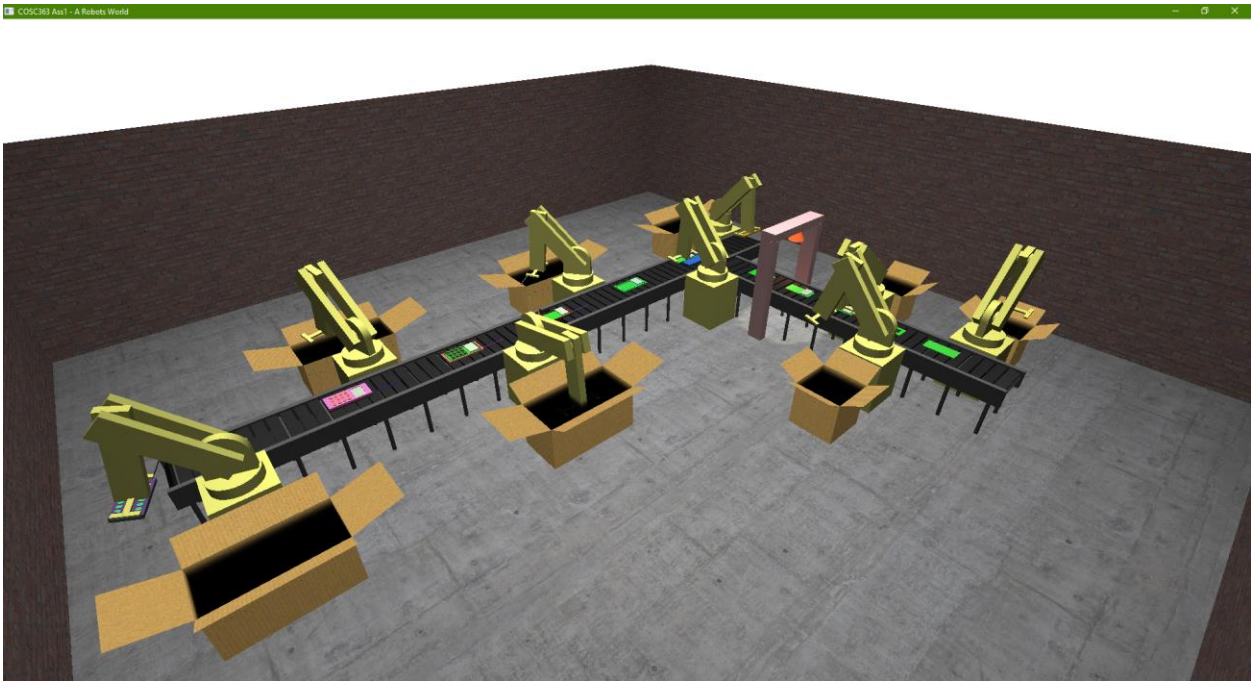
COSC363 Assignment 1 – A Robots World

Jack Wilkie – ID: 31481060 (jwi123)

Introduction

This scene was inspired by an interpretation of the phrase “Mobile robot”, which may have meant to refer to a robot that moves, but which was instead interpreted as a robot that makes mobiles (cell phones). This scene consists of a production line for mobile phones, where robotic arms grab parts out of boxes, and assemble them into completed mobile phones. The parts have randomized colors to demonstrate that the phones are distinct objects that move along the line.



Key bindings

- WASD: Movement
- QE: Up/down motion
- Arrows: Rotation
- +/-: Increase/decrease animation speed multiplier (High speeds may cause glitches)
- *: Reset to default speed
- 1: Set to freelook camera
- 2: Set to controlled/animated camera

Note: Window is resizable

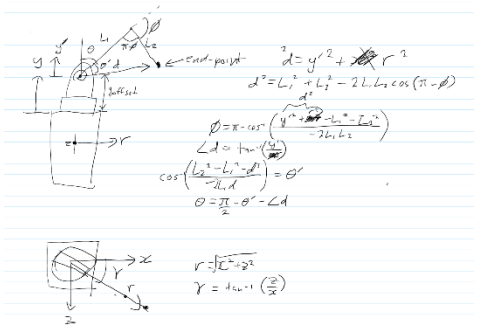
Notable/Extra Features

Robotic arm

I developed the robotic arm myself, and formulated some equations that can convert a point relative to the base of the arm into the required angles of the arm required for it to reach that point. One challenge

to overcome was using arctan to find the rotation on the 'core' axis, as arctan(x/z) generally only maps to ±90°, as -x/z is indistinguishable from x/-z, this was solved by adding PI to the result in the cases where arctan normally got it wrong.

