# Tech Review

Team 36, ViewCrobe Software

Product Name: FaceView

**The Many Faces of Microbial Communities**

The purpose of this project is to develop software which will allow users to input files containing sample data of microbial communities, and receive human model outputs based on the data within each sample. This mean that our program must accept some form of accepted, preferably common, file type for microbial sample data. It must also be able to convert this data into a data format suitable for our method of facial generation. It must then be able to generate a face based on various input values and output that face as some form of model.

## Model Generation

### MakeHuman

MakeHuman is a computer graphics software which allows users to create 3D human models through the use of a series of Blender scripts. One of the biggest benefits of MakeHuman is that it is open source with a very unrestricting license which allow it to be free used with commercial and non-commercial projects. This allows us to have control over every aspect of the technology without having to spend time creating our own scripts. MakeHuman also has an in depth API which allows for easy creation and exporting of models based on various criteria. It has proved itself as a high quality project when it was awarded the Suzanne Award for best Blender Python script in 2004. The primary foreseeable downside for MakeHuman is the potential difficulties of utilizing the API in our own software. Overall however, MakeHuman is a well-rounded project with a very clear method utilization, making it a great potential choice for our project.

### Fuse Character Creator

The second possible technology which could be used is the Fuse Character Creator. The Fuse Character Creator is a computer graphics software developed by Mixamo which allows the creation of 3D character models. The software is mainly designed for use by video game developers and the like. Overall, the models have great detail and could even be considered more realistic than the MakeHuman models. Fuse Character Creator also allows for users to easily import external user generated content, which could potentially be utilized in our project. Unfortunately, Fuse Character Creator carries with it some fairly heavy caveats. Firstly, the software is not open source, meaning we would not be able to easily access the core functionality of the project in addition to having no access to any API whatsoever. Secondly, the product licenses are required to use the software which costs money to obtain. Overall, the Fuse Character Creator is a great looking tool and would be great to use. Unfortunately, there seems to be a great deal of restrictions which would hold us back.

### Python Scripts for Blender

The third, and by far the most invested option would be to use python to generate scripts for Blender to create a human model. The primary benefit of this option is that it gives us an enormous amount of control over our code. Theoretically, the potentially of this option are limitless. Unfortunately, our project is confined to a 9 month period split between two different people. The amount of ground work required just to get something like this off the ground is astronomical for two people with no experience in the field. Overall, creating Python scripts for Blender simply is not a realistic option given the restrictions of our time and man power.

### Conclusion

Considering these option, it has been made abundantly clear that MakeHuman is the best option for us the use. The MakeHuman project provides all of the benefits of using the Fuse Character Creator without any of the immense downsides. Since MakeHuman is open source, we can still retain a level of control similar to that of generating our own Python scripts with all of the groundwork already completed.

## Data Formats

### BIOM

BIOM is essentially a special case of JSON format. The rows attribute provides taxonomy of each species, while the column attribute provides ids for the samples. The data field contains an array of each data point with ids for theirs corresponding sample and taxonomy entries. The advantage is that there are already libraries for parsing JSON for many languages, including the ones we consider.

### Classic QIIME OTU

Older versions of QIIME (Quantitative Insights Into Microbial Ecology) used a tab separated value format to represent data. In this format, the first column is an OTU id for each organism, followed by columns for each sample. The last column is the taxonomy name for the organism measured in that row. The advantage of this format is that it is well formatted when viewed as text.

### MEGAN CSV

This format is a comma separated value format in which the first column is the taxonomy of the organism, followed by data points for each sample. The advantage is that it’s extremely simple, and that there are libraries to parse such files for many languages.

### Conclusion

Since each format contains the same data, it makes little difference which one we choose. However, our client specifically requested that we support the tab-separated format, stating that it is well established and that it could be filled out by hand if necessary. Therefore, we choose the Classic QIIME OUT format.

## Language

We created a stretch goal to expand supported file formats, but initially, we chose BIOM. We would prefer to use a newer format, but otherwise, it doesn’t really matter which format we choose since they all present the same data but in a different way.

### Java

Java is a very popular general purpose language. Java comes with a plethora of frameworks for creating a UI. Both group members have a moderate amount of experience in writing with Java. A problem with using java would be using the Python API within a Java codebase.

