|  |  |  |
| --- | --- | --- |
| Date | Details of implementations | Details of problems and how they’re fixed |
| 27/10/21  (In tutorial) | Week 5, tutorial 1:   * Extended vertex class   + Included a ‘z’ dimension, included a float variable named z in the vector header file then added z to all methods in vertex class   + Adding z is the same as how x and y were implemented, just changed the variable to z and \_z for private variable * Replaced the 3x3 matrix class with a 4x4 matrix class   + Because of how the matrix class is implemented, the methods count through all the rows and columns, so all I needed to do was change the global variables ROWS and COLS to equal the value of 4 this means there is no need to change each method * Test program   + Vertices in the square have been set to 0   + Each matrix in the rasteriser.cpp file have been updated to handle a 4x4 matrix instead of the original 3x3 |  |
| 28/10/21  29/10/21  30/10/21  (free time) | Week 5, tutorial 2   * Removed DrawSquare method from the rasteriser class, as well as all the code from the update and render methods   + Render method has been updated to just contain a call to the bitmap clear method, changing the background to RGB white * Creation of a class to hold polygon information   + Creation of class and header named Polygon3D   + Polygon3D holds a container containing all the indexes for the list of vertices   + An array of 3 indices is used to store each vertex of a polygon, ‘getters and setter’ methods implemented   + A method created to set each value to 0, a setter, an accessor, and operator method * Creation of a model class, will hold the 3D model information – store list of vertices and list of polygons that will be loaded   + Creation of two methods, GetPolygons and GetVertices to return polygons vector and one to return vertices vector   + Another two methods, GetPolygonCount and GetVertexCount one to return the vector size of polygons and one for vertices   + Creation of AddVertex method to use method parameters, x,y,z to create a vertex of x, y and z coordinates, this vertex is pushed back onto the vertices vertex   + Creation of AddPolygon method to create a Polygon3D from the Polygon3D class using parameters, then the Polygon3D is pushed back onto the polygons vector * MD2Loader.cpp and MD2Loader.h files (MD2Loader class) added to project   + MD2Loader class used to populate vertex and polygon lists in the model * Instance of model class added to rasteriser class and inside the rasteriser initialise method the static method LoadModel is called to load the model into memory * Filename is changed to local directory to load the cube.md2 file |  |
| 3/11/21  (In tutorial) | * Updated model class to include two collections, one to load polygons from MD2 file and other for vertices. All transformations to be done of the vertices   + Method ApplyTransformToLocalVertices created, this method applies the multiplication transformation matrix to each vertex from the original vertex from the MD2 file and stores these results in the transformed vertex variable and then pushed back onto the \_updatedVertices vector. Using a separate vector variable allows the original vertices to be overwritten. updatedVertices is cleared every iteration to avoid erroneous data   + Method ApplyTransformToTransformedVertices created, this applied the transformation to each vertex in the transformed vertices collection. So each updatedVertices vector is multiplied by the transform and written back to that same updatedVertices pointer * Dehomogenize method needed, used too dehomogenize each vertex in the transformed vertices collection   + Dehomogenize method added to the Vertex class, each vertex x, y and z is divided by w and added back to their respected variable, e.g. \_x = \_x / \_w   + An ApplyDehomogenizeToTransformedVertices method is created to apply the dehomogenize from the vertex class to each vertex in the updatedvertices vector |  |
| 5/11/21  8/11/21  (free time) | * Creation of camera class   + Header includes a Camera constructor that passes the floats xRotation, yRotation, zRotation and a vertex containing the camera position   + Mutators included to change the initial values   + Accessors and mutators are added for x,y,z rotation   + Method added to the camera class to create and store the viewing camera matrix     - For the viewing matrix, 4 matrixes are created, an x,y,z rotation transformation and a translation transformation for the camera position. These 4 matrixes are multiplied together providing the viewing matrix |  |
