AI Face Simulation

Sophia Kolak - sdk2147@columbia.edu

***Abstract*— This report describes the design and implementation of a robotic face simulation, written in C++ using the Titan graphics library.**

I. Introduction

As human beings, we convey an incredible amount of information using only our facial expressions. Empathy, sympathy, love, admiration, respect, all of these relational emotions are commonly expressed in the way our faces transform in response to information. A significant part of our ability to relate to others naturally comes from this physical and visual facial movement. Perhaps the most fascinating aspect of this information transfer, is that both the action and its interpretation are typically understood implicitly.

For machines, however, these naturalistic human transformations do not mean anything. What a person automatically interprets as a frown is simply a re-arrangement of pixels to the robot. As a result, the question of how to respond to this motion requires learning from data, the same way many children likely figured out the socially expected way to respond to others.

Of course, the problem of accurately copying something as nuanced and emotionally charge as emotional facial expressions is a massive undertaking. In order to still approach the problem, however, we can break it down into three major sub-problems.

1) How does the machine interpret, store, and analyze the input (the view of another face)

2) How does the machine learn to respond to this input in digital space

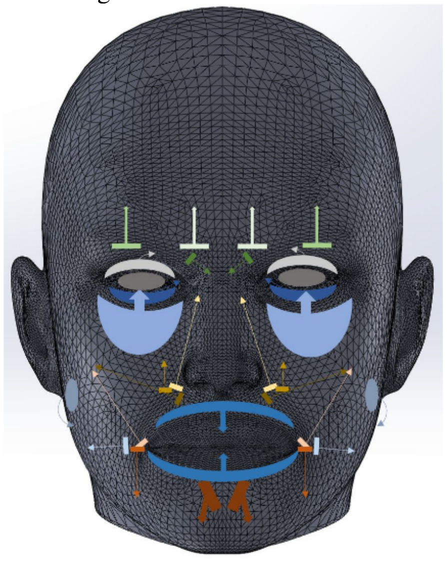
3) How can we translate this digital motion into the physical world?

The primary concern of this work is addressing the third problem—how can we map motion defined digitally into the physical world?

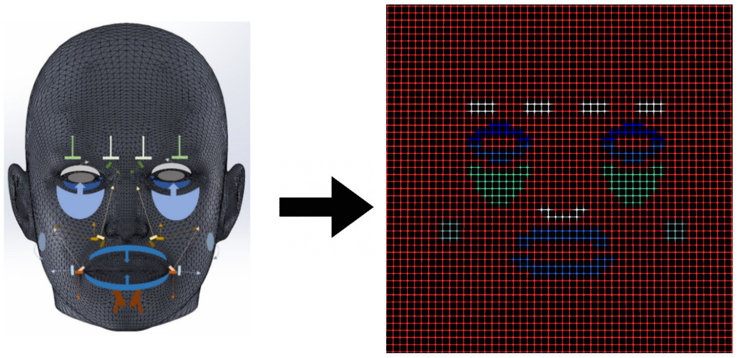
To bridge these two domains, I created a mass and spring simulation of the robotic face using Titan[1], a CUDA-based physics library. Through this simulation we can automatically translate between the digital motion represented in video and image datasets, and the real motions we want the robotic face hardware to make.

II. Robotic Face Model

The model for the physical robotic face consists of 16 muscles with directions of motion defined along the arrows displayed in the image below.



In simulation, there exists an analogous section of masses and springs for each muscle that appears on the robotic face. According to Lin, Huang, and Chen [2], the maximum displacement of the human face involved in facial expressions is approximately 15mm. Thus, the simulation was designed such that the corresponding muscles would expand by up to 15mm max in the defined directions.



