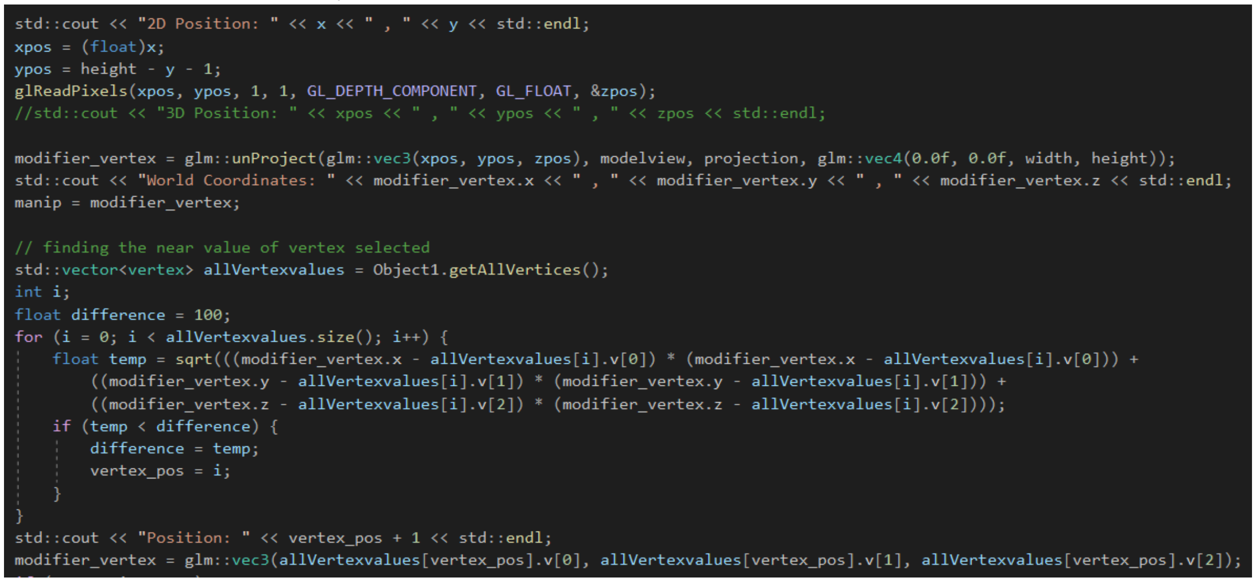
CS7GV5-A-SEM202-201920_REAL-TIME ANIMATION
Assignment 3 – Facial Animation

- **An interface for creating, selecting and moving vertex manipulators:**
  For selecting the vertex, I have used left mouse button click and the same button is used for moving the selected vertex around by mouse drag.
  The selection is done in the "***TwEventMouseButtonGLFW3***" function as we need only a single mouse position, while the dragging is implemented inside "*TwEventMousePosGLFW3*" because the position needs to be updated constantly.

  Selection: The mouse click gives us the 2D co-ordinates which are converted into 3D using "***glReadPixels***". These values are further modified into world space using "***glm::unProject***". After getting the world co-ordinates, the closest vertex is extracted from the mesh using Euclidean distance. The position and value of vertex is saved for future manipulations.



