CS4600

Andrew Fryzel u1070220

Homework 4

Part 1: Skeletal animation

I tried to implement the skeletal animation by using the trick found in Lecture 11 on slide 10:

> Better algorithm?
> Trick: assume that p(j) < j
> Compute F(j) in one for-cycle: F(j) = F(p(j)) R(j) T(j)
> - Assumes F(p(j)) was already computed
> - A(j) is easy special case (with T(j) = identity)

The most difficult part was understanding the parameters and making sense of what needed to go where. Reading the homework assignment description really helped to make sense of what was what, especially in relation to the code from the slides. I tried to transcribe and combine the code from the slides and the writeup descriptions as best as I could in the code. I commented the code to reflect the combinations of the "trick" formula and the homework description pdf information:

```
// TASK 1 comes here
//R = p_offset
//T = p_local

for (unsigned int j = 0; j < p_numJoints; j++)
{
    if (j == 0)
    {
        //From the homework description: p_global[j] = R(0) T(0)
        p_global[j] = p_offset[j] * p_local[j];
    }
    else {
        //p_jointParent has p_numJoints elements
        //F(j) = F(p(j)) R(j) T(j)
        p_global[j] = p_global[p_jointParent[j]] * p_offset[j] * p_local[j];
    }
}
```

Part 2: Linear blend skinning

I used the formula from slide 15 of Lecture 11 to solve this section of the assignment.

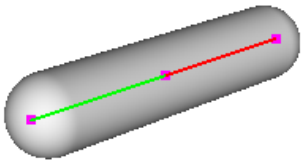
$$\mathbf{v}' = \sum_{i=1}^{m} w_i F(j_i) A(j_i)^{-1} \mathbf{v}$$

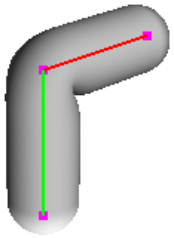
I wasn't confident if I should use a Vector3f or Vector4f and decided to err on the side of caution and use the Vector4f. After setting all the values to 0 in the vector, the final positions can be calculated by transforming the vertex toHomog, multiplying by the appropriate values at the right indexes, and then by computing fromHomog back to a final value. These final values are the p_deformedVertices. It took some trial and error to figure out what went where and with what indices.

```
// The following code simply copies rest pose vertex positions
// You will need to replace this by your solution of TASK 2
for (unsigned int v = 0; v < p_vertices.size(); v++)
{
    //Set all to 0, deformed vertices
    Vector4f homog2;
    homog2[0] = 0;
    homog2[1] = 0;
    homog2[2] = 0;
    homog2[3] = 0;

    //Iterate and set
    for (unsigned int i = 0; i < p_numJoints; i++)
    {
        homog2 += p_weights[i][v] * p_jointTrans[i] * p_jointTransRestInv[i] * toHomog(p_vertices.at(v));;
    }

    //From Homog
    p_deformedVertices[v] = fromHomog(homog2);
}
```

