Bryant Pong

ECSE-4750

11/29/14

**Computer Graphics Final Project – Robotic Arm Simulator**

**Project Overview**

Robotics is a relatively new field that has gained traction and interest quickly in recent years. However, there are two problems that budding roboticists face when learning about this exciting field. Robotics has a high initial barrier to entry; even the cheapest industrial robot is at least $10,000! Secondly, robotics requires a vast amount of theoretical knowledge before one can even implement basic functionality. These two barriers can easily be overcome with the creation of a robot simulation.

Unfortunately, many robot simulators today are currently either extremely expensive (to the tune of several hundred dollars) or are platform-specific. For example, a popular robot simulator is the Gazebo project (<http://www.gazebosim.com>). However, this software is currently only supported on Linux platforms and has a large set of software requirements, forcing interested roboticists into learning how to use Linux. Additionally, current simulators are difficult to use and have unintuitive controls. I decided to create a WebGL robotic arm simulator to allow users to simulate robotic arms on their browsers to avoid the problems of high-cost and steep learning curves.

**Design Requirements**

I first looked at the dimensions of industrial robotic arms to determine the scaling of the simulator’s canvas. From my research, I determined that the majority of robotic arms at zero-configuration (this is the position that the manufactures define as all joints in the arm are at 0 degrees; usually zero-configuration is when the arm is pointing straight out) are within a 10-foot workspace. This means that my canvas scaling was from -10 to 10 in all directions. As a result, my projection matrix was set using this constraint.

Next, I had to determine what the best possible viewing angle was. Originally, I wanted a wide and sweeping view of the robot from Cartesian location (1, 1, 1). However, after initial setting my model-view matrix to look at (1, 1, 1), I realized that the camera was zoomed out too far; the robot arm was rendering too small! To compensate, I set the camera at Cartesian (0.1, 0.1, 0.1); this gives the same effect as setting the camera at (1, 1, 1) except the field of view is narrower and thus the robot is more clearly seen. Thus, my model-view matrix is initially set to (0.1, 0.1, 0.1) while treating the positive Z-Axis as the “up” direction.

With the viewing out of the way, I now needed to determine the functions needed to calculate the robot’s forward kinematics. The forward kinematics is the procedure in which the orientation and location of each joint in the robot is determined. The forward kinematics is comprised of two parts: a rotational and a translational calculation. The rotational portion determines the angle that a joint is oriented; it is simply the sum of all the prior joint angles together. The translational forward kinematics determines the current Cartesian location of a joint. For a reference frame and a frame attached to

**Challenges Faced and Overcome**

My first challenge was encountered when writing the functions to calculate the robot arm’s forward kinematics. After reading through “MV.js”,