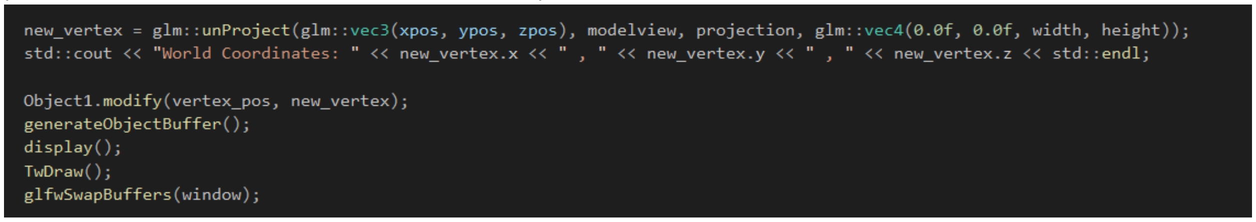
```
std::cout << "2D Position: " << x << " , " << y << std::endl;
xpos = (float)x;
ypos = height - y - 1;
glReadPixels(xpos, ypos, 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &zpos);
//std::cout << "3D Position: " << xpos << " , " << ypos << " , " << zpos << std::endl;

modifier_vertex = glm::unProject(glm::vec3(xpos, ypos, zpos), modelview, projection, glm::vec4(0.0f, 0.0f, width, height));
std::cout << "World Coordinates: " << modifier_vertex.x << " , " << modifier_vertex.y << " , " << modifier_vertex.z << std::endl;
manip = modifier_vertex;

// finding the near value of vertex selected
std::vector<vertex> allVertexvalues = Object1.getAllVertices();
int i;
float difference = 100;
for (i = 0; i < allVertexvalues.size(); i++) {
    float temp = sqrt(((modifier_vertex.x - allVertexvalues[i].v[0]) * (modifier_vertex.x - allVertexvalues[i].v[0])) +
        ((modifier_vertex.y - allVertexvalues[i].v[1]) * (modifier_vertex.y - allVertexvalues[i].v[1])) +
        ((modifier_vertex.z - allVertexvalues[i].v[2]) * (modifier_vertex.z - allVertexvalues[i].v[2])));
    if (temp < difference) {
        difference = temp;
        vertex_pos = i;
    }
}
std::cout << "Position: " << vertex_pos + 1 << std::endl;
modifier_vertex = glm::vec3(allVertexvalues[vertex_pos].v[0], allVertexvalues[vertex_pos].v[1], allVertexvalues[vertex_pos].v[2]);
```

Figure 1: Select Vertex

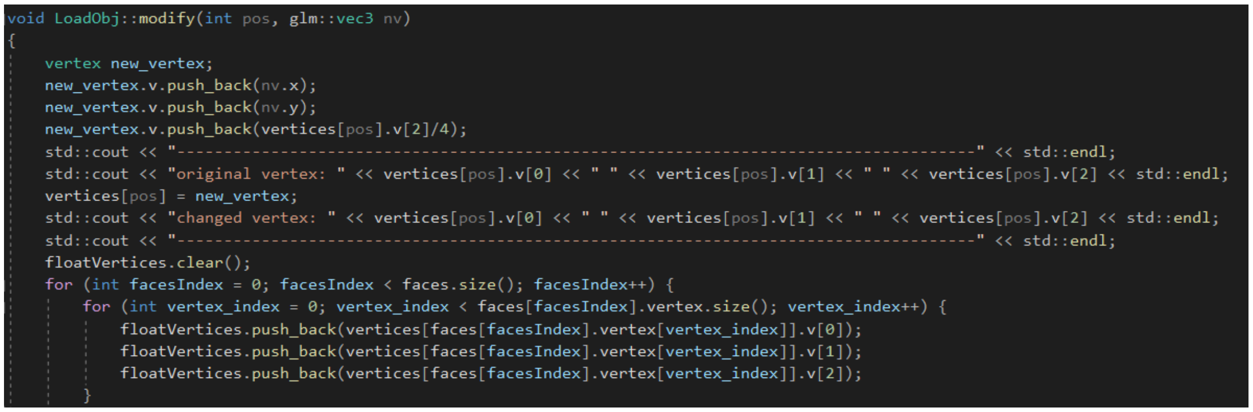
  Moving: From the selection process, the position of the desired vertex which will be changed is found. Now as the cursor moves in the screen, I have repeated the same process of world co-ordinate conversion here and passed that value to the mesh, so that it can be updated in real time.



```
new_vertex = glm::unProject(glm::vec3(xpos, ypos, zpos), modelview, projection, glm::vec4(0.0f, 0.0f, width, height));
std::cout << "World Coordinates: " << new_vertex.x << " , " << new_vertex.y << " , " << new_vertex.z << std::endl;

Object1.modify(vertex_pos, new_vertex);
generateObjectBuffer();
display();
TwDraw();
glfwSwapBuffers(window);
```