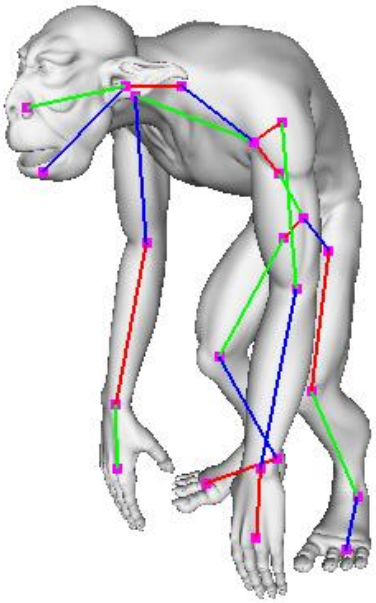
Images

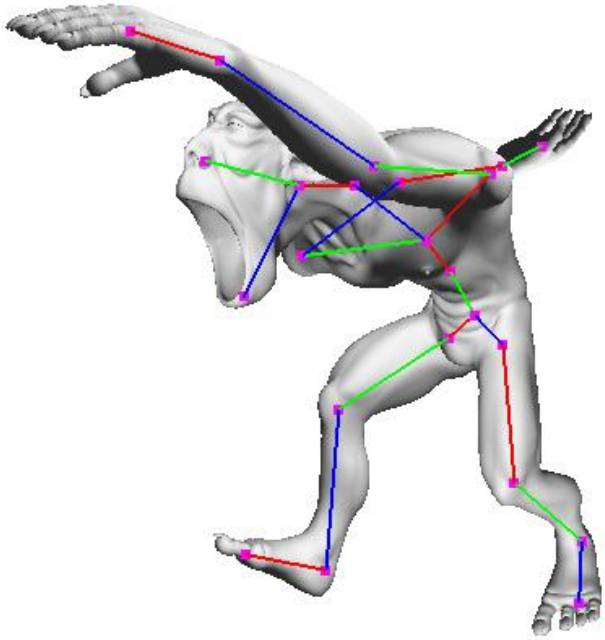
Capsule:





Ogre:



