CSE2015310886 박찬일 CG_A1 report

I started the project with cgcirc.sln, which is provided from cg.skku.edu. Most of the project is already set, and what I should modify is to create more than 20 circles and moving the circles.

For the vertex buffer, I should not upload CPU buffer to GPU buffer more than one time. Instead of creating a large vertex buffer(more than 20 circles and 73 vertexes per circle), I managed the information of circles in circle_t structure in circle.h, which is also provided.

In create_circles function in circle.h, I iterated the circle creating method for 20 times. The data of circles I created is stored in CPU memory. Because I have a vertex buffer for a circle and render the vertex buffer multiple times, I can make several circles from just one vertex buffer. The only data I should modify is the data of circles in CPU memory. The data will be transmitted to the GPU memory by uniform.

Anyway, I randomly created the circles 20times, but the circles were way a lot overlapped. I should have made circles not to overlap. So I checked all the circles create before whether it is too close or not. I calculated the length between two circles and maximized the limit of radius to that. This way I could avoid overlapping circles in creating step.

I added the two-dimensional velocity vector element to the circle_t structure, which is literally the velocity vector of the circle. In the "update" callable of circle_t structure, the center vector of circle moves by the velocity vector. In that way I implemented the moving circle.

Next was collision checking and resolving. For the collision check, it was way easier than resolving. I checked the distance between two circles and if the distance is shorter than the sum of two radius, it is in collision. And in case of border collision, I checked the center vector of the circle, whether if the distance between the center vector of the circle and border line is shorter than its radius, which means it is in collision with the border.

The collision resolving was quite difficult. In the case of border collision, I first flipped the velocity.x or velocity.y by its direction of collision. However, when a circle moves fast enough, it may pass over the border. So I brutally moved the center vector into the borderline. Also modified the direction of velocity vector into the borderline, instead of flipping its vector.

In case of collision between two circles, there was a well-defined vector-form formula of elastic collision in Wikipedia, so I accepted that. It was weird. Two circles stuck together and bounced wrong direction. I found the problem is from mutual resolving of collision. Two circles were resolving

collision mutually, so the calculation was overlapped. I modified the collision resolving condition to compare only circles created after itself. It worked only halfway. The circles bounced correct way, but still stuck together. Again, I brutally moved the center vector not to stuck together, as I did in the border case, by calculating the length of stuck part, and mutually pushing them in opposite direction.

Actually, what I should have done is not the vertex rendering or fragment rendering. It was quite physics-like problem. But I could be familiar to the concept of instancing. When I want to render multiple model which are topologically same, I can minimize the vertex buffer size by instancing.

I have never used C++ language for programming before, so several grammars were confusing. I could learn variety new C++ grammars working on this assignment. The most impressive one is random header. First I used rand() function to randomly create circles, but it was too slow changing the seed. When I tried to reset the circles, I should have wait for 1sec per change. I found random header for C++, and this made much faster randomizing function.

I was in trouble avoiding overlapping in circle creation, and I found Poisson Disk Sampling method. This time I could not use that one, but someday in future I will try it.