| 8/11/21  (In tutorial) | * + A mutator is added to get the viewing camera matrix from the camera class   + Instance of camera class is added to the rasteriser class header for the cameras default position of -50 on the z axis and a new instance of camera * Add methods to create perspective and screen matrices to the rasteriser class   + Two new methods added to the rasteriser class, GeneratePerspectiveMatrix and GenerateViewMatrix   + The GeneratePerspectiveMatrix takes two parameters, float d and float aspect ratio which is used to calculate the perspective matrix   + The GenerateViewMatrix takes 3 parameters, float d, int width and int height. These are used to generate the screen transformation matrix |  |
| 9/11/21  (Free time) | * Add a DrawWireFrame method to the rasterizer class   + Bitmap object is taken as a parameter, this will be used to generate the HDC, used to draw the graphical components   + A pen is created with the RGB colour blue, which will draw on the selected bitmap   + A for loop is used, using the GetPolygonCount from the model class to go through each polygon in the polygons collection   + Within the for loop 3 vertexes are initialised and will change through each iteration of the loop. These vertexes hold the 3 different vertices that make up the triangular polygon. For the points, a instance of model is used to call the getUpdatedVertices method, this method returns a vertex indices from the updated vertices vector using a call do GetPolygons and its GetIndex position   + Using these 3 vertices a line can be drawn from each point to draw each individual triangular polygon |  |
| 10/11/21  (In tutorial) | * In the rasteriser class, instance variables were added for the current model transformation and screen view transformation * Add code to the update method in the rasteriser class   + Update to the model transformation by creating a matrix that will rotate the model by the y-axis   + Assigning the newly created rotation matrix and storing it in the currentModelTransformation matrix   + A call is created to the GeneratePerspectiveMatrix to create the perspective matrix     - Two parameters are passed, d is set to 1 and the aspect ratio is the bitmap width divided by the bitmap height   + A call is created to the GenerateViewMatrix to create the screen matrix     - Three parameters are passed, d is 1, bitmap width and bitmap height | * Trouble converting bitmap.GetWidth and bitmap.GetHeight to float values that are accepted by the GeneratePerspectiveMatrix method that is called in the Update method within the rasteriser class.   + Quick fix, converted values to float |
| 12/11/21  (In tutorial) | * Add code to the render method in the rasteriser class   + Bitmap clear method call was kept in the create the white background bitmap   + Applied the model transform to the local vertices in the model by calling the ApplyTransformToLocalVertices method from the model class, passing through the model transform class variable which was updated to contain the current model transformation   + The viewing camera transform was applied to the transformed vertices by calling the ApplyTransformationToTransformedVertices method passing through the newCamera instance from the rasteriser class with a call to the getCameraMatrix method   + Perspective transform is applied to the transformed veryices by calling the ApplyTransformationToTransformedVertices and passing through the perspectiveMatix matrix that was updated within the GeneratePerspectiveMatrix method   + A call to the ApplyDehomogenizeToTransformedVertices method   + Apply the view screen transform to the transformed vertices by calling the ApplyTransformToTransformedVertices method and passing through the screenTransformation matrix that was updated within the GenerateViewMatrix   + A final call is made to the DrawWireFrame method, to draw all the lines, passing through the bitmap object   + Within the camera class, in each mutator method for the x,y,z rotation a call to the viewingMatrix is added to update the viewing matrix each time a rotation Is set | * Was having trouble viewing the cube, when the application is run the camera perspective was within the cube instead of -50 in the z axis away from the cube   + Within my rasteriser the newCamera class was being overwritten by an accidental call added within the rasteriser initialise method that was left from testing, this was removed and the cube is drawn correctly |
| 13/11/21  (Free time) | * Start of the backface culling implementation to remove the vertices that should be out of view certain perspectives |  |
| 17/11/21  (In tutorial) | * Within the model class, a new method CalculateBackfaces is called, with the parameter being an instance of Camera, which will be used later to get the camera position   + Using a for loop to go through each polygon is the model, using method GetPolygonCount to get the number of polygons loaded, 3 vertexes are created (vertex0, vertex1, vertex2), to store 3 vertices, each representing a point location of the triangular polygon   + 2 Vector3D instances are also created, these are construct a and b     - vectorA is the result of subtracting vertex 1 from vertex 0     - vectorB is the result of subtracting vertex 2 from vertex 0   + A normal vector is calculated by assigning the class variable \_normalVector the result from calling the VectorCrossProduct method passing through the parameters vectorA and vector   + Another Vector3D is created named eyeVector, calculated by subtracting camera position from vertex 0 |  |