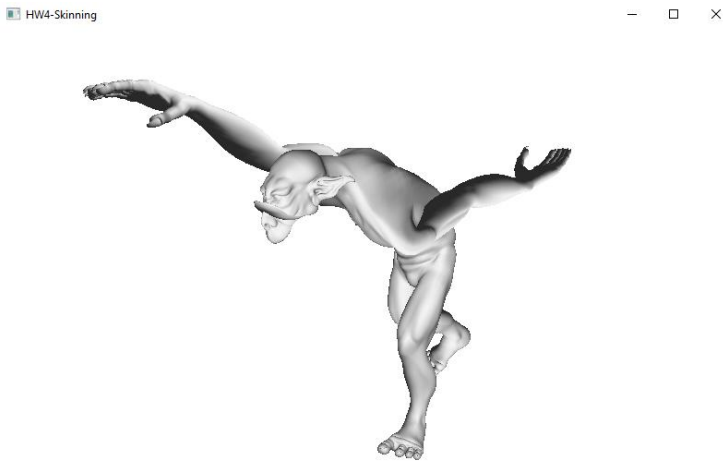


Part 3: Extra Credit

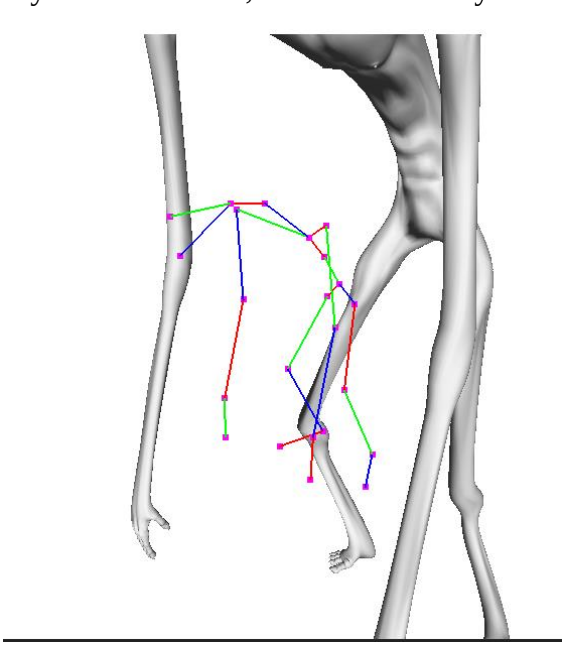
Dual Quaternion

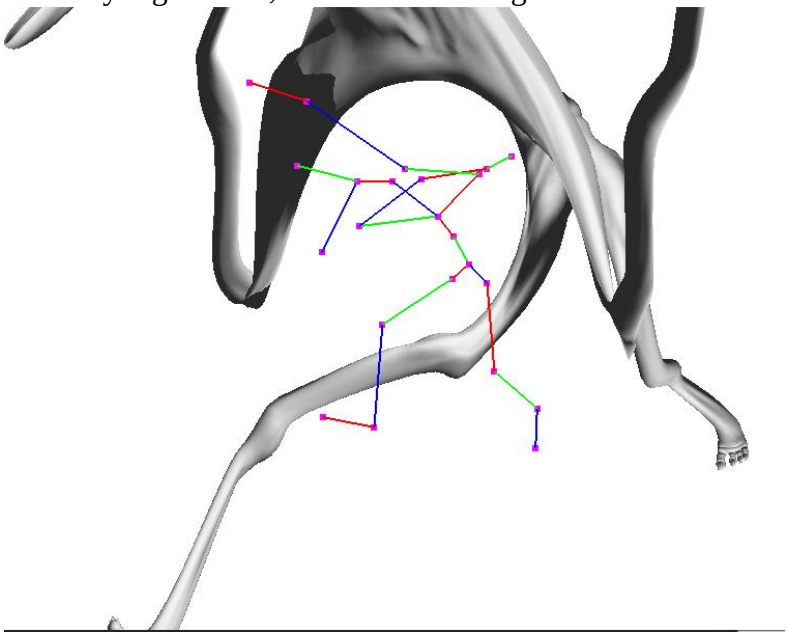
It took me a while to figure out most of this extra credit. I wasn't able to get the Dual Quaternion Skinning to work and had to remove it because I was getting some really wild results. Despite not getting it fully functional, I decided to include my work on it here.

The first problem was that certain parts of the model would overlap with each other and weren't correctly stopping when they were supposed to, for example:



My favorite results, that went horribly and will be in my nightmares, were the following:





After unsuccessfully implementing Dual Quaternion skinning, I went ahead and scrapped it and tried to focus on the animation and DMAT file.

Dual Quaternion Sources Used:

http://rodolphe-vaillant.fr/?e=29

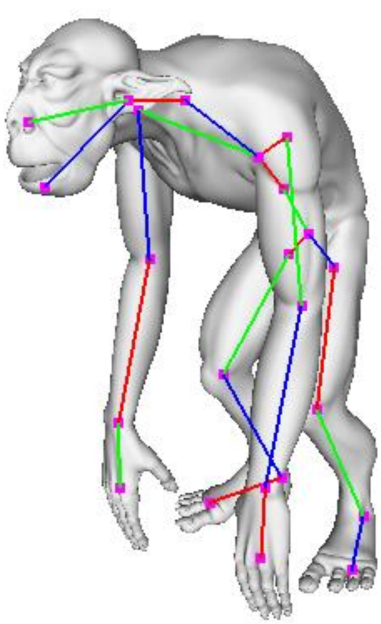
https://www.cs.utah.edu/~ladislav/kavan07skinning/kavan07skinning.pdf

https://www.cs.utah.edu/~ladislav/dq/index.html

DMAT and Animation

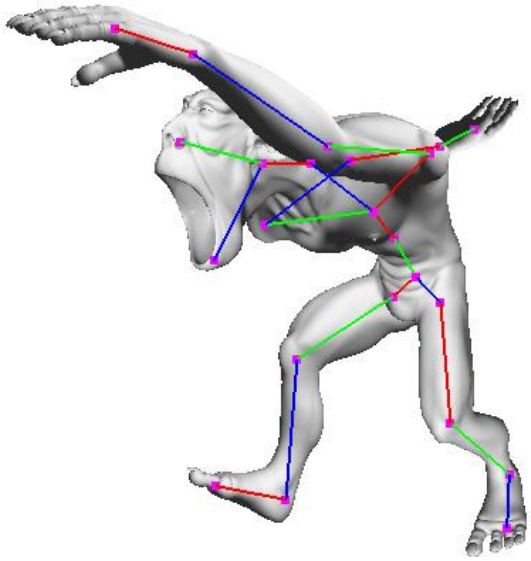
I had to use several outside resources to figure out how the DMAT file worked and I am still not completely confident that I fully understand it. Changing certain value seemed to have no effect but changing other had a huge impact. In the end I found the each chunk of matrice corresponded to one of the animation frames. So the first paragraph was the default pose, the second paragraph was second pose, and so in order to invert the walking, I flipped the positive and negative values and created a third paragraph. It took a little bit and I found that negating the second column values changed the mouth, changing the first column did the desired result, and changing the last two columns didn't seem to have a huge effect. In the end, I copied the second paragraph, negated the first column and it ended up flipping the final pose as desired. After this was done, the result was a smoother and fuller walking animation.

