The Many Faces of Microbial Communities

Senior Design
Winter Term

Thomas Albertine, Michael Phelps

Abstract

We intend for this project to provide an easier to use way to visualize microbial population data than some existing techniques, such as pie or bar charts, within the constraints provided by our client. As of the midterm progress report, we have had working backend components for loading population data files in the QIIME OTU format, generating 3d models based on the data in Wavefront obj format, and converting population counts into normalized values that can be used to generate 3D models of human faces, as well as a preliminary user interface (UI) for most of our project. Since then, we've improved the UI, connected the UI with backend functionality mentioned before, and done user testing to identify new improvements to be made. The remaining work is simply to fix the problems we've identified in our testing, continue testing to identify more problems, and possibly implement some of our stretch goals, such as expanding file format support.

CONTENTS

I. A NOTE REGARDING DIVISION OF LABOR IN THIS DOCUMENT AND THE CORRESPONDING VIDEO

During the midterm progress report, we split the work, such that Thomas primarily worked on the written report, and Michael primarily worked on the video. This was because Thomas had a cold and did not want to be in the video. However, we feel that dividing the work in this way allowed us to provide a better result than we would have otherwise, with a few exceptions due to poor communication.

For this report, we intend to do something similar, but with more thorough proofreading after the rough draft is complete. Therefore, Thomas will again not be appearing in the video, and will focus on the report instead. Unlike before however, Thomas will send Michael his notes on the video, and Michael will send Thomas his notes on this document, which should cut back on mistakes in both.

II. Purposes and Goals

The purpose of the project is to visualize microbial population data in a way that allows microbiologists (and secondarily other researchers) to more easily compare complex population data. To that effect, our client has requested that we model it as human-esque faces, in order to take advantage of humans' ability to recognize and compare those faces. Resulting models need not look like visually appealing humans, but they should represent the data.

To that effect, our project loads population data from files in supported formats and sample grouping data in corresponding metadata files. This data is converted to model attributes, values that determine how exaggerated some feature of the face, called a model parameter, is, based on associations provided by the user in the application. These attributes are passed into MakeHuman, an open source project that generates 3d models of humans. It generates 3D models in Wavefront obj format for each sample. Then the models are then loaded and displayed to the user, who can view smaller sets of samples based on grouping data, or can choose a subset of no more than six samples to view and inspect in more detail.

III. Completed Content

A. Backend

In the backend code of the project, Thomas has already written a wrapper around MakeHuman that can retrieve a list of valid model parameter names[1] as well as receive model parameter names and values in order to generate models. This particular code has a minor deficiency in that the models it produces have no eyes, so if we have time to improve upon it later, we intend to. That said, if it proves impossible, the current system is still sufficient to visualize the data. Figure 1 depicts a model generated from a data file.

Thomas[2] has also written code to load a QIIME OTU tab-delineated data file, which the client specified as the minimum for data loading, as well as a framework to allow us to easily add support for other file types later on. This is described in more detail in section VI later in this document, but in short, we create a python object that can parse a file format, and associate it with certain file extensions in a registry object that we have also created. When loading a file, simply pass the file path into the registry object, and it uses the extension from the file path to find the right parser, which is passed the file path. Each parser object is responsible for parsing the data file itself, as well as the metadata file that contains grouping information.

Thomas has also written code to normalize sample data based on the largest population in the sample (in order to compare ratios of populations between samples), based on the largest population in the file (in order to compare population sizes between samples), and based on a manual scaling value that can be set by the user, in order to compare between images of previously generated models, or models in separate files.

B. Frontend

Michael has written the data loading page, the main selection page, and the detailed viewing page. Currently, the grouping functionality, model generation, and microbe-model parameter mapping, are all connected to the UI

Thomas has written an obj viewer widget that Michael is using to display the obj files.

[1]Our requirements document specifies that we support 150 model parameters, but we only have enough facial features to support 138. We can extend that number significantly however, if we use other model parameters besides facial features, but our client mainly wants to work with facial features alone. We are considering adding the others as optional parameters.

[2]Originally, Michael started writing this component, but there was a miscommunication and Thomas panicked and wrote it instead.