| 18/11/21  19/11/21  (Free time) | * + The dot product is calculated by initialising a float variable, dotProductResult, and assigning it a float value by calling the DotProduct method from the Vector3D class passing through the parameters \_normalvector and eyeVector   + An if statement is then used to see if the dot products result is less then 0, if so the polygon is marked for culling and later shouldn’t be drawn by the DrawWireFrame method. Those that are marked for culling will have their value set to true, so each instance of polygon that is to not be drawn will be set to true * An else statement is also used to set the \_polygonCulled from that instance of the Polygon3D class to false * The DrawWireFrame class has been altered so only the polygons that are seen by the camera are drawn   + An if statement is added, if a polygons backface culling is equal to false, it is drawn | * Some lines are removed, but not all of them, from the debugging many dot product results that are below zero are still being recognised as false instead of true when called in the rasteriser class   + Issue resolved, an extra ‘-‘ sign was added into the crossProduct calculation, resulting in erronous data being passed through, this was removed and the cube is drawing correctly   + Now only the polygons that the camera is facing are drawn |
| 19/11/21  22/11/21  (In tutorial) | * Implementation of depth sorting (Painter’s sort algorithm)   + Depth sorting makes sure polygons further away from the camera are drawn first and the polygons closer are drawn last. By doing this, the polygons closest to the camera will always be on top of whatever was drawn first, so polygons that are behind are not drawn on top   + A new method is added to the model class, sort, sort will go through each polygon, collect the average z coordinate from the 3 different vertices and will store the average (mean) z vertices back within each of the polygon class objects   + In the polygon3D class two new methods are created, setPolygonZ and getPolygonZ, the accessor and mutators are used to store and retrieve the average z coordinate from the polygon class object   + First the method is created, with a for loop that will go through the number of polygons loaded from the model file (MD2 file).   + Within the for loop, each vertex in the polygon all 3 points are stored in 3 separate variables. Using these variables the mean average is calculated and stored into the vector holding the polygons through the use of the class object polygon3D, using the mutator setPolygonZ   + A new binary predicate (static method) is created within the model class, ascendingSort, this will be used when sorting the polygons in order from furthest away (smallest number) and closest (largest number). This method will be used when calling the library sort method from <algorithm>. This method will accept two parameters, the first being the left hand side (lsh) which is the beginning value in the \_polygons vector and the second parameter being the right hand side (rhs) being the end/last value in the \_polygons vector. Within the method, the comparison Boolean result from the lhs z value < rhs z value is returned, this is the iterator required by the sort() method call   + Outside the for loop, the sort library method is used, this method will sort all entities inside the vector \_polygons using the average z from furthest away to closest. The smallest number is further away and the biggest number is closer. 3 iterators (parameters) are passed through, the beginning of \_polygons, the end of \_polygons and the call to the binary predicate, using these iterators   + Finally within the rasteriser class, in the render method a call to the model class method ‘sort’ is added, this is added between the world transform and calculate backfaces methods |  |
