Team Name: Roblox 2.0

Team Members:

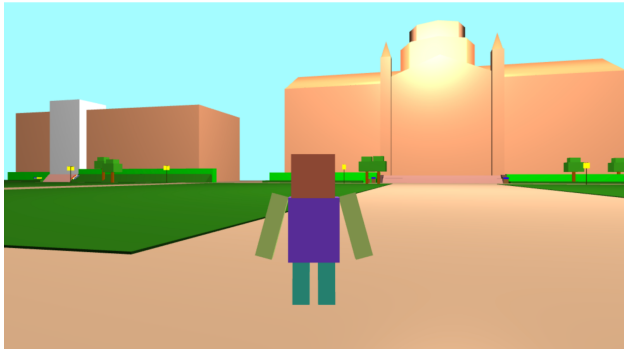
Yuxi Chang - 205309516
Da Yuen Kim - 305096821
Aurora Yeh - 305110110
John Houser - 405339211

Project Theme

Our group project expanded the features of a previous CS 174A project that one of our team members developed. The original goal of the project was to simulate the on campus environment with various iconic buildings during quarantine and online class. The project included a special multiplayer feature where multiple people could walk around the virtual environment and see each other.

In fact, if you visit cs174a.com you can see a live demo where users can see others also on the website. While this feature was not a part of our advanced features since it wasn't relevant to the class, we thought that this was a neat way to make our project stand apart. We accomplished this using a node express server along with socket.io to communicate between clients and the server.

The previous CS174A project recreated the area between Royce Hall and Powell Library:



The goal for *this* class was to improve the simulation using animation techniques we learned from CS 174C. These objectives would add more realism and complexity to the scene.

How to Run

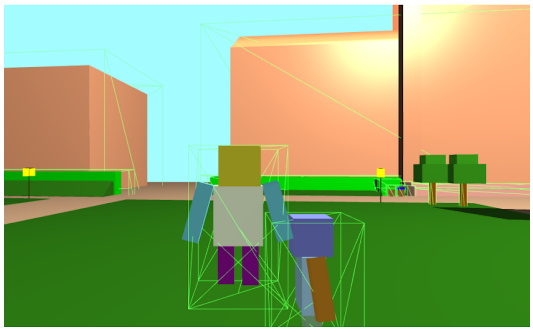
1. Unzip our submission folder
2. Type 'npm install' in CLI (get npm if you don't already have it)
3. Type ‘node index.js'
4. Visit localhost:5000 in your browser

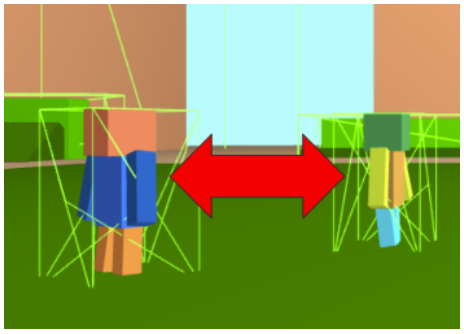
You can also access our project on www.cs174a.com

Computer Animation Algorithms

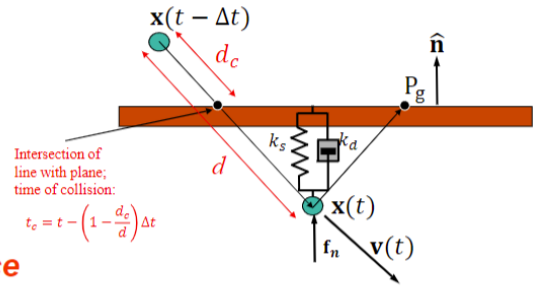
John: Collision Detection Using a Mass-Spring-Damper System

While we were able to reuse a lot of the code from the original project, we also inherited all of the bugs that the old version had. In particular, we had one frustrating issue with the old project. If two players moved towards each other it was possible that their collision boxes could collide and they would become trapped inside each other. The solution we explored to fix this problem involved using a mass-spring-damper system brought up in the lectures. Basically, whenever the game would detect a collision between two collision boxes, we would apply a spring force to push the player out of the collision:





Below is the slide that I referenced when I implemented the mass spring damper system [2]:



Normal force

$$\mathbf{f}_n = k_s\left(\left(\mathrm{P_g} - \mathbf{x}(t)\right)\cdot\hat{\mathbf{n}}\right)\hat{\mathbf{n}} - k_d(\mathbf{v}(t)\cdot\hat{\mathbf{n}})\,\hat{\mathbf{n}}$$

$$\mathbf{f}_n \cdot \hat{\mathbf{n}} > 0$$

One important component of our project that adds more difficulty is that the game is multiplayer. This introduces a lot of new challenges that were not present before. For example, deciding whether the server or clients should be in charge of updating the physics of the world was a new

complexity that we had not encountered in assignment 1 or 2. If one of the clients were to update the position of the other clients it was possible that they might have race conditions. Instead, we decided that each individual client should be responsible for their own physics. This way, there would be no race conditions when it came to updating the physics of players. So, when a player's collision box intersects with another collision box, only that player will be able to update its own position.

