TED UNIVERSITY

CMPE360

Project 8

WebGL

Robot Arm Simulation

Section 02

Erkan Sancak
44293566706

Elif Aysu Kürşad
14162766452

Table of Contents

I. Introduction

The **WebGL Robot Arm Simulation** is a project that extends from Project 7. the goal is to enhance the existing robot arm simulation by incorporating articulated fingers and a thumb. The simulation manipulates WebGL (Web Graphics Library) with JavScripts to create an interactive experience, allowing users to manipulate the robot arm in a three-dimensional space.

II. Implementation

Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12 Server at localhost Port 80

Render Function

At the core of the WebGL Robot Arm Simulation lies the **render function**, a pivotal component responsible for visually representing the articulated robot within the WebGL canvas. This function meticulously handles the rendering process, showcasing the robot arm, fingers, and thumb with interactivity.

The initial segment of the render function manages the rendering of the shoulder, elbow, fingers and a thumb. These joints play a crucial role in defining the overall posture of the robot arm. The shoulder joint responds dynamically to user input, allowing for a wide range of motion. Simultaneously, the elbow joint adjusts its position based on user commands, contributing to the arm's flexibility.

a.Drawing the Thumb

```
// Thumb positions
var thumbPositions = [
    vec3(0.3, -0.2, 0.0), // thumb 1
    vec3(0.3, -0.2, 0.0)  // thumb 2
];

// Position thumbs
for (var i = 0; i < 2; i++) {
    var posT = thumbPositions[i];
    matStack.push(mv);
    mv = mult(mv, translate(posT[0], posT[1], posT[2]));
    mv = mult(mv, rotate(thumb, vec3(0, 0, 1)));
    mv = mult(mv, translate(1.0, 0.0, 0.0));
    mv = mult(mv, scale(0.4, 0.14, 0.2));
    gl.uniformMatrix4fv(mvLoc, gl.FALSE, flatten(transpose(mv)));
    gl.drawArrays(armShape.type, armShape.start, armShape.size);
    mv = matStack.pop();
}
```

This code handles the drawing of the thumb in the WebGL canvas. It utilizes a loop to position two components of the thumb based on the predefined thumbPositions array. The transformations involve translation, rotation, and scaling to achieve the desired thumb appearance. The mv matrix stack is employed to manage transformations and maintain the state.

b. Thumb Movement

```
//thumb
if (shift && isPressed("F"))
{
   if (thumb>0) thumb = (thumb - y);
   else thumb = 0;
}
if(!shift && isPressed("F"))
{
   if(thumb<10) thumb = (thumb + y);
   else thumb = 10;
}
```

By positioning the thumb joints and applying rotations in response to user input, the simulation achieves a lifelike representation. The thumb's autonomy enhances the overall functionality of the robotic hand.

This code handles the keyboard events related to thumb movement. When the "F" key is pressed with or without the shift key, it adjusts the thumb variable accordingly, controlling the rotation of the thumb in either the positive or negative direction. The amount of rotation is calculated based on the elapsed time (timePassed) and a predefined rotation speed (s).

c. Drawing the Fingers

```
// Finger positions
var fingerPositions = [
    vec3(0.2, 0.1, -0.5),
    vec3(0.2, 0.1, 0.),
    vec3(0.2, 0.1, 0.5),

];

// Position fingers
for (var j = 0; j < 3; j++) {
    var posF = fingerPositions[j];
    matStack.push(mv);
    mv = mult(mv, translate(posF[0], posF[1], posF[2]));
    mv = mult(mv, rotate(finger, vec3(0, 0, 1)));
    mv = mult(mv, translate(1.0, 0.0, 0.0));
    mv = mult(mv, scale(0.4, 0.14, 0.2));
    gl.uniformMatrix4fv(mvLoc, gl.FALSE, flatten(transpose(mv)));
    gl.drawArrays(armShape.type, armShape.start, armShape.size);
    mv = matStack.pop();
}
```

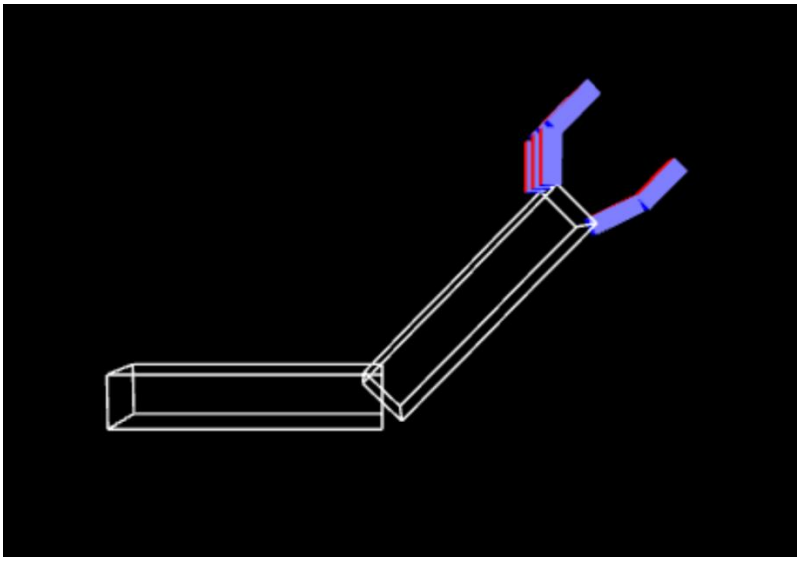
This code handles the drawing of the fingers in the WebGL canvas. It utilizes a loop to position three components of the fingers based on the predefined fingerPositions array. Similar to the thumb drawing, it involves translation, rotation, and scaling to achieve the desired finger appearance.