### Visual C#

Visual C# is a very popular language for creating UI’s. It has easy and intuitive methods for creating a graphical interface without much the need to set up a great deal of ground work. It requires access to Visual Studio in order to use which is readily available to the group. A problem with Visual C# is the difficulties of compiling a project for either Mac or Linux. We would undoubtedly need to come up with a work around in order to solve this problem. There is also the difficulty of smoothly using out MakeHuman API which is based on a python project.

### Python

While not as popular as Java, nor as well suited for creating a UI as Visual C#, Python does provide several popular GUI frameworks. Additionally, Python is scripting language which lends itself well to the implementation of algorithms. This could be very beneficial in the process of converting the sample files into a data for the MakeHuman API. Additionally, because MakeHuman is made in Python, the process of using its API becomes far easier than with other programming languages.

### Conclusion

Overall, while the only language that gives us everything we want is Python. It may not have the greatest GUI frameworks but they should be suitable for a project such as this which requires very few individual components in the interface. The easy natural compatibility with the MakeHuman API is also an excellent plus for keeping this project simple enough for a two person group.

## UI Library

### Tk

Tk is a cross platform gui toolkit. There is an OpenGL interface for Tk, which will make rendering the 3d models possible easier. The advantages of Tk are that it is cross platform and that it is included with Python, which we expect to use. However, the OpenGL interface and the Python modules to include that interface are not included.

### Qt

Qt is also a cross platform gui toolkit which also has an OpenGL interface. The advantages of Qt are that it is also cross platform and that there also exists a Python module for it. The disadvantages are that the Python module for Qt is not included in the standard installation, so we would have to include that in our project manually.

### SDL

SDL or Simple DirectMedia Layer provides a cross platform interface for drawing to the screen, playing audio, receiving input from devices, and other operating system specific operations. It also supports plugins for other tasks like networking or rendering fonts. The advantage of using SDL over a GUI toolkit is that SDL allows more flexibility, since any work takes place at a lower level. One disadvantage is that it would make UI development more difficult, as the toolkits implement subtle features of the native look and feel that users come to expect. Another disadvantage is that using it with Python puts more Python code between the user and the functionality, which would be slower than the C code that other libraries use.

### Conclusion

We have chosen to use Qt. We realize that Tk and Qt are very similar, so it should be sufficient to choose one arbitrarily. We also recognize that we don’t need to go as low level as SDL allows, and to do so would fail to provide features that users take for granted unless we were to implement them manually. For example, copying and pasting text. Reinventing the wheel is unnecessary in this instance, and would also open us up to writing more bugs.

## Model Format

### .obj

Object files represent the geometry of a model and can be associated with material files (.mtl) which can represent the colors and textures on an object. The advantage of this format is that it is relatively simple and does not bother to describe other scene elements, which would make loading the models easier. The disadvantage of this format is that it does not support hierarchical models, so users of our application would not be able to pose the generated models.

### Renderman

RenderMan format is a more complex format than obj. In addition to storing model geometry mapping details, it can also store world and lighting details as well as shaders. This allows an entire scene with many objects to be stored in a single .rib file. The disadvantage is that these details add a significant amount of complexity which may be unnecessary to our project.

### Pov-Ray

Pov-Ray is a format used by the Pov-Ray ray tracing software. Like Renderman, it represents entire scenes in a single file. Since it was designed with ray tracing in mind, Pov-Ray format does not support user created shaders like Renderman does. It does however support programming language features like conditionals, loops, variables, and functions. Like Renderman, Pov-Ray supports many features unnecessary to our project that increase complexity.

### Conclusion

We have decided to use obj format to store our models. It’s a relatively simple format, and we don’t need the features that add complexity. Specifically, we only need to store one model per file, and they need not be poseable. If we were to use Renderman, we would ignore many features that are built in to the format such as lighting details, camera position, custom shaders, or multiple models. If we were to use Pov-Ray, we would have to ignore similar features, as well as features that lend themselves to ray tracing. This is because ray tracing is significantly slower (and thus less responsive) than a shader-based system, which would prevent users from rotating models in real time.