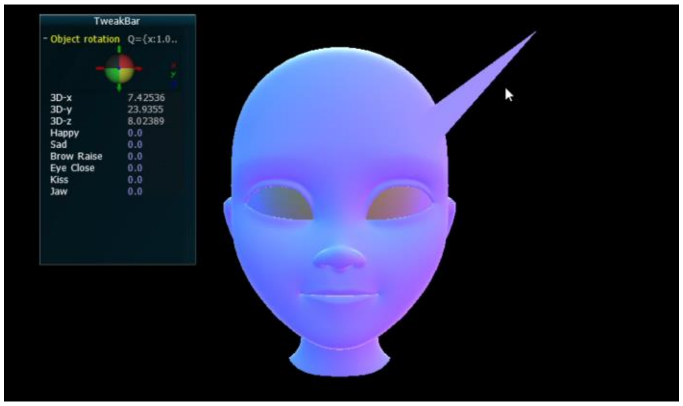
Figure 2: Call to modify function after getting new vertex



```cpp
void LoadObj::modify(int pos, glm::vec3 nv)
{
    vertex new_vertex;
    new_vertex.v.push_back(nv.x);
    new_vertex.v.push_back(nv.y);
    new_vertex.v.push_back(vertices[pos].v[2]/4);
    std::cout << "----------------------------------------------------------------------------------" << std::endl;
    std::cout << "original vertex: " << vertices[pos].v[0] << " " << vertices[pos].v[1] << " " << vertices[pos].v[2] << std::endl;
    vertices[pos] = new_vertex;
    std::cout << "changed vertex: " << vertices[pos].v[0] << " " << vertices[pos].v[1] << " " << vertices[pos].v[2] << std::endl;
    std::cout << "----------------------------------------------------------------------------------" << std::endl;
    floatVertices.clear();
    for (int facesIndex = 0; facesIndex < faces.size(); facesIndex++) {
        for (int vertex_index = 0; vertex_index < faces[facesIndex].vertex.size(); vertex_index++) {
            floatVertices.push_back(vertices[faces[facesIndex].vertex[vertex_index]].v[0]);
            floatVertices.push_back(vertices[faces[facesIndex].vertex[vertex_index]].v[1]);
            floatVertices.push_back(vertices[faces[facesIndex].vertex[vertex_index]].v[2]);
        }
```

Figure 3: Modify function in loader to change the mesh

Output:



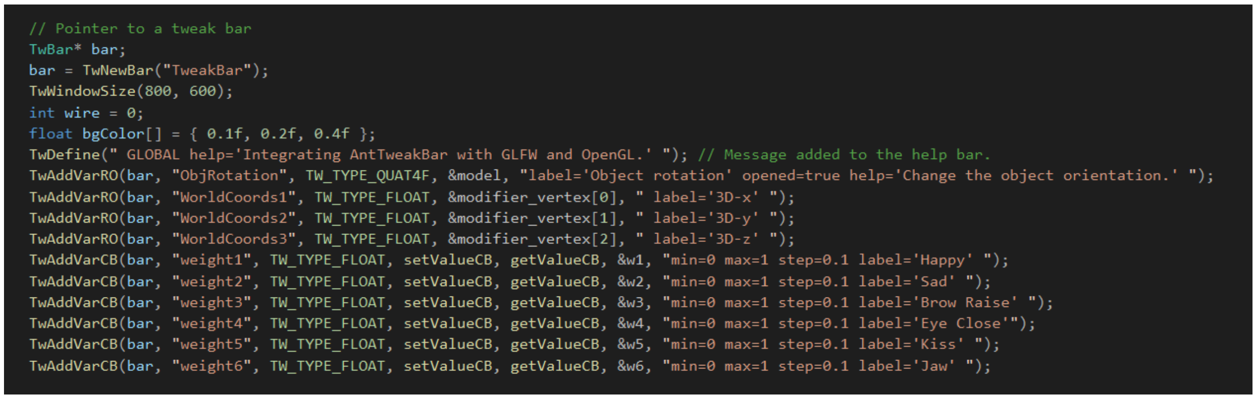
```
3D-x         7.42536
3D-y         23.9355
3D-z         8.02389
Happy        0.0
Sad          0.0
Brow Raise   0.0
Eye Close    0.0
Kiss         0.0
Jaw          0.0
```