d. Finger Movement

```
//finger
if (shift && isPressed("F"))
{
    if (finger<0) finger = (finger + y);
    else finger = 0;
}
if(!shift && isPressed("F"))
{
    if(finger>-10) finger = (finger - y);
    else finger = -10;
}
```

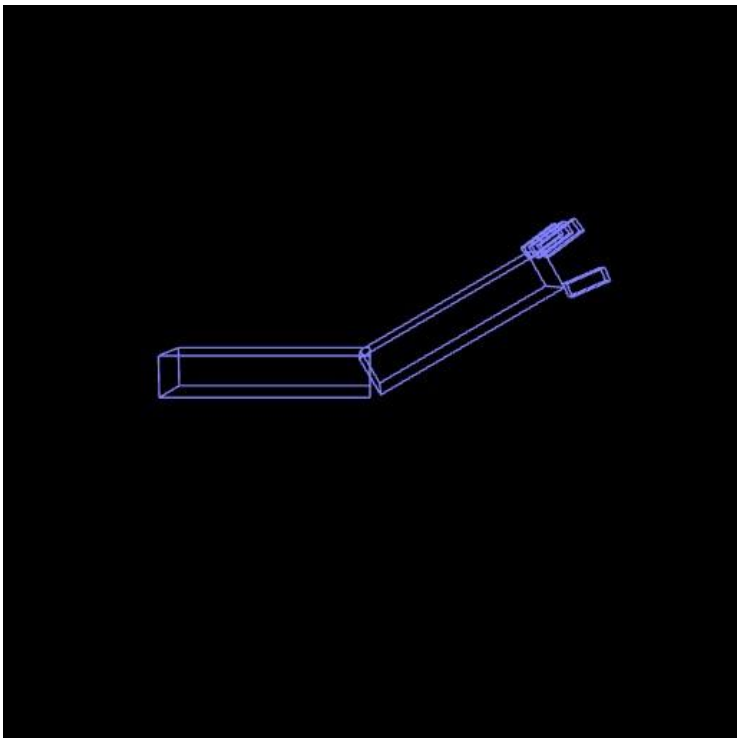
Following a parallel approach, moving the fingers contribute to the overall dexterity of the robotic hand. These code handles the keyboard events related to finger movement. When the "F" key is pressed with or without the shift key, it adjusts the finger variable accordingly, controlling the rotation of the fingers in either the positive or negative direction. The amount of rotation is calculated based on the elapsed time (timePassed) and a predefined rotation speed (s).

III. Results and Analysis



In the WebGL Robot Arm Simulation, the implementation of fingers involves positioning them both before and after the finger joint. Additionally, a similar approach is applied in the manner for the thumb.

However, during the implementation of fingers and a thumb in the, a challenge was encountered regarding the color consistency between the fingers and arm. The attempt to implement fingers before the finger joint and fingers after the finger joint, and the thumb, led to a noticeable mismatch in colors.



The final implementation has modification which the fingers and thumb components after the joint were removed. This adjustment in the design and structure of the arm has noteworthy impacts for both the visual representation and the underlying mechanics of the simulation.

The removal of components after the joint implies a more straightforward articulation mechanism. Thus, when the 'F' key is pressed, the fingers extend in an upward direction resembling a claw, while the thumb simultaneously moves downward, and vice versa.

Video representation of the robot arm can be found through this link (Google Drive).

IV. Conclusion

In conclusion, project aims to enhance the existing robot arm simulation by incorporating articulated fingers and a thumb. Utilizing WebGL (Web Graphics Library) with JavaScript, the simulation provides an interactive three-dimensional experience, allowing users to manipulate the robot arm. The implementation involves rendering the arm components, including fingers and thumb, and enabling movement through keyboard inputs.

A fundamental aspect of the project involves the adept use of matrices, particularly the mv matrix, to control the positions and rotations of objects within the WebGL environment. The project also utilizes functions such as translate, rotate, and scale to update the mv matrix, determining the components position and shape. These transformations play a vital role in enabling the movement and rotation of the components. Despite being partially completed, the project demonstrates progress in achieving a dynamic and responsive robot arm simulation.