| 24/11/21  25/11/21  (In tutorial) | * Drawing a solid figure   + A new method is created in the rasteriser class, DrawSolidFlat, this method will the replacing DrawWireFrame. DrawSolidFlat will be using the polygon() method from the c++ standard library to draw actual polygons instead of lines to create the triangle shapes   + The method is created with the bitmap as a parameter, just like DrawWireFrame, within the method the hDc declared in a variable and the pen and brush is created. With the colour set to a light blue RGB(73,243,238)   + A for loop is used to go through each polygon loaded from the model, this loop will be used to draw the polygons     - Within the for loop each polygon updated vertex containing the x,y,z for each of the vertices is retrieved and stored in a vertex variable. So 3 different vertex variables are used to store each point of the polygon.     - An if statement is used to detect if the polygon is culled or not, just as in the DrawWireFrame method. Within this if statement 3 POINT type objects are created to store the 3 x and y points of the polygon. An array is created of type point to store each of the 3 points used to form that specific polygon. Next within the if statement the polygon is drawn using the polygon() default library method. This method takes in 3 parameters, the hDc brush, the polygon array and the size of the array.   + A call is made to the method DrawSolidFlat in the render class replacing the DrawWireFrame call | * Was getting errors when retreiving the GetX and GetY for the POINT type objects, was getting an error due to can’t convert type float to long   + Issue was reolved by adding a (long) declaration before each of the point accessors for X and Y |
| 26/11/21  (Free time) | * Added reflection coefficents to the model class, within the model class these were created and within the Model method these were set |  |
| 26/11/21  27/11/21  (In tutorial) | * Implementing directional light source   + Creation of a new class, DirectionalLighting, this is to hold the information needed for a directional light   + The information stored is r,g,b light intensitys which will be a value 0-255 and a vector object to store the direction of the light source   + Mutators and accessors added   + Within the rasterier class a vector collection was declared with the object type DirectionalLighting, a Vector3D object for the lightDirection and a DirectionalLighting object which will store the r.g.b and light direction which will be pushed onto the collection   + Within the rasteriser class and in the initialise method the light direction is declared as well as the directional lighting which is pushed onto the vector collection   + Within the model class a new method is created, CalculateLightingDirectional, which takes the vector collection as a parameter   + Temp r,g,b holders are created, these will be altered for each polygon in the model, these temp values are assigned the r,g,b light intensity that was initially declared in the rasteriser class   + The temp values are then multiplied by the r,g,b corresponding diffuse reflectance coefficients   + The light direction is normalised and the normal vector is normalised. These are used to calculate the dot product,. The dot product is used to indicate which polygons are in view and which need directional lighting or not   + Using the dot product only the polygons that are seen by the directional lighting have the lighting applied to. The total r,g,b is then equal to the temp r,g,b mulitplied by the dot product. With the use of clamp, the r,g,b values are checked to see if they’re in the range 0-255 and then applied back to the polygons set colour property. This is done for each directional light in the collection for each polygon   + Within the rasteriser class, in the render method the CalculateLightingDirectional call is added after the CalculateBackfaces, passing the directionLights parameter |  |
| 29/11/21  (In tutorial) | * Implementation of ambient light   + New class created to store the details of the ambient lighting, storing r,g,b intensity   + Within the rasteriser class a AmbientLighting object is created to declare the ambient light insensity   + In the initialise method in the rasteriser class the ambient lighting intensity is set   + Within the model class a new method is created, CalculateLightingAmbient   + Within this method the light intensity declared in the rasteriser class is multiplied by the reflectance coefficients and stored back into the polygons setcolourproperty to be drawn   + Within the rasteriser class, render method, a call is made to the CalculateLightingAmbient method, this is culled just before the directional lighting and behind CalculateBackfaces and this passed the parameter ambientLights |  |
| 30/11/21  (Free time) |  | * Was receiving memory leak errors   + After debugging I found its source to be the DrawSolidFlat method. The brush object wasn’t being deleted after every creation, so deleteobject was added to fix this |