Figure 4: One vertex selected and moved in the mesh

Note: The selected vertices could be seen on the AntTweakBar.

- **A user interface to set blendshape weights:**

The user interface is created using **AntTweakBar**. Once the bar is initialized and its window is created, "***TwAddVarCB***" is used to set and get the weights for the blendmeshes. The reason why CallBack function is used instead of ReadWrite is because on every weight update, the meshes were being manipulated and a function was needed to do that.



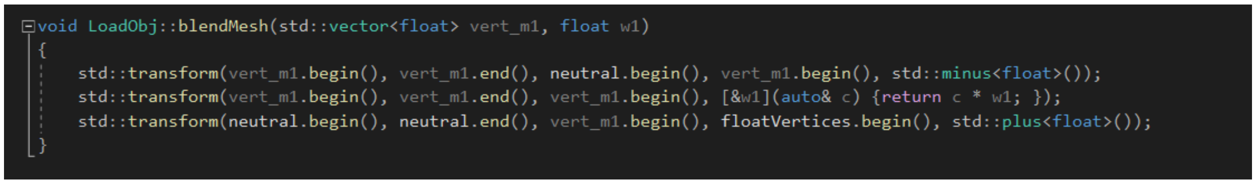
```cpp
// Pointer to a tweak bar
TwBar* bar;
bar = TwNewBar("TweakBar");
TwWindowSize(800, 600);
int wire = 0;
float bgColor[] = { 0.1f, 0.2f, 0.4f };
TwDefine(" GLOBAL help='Integrating AntTweakBar with GLFW and OpenGL.' "); // Message added to the help bar.
TwAddVarRO(bar, "ObjRotation", TW_TYPE_QUAT4F, &model, "label='Object rotation' opened=true help='Change the object orientation.' ");
TwAddVarRO(bar, "WorldCoords1", TW_TYPE_FLOAT, &modifier_vertex[0], " label='3D-x' ");
TwAddVarRO(bar, "WorldCoords2", TW_TYPE_FLOAT, &modifier_vertex[1], " label='3D-y' ");
TwAddVarRO(bar, "WorldCoords3", TW_TYPE_FLOAT, &modifier_vertex[2], " label='3D-z' ");
TwAddVarCB(bar, "weight1", TW_TYPE_FLOAT, setValueCB, getValueCB, &w1, "min=0 max=1 step=0.1 label='Happy' ");
TwAddVarCB(bar, "weight2", TW_TYPE_FLOAT, setValueCB, getValueCB, &w2, "min=0 max=1 step=0.1 label='Sad' ");
TwAddVarCB(bar, "weight3", TW_TYPE_FLOAT, setValueCB, getValueCB, &w3, "min=0 max=1 step=0.1 label='Brow Raise' ");
TwAddVarCB(bar, "weight4", TW_TYPE_FLOAT, setValueCB, getValueCB, &w4, "min=0 max=1 step=0.1 label='Eye Close'");
TwAddVarCB(bar, "weight5", TW_TYPE_FLOAT, setValueCB, getValueCB, &w5, "min=0 max=1 step=0.1 label='Kiss' ");
TwAddVarCB(bar, "weight6", TW_TYPE_FLOAT, setValueCB, getValueCB, &w6, "min=0 max=1 step=0.1 label='Jaw' ");
```

