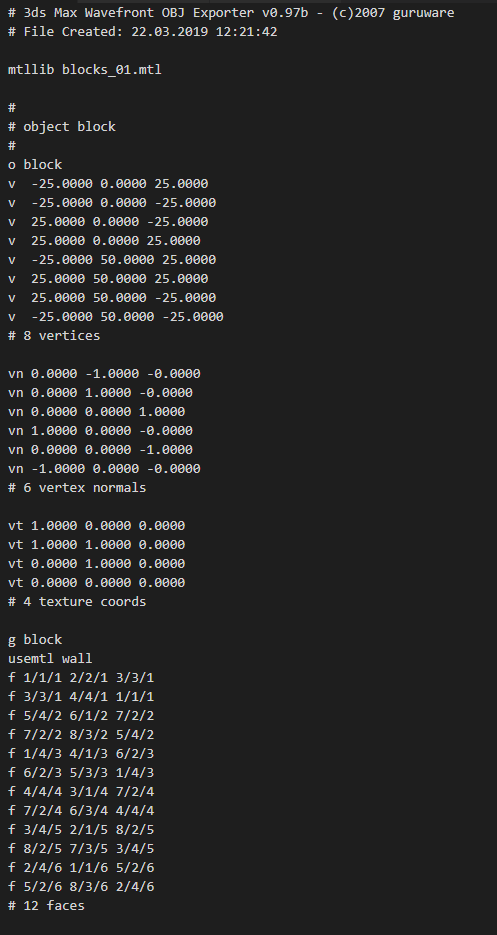
# Loading Object Files

## Brief discussion of OBJ format

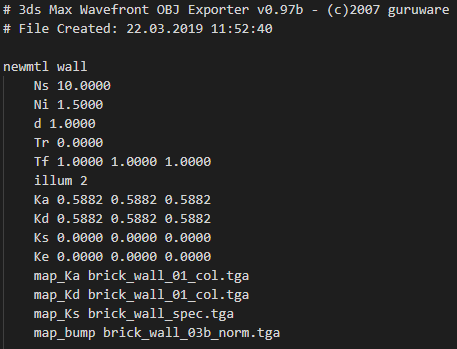
Let’s start by talking about what an OBJ file is made of. If you open an OBJ file in you text editor of choice, you will notice that it’s just plain text.



You should also notice that almost every line starts with a letter of a # symbol.

|  |  |
| --- | --- |
| Symbol | Meaning |
| # | This character indicates that the following line is a comment and not part of the mesh data. |
| o | The name of the object (as defined in the 3D package that created it) |
| v | The coordinates for a vertex. Should be followed by 3 numbers. |
| vn | Vertex normal information.  Which way the normal faces. |
| vt | Vertex texture coordinates.  The UV information for the vertex. |
| g | The name of this part of the geometry. Objects can be comprised of 1 or more geometric pieces. Usually this is named the same as the object itself. |
| f | Index data to construct a face.  All faces are assumed to be triangular.  3 sets of 3 numbers are used to reference the vertex position, the texture coordinates and the vertex normal. These 3 combinations of data, define a unique vertex point. |
| mtlib | Defines the material library file to use for this object |
| usemtl | Defines which material to use from the material library. |

If materials have been applied to the object, then an MTL file is generated. MTL files are used to reference the textures used on the object or objects and the ambient, diffuse and specular properties of the material.



|  |  |
| --- | --- |
| Symbol | Meaning |
| # | Defines that the following line of text is a comment |
| newmtl | Defines that the following information is for a new material and the materials name. this name is used in the OBJ file as a reference. |
| Ns | Specular exponent. Used to weight the specular value. In range 0 to 1000. |
| Ni | Refraction index. Rage 0.001 to 10.  Values of 1 mean light dose not bend as it passed through the object. |
| d | Transparency. Range 0 to 1. 1 = fully opaque. |
| Tr | Transparency. Used in different implementations and is revers to d.  Tr= 1-d |
| Tf | Transmission filter.  Used to specify how much light passes through the material.  Usually followed by 3 numbers RGB or a .rfl (spectral factor file). |
| illum | Illumination model.  Used one of the following modes for the material.  0. Colour on and Ambient off  1. Colour on and Ambient on  2. Highlight on  3. Reflection on and Ray trace on  4. Transparency: Glass on, Reflection: Ray trace on  5. Reflection: Fresnel on and Ray trace on  6. Transparency: Refraction on, Reflection: Fresnel off and Ray trace on  7. Transparency: Refraction on, Reflection: Fresnel on and Ray trace on  8. Reflection on and Ray trace off  9. Transparency: Glass on, Reflection: Ray trace off  10. Casts shadows onto invisible surfaces |
| Ka | Ambient colour of the material. Should be followed by 3 numbers, RGB values in the range 0 to 1; |
| Kd | Diffuse colour of the material. Should be followed by 3 numbers, RGB values in the range 0 to 1; |
| Ks | Specular colour of the material. Should be followed by 3 numbers, RGB values in the range 0 to 1; |
| Ke |  |
| map\_Ka | Defines the name of the ambient map used for this material. |
| map\_Kd | Defines the name of the diffuse map used for this material. |
| map\_Ks | Defines the name of the specular map used for this material. |
| map\_bump | Defines the name of the bump (normal) map used for this material. |