| 01/12/21  (In tutorial) | * Alterations to the CalculateLightingDirectional method within the model class   + To include the colour intensity already set by the ambient lighting, the colours already set by the ambient lighting are required within the method. To do this a colour reference object is created to hold the colours from the selected polygon, the r,g,b values are then collected from the colour reference and assigned to the total r,g,b values. By doing this, the polygons that are not in the direction of the directional lighting are not altered and stay the same colour set by the ambient lighting, which creates the directional lighting effect   + So the polygons that are not effected by the dot product stay the same ambient colour, those that are effected have the directional lighting colour added to the ambient lighting   + To test and demonstrate the effects of the ambient and directional lighting I set the ambient lighting to a burgundy colour and the directional lighting is set to a green |  |
| 02/12/21  03/12/21  (Free time) | * Implementation of the point light   + A new class is created, point lighting, this will store the information used to correctly draw the point lights   + Within the point lights class, methods are used to access and mutate r,g,b intensity, a,b,c coefficents (used to calculate the attenuation) and the point light direction   + Similar to the directional lighting, in the rasteriser class objects are created for the vector collection housing PointLighting objects , a vertex object to hold the lightPosition and a PointLighting object   + Within the rasteriser class, initialise method the light position was set. Light intensity and a,b,c coefficients were set and these were all pushed back onto the pointLights collection |  |
| 04/12/21  (Free time) | * + Within the model class, a CalculateLightingPoint method is created. This will be used to calculate the point lighting   + For each polygon the r.g.b values are collected and stored, this will be used to either set the polygons the same colour as set in the directional lighting or apply the point lighting ontop of certain polygons, this will all depend on the dot product and the direction of the point light light source   + For each light source from the collection, the colour intensity coefficients are applied to the colour intensity   + The first vector from each polygon is collected, this will be used to calculate the distance between the light source to the vector. Each light position is subtracted from the vector (x-x,y-y,z-z) and assigned to a Vector3D object vectorToLight   + Using the Vector3D object the distance is calcuated   + The vectorToLight is then normalised |  |
| 06/12/21  (In tutorial) | * + The attenuation is calculcated using the pointLights a,b,c coefficents and is multiplied by 100, as a ‘fudge factor’   + Another Vector3D is object is needed, this hold the normalVector from the current polyon. This is then normalised   + The dot product is then calcualted using the vectorToLight and normalVectorNormalise   + If the dot product is more then zero, the polygons that are within the view of the point light are changed. The temp r,g,b values are multiplied by the attenuation and then clamped if necessaryto keep them between 0-255   These are then applied back to the setColourProperty for each polygon |  |
| 09/12/21  (Free time) | * After resolving the issue, all 3 lights are working, ambient is purple, directional is green and point light is blue | * Had trouble drawing the pointlighting, was drawing incorrectly, colours were rendered very glitchy and static      * + Issue resolved, vector to light calculation was calculated incorrectly, mistakenly took away the vertex points from the point light, instead of point lights from the vertex points   + When also calculating the distance, I didn’t include the z coordinate, this was added and the code runs smoothly |
| 10/12/21  (In tutorial | * Implementation of flat shading using my own rasterzation code * For my own rasterization code I will be following the standard algorithm that calculated the slope for each side, filling in a flat-bottom triangle and a flat-top triangle * I chose to use the **standard algorithm** because from readying articles about the different algorithms standard works the best in terms of execution latency performance, it runs smoother compared to Barycentric algorithm which may have simplicity but that does come at a price of low execution speed comapred to the other two approaches. I prefer the approach of the standard algorithm where it draws the flat top and flat bottom triangles using interpolation * Addition of new methods in the rasteriser class: MyDrawSolidFlat, FillPolygon, FillBottomFlatTraingle and FillTopFlattriangle methods |  |
| 10/12/21  (Free time) | * Created GetVertexRGB and SetVertexRGB methods within the Vertex class, will be used when generating normal vectors for each vertex later on   + Accessors and mutators are added to the Vertex class, these will access and alter the colour of the RGB values for each vertex within a polygon, these colours are stored within the class object and will be used for my rasteriser using the Gouraud shading algorithm |  |
