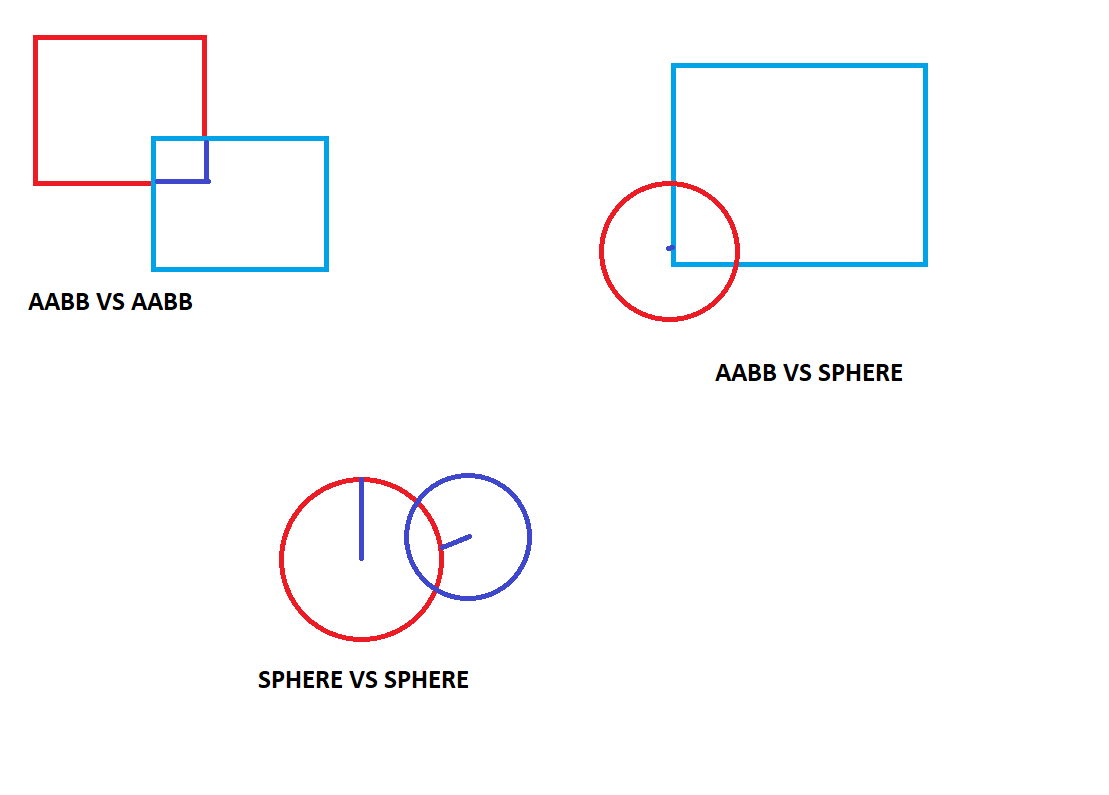
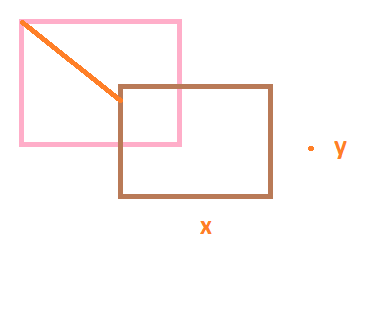
This is my 217CR Report for the July Resit. **By Nikolay Naydenov SID: 7118638**

The task is to Implement a library in C++ with Collision detection between AAB and a ball, ball and ball, bouncing of balls and implementing the physics for changing direction and velocity given the forces applied.

I decided to start with a simple 2D game first created with legacy OpenGl to get a grasp of the AABB vs AABB collision before I jump into 3D.



We start with the one in the top left corner. Just by looking at it you can see that collision between the objects is initiated when they overlap but let’s get more specific. The collision between the two could be mathematically detected a in a few ways. If either of the diagonals is shortened see the diagonal of the pink box hitting the brown means, there is something in it’s way inside the box or if it’s x and y axes are both crossed.

