Assignment 2 Report

Worked as a team
Simon Roy 40030996
Jeffrey Maher 40018878
Yefei Xue 26433979

our repo
https://github.com/felixyf0124/COMP477_F19_A2

Dependencies or libs
GLFW
GLEW
GLM
OpenNI2
NiTE 2
FreeImage
https://github.com/frtru/GemParticles

Part 1

For Part 1, we created a very simple cube made out of 27 total points (9 points per face). These points are interpolated from their original position to a position along the surface of a sphere. The position is calculated to be *r* units away from the center of the cube, *r* being the sphere's radius. The animation alternates between cube and sphere, with 2 second intervals between each, and a 2 second pause between interpolations.

The end result is a sphere that is very polygonal. In theory, we could create a smoother sphere with more points, though generating the triangles would be a tedious process without implementing some algorithm.

Here are the controls to move around and get a different angle of the cube's animation:
X-axis: +: D
        -: A
Z-axis: +: W
        -: S
Y-axis: +: Q
        -: Z

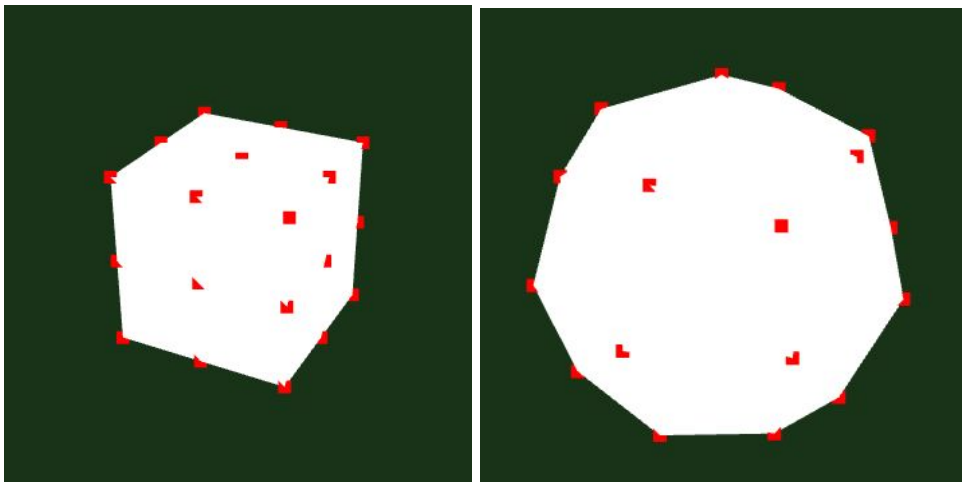


Fig.1: The cube and sphere shapes.

Part 2

For Part 2, we generated a simple linked structure with 3 joints. The structure will try to move from its starting position to a given position along a given time frame. You can click to change the position of the target point.

To make sure the structure always moves, we have to ensure its joints are always bent to some degree. To solve this issue, we have limited the range of points to a circle of a radius that is just below its maximum range (that is, all joints fully extended). If we allowed the machine to fully stretch out, we would encounter singularities in our Jacobian matrix, rendering calculations impossible.

