The purpose of this project is to apply the mathematical concepts of matrices and transform a 3D-object that is drawn on a 2D screen. These transformations are rotations over the object, rotations over the origin, and translations to a specific point. The program uses these matrices to calculate the new point of where the old point will be after the transformation is completed.

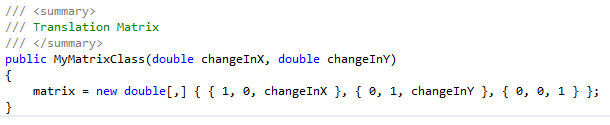
**Translation**:

To be able to translate an object to a given points, an identity matrix is required. This matrix takes the points of the object, and performs a simple matrix multiplication with a vector.

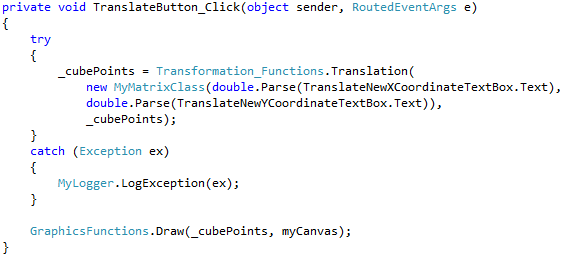
\* =

This calculates a new point which can then be used to draw the new point on the screen. This can be done for every point in the object to move the entire object.

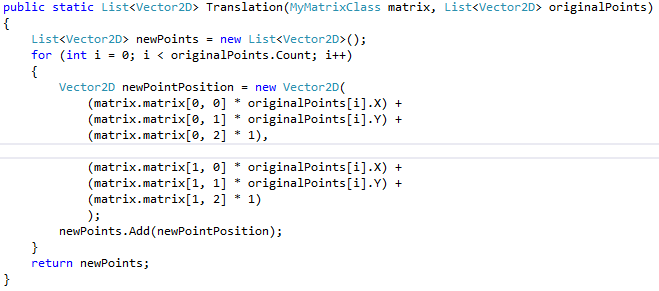
In code, the first step is to let the user enter the point to which they want to translate the point through a user interface textbox. This text is parsed and then given to the MyMatrixClass which stores the entered points in the identity matrix above as ∆ and ∆.



Once this is completed, the translation can be started. The Transformation\_Functions class contains the three basic transformations. The method that is called for the translation is the Transformation\_Functions.Translation method, which takes two parameters, the matrix that was just created, and all of the original points with which the matrix is multiplied.



In the Transformation\_Functions.Translation method, a for loop goes through each point in the list where each point is multiplied by the first, second, and third element of the matrix in order to complete the matrix multiplication.



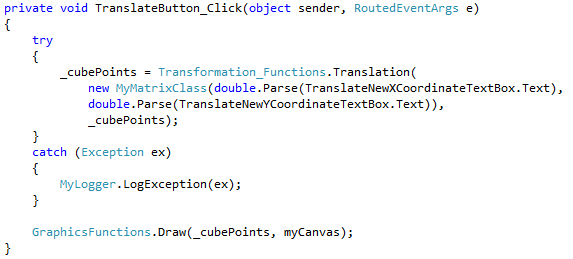
The points are then redrawn through the GraphicsFunctions.Draw method. The translation is then finished with the translated object shown on the screen.

**Rotation around the Origin:**

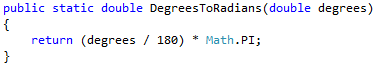
To rotate the object around the origin, a rotation matrix is required. The user inputs an angle, in degrees, to determine how much the object will be turned. The rotation matrix is then multiplied by the vector containing the original and coordinates.

\* =

The end result of this calculation, is a new point which has been rotated over the origin. This needs to be done for every point that is part of the object in order to rotate the entire object.