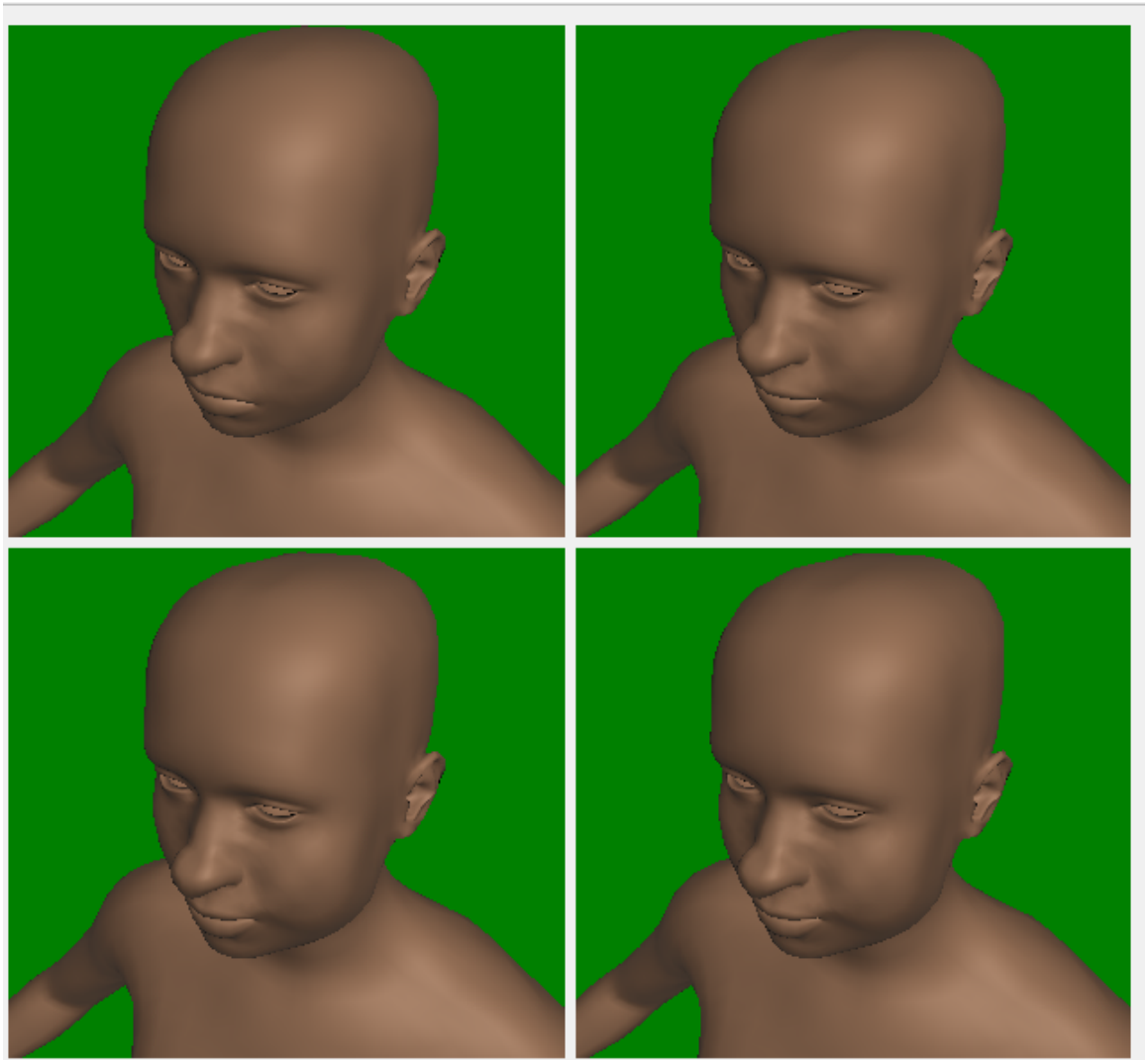

Fig. 1. A model generated from a sample data file. This file had few enough organisms that only some features were modified. The rest were left at their default values to avoid distracting from the significant features of the data set. This image was generated using an obj viewer widget that Thomas wrote and the analysis view page that Michael wrote.

The Data Loading page (see Figure 2) allows users to select a data file and a groupings file. These files will be loaded, and then used to populate the table on the bottom, which allows users to associate model parameters with organisms in the sample. On the top right, a series of radio buttons can be used to specify which normalizing strategy to use. That is, whether the generated models should reflect ratio of each organism to the whole of the sample, or whether the generated models should reflect the ratio of each organism to the largest population in the file, or whether the models should be normalized in terms of some other number entirely. The first is good for identifying samples with similar ratios of an organism to the population in the sample. The second is good for identifying samples with similar total populations of an organism. The third is good for comparing models in a different file.

The Main Selection page (see Figure 3) will allow users to select a few samples out of the file to examine in more detail. The box on the left is where images of the samples' models will appear, while the elements

