Fall 2018 ME459 Final Project Report
University of Wisconsin-Madison

Default Project: Collision of Triangular Mesh with a collection of Spheres

Guannan Guo

December 18, 2018

Abstract

The project is for solving a sphere-triangle collision detection problem, which is important for computer games and physics-based problem. Computational geometry methods such as determining the distance between a point and a triangle in space and the distance between a point and a segment line in space are implemented in this project.

Contents

1. Problem statement

The project is to carry out a collision detection step and this is very important in computer games and physics-based simulation.
Imagine you have one sphere and you have a mesh, which is simply a collection of triangles. This collection of triangles might come from a meshing of a bunny, for instance. In addition to the mesh, you are given a collection of spheres, all of the same radius. You have to take each sphere and figure out if it touches any triangle of the mesh. You have to check all spheres.

2. Solution description

To detect the collision between a sphere and a triangle, the most important part is to calculate the distance between the point which is the center of the sphere and the triangle. From computational geometry point of view, this is divided into two cases. **Case 1**: The projection point P of center point O to the plane which the triangle lies in is within the triangle. **Case 2**: point P is not within the triangle. In case 1, the distance can be calculated by measuring the distance between point P and point O. In case 2, the distance can be calculated by measuring the minimum distance between O and the three edges of the triangle. **So, the first problem needs to be solved is how to determine whether point P is within the triangle.**

Suppose point $P(x_0,\ y_0,\ z_0)$ and the three vertices of the triangle $A(x_1,\ y_1,\ z_1)$, $B(x_2,\ y_2,\ z_2)$, $C(x_3,\ y_3,\ z_3)$. At this moment, the normal vector of the plane ABC (l, m, n) can be calculated as below.

$$l = \begin{vmatrix} 1 & y_1, & z_1 \\ 1 & y_2, & z_2 \\ 1 & y_3, & z_3 \end{vmatrix}$$
$$m = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$
$$n = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

And thus, the plane equation for ABC can be described as

$$lx + my + nz = lx_1 + my_1 + nz_1$$

The line which is through P and vertical the ABC plane can be formulized as

$$\begin{cases} x = x_0 + tl \\ y = y_0 + tm \\ z = z_0 + tn \end{cases}$$

Substitute this into the ABC place equation, the coefficient t can be solved as

$$t = \frac{l(x_1 - x_0) + m(y_1 - y_0) + n(z_1 - z_0)}{l^2 + m^2 + n^2}$$

And then substitute t into the vertical line equation, the coordinate of point P(X, Y, Z) can be expressed as

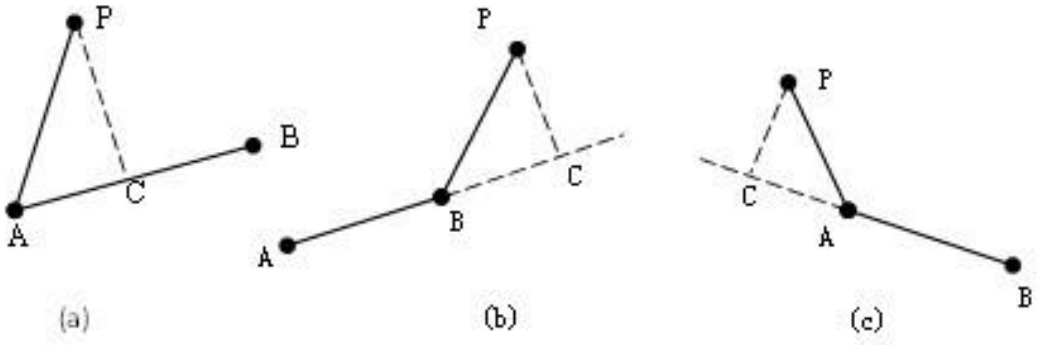
$$\begin{cases} X = x_0 + tl \\ Y = y_0 + tm \\ Z = z_0 + tn \end{cases}$$

To determine whether point P is within triangle ABC, this is can be done by the following procedure.
Calculate the absolute values of these three determinants and check if the summation is equal to the absolute value of n. If so, then point P is within the triangle ABC.

$$\begin{vmatrix} X & Y & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad \begin{vmatrix} x_1 & y_1 & 1 \\ X & Y & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ X & Y & 1 \end{vmatrix}$$

So far, we have solved the first part, if point P is within triangle ABC, the distance between point O and the triangle ABC is $|OP|$. **Next, we are going to deal with another situation where point P is not within the triangle ABC.**

If point P is not within the triangle ABC, then we have to calculate the minimum distance between point O and those three edges of triangle ABC. There are three cases in this situation as the following figure. In the following part, P represents any point we are interested in.