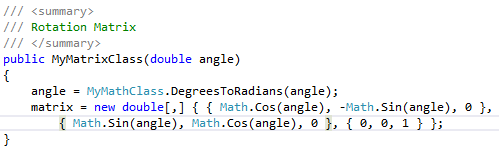


In code, the user first enters the degrees that he wants to rotate the object. This value is parsed from a textbox and the degrees are converted to radians in the MyMathClass.DegreesToRadians

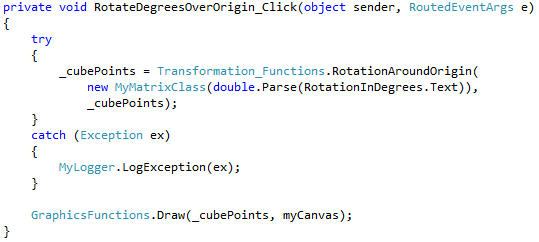


which uses the following formula:

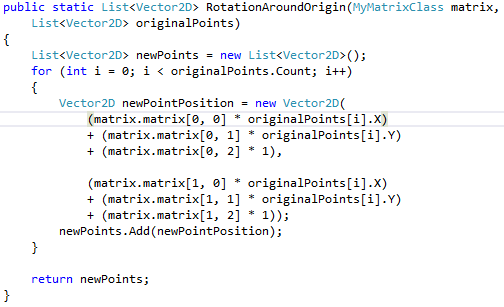
These degrees are passed on to the MyMatrixClass where the radians are stored for later calculations.



The next step is to perform the calculation which is done by calling the Transformation\_Functions.RotationAroundOrigin Method.



The Transformation\_Functions.RotationAroundOrigin method multiplies the recently created rotation matrix with all points of the object.



The new points are then returned so that they can be drawn on the canvas.

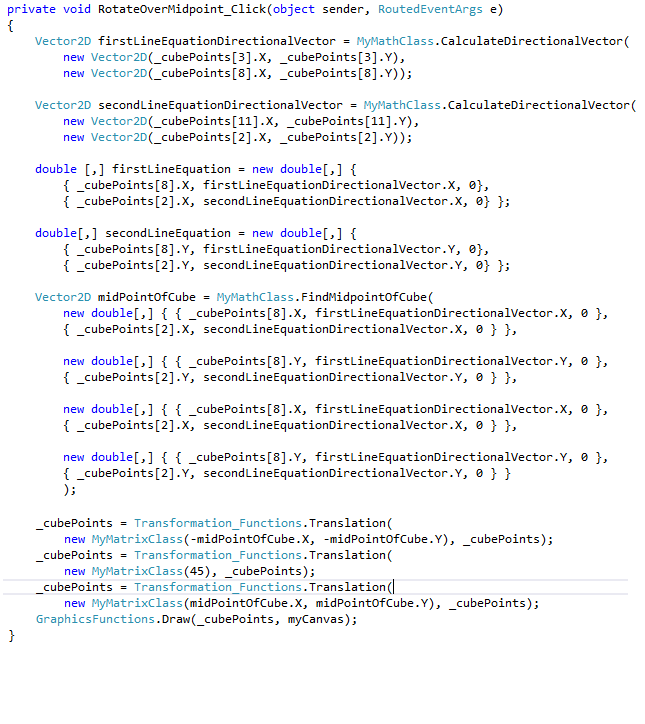
**Rotation around the object:**

The last basic transformation uses both the identity matrix and the rotation matrix. The reason for this is that the object first needs to be moved to the origin. It is then rotated the degrees that the user enters, and then moved back to its original position. To move the object back to its original position, the midpoint is found since the midpoint of the object doesn’t change unless the object is moved to place besides its original position. First, the program needs to find the midpoint of the object. In this case, the object is a cube. The midpoint of the cube is where 2 diagonals meet. The diagonals are created by setting up the two equations of a line using the following form:

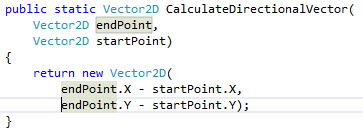
g:X =

This is done twice and then they must be set equal to each other. The next step is to set up a system of equations and solving for the variables. Once the variables have been solved for, they must be plugged back into one of the original line equations. The midpoint has then been found. The second step is to use the translation matrix to move the points the negative amount of the midpoint in order to get to the origin. Once the points have all been moved, the rotation matrix is used to rotate the points around the origin. Lastly, the points are all moved back with the translation matrix by using the positive values of the midpoint. The object has then been rotated successfully.