Yuxi: New Humanoid Animations Using Forward Kinematics (and Inverse Kinematics)

The local human model is completely reworked; it is implemented as an articulated model with each part stored in a tree structure using arc and node nodes similar to assignment 2. The walking animation is done using forward kinematics, with shoulder and hip joint angles following a sinusoidal function of animation time. We also implement the original jumping animation by giving the root node x-axis rotation degree of freedom.

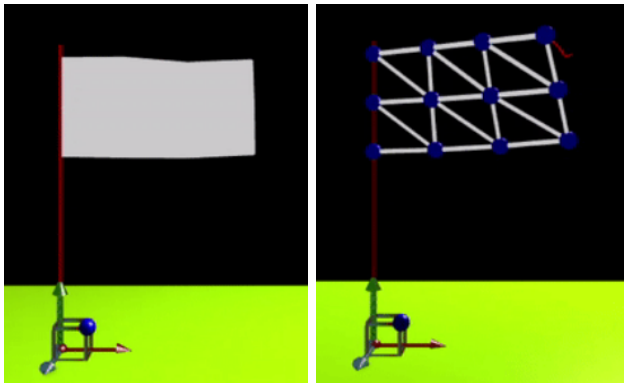
A feature implemented after the final demo is that the new human model cheering animation is animated using inverse kinematics (IK), with the end effector at the right hand position. Pseudo-inverse technique is used when implementing the update part of IK. The cheering animation follows a simple linear function dynamically defined whenever the cheering button is pressed.

When the player moves, we also added a particle effect on the ground to represent dust. We reused and modified part of Yuxi Chang's 174a project code[1] to implement the particle system.

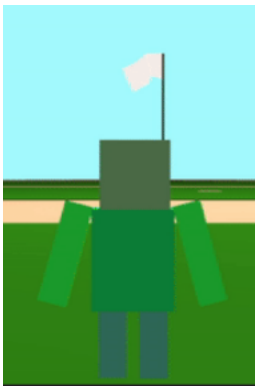
Each particle is spawned randomly at a random position around the player's feet with predefined color, shape, spawn rate etc and randomized initial velocity and living time etc. Each particle is stored in an array, and for each frame, their states get updated iteratively and rendered one by one.

Aurora: Flag Simulation With a Mesh of Particles (class Flagpole)

Another component that used a mass-spring-damper system was the waving flag simulation. This used the system we implemented in part 3 of assignment 1 (the chain), where the springs had high values for the spring constants to resist stretching. The flag was defined as a mesh of particles with triangles filling in the spaces to give an impression of a solid piece of cloth, with the triangles positions calculated by change of basis matrices. The motion of the flag follows a hermite spline, with arbitrarily set points and tangents, that lets it wave back and forth. The update step uses symplectic euler integration, which finds the updated position using the updated velocity, to calculate the motion.



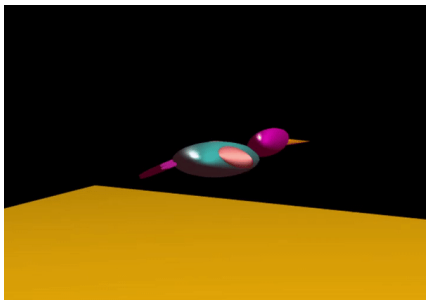
Small model flag showing both solid and mesh



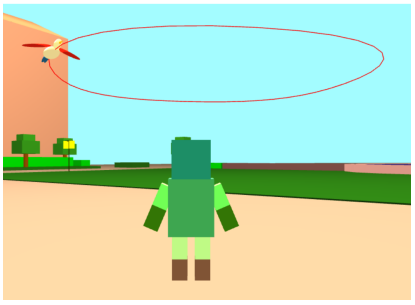
Flag in world

Da Yuen: Movement Along Hermite Spline and Frenet Frames

We added a new element to the overall environment of the campus by adding birds. These birds' movements are determined by a set hermite spline that is predetermined in the code. This is done using part 1 of assignment 1 (hermite spline), to draw the spline and keep a record of its curve function. Using the curve function, we were able to update the model's translation to follow the points along the curve. Following the final demo, we additionally made the bird move in a more natural way, facing the direction of the spline movement. This is done by updating and using the bird's Frenet Frame unit vector tangent **T**, normal unit vector **N**, and binormal unit vector **B** along the curved path.



Model of Bird



Movement along Spline

Reference:

[1] particle.js from: https://github.com/CcccYxx/cs174A-project.git
[2] slides06-collisions from:
https://bruinlearn.ucla.edu/courses/129717/files/9244846?module_item_id=5020160