Figure 5 Setup of AntTweakBar

- **Load facial expressions, save them internally as delta-blendshapes and update a facial expression after blendshape weights have been manipulated:**
  I have used the blendshapes given on Blackboard. I have loaded them using my own loader file which reads the object file and writes its contents into respective vectors for generating buffers and rendering. I have used *11 blendshapes [Neutral, Happy(left + right), Sad(left + right), Brow Raise(left + right), Eye Closed(left + right), Kiss and Jaw Open]* including the neutral for the project.
  I have also created a header file, named ***obj_manip.h*** which is used to read and write the object files and save them internally.
  Once the blendshapes are loaded and weights get changed, the function "***blendMesh***" from loader.cpp is called which takes all the meshes and their weights as arguments and manipulate them accordingly. I have used this formula for the same:

$$f(w) = f_0 + \sum_{k=1}^{K} w_k (f_k - f_0)$$

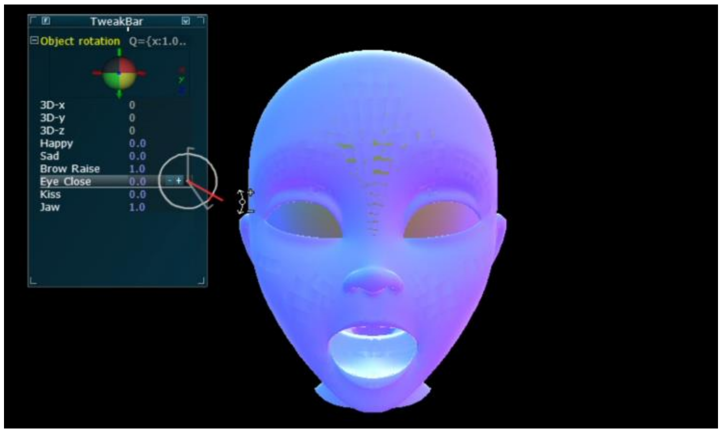
$$w = [w_1, \dots, w_k]^T$$



```
void LoadObj::blendMesh(std::vector<float> vert_m1, float w1)
{
    std::transform(vert_m1.begin(), vert_m1.end(), neutral.begin(), vert_m1.begin(), std::minus<float>());
    std::transform(vert_m1.begin(), vert_m1.end(), vert_m1.begin(), [&w1](auto& c) {return c * w1; });
    std::transform(neutral.begin(), neutral.end(), vert_m1.begin(), floatVertices.begin(), std::plus<float>());
}
```

Figure 6: Overloaded version of blendMesh which takes only one Blendshape, same formula is used for more meshes in the code.

Output:



```
TweakBar
Object rotation   Q={x:1.0..
3D-x          0
3D-y          0
3D-z          0
Happy         0.0
Sad           0.0
Brow Raise    1.0
Eye Close     0.0
Kiss          0.0
Jaw           1.0
```

Figure 7: Adjusting weights using UI

- **Compute blendshape weights from manipulators:**
  I have changed the Eigen direct manipulation code that was given with the assignment. By using thw following formula, I tried to implement it:

$$A^T A \;\; x = A^T c \;\rightarrow\; \left(\bar{B}^{\mathrm{T}}\bar{B} + (\alpha + \mu)I\right)w = \bar{B}^{\mathrm{T}}(m - m_0) + \alpha w_{t-1}$$

Here, the B matrix is the collection of vertices from all the meshes which are constrained (3, in the code i.e. B has size of 9x11, because every selected vertex adds 3 rows to the matrix and I am using 11 vertices).
The m0 is the original position of the vertex and m is the final position where the mouse drags it. We calculate w based on the current weights.
Also, the selection of constraints is done using ***right mouse button*** and changes have been made by dragging the same.

```
A = B.transpose() * B + (alpha + mu) * Eigen::MatrixXf::Identity(num_cols, num_cols);
b = B.transpose() * b + alpha * wt;
Eigen::LDLT<Eigen::MatrixXf> solver(A);
Eigen::VectorXf w = solver.solve(b);
```