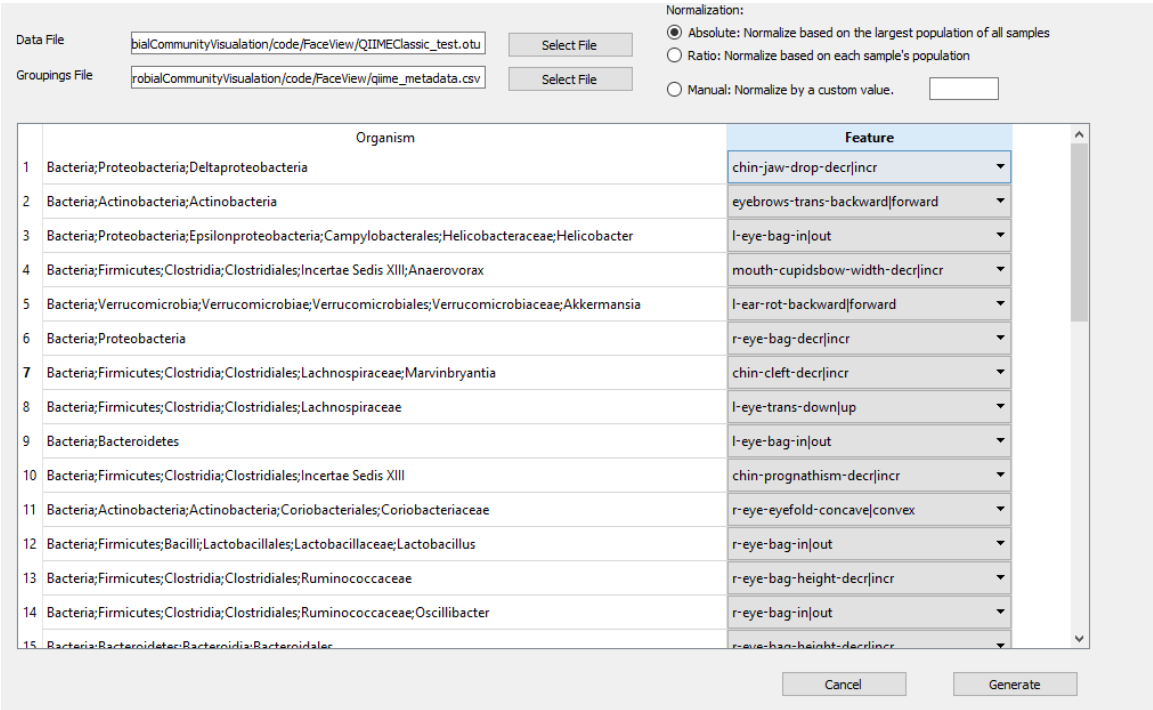

Fig. 2. The UI page that allows users to perform tasks related to loading data files.

on the right allow users to organism samples into groups, or view groups from the metadata file.

IV. Remaining Work

There still exist some notable bugs. Specifically, once the user goes forward to a new page, he or she cannot go back at this time. When the user opens the analysis view, the thumbnail view does not close, but becomes unresponsive, which could lead to confusion for users. Finally, when the application is closed, Windows displays an error message that python is not responding, which does not directly inhibit usability, but should still be fixed before release.

Additionally, our user testing has revealed several opportunities for improvement that we'd like to implement. One of our test subjects suggested that the blue and green backgrounds in the thumbnail view which denote whether the sample is selected or not are not intuitive, and that we should use different colors. More than one of our subjects suggested that the model parameters are difficult to understand. We believe that we can resolve this by adding a system to alias the model parameters taken directly from MakeHuman to something more easily understood. Another test subject pointed out that the images of the models are not drawn with the names of their corresponding samples, which could significantly improve usability, but should be easily fixed. Test subjects would also click the model generator button multiple times because there was no cue for them to know that the first click had done something. We'd like to either add something to the window to show that, or close that window to reveal the console so that the user knows that the application is making progress. Finally, our last subject suggested that not every model parameter is visible from the front view provided by thumbnail view. We think that this could be resolved by turning the models back and forth slowly.

