Comp 410: Computer Graphics

Term Project Report

Sitare Arslantürk 57677

Furkan Yakal 59943

BODY MODEL ANIMATION

Motivation:

The main goal of this project is to successfully simulate the human body movements basics such as walking and running under different gender and age groups, using OpenGL. The user is given the ability to start-stop the motion and rotate the body-model for observing the movements from different angles in the graphical interface.

Background Information:

Golden Ratio in Human Body:

The use of golden ratio is used to give aesthetic pleasure in design. This ratio is found by dividing a line into two parts in which the ratio of the longer part divided by the smaller part is equal to the ratio of total length divided by the longer part. Leonardo da Vinci's 'Vitruvian Man' is assumed to be drawn in golden ratio. *Figure1* illustrates the 'Vitruvian Man'. The artwork was provided with notes such as "from above the chest to the top of the head is one-sixth of the height of a man". Our human-body-model followed a similar principle to that of "Vitruvian Man"s.

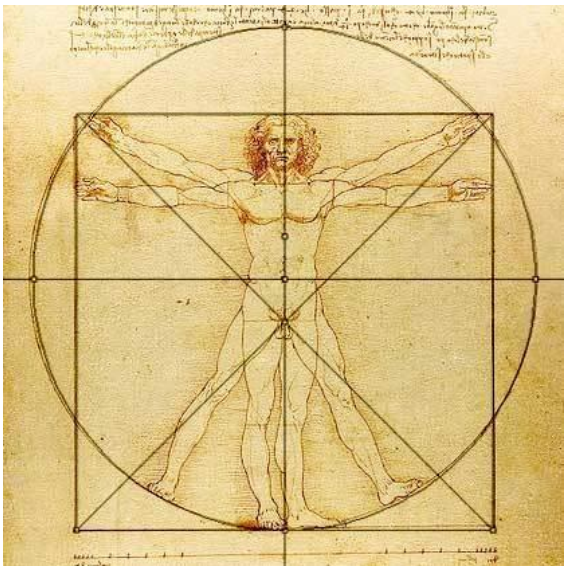

Figure 1: Leonardo da Vinci's 'Vitruvian Man'

In addition to the golden ratio, human bodies show certain similarities in terms of mathematical concepts. They are symmetrical in y-axis. The ratios of the length of one body-part to the length of another is similar in all humans. For example, the ratio of arm span (the distance from the middle fingertip of the left hand to that of the right hand) to the body height is equal to one in all humans. Thus, it is concluded that body parts are proportional to each other in terms of their length.

In our implementation, different nodes were used to represent one specific body part. The nodes selected for use were the main joints such as the wrists, elbows, shoulders, hips, knees and ankles. A center point was chosen as the middle of the height inspired by the "Vitruvian Man" and all the nodes were connected to the center in a hierarchy. On the same-y axis of the center, the chest and the head were placed. As a result, Center, Chest, Head, Right-Hip, Right-Knee, Right-Ankle, Left-Hip, Left-Knee, Left-Ankle, Right-Shoulder, Right-Elbow, Right-Wrist, Left-Shoulder, Left-Elbow and Left-Wrist composed the fifteen nodes.

A ratio of a node's distance to it's previous hierarchical node in both x-axis and y-axis to the total length was calculated for each node. This ratio was then used to calculate each node's positioning. *Figure2* represents the calculation step of the distances of each node to its hierarchiecal node.
Calculation:

- previous joint's x coordinate + (their ratio in x-axis * the body height)
- previous joint's y coordinate + (their ratio in y-axis * the body height)

There are differences in ratios of male and female body. For example, female body have wider hip gap than the male body and the distance between shoulders is narrower than the male's.

Age option simply sets the height to a totally new value. Since the ratios are known, places of the nodes can be determined by applying the new height.

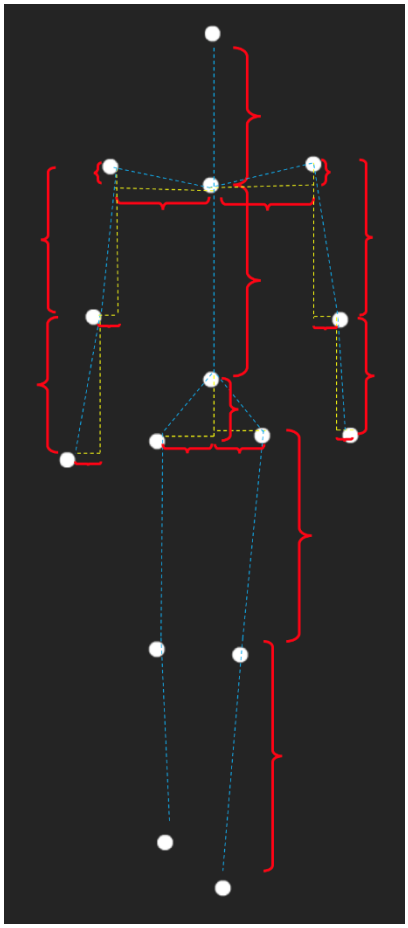

Figure 2: Distance Calculation

Applied Concepts:

Hierarchical Modeling

For replicating the movement process of the body with the necessary rotations in the correct axis a hierarchical model is formed. In the implemented hierarchy, every node is written in terms of the center node directly or indirectly. Therefore, when any rotation is applied to the center node, all the other nodes will be affected from that rotation. *Figure3* shows the hierarchical tree model of the nodes.

In addition, when the arm is taken into account, any rotation applied on the shoulder will also be applied on the elbow and the wrist. This is inspired by the robot-arm example presented in *Figure4.* With the same logic if any rotation is applied on the elbow, this will show its effect on wrist as well. Very same pattern is also valid for the legs.

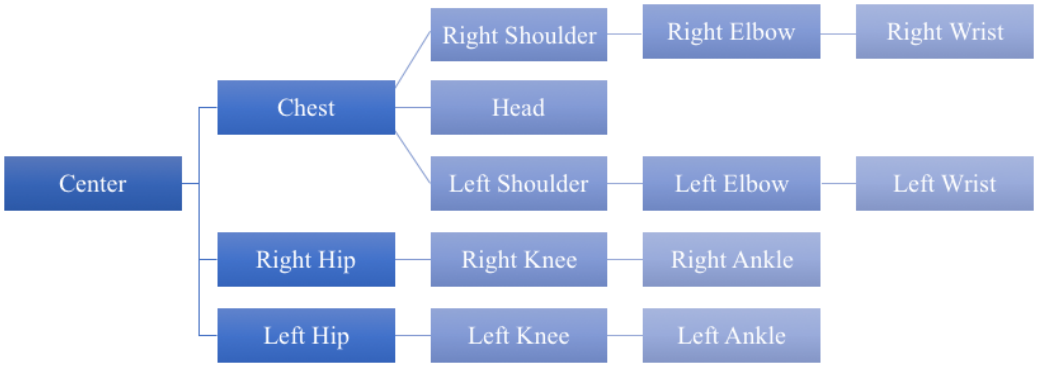

Figure 3: Hierarchical Model

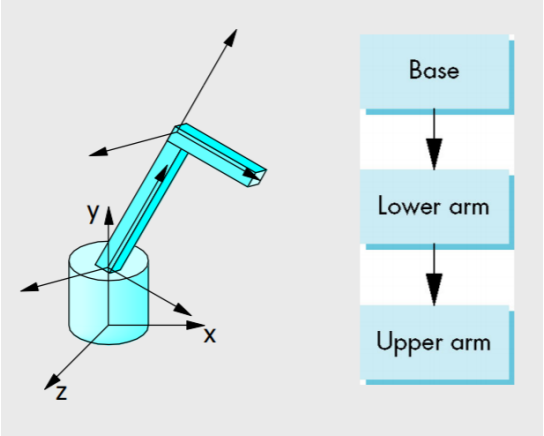

Figure 4: Robot Arm

For achieving this, each of the nodes' model-view matrices should be written in terms of the *its parent model-view matrices * its own rotation * its own translation.*

Example; Model-View[HEAD] = Model-View[CHEST]*Rotate[HEAD]*Translate[HEAD]

Each node has its parent's model-view so any rotation or translation applied to its parent will carry out its effect on its children.