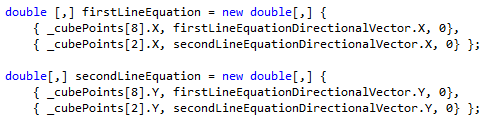
In code, the MyMathClass.CalculateDirectionalVector is called twice to calculate the directional vector needed for the two line equations.



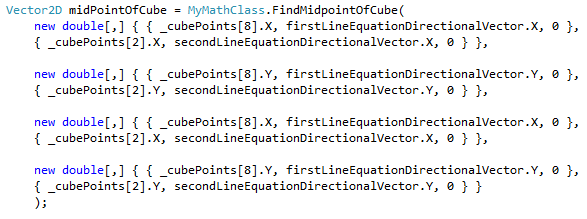
The MyMathClass.CalculateDirectionVector calculates the directional vector needed for the line equation by subtracting the starting point from the end point.



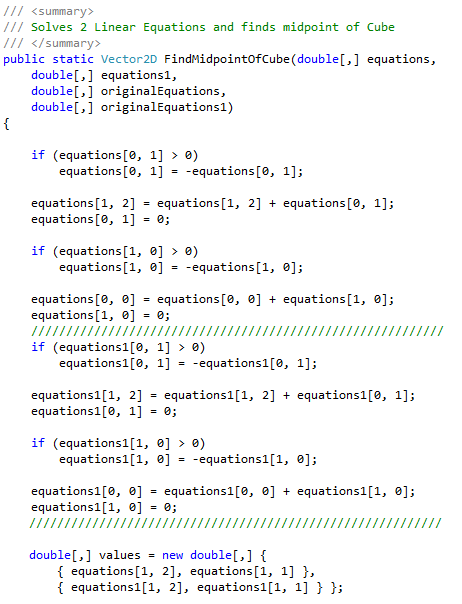
Once done, two new 2D arrays are declared where the equations are set up.



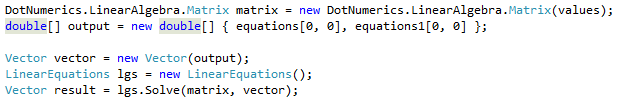
After this step, the computer can calculate the midpoint of the cube with the MyMathClass.FindMidpointOfCube using these two 2D arrays.



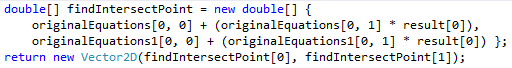
The linear system of equations is solved by checking first element of the first 2D array containing the first 2 equations. If it is positive, it is changed to being negative and then added to the first element of the second equation. The second element of the second equation is then checked if it is positive. If so, it is changed to the negative value and then added to the first equation. This is done with the second 2D array to get the variables on one side and the normal values on the other side.



To solve the system, the DotNumerics dll is used. It requires a matrix of variables and a vector of expected results.



This returns a vector which are the x and y values. To find the intersecting point of these two lines, either the x or the y values needs to be plugged into the original equations and the results needs to be returned.



Now, the translation can take place the same way the translation is performed in a normal translation. Once it has been translated to the origin, the object can be rotated at the angle which the user specified. This rotation is the same kind of rotation as a normal rotation over the origin. Now that the object has been rotated correctly, it can be translated back to the original midpoint. This is done by performing another translation just this time using the positive coordinates of the midpoint instead of the negative coordinates.