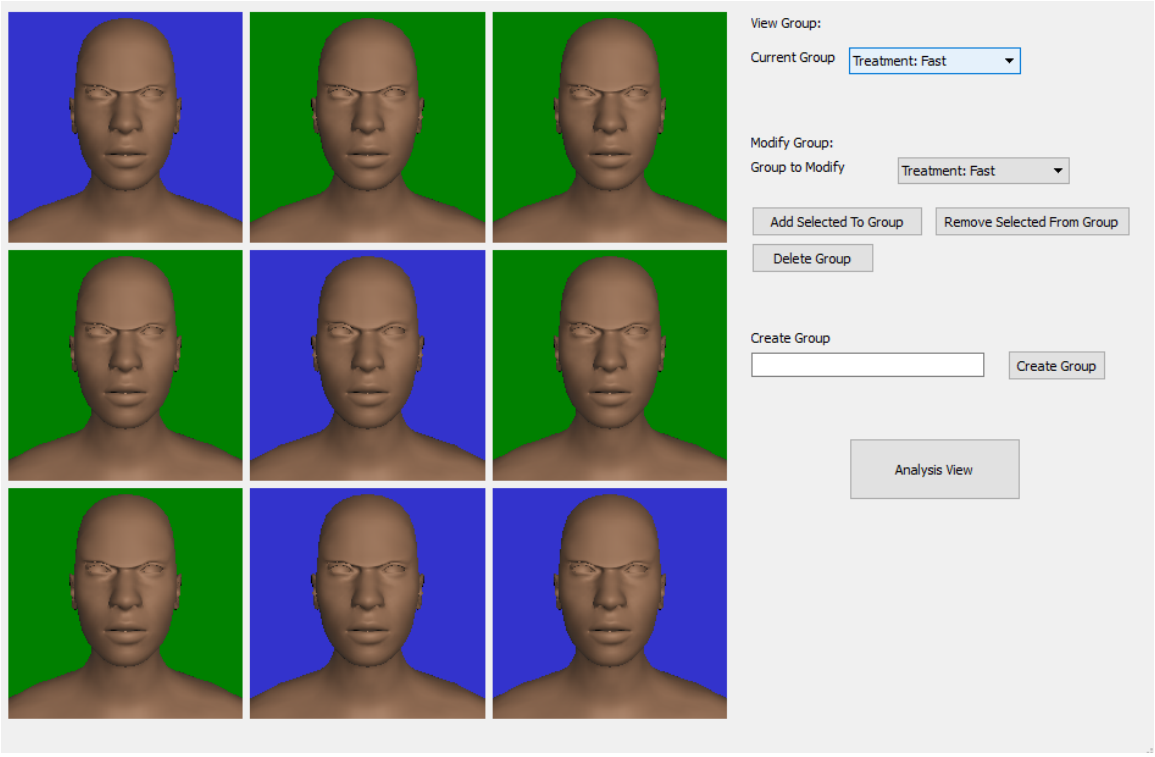

Fig. 3. The UI page that allows users to perform tasks related to selecting data files.

When these issues are resolved at least, then we will consider version 1.0 complete, although if we can fit any of our stretch goals in too, so be it.

As mentioned in the backend section, if time allows, we would also fix the eye problem with model generation and improve our testing.

V. Significant Problems and Associated Solutions

The most difficult problem that Thomas faced was simply trying to read and decypher the MakeHuman code enough to write the wrapper. Since Python is loosely typed, he couldn't just match the types to see how things were being used. Instead (at least at first), he had to walk through the code jumping from function definition to function definition, searching through multiple files. After doing that for a few hours, he remembered that python objects have a dictionary of their attributes, so he was able to import MakeHuman modules and examine the the contents of objects returned from functions. This sped up the process considerably. Additionally, he figured out how to use grep-like functionality from within PowerShell, which made it much easier to look for function definitions.

Additionally, the obj viewer widget proved challenging simply because the the error policy for OpenGL is to fail silently and not draw anything. Most of this can be mitigated by occasional calls to $glGetError()$, but some mistakes are not technically errors, yet still don't produce the intended output.

For Michael, the most difficult problem was figuring out how to display the obj files within the Qt based interface. There is surprisingly little documentation on how to perform such a task. As of now, the obj

viewer widget should solve these problems, but nevertheless, it was a concern earlier on, and one that Michael put an exceptional amount of work into.

VI. Interesting Code

A. *fvParser*

The following is the framework for easy addition of new parsers. minParser represents the minimum functionality required for a parser. If this were C++, it would be a virtual class, but thanks to Python's duck typing, we don't actually have to inherit from anything, so it's more of a template for us to copy later.

The ParserRegistry itself is little more than a wrapper around a dictionary, mapping a file extension to a parser object, but with some additional functions added to it to make its usage more obvious. Finally, an instance of the Registry is created as a global, so that it can be referenced anywhere by importing the module.

```
import os

#minimum functions required for a parser.
#Should also register itself with the parser registry
#Returns list of samples and a list of groups
class minParser:
        def parseFile(self, filePath, metaPath):
                "noop"

#The parser registry
class parserRegistry:
        def __init__(self):
                self.mapping = {}

        def registerParser(self, parser, extension):
                self.mapping[extension] = parser

        def getParserForExtension(self, extension):
                if self.extensionIsSupported(extension):
                        return self.mapping[extension]
                else:
                        return None

        def extensionIsSupported(self, extension):
                return self.mapping.has_key(extension)

        def parse(self, filePath, metaPath):
                fileExt = os.path.splitext(os.path.normpath(filePath))
                    [1]
                parser = self.getParserForExtension(fileExt)
                if parser is not None:
                        return parser().parseFile(filePath, metaPath)
                else:
                        return None
```

```
pReg = parserRegistry()
```