Rotation Logic

Rotations in the movement process need to be done in their fixed axis. At the beginning it should translate itself to its parent node, after that should apply its corresponding rotation, and at the end it should translate itself back to its location. *Figure5* displays the rotation logic.

$$M = T(p_f)\,R(\theta)\,T(-p_f)$$

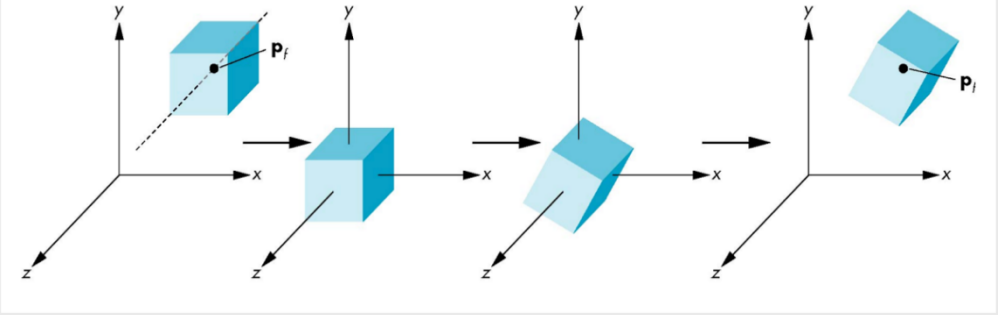

Figure 5: Rotation Logic

Implementation

Ratios

Figure6 displays the ratios between each body part. When the height changes, since the ratios are known, new position of the nodes can be determined easily. From the menu the user can change the age mode to child, adult or elder. In that scenario height parameter is updated and then the model-view matrices are updated accordingly to fit the new looking.

```
80  float maleBodyHeight = 50;
81  float femaleBodyHeight = 50;
82  //male body ratios
83  float male_center_chest = 1.9/7.7;
84  float male_chest_head = 1.4/7.7;
85
86  float male_center_hipX = 0.3/7.7;
87  float male_center_hipY = 0.6/7.7;
88
89  float male_hip_kneeX = 0.1/7.7;
90  float male_hip_kneeY = 2/7.7;
91
92  float male_knee_ankleX = 0.1/7.7;
93  float male_knee_ankleY = 2.2/7.7;
94
95  float male_chest_shoulderX = 1/7.7;
96  float male_chest_shoulderY = 0.3/7.7;
97
98  float male_shoulder_elbowX = 0.1/7.7;
99  float male_shoulder_elbowY = 1.4/7.7;
100
101 float male_elbow_wristX = 0.1/7.7;
102 float male_elbow_wristY = 1.3/7.7;
```

```
104 //female body ratios
105 float female_center_chest = 1.8/7.4;
106 float female_chest_head = 1.4/7.4;
107
108 float female_center_hipX = 0.6/7.4;
109 float female_center_hipY = 0.6/7.4;
110
111 float female_hip_kneeX = 0.3/7.4;
112 float female_hip_kneeY = 2.1/7.4;
113
114 float female_knee_ankleX = 0.1/7.4;
115 float female_knee_ankleY = 2.2/7.4;
116
117 float female_chest_shoulderX = 0.8/7.4;
118 float female_chest_shoulderY = 0.2/7.4;
119
120 float female_shoulder_elbowX = 0.15/7.4;
121 float female_shoulder_elbowY = 1.4/7.4;
122
123 float female_elbow_wristX = 0.2/7.4;
124 float female_elbow_wristY = 1.3/7.4;
```

Figure 6: Code Snippet of Ratios

Figure7 shows the observations in female body with different height parameters.

- Adult Female Body

- Child Female Body

- Elder Female Body

Figure 7: Height Observation

Angles

A 3D angle array is initialized with given angles. First parameter of the 3D angle array indicates whether movement type is walking or running. Second parameter indicates the related body part. The third parameter holds the angles of the indicated body part in x, y, z axis. When any rotation takes place, the corresponding angles are updated. One step movement is completed when the arm and leg with negative (-) angle reaches the value of arm and leg with positive (+) angle, and vice versa.*Figure8* shows the 3D array.

```
GLfloat Angles[2][15][3]={{//WALKING
    {0,-30,0},//CENTER
    {0,0,0},//CHEST
    {0,0,0},//HEAD

    {-35,0,0},//RIGHT HIP
    {30,0,0},//RIGHT KNEE
    {0,0,0},//RIGHT ANKLE

    {25,0,0},//LEFT HIP
    {5,0,0},//LEFT KNEE
    {0,0,0},//LEFT ANKLE

    {40,0,0},//RIGHT SHOULDER
    {-40,0,0},//RIGHT ELBOW
    {0,0,0},//RIGHT WRIST

    {-20,0,0},//LEFT SHOULDER
    {-40,0,0},//LEFT ELBOW
    {0,0,0},//LEFT WRIST
},
    {//RUNNING
        {0,-30,0},//CENTER
        {0,0,0},//CHEST
        {0,0,0},//HEAD

        {-45,0,0},//RIGHT HIP
        {30,0,0},//RIGHT KNEE
        {0,0,0},//RIGHT ANKLE

        {35,0,0},//LEFT HIP
        {90,0,0},//LEFT KNEE
        {0,0,0},//LEFT ANKLE

        {45,0,0},//RIGHT SHOULDER
        {-90,0,0},//RIGHT ELBOW
        {0,0,0},//RIGHT WRIST

        {-35,0,0},//LEFT SHOULDER
        {-90,0,0},//LEFT ELBOW
        {0,0,0},//LEFT WRIST
    }
};
```