The outer-layer of the robot's face, represented in Figure 1 is made of rubber, and each of the colored muscles corresponds to a wire anchor on the interior of the face. The wires attached to the anchors are then controlled by an underlying set of servos. At the level of abstraction of this simulation, it is important only to note that the range of motion for each muscle is a potential expansion of 15mm in the direction indicated by the arrows, and their representation

[1]https://github.com/ja3067/Titan

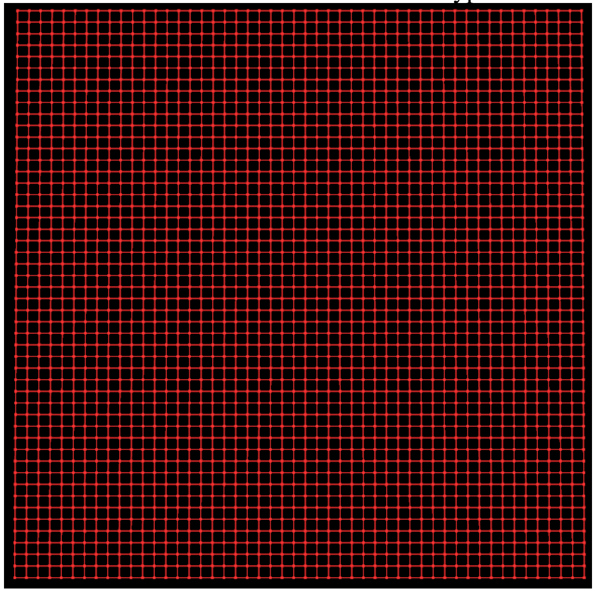
[2]Lin, Chyi-Yeu, Chun-Chia Huang, and Li-Chieh Cheng. "An ExpressionalSimplified Mechanism in Anthropomorphic Face Robot Design." Robotica 34.03(2014): 652-70. Web

in simulation appears as shown on the red grid in Figure 2.

III. The Grid

The points in this grid representation are masses, and the lines between them are springs. While these objects were defined in code using Titan's mass and spring objects, underneath this naming structure they are ultimately CUDA objects.

Each of 50 rows of masses contains 50 masses and 49 springs, making this a grid of 2500 masses and 4950 springs. The dimension of the grid is a global variable that can easily be reset to define grids of different sizes. For the sake of this task, 50x50 was a near optimal choice, since it provides enough detail for defining the muscles accurately, but not too much detail as to burden the GPU on a typical machine.



All masses in the grid are stored in a vector called `mass_vec`, where `mass_vec[0]` is the upper left corner and `mass_vec[2499]` is the bottom right corner.

Spring objects are defined between two given Masses. Where spring objects record the masses that they connect in Titan, mass objects contain no information about which springs are between them. As a result, there are two different vectors which store springs, organized such that there is a natural connection between regions of the grid (masses) and the springs. `spring_vec` contains all of the horizontal springs, and `spring_vec_vert` contains all of the vertical springs.

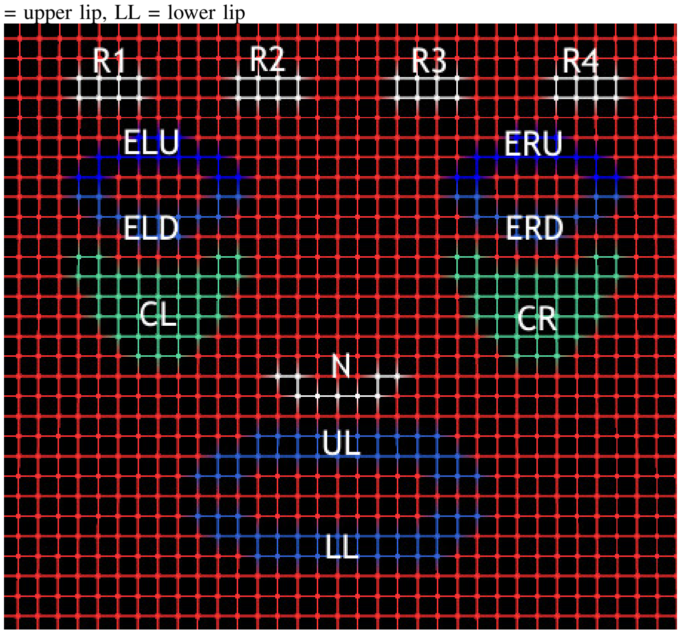
```
void make_rows(Simulation *sim){
    /**creates row of springs **/
    int row = 0;
    for(int i = 0; i < dim; i++){
        for(int j = 0; j < dim-1; j++){
            //scale by row*dim to make all the rows
            Spring * x =
            sim->createSpring(mass_vec[j+(row*dim)],
            mass_vec[j+ (row*dim) + 1 ] );
            spring_vec.push_back(x);
            full_springs.push_back(x);
        }
        row++;
    }
}

void make_columns(Simulation * sim){
    /** creates columns of springs **/
    int c = 0;
    for (int i = 0; i < dim; i++){
        for(int j = 0; j < mass_vec.size();
        j = j + dim){
            Spring * x = sim->createSpring(
            mass_vec[i+j],
            mass_vec[i+j+dim]);
            spring_vec_vert.push_back(x);
            full_springs.push_back(x);
            c++;
        }
    }
}
```