B. objWidget

The following is a PyQt widget written by Thomas which inherits from the QOpenGLWidget in standard PyQt. Additionally, the file includes additional functions used by the class for creating transformation and projection matrices. The class itself accepts a 3d model object

```
import OpenGL.GL as GL
import OpenGL.GLU as GLU
from PyQt4 import QtGui
from PyQt4 import QtCore
from PyQt4.QtOpenGL import *
import numpy
import os.path
import math

from ctypes import c_void_p

#By Thomas Albertine
#It should allow you to view obj files in a little window in your app.

def genTranslationMatrix(x, y, z):
        return numpy.matrix([
                [1, 0, 0, x],
                [0, 1, 0, y],
                [0, 0, 1, z],
                [0, 0, 0, 1]
                ], dtype='float32')

def genScaleMatrix(x, y, z):
        return numpy.matrix([
                [x, 0, 0, 0],
                [0, y, 0, 0],
                [0, 0, z, 0],
                [0, 0, 0, 1]
                ], dtype='float32')

def genRotationMatrixX(angle):
        angle = math.radians(angle)
        return numpy.matrix([
                [1, 0, 0, 0],
                [0, math.cos(angle), math.sin(angle), 0],
                [0, -math.sin(angle), math.cos(angle), 0],
                [0, 0, 0, 1]
                ], dtype='float32')

def genRotationMatrixY(angle):
        angle = math.radians(angle)
```

```
        return numpy.matrix([
                [math.cos(angle), 0, -math.sin(angle), 0],
                [0, 1, 0, 0],
                [math.sin(angle), 0, math.cos(angle), 0],
                [0, 0, 0, 1]
                ], dtype='float32')

def genRotationMatrixZ(angle):
        angle = math.radians(angle)
        return numpy.matrix([
                [math.cos(angle), math.sin(angle), 0, 0],
                [-math.sin(angle), math.cos(angle), 0, 0],
                [0, 0, 1, 0],
                [0, 0, 0, 1]
                ], dtype='float32')

def genRotMatFromQuat(x, y, z, angle):
        angle = math.radians(angle)
        xx = x * x
        xy = x * y
        xz = x * z
        xw = x * w
        yy = y * y
        yz = y * z
        yw = y * w
        zz = z * z
        zw = z * w
        return numpy.matrix([
                [1 - 2 * (yy + zz), 2 * (xy - zw), 2 * (xz + yw), 0],
                [2 * (xy + zw), 1 - 2 * (xx + zz), 2 * (yz - xw), 0],
                [2 * (xz - yw), 2 * (yz + xw), 1 - 2 * (xx + yy), 0],
                [0, 0, 0, 1]
                ], dtype='float32')

def genIdentity():
        return numpy.matrix([
                [1, 0, 0, 0],
                [0, 1, 0, 0],
                [0, 0, 1, 0],
                [0, 0, 0, 1]
                ], dtype='float32')

def genPerspective(vFov, aspect, near, far):
        fov = math.radians(vFov)
        tanHalf = math.tan(fov / 2)

        #return numpy.identity(4)
        return numpy.matrix([
                [1 / (aspect * tanHalf), 0, 0, 0],
                [0, 1 / tanHalf, 0, 0],
```

```
                [0, 0, -(far + near) / (far - near), -(2*far*near) / (
                    far - near)],
                [0, 0, -1, 0]
                ], dtype='float32')

class ObjWidget(QGLWidget):
        # Any default values should be stored in the class

        def __init__(self, model, highlighted = False, parent=None,
            aspectRatio=1.0):
                QGLWidget.__init__(self, parent)
                self.data = model
                self.vao = None
                self.verts = None
                self.shaderProg = None
                self.shaderVars={}
                self.vertexComps = -1
                self.indexBuffer = -1
                self.aspect = aspectRatio

                #Various positioning data
                self.objScale = 1.5
                self.yRotate = 0
                self.xRotate = 0
                self.yPan = 0

                self.MODEL_HEIGHT = 16.594

                #Various matrices
                self.perspective = None #To be populated on resize
                self.modelToWorld = None
                self.worldToCamera = None
                self.modelNormalToCamera = None
                self.updateMatrices()

                self.highlighted = highlighted




        def highlight(self):
                print "hello"

        def pan(self, y):
                'Allows the user to pan vertically around the model'
                self.yPan = max(min(y, self.MODEL_HEIGHT * 0.8), -self.
                    MODEL_HEIGHT * 0.92)
                self.updateMatrices()
                self.update()
```

```
def relativePan(self, y):
        self.pan(self.yPan + y)

def rotate(self, x, z):
        if z > 360 or z < 0:
                z = z % 360
        if x > 180:
                x = 90
        if x < -90:
                x = -90
        self.xRotate = x