| 12/12/21  (Free time) | * Implementing algorithms within the MyDrawSolidFlat, FillPolygon, FillBottomFlatTraingle and FillTopFlattriangle methods * MyDrawSolidFlat   + For each polygon, all 3 vertices within the polygon are assigned to a local variable   + For the 3 local variables, the VertexRGB value is set to the colourref value from the polygon   + If the polygon is backfaced culled, the 3 vertices are pushed back onto a vector of type vertex and passed through to the FillPolygonGouraud method as a parameter   + After the method instance for FillPolygonGouraud has been executed the culledPolygonVertices vector is cleared * FillPolygonGouraud   + A new method is created, sortVerticesAscendingByY, used for standard algorithm, sorts verticies by Y values, smallest to largest, used for the sort <algorithm> library method   + Sort <algorithm> library method used, passing parameters the first vertice and last vertice in the vector store, with the sortVerticesAscendingByY method   + Once the vertices are sorted, the 3 vertices stored within the vector list are assigned to local variables   + The RGB values stored within the vertices are accessed and stored into COLORREF local variables   + If statements are used to determine if the polygon is FillBottomFlatTriangle or FillTopFlatTriangle * FillBottomFlatTriangle   + In this method as we will not fill in a complete line using polygon, but instead we will loop over all pixels of a horizontal line, we will be using a predefined direction from left to right. So x1 must be smaller than x2 so slope1 must be smaller than slope2. If not, we do a swap for both position and colour.   + With every horizontal line that is drawn, x1 and x2 values have slope1 and slope2 added to them respectfully, as well as their colour values * FillTopFlatTriangle   + In this method as we will not fill in a complete line using polygon, but instead we will loop over all pixels of a horizontal line, we will be using a predefined direction from left to right. So x1 must be smaller than x2 so slope1 must be smaller than slope2. If not, we do a swap for both position and colour.   + With every horizontal line that is drawn, x1 and x2 values have slope1 and slope2 values taken away from them respectfully, as well as their colour values taken away * Render method altered to include MyDrawSolidFlat method to be rendered |  |
| 13/12/21  (Free time) | * For the ‘for’ loop that draws each individual pixel using scanlineY variable comparrision to the Y axis, converted float values to integers and rounded the integers | * Debugging code, altering C++ code that was converted from java script, finalising a few things to get it working |
| 14/12/21  (Free time) | * Altered the Y Axis sort method by changing the sign to less than ‘<’ and the model draws again using setpixel (changed background to white for demonstration purposes) | * Colours are not being drawn on the model correctly using my rasteriser method, need to debug more to solve issue   + Implementation of colour solved, within the vertex class I needed to add constructors to allow VertexRGB colours to be set using the ‘=’ operator |
| 15/12/21  (In tutorial) | * An additional value needed within the Vertex class to store the normal vector, accessor and mutators added for the NormalVector, GetNormalVector and SetNormalVector * An additional value needed within the Vertex class to store the vector contribution count, accessor and mutators added for the count, GetVertexCount and SetVertexCount * Implementation of CalculateNormalVectors method for Gouraud shading within the model class   + For each vertex, normal is set to (0,0,0) and contributing count to 0   + For each of the 3 vertices, for the normal vertex, the polygon normal is added to its value and the contributing count is incremented by 1   + After all the polygons are processed for each vertex the summed vertex normals are divided by the number of times   + Next the NormalVector is normalised | * Error with CalculateNormalVectors implementation, code is not correct, need to debug to figure out the issue |
| 15/12/21  (Free time) |  | * After each polygon is processed, and when the summed vertex normals are to be divided by the count, the normal vector values are set to nan(ind), an error has occurred somewhere    + Issue revolved by including the NormalVector set inside the count for loop instead of its own separate for loop * Need to store the new normalVector for each vertices |
| 16/12/21  17/12/21  (Free time) | * Implemented DrawString method, this will display on screen what type of rotation, scaling and drawing method is manipulating the 3D model on screen * Altered render method to render different transformations, scaling, drawings separately in order |  |
| 17/21/21  (In tutorial) | * Modified CalculateNormalVectors class as colours for directional lighting were not showing at all | * Error running CalculateNormalVectors method    + CalculateNormalVector call commented out |