Full Code at https://github.com/sophiakolak/aiface_sim

A. The Muscles

Fig. 1. R1 = rectangle 1 R2 = rectangle 2, R3 = rectangle 3, R4 = rectangle 4, ELU = eyelid left up, ELD = eyelid left down, ERU = eyelid right up, ELD = eyelid right down. CL = cheek left CR = cheek left. N = nose, UL = upper lip, LL = lower lip



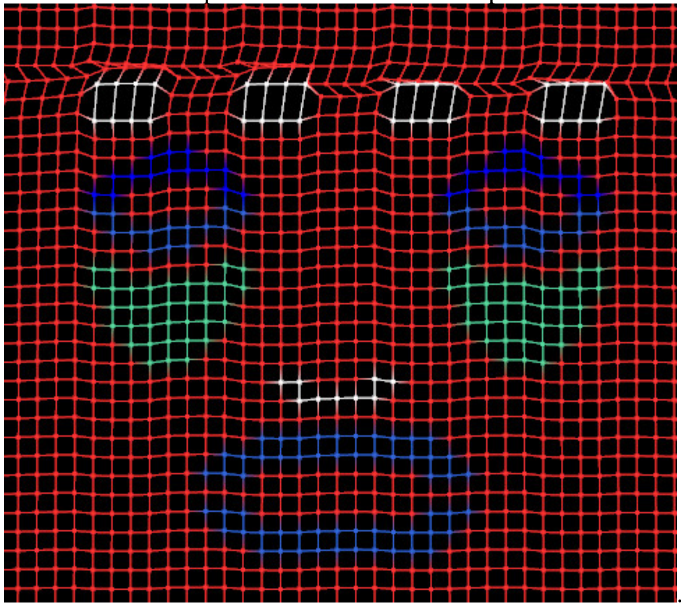
As shown in the image above, all facial regions are defined and labeled according to the region of the face they represent. A given muscle is simply a container of masses and springs. A container is a Titan object that allows for the manipulation of springs and masses as one unit. To provide the functionality for moving these regions together, two containers are defined for each muscle.

First, there is the underlying container which allows for the color change and index identification on the grid. Secondly, there is a container which only includes the springs and masses for the direction of intended motion. For instance, R1-R4 allow for upwards vertical motion. Thus we have both the definition of `rn` and `rn_vert` for each rectangle.

```
void make_r1(Simulation *sim, int init_row)
    Container * r1 = make_rec(init_row,sim);
    Container * r1_vert = make_rec_vert(init_row,sim);
```

B. Muscle Motion

In order to give the appearance of real muscles, it was necessary that some of the springs stayed in their fixed position. Again using the rectangular example, the vertical container `r1_vert` was defined such that each spring composing the lower row would remain fixed using the function `mass->fix()`. This can be observed in the upward motion of R1-R4 pictured below.



