The program begins by accepting a .obj voxel file uploaded by the user. It then reads through the file line by line, looking for the following at the beginning of each line: “vn”, “vt”, “v” or “f”. It ignores the other lines. It sorts the usable lines in the following manner:

* “vn” lines are added to an array of vertex normals (they indicate the outward facing side of the face)
* “vt” lines are added to the texture coordinate array (these indicate the position of the texture/color in the .png associated with the .obj file)
* “v” lines are added to the vertex array (these contain the x/y/z position of the vertex they represent). It also compares these to the current largest and smallest values for each, to determine the eventual maximum and minimum values of the object.
* “f” lines contain the per-face information and are sent to the “setMatrix” function, which will be explained later (these lines contain code for the normal, texture coordinate and three coordinates associated with the face they represent)

“setMatrix” extracts the values of the normal, texture coordinate and the three vertex coordinates from the line and uses them, in conjunction with the aforementioned arrays, creates a new “face” object, containing all the pertinent information about each face. It then normalizes the coordinates by rounding them to the nearest whole number and adding the lowest x, y or z value (depending on which axis they belong to), to them to ensure the lowest possible coordinate is 0 while maintaining the correct orientation with the other blocks. It adds this “face” object, with the normalized values to the array of “face” objects.

From there these objects are sent to the “addCoords” function, where the normal are used to deduce which two dimensions are being altered in the face. It then compares the relevant coordinates of each of the three vertexes to determine which is the right angle (which will be used later). After it has set this information, it uses it to create an “angle code”, indicative of the right angle of the face e.g. an increase in the x axis and a decrease in the z axis result in a code of +/0/-.

From here it enters “buildMatrix”, where the final, three-dimensional matrix is created using the max values of each dimension. “buildMatrix” sends the faces one at a time into the function “add”. “add” is the crux of this section of the code. Using the normal to deduce the two altering dimensions, it uses the two angles, other than the right angle, to calculate the values of gradient and intercept in the straight line formula y = mx + c (or y = mz + c or z = mx + c). It then uses the resulting formula to calculate the location of the coordinate of the slope of the hypotenuse for each value of the corresponding, alternate side. Using the angle code to infer the orientation of the face, it determines whether the point calculated on the hypotenuse is the starting point or end point of the next function “putIn” which cycles through each level of the face, adding each block from the point of origin to the end point (each being either the point on the side of the face or the point calculated on the hypotenuse). Once this process is complete, the final matrix is ready to be transferred.