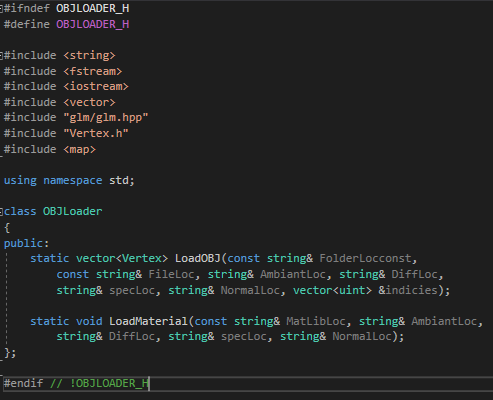
## Reading OBJ file

Now that we know what an obj file looks like, lets read one in to our program.

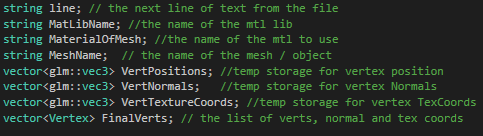
I’m going to define a new class called OBJLoader and create some static functions for loading my OJV files.

NOTE:  
the code demonstrated here is, in no way, presented as “the best approach”. The code has serious limitation and will not scale or return optimized data. It is up to you to refine it.

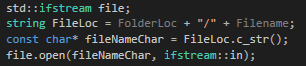
This forms part of your assessment.



In the LoadOBJ function, define the following:



Next, we’ll open the file for reading:

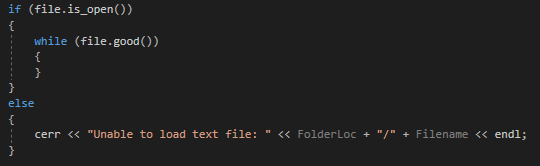


FolderLoc is the folder containing our OBJ. File name is the file name.

We need to keep them separate as we will use the FolderLoc to load our mtl fil and the textures, later.

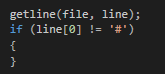
Now we need to check that the file is open. If it is not, then some thing has gone wrong and we need to print and error message.

If the file is open, then we want to enter a while loop until we are finished reading the file.



Next, we need to get a line of text and check to make sure its not a #. If it is, then this line is a comment and we don’t need to process it.

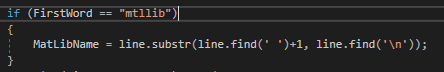
Enter the following in to the while loop:



Now we need to read the first letter or word from this line. Since the definitions of OBJ files states that all parameters and values must be separated with spaces, we can split our line in to 2 strings, based on the first space.

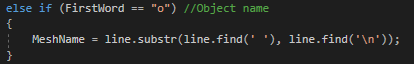


Now we can check that word to see what it is.



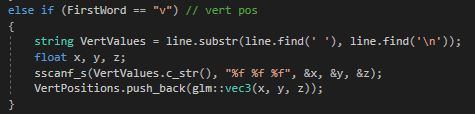
If this line is the mtllib line, get the data after the first space and save it in the MatLibName var.

Same for object name:

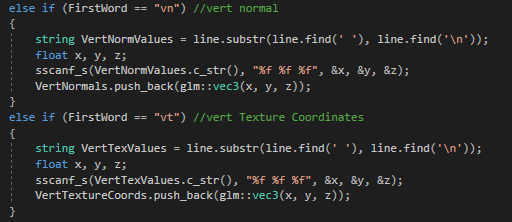


If it’s a V, wee need to read the data after the V, in to 3 separate float vars. This data is then pushed in to the VertPositions vector, this is our temporary position data structure.