        self.yRotate = z
        self.updateMatrices()
        self.update()

def relativeRotate(self, x, z):
        self.rotate(self.xRotate + x, self.yRotate + z)

def scale(self, zoom):
        self.objScale = man(min(zoom, 3), 0.1)
        self.updateMatrices()
        self.update()

def relativeScale(self, zoom):
        self.scale(self.objScale + zoom)

def updateMatrices(self):
        "Updates every matrix except projection, which will be
           updated on resize"
        newMatrix = genIdentity()
        #Center on the face
        newMatrix = genTranslationMatrix(0, self.MODEL_HEIGHT *
           -.42, -0.66) * newMatrix
        newMatrix = genScaleMatrix(self.objScale, self.objScale,
            self.objScale) * newMatrix
        newMatrix = genRotationMatrixY(self.yRotate) * newMatrix
        newMatrix = genRotationMatrixX(self.xRotate) * newMatrix
        newMatrix = genTranslationMatrix(0, self.yPan, 0) *
           newMatrix
        self.modelToWorld = newMatrix

        newMatrix = genIdentity()
        newMatrix = genTranslationMatrix(0, 0, -5) * newMatrix
        self.worldToCamera = newMatrix

        self.modelNormalToCamera = numpy.linalg.inv((self.
           worldToCamera * self.modelToWorld)[0:3,0:3]).T


def __del__(self):
```

```
            if callable(hasattr(super(type(self), self), "__del__"))
                :
                    super(type(self), self).__del__(parent)
            #Don't call unloadData because it's about to unload
                anyway.

    def paintGL(self):
            #For a test
            if self.highlighted:
                    GL.glClearColor(0.0, 0.5, 0.0, 1.0)
            else:
                    GL.glClearColor(0.2, 0.2, 0.8, 1.0)
            GL.glDisable(GL.GL_CULL_FACE)
            GL.glClear(GL.GL_DEPTH_BUFFER_BIT | GL.
                GL_COLOR_BUFFER_BIT)

            GL.glUseProgram(self.shaderProg)

            GL.glBindBuffer(GL.GL_ARRAY_BUFFER, self.vertexComps)
            offset = 0
            for i in self.attrs:
                    name, type, length, data = i
                    size = data.size * data.itemsize
                    varIndex = self.shaderVars[name]
                    if varIndex > -1:
                            GL.glEnableVertexAttribArray(varIndex)
                            GL.glVertexAttribPointer(varIndex,
                                length, type, GL.GL_FALSE, 0,
                                c_void_p(offset))
                    offset += size

            #Matrices
            loc = GL.glGetUniformLocation(self.shaderProg, '
                uModelWorld')
            if loc > -1:
                    GL.glUniformMatrix4fv(loc, 1, GL.GL_TRUE, numpy.
                        array(self.modelToWorld))
            loc = GL.glGetUniformLocation(self.shaderProg, '
                uWorldCamera')
            if loc > -1:
                    GL.glUniformMatrix4fv(loc, 1, GL.GL_TRUE, numpy.
                        array(self.worldToCamera))
            loc = GL.glGetUniformLocation(self.shaderProg, '
                uCameraProjection')
            if loc > -1:
                    GL.glUniformMatrix4fv(loc, 1, GL.GL_TRUE, numpy.
                        array(self.perspective))
            loc = GL.glGetUniformLocation(self.shaderProg, '
                uModelNormalCamera')
            if loc > -1:
```

```
                GL.glUniformMatrix3fv(loc, 1, GL.GL_TRUE, numpy.
                    array(self.modelNormalToCamera))

        #Colors
        loc = GL.glGetUniformLocation(self.shaderProg, '
            skinColor')
        if loc > -1:
                data = numpy.array(self.data.skinColor(), dtype=
                    'float32')
                GL.glUniform4fv(loc, 1, data)
        loc = GL.glGetUniformLocation(self.shaderProg, '
            skinSpecularColor')
        if loc > -1:
                data = numpy.array(self.data.skinSpecColor(),
                    dtype='float32')
                GL.glUniform4fv(loc, 1, data)

        GL.glBindBuffer(GL.GL_ELEMENT_ARRAY_BUFFER, self.
            indexBuffer)
        GL.glDrawElements(GL.GL_TRIANGLES, #Topology
                numpy.array(self.data.indices(), 'uint32').size,
                     #Count in points
                GL.GL_UNSIGNED_INT, #Type
                None)
        for i in self.attrs:
                name = i[0]
                varIndex = self.shaderVars[name]
                if varIndex > -1:
                        GL.glDisableVertexAttribArray(varIndex)

def resizeGL(self, width, height):
        self.perspective = genPerspective(50.0, float(width) /
            float(height), 0.1, 20.0)
        GL.glViewport(0, 0, width, height)

def initializeGL(self):
        QGLWidget.initializeGL(self)
        GL.glEnable(GL.GL_DEPTH_TEST)


        self.attrs = [('uModelPos', GL.GL_FLOAT, 4, numpy.array(