Figure 8: Code Snippet of 3D Array

Modelview Matrices

All fifteen modelview matrices were constructed in terms of the node's parent model-view matrix with its own rotation and translation matrices. Inside the modelview matrices, helper functions of *getDisplacement* and *getRotationMatrix* are used. *getDisplacement* function is defined for returning the displacement vector for desired body part and the *getRotationMatrix* function is defined for returning the rotation matrix for the desired body part. *Figure9* shows the implementation of modelview matrices.

```
vec3 getDisplacement(int index){
    return vec3(coordinates[gender][index].x,coordinates[gender][index].y,coordinates[gender][index].z );
}

mat4 getRotationMatrix(int index){
    return RotateX(Angles[movType][index][0])* RotateY(Angles[movType][index][1])*RotateZ(Angles[movType][index][2]);
}

void initializeModelViews(int gender){
    model_view[Center] = Scale(0.02, 0.02, 0.02)*Translate(getDisplacement(Center))*getRotationMatrix(Center);
    model_view[Chest] = model_view[Center]*Translate(getDisplacement(Chest))*getRotationMatrix(Chest);
    model_view[Head] = model_view[Chest]*Translate(getDisplacement(Head))*getRotationMatrix(Head);

    model_view[RightHip] = model_view[Center]*getRotationMatrix(RightHip)*Translate(getDisplacement(RightHip));
    model_view[RightKnee] =model_view[RightHip]*Translate(getDisplacement(RightKnee))*getRotationMatrix(RightKnee);
    model_view[RightAnkle] = model_view[RightKnee]*Translate(getDisplacement(RightAnkle))*getRotationMatrix(RightAnkle);

    model_view[LeftHip] = model_view[Center]*getRotationMatrix(LeftHip)*Translate(getDisplacement(LeftHip));
    model_view[LeftKnee] = model_view[LeftHip]*Translate(getDisplacement(LeftKnee))*getRotationMatrix(LeftKnee);
    model_view[LeftAnkle] = model_view[LeftKnee]*Translate(getDisplacement(LeftAnkle))*getRotationMatrix(LeftAnkle);

    model_view[RightShoulder] = model_view[Chest]*getRotationMatrix(RightShoulder)*Translate(getDisplacement(RightShoulder));
    model_view[RightElbow] = model_view[RightShoulder]*Translate(getDisplacement(RightElbow))*getRotationMatrix(RightElbow);
    model_view[RightWrist] = model_view[RightElbow]*Translate(getDisplacement(RightWrist))*getRotationMatrix(RightWrist);

    model_view[LeftShoulder] = model_view[Chest]*getRotationMatrix(LeftShoulder)*Translate(getDisplacement(LeftShoulder));
    model_view[LeftElbow] = model_view[LeftShoulder]*Translate(getDisplacement(LeftElbow))*getRotationMatrix(LeftElbow);
    model_view[LeftWrist] = model_view[LeftElbow]*Translate(getDisplacement(LeftWrist))*getRotationMatrix(LeftWrist);
}
```