The original working is shown to the right, and the final equations are (separated out for readability):



$$r = \sqrt{x^2 + z^2}\,, \quad y' = y - y_{offset}$$

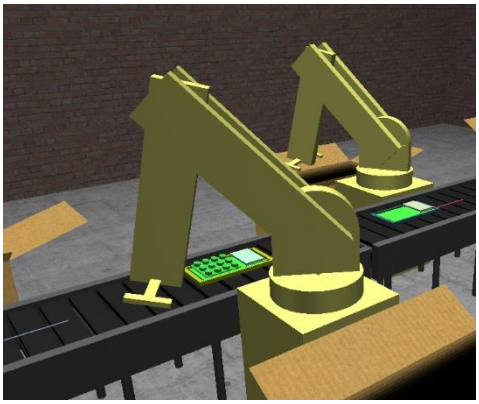
$$d = \sqrt{y'^2 + r^2}, \quad \angle d = \tan^{-1}\frac{y'}{r}$$

$$\theta' = \cos^{-1}\left(\frac{L_2^2 - L_1^2 - d^2}{-2L_1 d}\right), \quad \theta = \frac{\pi}{2} - \theta' - \angle d$$

$$\varphi = \pi - \cos^{-1}\left(\frac{d^2 - L_1^2 - L_2^2}{-2L_1 L_2}\right), \quad \gamma = \tan^{-1}\frac{z}{x}$$

Where gamma, theta, and phi are the angles of the core axis, arm 1, and arm 2, respectively (as per the diagram).

The robot arm was constructed entirely from built-in primitives.



Variable Frame Timing (delta-time), Key Hold Handling, Smooth Camera

I decided to implement my own system to wrap and improve on the glut callback system that we were taught for animations. At the end of the display function, a timer is set for 0ms to call the animation/ processing function as soon as possible. The processing function uses chrono::high_resolution_clock to find the exact time elapsed (dt) since the last frame and then passes this to all animation functionality. This means that no matter what framerate the program runs at, the animations should always take the same time, and run at the same speed.

In addition, I created my own wrapper functionality around "glut[Keyboard/Special]Func" and glut[Keyboard/Special]UpFunc, so that the keys currently being held are stored in an array, and can be queried anywhere in the application. This was used to give smooth camera control, by detecting if certain camera motion buttons were down at each frame, and then moving the camera in the appropriate way, by an amount equal to a preset value per second, multiplied by the dt value from the variable frame timing system. This gave smooth camera movement at a speed independent of framerate.

Mobile model

The mobile phone model used in the application was developed in blender as several parts. The meshes were loaded separately, and can be separately rendered. Any mobile phone part in the scene is an instance of the Mobile class, with a 'state' variable (which is a enum that also doubles as a bitfield of which parts to render) describing what kind of part it is, or in what stage of assembly the phone is in. The meshes are only loaded once for all the mobile models in the scene by keeping track of the number of mobiles created, and only loading them on the first constructor, and deleting them on the last destructor. The mobile object has an array that stores the color values of all of its parts.

The mobile object contains a combine method, that combines it and another mobile object, generally preserving the colors of the two original mobile objects, this is used in my animation to combine the parts from the robots into the mobiles on the assembly line, preserving the existing colors of both objects.

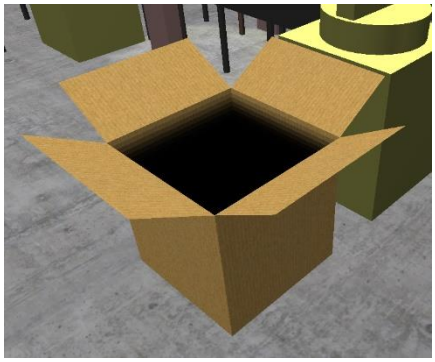
Moving spotlight/Soldering oven

This part was supposed to look like some sort of 'soldering oven' to solder the electronic parts(Chip/screen) onto the green circuit board, before it was placed in the phones casing. The frame is a simple static object made of 3 scaled glutSolidCube()'s, and the 'heat lamp' it shown as a cone. The motion of the 'heat lamp' is determined by a sin function of the current position in the main animation cycle. Both the cone and the light have positions controlled this way.

Animated camera

The camera animation will be enabled when the '2' button is pressed, and will move the camera to the start of the animation, but it will only start when the robot arm releases the part to be tracked on the conveyor, this is to ensure the camera animation is synchronized with the scene. The animation starts a timer (which is incremented by dt each frame), and if, if else, else statements decide what action to take on the camera based on the progress through the animation. One notable thing I did in this animation was interpolate between two points (very last part of the animation), however I got the linear proportion (in the range 0-1), and passed it through a cosine function, to make the beginning and end of the camera motion much smoother.

Part box



The part box was used to replace the glutSolidCube()'s that were previously being used to pull parts out of. The idea was to have the robot arms descend into the dark box smoothly, rather than phase through what appeared to be a solid surface. I had to use glEnable(GL_BLEND); and glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA); to enable blending of transparent surfaces. I made several layers of black planes, with increasing opacity going down, so that it was clearly not just a black surface, but was sufficiently dark at the bottom to prevent the appearance of objects out of thin air being visible. Because of Z-Culling, I had to make sure to render the boxes after the robots, so that the transparent black surfaces would correctly render over the top, and to render the planes from the bottom up, so that the closer upper layers wouldn't block the lower layers from drawing.

Resources

Cardboard texture: www.textures.com/download/cardboardplain0008/28990
Brick texture: www.textures.com/download/bricksmallbrown0270/66235
Concrete texture: www.textures.com/download/concretebare0433/108718
loadTGA, and LoadBMP from the labs were included, but only loadTGA was utilized.
I created my own .obj file loader from scratch.