Since V’s, VN’s and VT’s are usually clumped together and the f’s are defined at the end of the object definition, we can assume that all we will read in all this object vert’s, normal and texture coordinates before we try to construct your final verts/faces. Though this might not be true in all cases.



VN and VT are the same, we just put the data into the respective vectors.



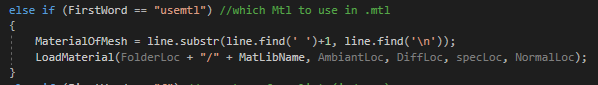
For usemtl, we need to read the name of Material we want to use, load the mtl lib and retrieve the date that we need. We read the material name into MaterialOfMash and pass it to LoadMaterial….. wait… no we don’t!

Have I missed something?

Yes. I’m assuming that we only have one object, with one material. This is a limitation that you can solve. You can fix this right 😉

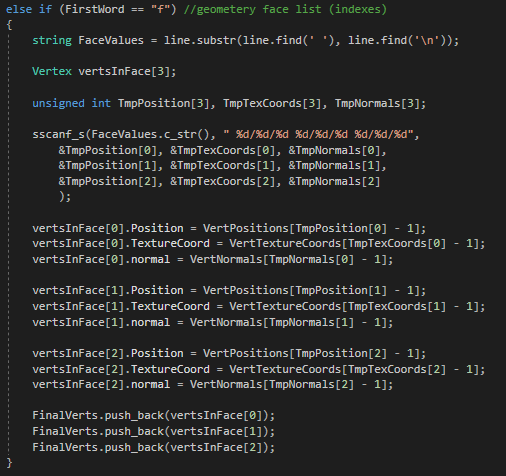
To LoadMaterial we pass the concatenation of the folder location, a”/” and MatLibName for the mtl file location. A reference to our Ambient var, our diffuse var, our spec var and our normal var.

These variables will be used to return the names of the image files to be loaded for our textures.



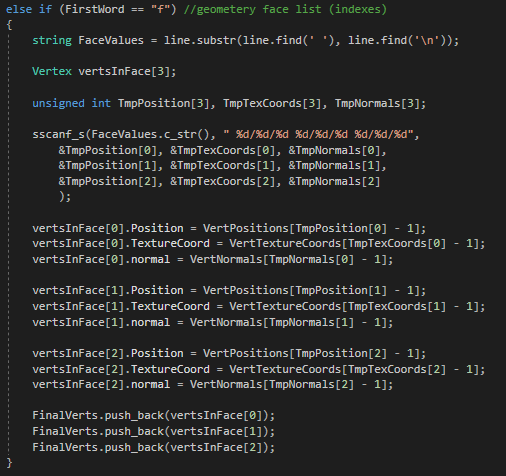
We’ll come back to the LoadMaterial function in a minuet.

Now let’s move on to the final symbol, f.



Blimey!

Let’s unpack this.



6

5

4

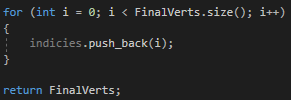
3

2

1

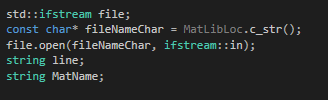
1. Read the text after the ‘f’, this will contain out face index values.
2. Make an array of 3 Vertex object. These will be the final verts we will push on to our FianlVerts List.
3. These 3 arrays are temporary data stores for the values we are about to read in. We need 3 for our position indices, 3 for our texture coordinate indices and 3 for our normal indices.
4. We use sscanf\_s to read the data from our FaceValues string (we convert the string to a c string as SScanf\_s doesn’t understand string. Then we use the follow string to letters and symbols to mask out the data we need from the string. “ %d/%d/ %d %d/ %d/ %d/ %d/ %d/ %d“. its hard to see here but the each set of %d/%d/%d is separated by a space and there is a space at the beginning of the line. The %d means “read this chunk of text” up to the first ‘/’. Then we read another %d up to another ‘/’ and a final %d up to the next SPACE.  
   wash,  
   rinse,  
   repeat.  
   each of these %d’s read data the relevant arrays and elements, listed just below, respectively.
5. This section uses the temporary data, read above, to grab the appropriate position, texture and normal coordinates from our vectors, above, and compiles to in to 3 vertex objects.   
   we -1 from the index values, we just read in, as they start a 1 and our arrays start a 0.
6. Finally, we push the vertex objects in to the FinalVerts vector.

Finally, we need to construct a list of indices to pass to our mesh class. Since we have not been smart about our data, we have constructed no unique vertices and so, our indices vector is just going to be a count of our FinalVerts vector, in order.