Pose 1 (Paragraph 1)



```
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
```

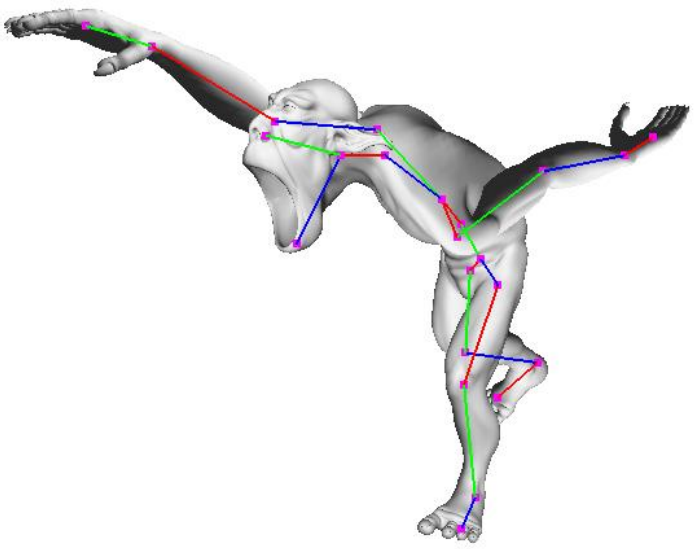
Pose 2 (Paragraph 2)



```
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
-0.9848 0.1736 0.0 0.0
-0.9848 0.1736 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
-0.9961 -0.08715 0.0 0.0
-0.9961 -0.08715 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
-0.9239 -0.3827 0.0 0.0
-0.8660 -0.5 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
0.9239 -0.3827 0.0 0.0
0.8660 -0.5 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
0.9848 0.1736 0.0 0.0
1.0 0.0 0.0 0.0
0.9848 -0.1736 0.0 0.0
1.0 0.0 0.0 0.0
```

Pose 3 (Paragraph 3)

Interestingly enough, changing the final two 0.9848 values to negative messed up the mouth. Honestly, the DMAT structure is still pretty confusing, but through trial and error I was able to reverse the second pose to create the third pose.



```
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
0.9848 0.1736 0.0 0.0
0.9848 0.1736 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
0.9961 -0.08715 0.0 0.0
0.9961 -0.08715 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
0.9239 -0.3827 0.0 0.0
0.8660 -0.5 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
-0.9239 -0.3827 0.0 0.0
-0.8660 -0.5 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0
0.9848 0.1736 0.0 0.0
1.0 0.0 0.0 0.0
0.9848 -0.1736 0.0 0.0
1.0 0.0 0.0 0.0
```

The code for the animation also took a little while to fully figure out. The hardest part was knowing what to change.