Figure 9: Code Snippet of Modelview Matrices

Movement

Figure10 displays the code snippet for the rotation in right leg.

```
void updateRightLeg(){
    model_view[RightHip] = model_view[Center]*getRotationMatrix(RightHip)*Translate(getDisplacement(RightHip));
    model_view[RightKnee] =model_view[RightHip]*Translate(getDisplacement(RightKnee))*getRotationMatrix(RightKnee);
    model_view[RightAnkle] = model_view[RightKnee]*Translate(getDisplacement(RightAnkle))*getRotationMatrix(RightAnkle);
}

void rotateRightLeg(){
    if(rotateRightLegForward){
        legRotForMovType[0]=(float)25/60;
        legRotForMovType[1]=(float)-60/80;

        model_view[RightHip]=model_view[RightHip]*Translate(getDisplacement(RightHip))*RotateX(-1)*Translate(-getDisplacement(RightHip));
        Angles[movType][RightHip][Xaxis]-=1;
        updateRightLeg();

        model_view[RightKnee]=model_view[RightKnee]*Translate(getDisplacement(RightKnee))*RotateX(legRotForMovType[movType])*Translate(-getDisplacement(RightKnee));
        Angles[movType][RightKnee][Xaxis]+=legRotForMovType[movType];
        updateRightLeg();
    }else if(rotateRightLegBackward){
        legRotForMovType[0]=(float)-25/60;
        legRotForMovType[1]=(float)60/80;

        model_view[RightHip]=model_view[RightHip]*Translate(getDisplacement(RightHip))*RotateX(+1)*Translate(-getDisplacement(RightHip));
        Angles[movType][RightHip][Xaxis]+=1;
        updateRightLeg();

        model_view[RightKnee]=model_view[RightKnee]*Translate(getDisplacement(RightKnee))*RotateX(legRotForMovType[movType])*Translate(-getDisplacement(RightKnee));
        Angles[movType][RightKnee][Xaxis]+=legRotForMovType[movType];
        updateRightLeg();
    }
}
```

Figure 10: Code Snippet of Rotation in Right Leg

Rotation in leg which we call the walking movement starts with the hip. Since it is a fixed axis rotation, first it is translated to its parent. Then, rotation is applied. Finally it is translated back to its original position. After completing the rotation in hip. Global Angle array's corresponding values should are set to their new values.

Since the knee is connected to the hip and the ankle is connected to the knee, right leg needs to be updated due to the rotation needed for the knee and the ankle. Same procedure is applied for knee to complete the leg movement. The same logic is applied for the other leg and arm. *Figure11* displays the walking and running frameshots of the model.

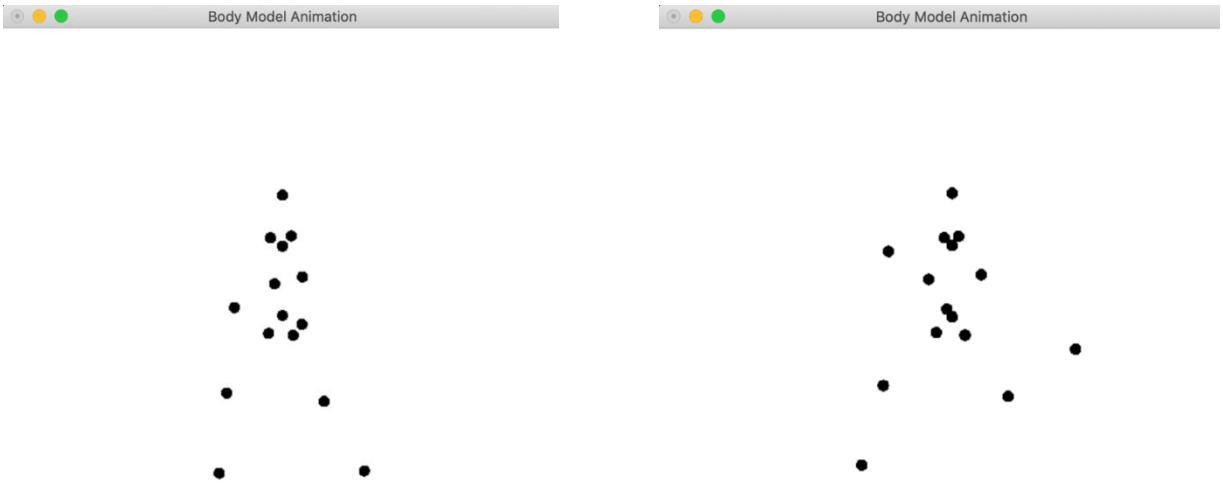

Figure 11: Walking vs Running odel

Conclusion

In conclusion, this project manages to display the human body movements successfully with providing an interface for user to change between the type of the movement, gender, and age modes. However, since the angles of the corresponding movement type is determined by the trial and error, it is prompt to errors.

Resources

1. *"Vitruvian Man" by Leonardo Da Vinci and the Golden Ratio*, www.crl.nitech.ac.jp/~ida/education/VitruvianMan/

2. "The Human Body." *The Myth of the Golden Ratio*, goldenratiomyth.weebly.com/the-human-body.html.

With special thanks to,

3. "COMP410/510 - Computer Graphics." *Comp 410*, home.ku.edu.tr/~yyemez/Comp410/index.htm.