## Loading Materials

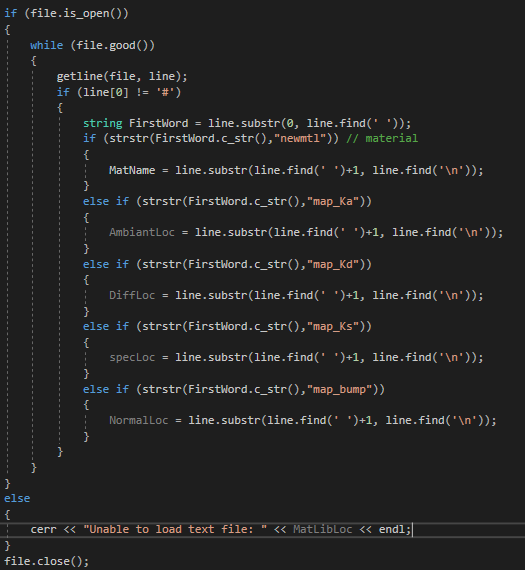
In the LoadMaterial function, enter the following;



Since the mtl file is also a text file, we just need to open it, just as we did the obj file.

We also define a string for the current line we are reading and a string for the Material name, thought we do nothing with it. It might be helpful for you in the future, when you some to load multiple materials.

Then we parse the mtl file, like we did the obj file.



Nothing here is new to you. The only bit to take note of is the +1 when reading the texture names in.

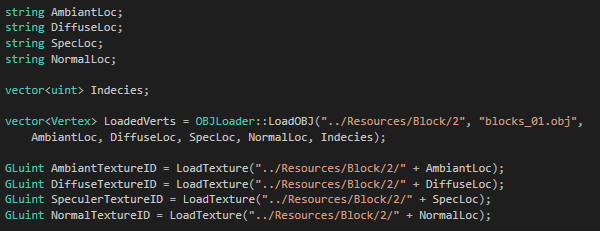
We add +1 to the find function as this function will return the index of the first SPACE it finds and read up to the first \n. this is fine but will return the string with a leading SPACE. So, when we come to load our textures from disk, we will end up with a file path like so, “C:/Folder/ File.dot”. This implies that the file name will start with a SPACE, which probably won’t be true (how many times have you named a file with a leading SPACE?). by adding 1 to the file function we are essentially trimming the leading space off the string.

Note:

This code only reads in the textures defined in the mtl file. It ignores any other values in the mtl and thus is not a true representation of our artists work.

We should probably read in the other values too, but I’m going to leave that for you 😉

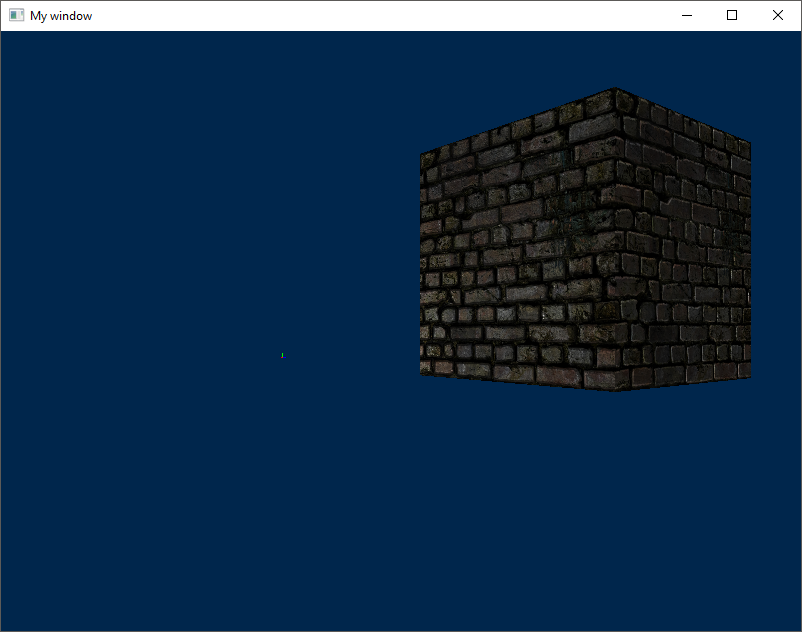
Now we just need to use the functions.



Something like this should do.

Don’t just blindly copy, make sure you are pointing the code are the right file and folder location.

And, with a little luck, you should end up with something that looks like this.



Isn’t it beautiful?......

No?...

Fine… load your own damn model then…… see if I care ….

☹