            self.data.vertices(), dtype='float32')),
                ('uST', GL.GL_FLOAT, 2, numpy.array(self.data.
                    texCoords(), dtype='float32')),
                ('uModelNormal', GL.GL_FLOAT, 3, numpy.array(
                    self.data.normals(), dtype='float32'))] #
                    name, gl type, vector length, data

        self.loadShaders('../FaceView/objViewer/vert.glsl', '../
            FaceView/objViewer/frag.glsl')
        self.loadData()
```

```
def unloadData(self):
        arr = numpy.array([self.vertexComps], dtype='uint32')
        GL.glDeleteBuffers(1, arr)
        self.vertexComps = -1
        GL.glDeleteBuffers(1, numpy.array([self.indexBuffer],
            dtype='uint32'))
        self.indexBuffer = -1

def reloadData(self):
        self.unloadData()
        self.loadData()

def loadData(self):
        self.vao = GL.glGenVertexArrays(1)
        GL.glBindVertexArray(self.vao)

        totalSize = 0
        for i in self.attrs:
                data = i[3]
                totalSize += data.size * data.itemsize
        #Generate data buffer
        self.vertexComps = GL.glGenBuffers(1)
        GL.glBindBuffer(GL.GL_ARRAY_BUFFER, self.vertexComps)

        #Get memory
        GL.glBufferData(GL.GL_ARRAY_BUFFER, totalSize, None, GL.
            GL_STATIC_DRAW)

        #Put vertices in buffer
        offset = 0
        for i in self.attrs:
                data = i[3]
                size = data.size * data.itemsize

                GL.glBufferSubData(GL.GL_ARRAY_BUFFER, #target
                        offset, #offset into the buffer to put
                            the data
                        size, #size of data in bytes
                        data) #data
                offset += size

        #Generate index buffer
        indices = numpy.array(self.data.indices(), dtype='uint32
            ')
        self.indexBuffer = GL.glGenBuffers(1)
        GL.glBindBuffer(GL.GL_ELEMENT_ARRAY_BUFFER, self.
            indexBuffer)
        GL.glBufferData(GL.GL_ELEMENT_ARRAY_BUFFER, indices.
            itemsize * indices.size, indices, GL.GL_STATIC_DRAW)
```

```
def loadShaders(self, vertexPath, fragmentPath):
        if self.shaderProg is not None:
                GL.glDeleteProgram(self.shaderProg)

        vert = GL.glCreateShader(GL.GL_VERTEX_SHADER)
        frag = GL.glCreateShader(GL.GL_FRAGMENT_SHADER)

        #Read shader text
        vertexText = ""
        fragmentText = ""
        with open(vertexPath, 'rU') as f:
                vertexText = f.read()
        with open(fragmentPath, 'rU') as f:
                fragmentText = f.read()

        #Compile vertex shader
        GL.glShaderSource(vert, vertexText)
        GL.glCompileShader(vert)
        print os.getcwd()
        if GL.glGetShaderiv(vert, GL.GL_COMPILE_STATUS) != GL.
            GL_TRUE:
                raise RuntimeError("\n\nVertex Shader: " + GL.
                    glGetShaderInfoLog(vert))

        #Compile fragment shader
        GL.glShaderSource(frag, fragmentText)
        GL.glCompileShader(frag)
        if GL.glGetShaderiv(frag, GL.GL_COMPILE_STATUS) != GL.
            GL_TRUE:
                raise RuntimeError("\n\nFragment Shader: "  + GL
                    .glGetShaderInfoLog(frag))

        #Link
        prog = GL.glCreateProgram()
        GL.glAttachShader(prog, vert)
        GL.glAttachShader(prog, frag)
        GL.glLinkProgram(prog)
        if GL.glGetProgramiv(prog, GL.GL_LINK_STATUS) != GL.
            GL_TRUE:
                raise RuntimeError(GL.glGetProgramInfoLog(prog))
        GL.glDetachShader(prog, vert)
        GL.glDetachShader(prog, frag)
        GL.glDeleteShader(vert)
        GL.glDeleteShader(frag)
        self.shaderProg = prog

        for i in self.attrs:
                name = i[0]
                self.shaderVars[name] = GL.glGetAttribLocation(
                    self.shaderProg, name)
```

```
                for key, value in self.shaderVars.iteritems():
                        if value == -1:
                                print "Double check " + str(key)+ "."
                                print "It was not found, but it may have
                                    been optimized out of the shaders."

        def setHighlighted(self, hl):
                self.highlighted = hl
                self.updateGL()

if __name__ == '__main__':
        import objModel
        import os
        model = objModel.ObjModel('../../models/PC.354.obj')
        app = QtGui.QApplication(["Thomas' "])
        widget = ObjWidget(model)
        widget.resize(300, 300)
        widget.highlight()
        widget.show()
        timer = QtCore.QTimer()
        def timeout_func():
                widget.relativeRotate(0, 0)

        timer.timeout.connect(timeout_func)
        timer.start(16)
        app.exec_()
```