In code, this effect was achieved by the fixing one row as shown. The index is scaled by

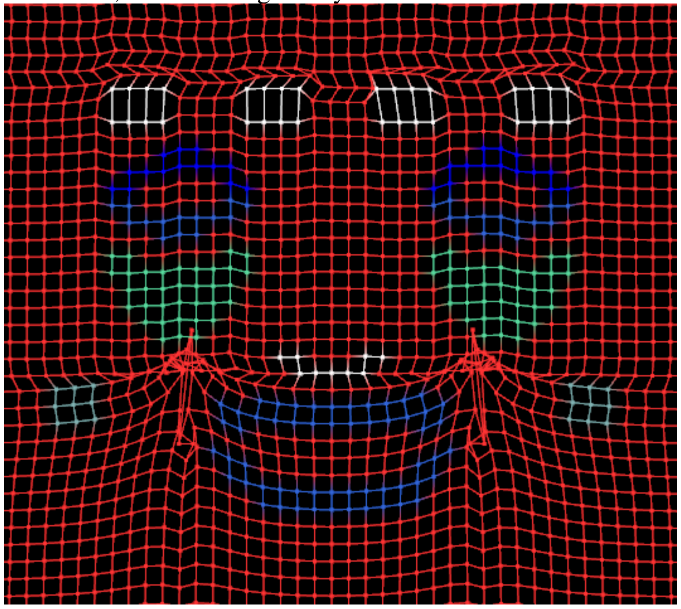
```
Container * make_rec_vert(int init_row, Simulation *sim){
  Container * r = sim->createContainer();
  for (int i=init_row; i < init_row+4; i++){
    (spring_vec_vert[
    (i*dim) + init_row]->_left)->color = Vec(1,1,1);
    (spring_vec_vert[
    (i*dim) +  ]->_right)->color = Vec(1,1,1);
    r->add(spring_vec_vert[(i*dim) + init_row]);
    (spring_vec_2[(i*dim)+row_init]->_right)->fix();
    r->add(spring_vec_vert[(i*dim)+row_init]->_left);
    r->add(spring_vec_vert[(i*dim)+row_init]->_right);
  }
  return r;
}
```

To create deformations in the grid which appeared to be skin-like or rubbery, the spring constant k was set to be extremely high (800) within the simple `set_all_k` function, meaning that the grid was generally quite loose, and a muscle's motion could impact other muscles in distant regions of the grid.

```
void set_all_k(Simulation *sim){

    for (int i = 0; i < full_springs.size(); i++){
        full_springs[i]->_k = 800;
    }
}
```

Creating the actual motion was achieved by manipulating the rest lengths of mass containers corresponding to directions of motion of the muscle in question. Using the Titan function `setAllSpringValues(val)`, which acts on a `container` object. Figure 6 shows the effect achieved when we expand the rest length of springs at the corners of the mouth, while raising the eyebrows.



IV. Obtaining Position

Titan has many pre-existing functions for obtaining the motion of the masses, and thus, determining how the over-arching structure has moved in response to the simulated actions. Using the built in functions such as `getPosition`, the simulation can be used to inform motion in the physical world.

V. Conclusions and Future Work

While the simulation is quite minimalistic, it accurately captures the physical aspect of facial motion at a level of detail appropriate for the robotic face it was designed for, providing an essential piece in the ongoing effort to design a machine that can learn facial responses.

Using this underlying simulation technique as well as texture mapping from images and videos of human faces, the physical robotic face could be trained to respond actively to other faces it encounters. By testing motion and developing learning algorithms on the simulation rather than the real robot, a great deal of time and resources are saved. Testing motion on the physical face is both risky in that in can lead to hardware malfunctions, and time consuming since there is only one physical machine.

Future work that naturally follows from this underlying structure involves connecting the simulation software with the deep learning algorithm for generating facial responses. All code written for this project is publicly available at https://github.com/sophiakolak/aiface_sim

References

[1] Lin, Chyi-Yeu, Chun-Chia Huang, and Li-Chieh Cheng. "An ExpressionalSimplified Mechanism in Anthropomorphic Face Robot Design." Robotica 34.03(2014): 652-70. Web