 0.537 | 0.898 | 2.084 |

GA Results with Population size of 50 and Chromosome size of 20 and 100 iterations:

|  |  |  |  |
| --- | --- | --- | --- |
| 10 Tournament  Single Point | 10 Tournament  Double Point | N Tournament  Single Point | N Tournament  Double Point |
| 13.298 | 10.372 | 12.807 | 12.183 |
| 8.502 | 9.587 | 10.287 | 11.547 |
| 10.829 | 12.122 | 11.105 | 12.667 |
| 10.295 | 12.259 | 11.325 | 13.357 |
| 10.681 | 11.496 | 13.435 | 13.225 |
| 12.428 | 13.371 | 14.191 | 12.334 |
| 10.502 | 13.054 | 12.115 | 11.762 |

Hill Climbing Results with population size of 50, Chromosome size of 4, and 100 iterations:

|  |
| --- |
| 4.581 |
| 4.25 |
| 3.634 |
| 4.338 |
| 4.303 |
| 4.863 |
| 3.438 |

Hill Climbing Results with population size of 50, Chromosome size of 20, and 100 iterations:

|  |
| --- |
| 14.826 |
| 14.563 |
| 15.451 |
| 13.312 |
| 15.135 |
| 14.776 |
| 14.855 |