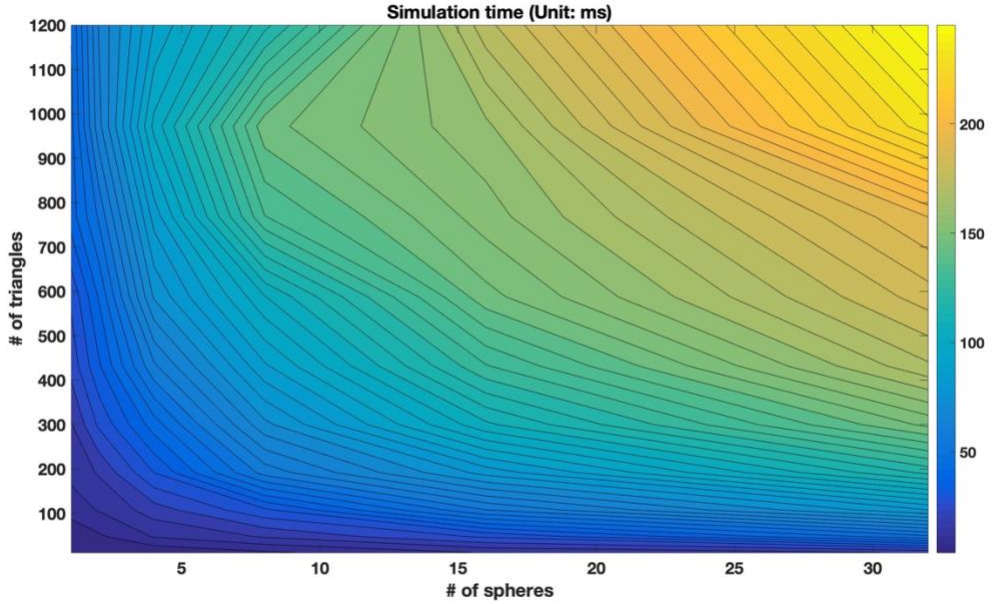
$$\overrightarrow{AC} = \frac{(\overrightarrow{AP} \cdot \overrightarrow{AB})}{|\overrightarrow{AB}|^2} \overrightarrow{AB} = \frac{(\overrightarrow{AP} \cdot \overrightarrow{AB})}{|\overrightarrow{AB}|} \cdot \frac{\overrightarrow{AB}}{|\overrightarrow{AB}|}$$

Define $r = \frac{\overrightarrow{AP} \cdot \overrightarrow{AB}}{|\overrightarrow{AB}|^2}$, the minimum distance can be calculated accordingly

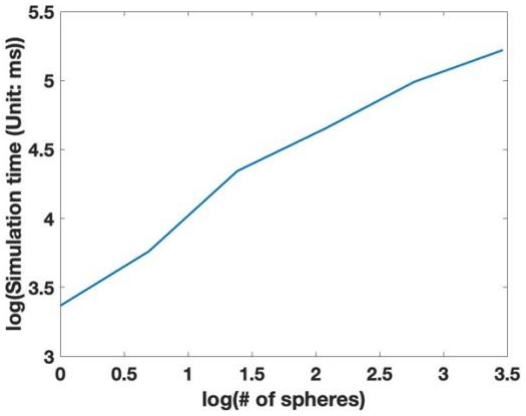
$$d = \begin{cases} |\overrightarrow{AP}| \; if \; r \leq 0 \\ |\overrightarrow{BP}| \; if \; r \geq 1 \\ |\overrightarrow{CP}| \; otherwise \end{cases}$$

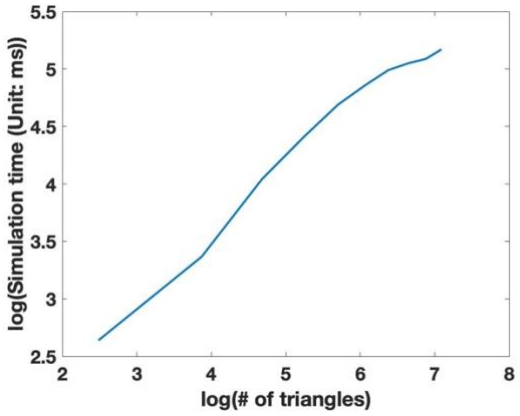
So far, we have figured out how to calculate the distance between the center of the sphere O and the triangle ABC in two cases. **Once we get the distance, we just need to compare the distance with the radius of spheres, if the distance is smaller than or equal to the radius, then there is collision, otherwise, there is no collision.**

3. Overview of results. Demonstration of your project



Above is the contour plot of the simulation time in ms, x-axis is the number of spheres and y-axis is the number of triangles.





Above are the log-log plots, the slopes of these two curves are approximately equal to 0.5, so the executable time goes up in $C\sqrt{n}$ where C is a constant and n is the number of spheres or the number of triangles. I think this is quite reasonable, for each pair of triangle and sphere, there is one series of operations that determines whether there is collision, so the time complexity should be $C'n$, where n is the number of triangle-sphere pairs.

4. Deliverables:

This report
Source code in ME459 repo, under folder **FinalProject**
Documentation generated by doxygen
To compile the code, run "module load cmake" first, and then use CMake to compile it. After the compiling, there is a "Main.jar" file in the directory, to run it, use command "java -jar Main.jar", then it will generate the final result in a file called "collision_detection.out".