Figure 8: snippet from the function where weight calculation is being done

Output:

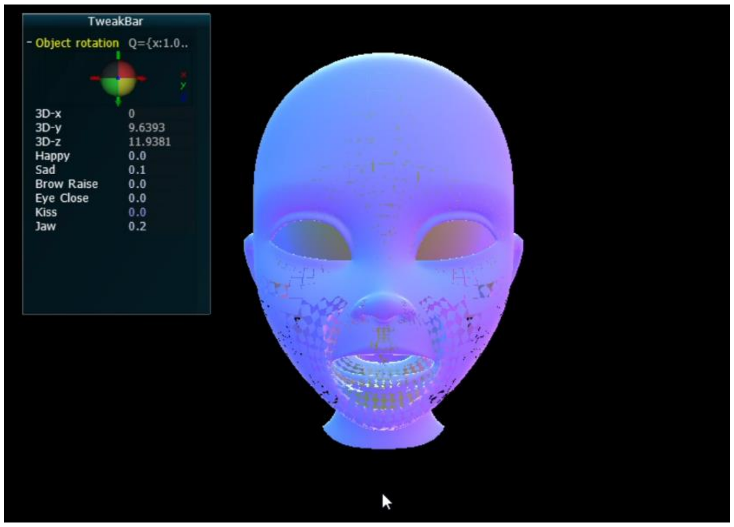

```
TweakBar
- Object rotation   Q={x:1.0..
3D-x         0
3D-y         9.6393
3D-z         11.9381
Happy        0.0
Sad          0.1
Brow Raise   0.0
Eye Close    0.0
Kiss         0.0
Jaw          0.2
```

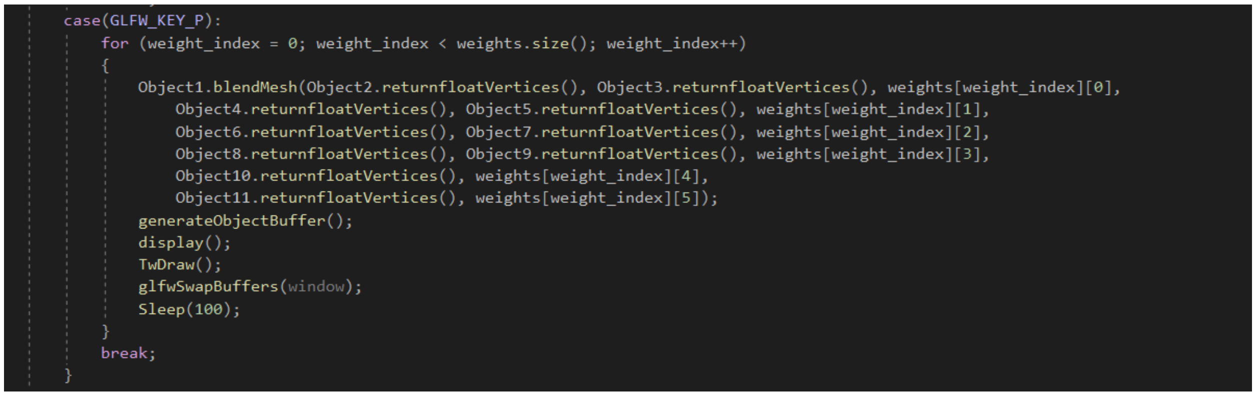
Figure 9: Changing jaw position on mouse drag

Note: The selected vertices could be seen on the AntTweakBar.

