CS324 Graphics Coursework - Robot Arm

Justin Riddell u1419657

January 18, 2017

Features of solution and design

The first task undertaken was broad phase collision detection using Axis Aligned Bounding Boxs ([1]). Initially it was attempted to then orient these bounding boxes as the shapes were rotated - however this was increasingly difficult using Euler angles. Quaternions have therefore been used to store orientation of objects to prevent Gimbal lock and they have been easier to use in the long run. Subsequent to this Separating Axis Theorem ([2]) was used for precise collision detection, use of which in this implementation could be extended for convex polygons other that cuboids. In order to increase the scalability of the solution, Octtrees ([3]) have been employed to recursively sub-divide the space such that the collisions only need be checked for objects that are close to each other.

When the arm is moved, all the parts above it should also move in the same way. This was achieved by translating them an equal amount if a translation occurred such as movement of the base and by using the quaternion rotation of each part of the arm about the initial parent point that moved if an arm part is rotated. The arm is defined simply as a vector of vectors, with vectors in the vector i+1 depending on the vector in i. This tree structure allows for the correct movement and rotation of the arm when a part lower down the tree is moved. The claw grabbing action is achieved by generating a quaternion and then using spherical linear interpolation ([4]) to change between the original position of the claw parts and the center of the "platter" (flattened cube on which they rest). Small parts connect the major parts of the arm together so that the edges of the cuboids do not always collide with adjacent parts when rotated.

Grabbing and picking up objects can be achieved when all parts (in this case 2) of the claw make contact with the object simultaneously. The picked up objects will then be added to the vector of ARM_PARTS and thus act as if it were a part of the arm. The right mouse button releases the held object by resetting the claw orientations to that of the platter below them.

The only OpenGL specific/GLSL features that have been used are a very simple vertex and fragment shader, that takes the colour and positional data from the data passed in and calculates the position from the model view projection matrices also passed in from the main "render" loop. Basic OpenGL features like VAOs/VBOs are used too.

Disclaimer - unfortunately, I originally read the specification as saying that the game Towers of Hanoi should be playable and solvable by a robot-arm rather than automated. I do realise that, nevertheless, this submission does not fully meet the specification nor does it employ extensive use of OpenGL.

How to run

Running the program should be done by executing the temp.sh script that will compile and run the application called "arm". All development has been done on DCS machines.

Keys for movement are r,y,z for yaw pitch and roll rotation, capitalised versions for rotation the other way. h,j,k,l for movement of the base of the arm. Left click to pincer the claw together to grab things. Right click to reset the claw. Left and right arrow keys to change selected shape so that other parts may be rotated or moved. w and s are camera motion in the direction you are facing and against it respectively. Please note you cannot move when colliding: if a part of the arm has collided, it must be remedied before further movement/rotation of any parts of the arm - usually just by reversing the movement that was just

made. Picking up and dropping of blocks is possible.

References

[1] 3d collision detection. https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection. Accessed: 10.12.17.

[2] Collision detection using the separating axis theorem. https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169. Accessed: 11.12.17.

[3] Octree. https://en.wikipedia.org/wiki/Octree. Accessed: 11.12.17.

[4] Quaternion interpolation. https://theory.org/software/qfa/writeup/node12.html. Accessed: 12.12.17.