VII. User Study

We have completed our first user study and gained some suggestions and insight to improve the project's usability.

A. Users

For our test subjects, we found college students of various disciplines and backgrounds. This was more than sufficient to provide suggestions about, and identify flaws in, our UI, especially since it is relatively new and untested.

B. Methods

We used a set of 5 moderated, in-person tests. This was enough to identify several bugs and other opportunities for improvement, but not so many that it took an impractical amount of time to moderate it.

C. Tasks

Before each task, we asked users how confident they are that they can complete the task, and timed them while they performed the task, recording the time it took them. When we saw them struggling over a particular step in the task, we made a note of it. If the user seemed unable to make progress, we made a

note of what step they were stuck on, politely interrupted them, and completed the step ourselves so that the user could see how it was done. When users requested that we repeat the task statement or some detail of it, we did. None requested clarification, but if they had, we would answer their questions and make a note of that as well, so that we could design a better test in the future.

Our tasks are as follows, starting immediately after opening the application for the first time, and given a sample data file:

1) Load the data file.
2) Associate Bacillus subtilis with head squareness[3].
3) Modify settings so that you can compare ratios of different populations between samples.
4) Generate the models.
5) Select some samples to compare in more detail.
6) Find the two samples with the most similar ratios of Bacillus subtilis to total population and view them in more detail.
7) Find the two samples with the most similar total population of bacillus subtilis and view them in more detail.

After the first few tests, we removed a task "Manipulate Models" because users invariably started playing with that feature before we had a chance to time them or ask them anything about it.

The first five tasks are primarily to identify how intuitive the application is to a new user. The last two are to identify how easy the application is to use once someone has seen how it works.

D. Results

From our testing, we were able to better understand which areas of our UI needed the most improvement. These improvements are described in section IV, Remaining Work.

Figure 4 contains our data for the ui testing. In addition, we also have a series of observations that we made while watching our subjects, and comments that our subjects made when we asked them afterwards.

1) Subject 1: This subject is a junior majoring in biology. During task 5, she found that the blue and green colors we currently use to denote whether the models are not particularly intuitive. During task 6, she spent longer than expected looking through the model parameter options than we expected, which delayed her completion time. Afterwards, she commented that the names in the model parameter dropdown were not particularly intuitive.

2) Subject 2: This subject is a freshman majoring in his mechanical engineering, and who's first language is not English. He discovered that not every selected sample appears in the analysis page when the analysis view button is clicked, a bug that has since been fixed. He also commented that the model parameter list is "detailed", and that the samples do not show their sample name in the thumbnail view.

3) Subject 3: This subject is a freshman majoring in computer science. He showed signs of being competitive regarding completion times, but got distracted looking through the model parameter list, which delayed his completion time. He commented that he was confused about how the data sets corresponded to the generated faces, but later said that the relationship was intuitive if the user understands what the input data contains.

4) Subject 4: This subject is a college freshman majoring in business. He showed signs of being tired or not having slept enough recently, which may have affected his results. He made jokes about sleep deprived

[3]This organism and this model parameter need not be the ones used in the test

Task	Subject					
	1	2	3	4	5	Average Time
1	“Yes” 0:31	“Yes” 0:44	“Medium” 0:20	“In the middle” 0:56	“Yes” 0:38	0:38
2	“No” 1:00	“Don’t Know” 0:12	“Very Confident” 0:01	“Medium” 2:00	“No” 0:11	0:41
3	“Yes” 0:02	“Yes” 0:03	“Confident” 0:01	“4/10” 2:27	“No” 0:22	0:35
4	“Yes” 0:10	“Yes” 0:12	“Confident” 0:10	“4/10” 0:10	“Yes” 0:10	0:10
5	“No” 1:20	“No” 0:09	“Confident” 1:00	“5/10” 0:09	“No” 0:11	0:34
6	“Yes” 1:26	“Yes” 0:53	“Maybe” 3:03	“6/10” 3:08	“No” 0:59	1:54
7	“Yes” 0:57	“Yes” 1:27	“Yes” 0:50	“Yes” 1:40	“Yes” 1:54	1:22

Fig. 4. This is the data from our UI testing. The middle cells contains the user’s description of his or her confidence before attempting each task as well as the time it took the user to complete the task

researchers which seems to confirm my observation. Initially, the subject had difficulty using functionality provided by Windows, accidentally minimizing the window, and then closing it while trying to unminimize it, but he did better once we reopened it for him.

During task 2, he commented that the tool was “very thorough”. During task 3, he was unable to find the radio button that specifies that data be normalized as a percentage of the total sample population, requiring me to point it out to him. During task 4, while waiting for the models to generate, he suggested adding something to let the user know that the application is making progress.

Afterwards, the subject said that the tasks were difficult and that he would like to try again for a shorter completion time. He also mentioned that the relationship between the model parameters and their effect on the generated model was not particularly intuitive.

5) Subject 5: This subject is a college senior majoring in fish and wildlife who claims to be bad with computers. During task 7, he mentioned that not all model features are visible from the front view provided from the thumbnail view page, and suggested making the models rotate.