➢ **Extra marks for animation playback:**
For the playback part, I have read the weights that were given on BlackBoard. Since, I did not use all the meshes, I have only read the first few weights which were needed. I have stored these weights into a vector and as the user presses the key **P**, the playback starts. A loop runs from **0** to the **weights.size()** and all the calculation is done using **blendMesh** for all the weights. For the animation to be smoother, I have used **Sleep(500)**, so that every change stays on the screen for 0.5 seconds.

```
std::vector<float> w;
std::stringstream stringstream(line);
for (int i = 0; i < 6; i++)
{
    stringstream >> key >> std::ws;
    w.push_back(key);
}
weights.push_back(w);
```

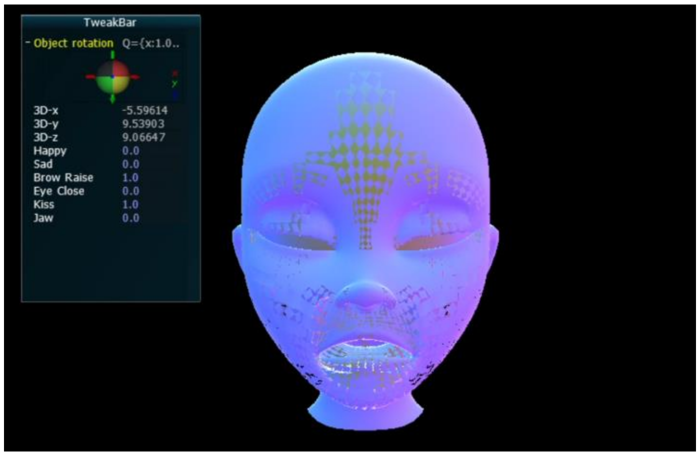
Figure 10: saving the weights from file in a vector



```
    case(GLFW_KEY_P):
        for (weight_index = 0; weight_index < weights.size(); weight_index++)
        {
            Object1.blendMesh(Object2.returnfloatVertices(), Object3.returnfloatVertices(), weights[weight_index][0],
                Object4.returnfloatVertices(), Object5.returnfloatVertices(), weights[weight_index][1],
                Object6.returnfloatVertices(), Object7.returnfloatVertices(), weights[weight_index][2],
                Object8.returnfloatVertices(), Object9.returnfloatVertices(), weights[weight_index][3],
                Object10.returnfloatVertices(), weights[weight_index][4],
                Object11.returnfloatVertices(), weights[weight_index][5]);
            generateObjectBuffer();
            display();
            TwDraw();
            glfwSwapBuffers(window);
            Sleep(100);
        }
        break;
    }
```

Figure 11: KeyPress for Animation

Output:



```
TweakBar
- Object rotation   Q={x:1.0..
3D-x         -5.59614
3D-y         9.53903
3D-z         9.06647
Happy        0.0
Sad          0.0
Brow Raise   1.0
Eye Close    0.0
Kiss         1.0
Jaw          0.0
```

Figure 12: A still from playback

Note: The weights from the file are not shown on the UI.

- **Additional Libraries used:**
  - GLFW3
  - Glew
  - GLM
  - AntTweakBar
  - Eigen

- **Shortcomings in the project:**
  - Although the code is working fine without it, but I could not get the sphere or circle to point on the selected vertex (I have included the code for it in the files).
  - In direct manipulation, only a few weights change on mouse drag. Maybe it's supposed to be like that only, but I found it odd.
- **References:**
  - Lecture notes for facial animation.
  - https://sourceforge.net/p/anttweakbar/code/ci/master/tree/examples/TwSimpleGLFW.c#l141

- http://antongerdelan.net/opengl/blend_shapes.html
- https://stackoverflow.com/questions/23834680/anttweakbar-glfw3-opengl-3-2-not-drawing
- https://gamedev.stackexchange.com/questions/71472/using-gluunproject-to-transform-mouse-position-to-world-coordinates-lwjgl
- https://stackoverflow.com/questions/24175414/glitchy-facial-morph-target-animation-in-opengl-c