The first section of code that I changed was making the static image display when the animation poses are cycled through using the ‘A’ key. I needed to add an extra case to go to the third pose. The value I passed into setJointRotation was mostly a lucky guess based on having to invert all of the DMAT file values. I figured that inverting the second poses value (the value 1.0f) would work and it did.

```
void animate()
{
    //allow negative values
    const float t = sinf(getTime());

    switch (g_enableAnimate)
    {
    case 0: setJointRotations(t); break;

    case 1: setJointRotations(0.0f); break;

    case 2: setJointRotations(1.0f); break;

        //allow negative
        //changed
        //allow the 3rd image
    case 3: setJointRotations(-1.0f); break;

    }
    computeJointTransformations(g_jointRot, g_jointOffset, g_jointParent, g_numJoints, g_jointTrans);
    skinning(g_vertices, g_numJoints, g_jointTrans, g_jointTransRestInv, g_weights, g_deformedVertices);
}
```

The code that took the longest to figure out, and was also the smalles, was this:

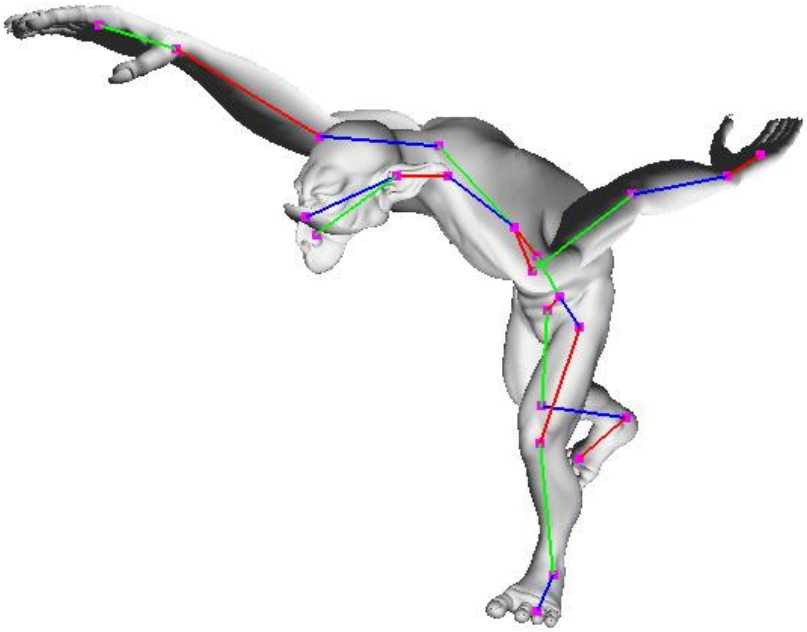
```
if (p_key == GLFW_KEY_A && p_action == GLFW_PRESS)
{
    //there are now 4 animation frames / things
    g_enableAnimate = (g_enableAnimate + 1) % 4;
}
```

When I was cycling through the poses, my final pose would never actually show up and I couldn’t figure out why. I needed to add this code since the were now 4 cases, I had to also mod by 4 and not by 3.

The final section of code changed was in the setJointRotations method:

```
//changed
int val = 1;

if (t < 0)
{
    //set t to positve val
    t = (-1) * t;
    val = 2;
}

for (unsigned int jointID = 0; jointID < g_numJoints; ++jointID)
{
    //changed
    Quaternionf qB = Quaternionf(g_poses[val][4 * jointID], g_poses[val][4 * jointID + 1], g_poses[val][4 * jointID + 2], g_poses[val][4 * jointID + 3]);
```

The challenge here was figuring out what to change the ‘val’ value to. When it was just set to 0, nothing happened at all; when it was set to 1 or 2 the third pose’s mouth clipped through the head:



Since the value being passed in to setJointRotation from the third case was negative (-1.0f), I had to do a check for values < 0, and then set the values to a positive value. Setting the value to 1 after the check didn’t switch to pose 3 but setting it to 2 did. Through a bunch of testing and trying out different thing, I found the code that worked. Also, the “t” value has to be changed to a non negative number other wise the lower jaw clips through the head again like above.

Overall, I was able to extend the animation to create a more complete walking animation. I attempted to implement Dual Quaternions but wasn’t successful.

DMAT and Animation Resources Used:

https://igl.ethz.ch/projects/libigl/file-formats/dmat.html