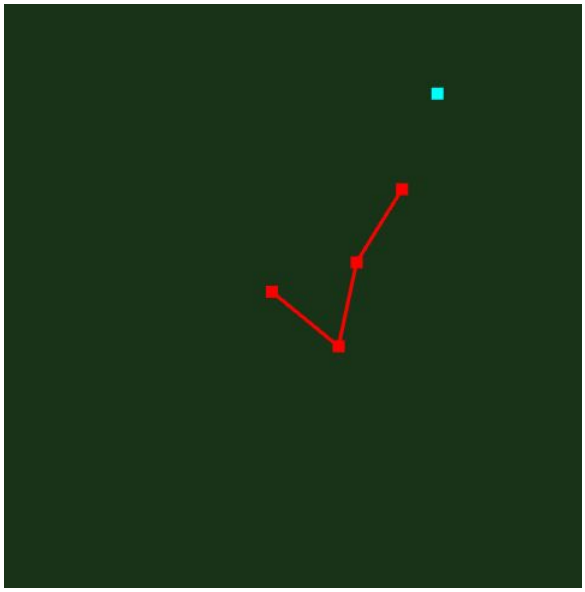


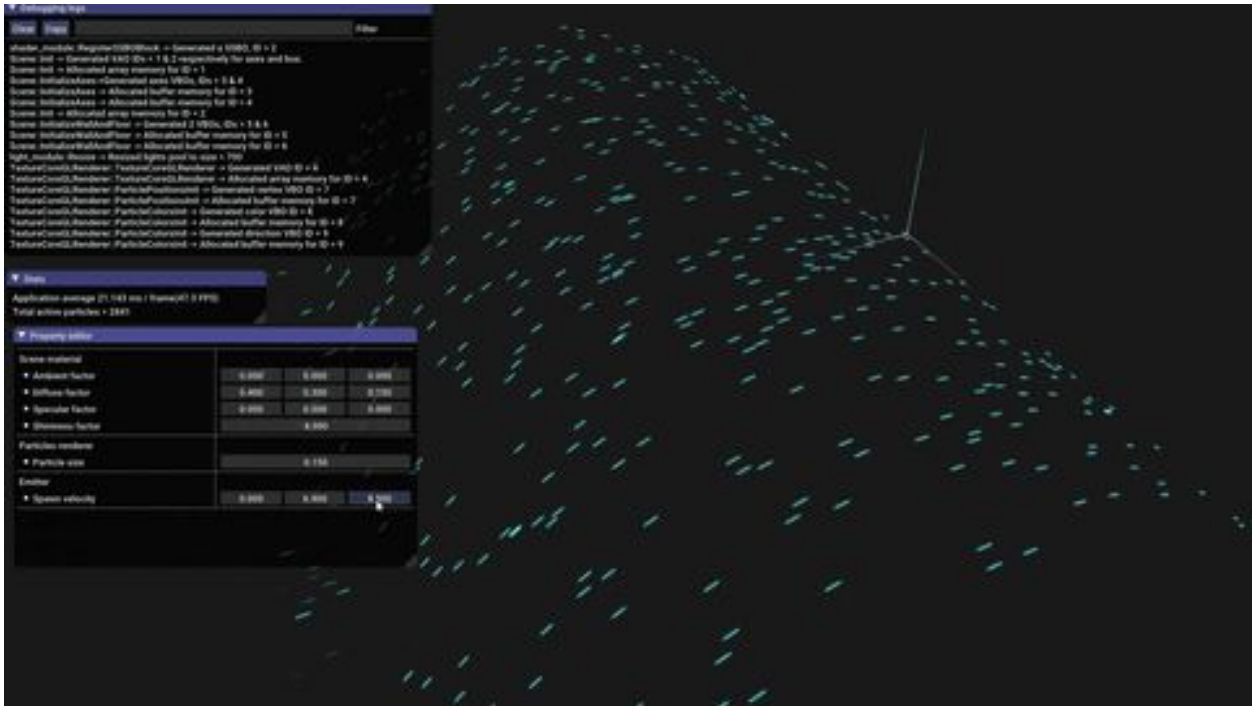
Fig.2: The structure (red) and its joints, moving towards the target (cyan).

Part 3

For the particle system, we decided to modify a pre-existing environment called Gem-Particle on GitHub:
https://github.com/frtru/GemParticles

Our objective was to kickstart the initial steps our project, which focuses on water and explosion simulation. Gem-Particles had a lot of work put into it compared to other systems found on GitHub, and their UI allowed to easily change variables at runtime.

We started from the first sample of the repos called ***lit_particules*** which bounces small particle that emit light and bounce off surfaces. We recreated the project and called our version ***wave***, and we first started by creating a new emitter that would spawn particles in from a line. The system had a spherical emitter and a simple point emitter, but it was very easy to add my own. Once completed, I proceeded to simplify the velocity of the particles, by removing the random motion in the y and z axis. We now had a simple, wide flow from our emitter that would hopefully be a base for our water simulation.



The next step was to implement a disruption to the flow of particles, but we ran into issues with the API made it difficult to add interactive functionality to our project. We wanted to place **obstacles in the water that would divert particles from their paths**.We are still having issues at this time, but we hope to be able to use the system for our full project.

We worked on a fork of the engine, and you can test our simple project using either our main project of the aforementioned fork. You can move the camera with **WASD** and drag the right mouse button to move the camera.

Fork path:
COMP477_F19_A2\GemParticles-Fork\src\projects\wave

You can also just run it from the main project by pressing 3 when you run the console.

Part4

We are using Kinect 2 device with libfreenect2, OpenNI2 and NiTE2 lib. The kinect 2 requires USB 3.0 slot for the device.
We tried to make it available to record through different PCs but so far there is only one desktop working properly with the kinect 2.

To record (only works on Yefei's desktop so far)

- modified UserViewer from NiTE2, so it can dump tracked data (coordinate) into a raw data file **raw_data_1**

- to record, connect Kinect 2, and run UserViewer.exe, stand in front of the Kinect 2. Make sure your whole body is within the camera sight. (To build UserViewer.exe, you have to build a x64 version.)
- when you are done with recording, just disappear from the camera scene for about 20 - 30 second, and wait for the raw data file to be dumped.

To replay the raw data recorded

- run the problem 4
- turn around camera with the **Left & Right** key till you see the replay

To replay .anima

- copy the generated **A2_sample.anima** file to the sample assignment from the course and load it and play it.
- notice that the original sample .anima is at the format

frame_index_0 quaternion1_w quaternion1_x quaternion1_y quaternion1_z quaternion2_w ...
frame_index_1 quaternion1_w quaternion1_x quaternion1_y quaternion1_z quaternion2_w ...
…

- and we couldn’t fully figure out the full detailed algorithm of the play animation since there is no specific instruction or description document.
- but the general logic pipline is
    - record tracked joints coordinate data
    - dump out
    - converted to quaternion through frames
    - save the converted data as .anima format