Using Boolean logic, we can represent it like so:

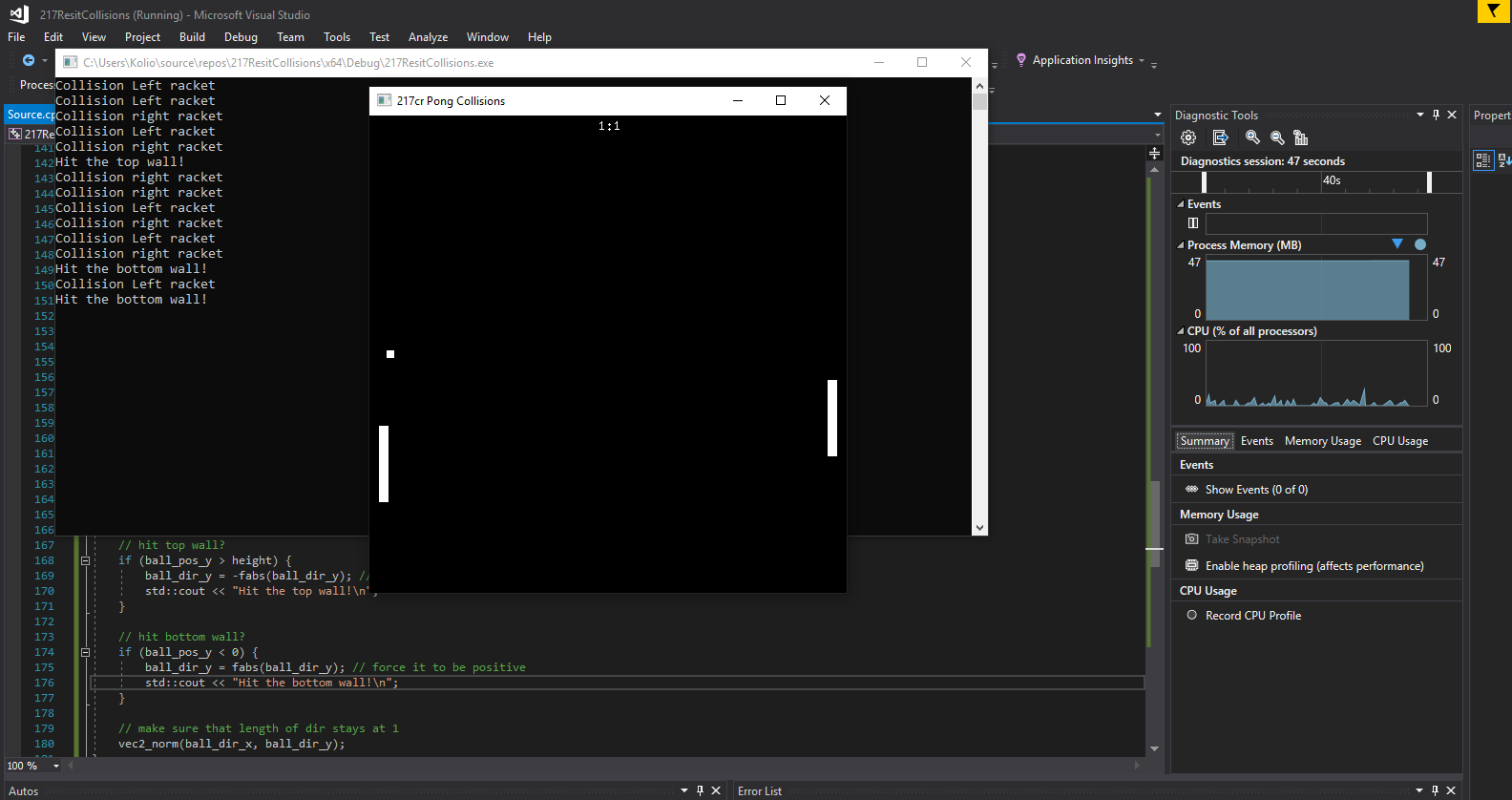
bool DoBoxesCollide(Box a, Box b) {

return (abs(a.x - b.x) \* 2 < (a.width + b.width)) &&

(abs(a.y - b.y) \* 2 < (a.height + b.height));

}

When we implement it is only half of the job done because now we know that they collided and it’s time to take some action. In my small 2D pong game after I detect the collision I want to know what I collided with so I could take an appropriate action. For that we have collision tests in debug mode or just a printed statement in the console.



Now that we know what we hit we can do multiple thing delete the object, restart the game, add score or the one we are interested here react to the object. When the object(ball) jets hit by one of the balls we want to calculate it’s new position and velocity and in this case the position is opposite of the direction the ball was heading. Let’s say we hit it with the left raket now the position direction and speed of the ball changes also if it hit the top bottom or middle part of it so we can add or deduct some of the angular movement. There is the representation.

void updateBall() {  
    // free fly   
    ball\_pos\_x += ball\_dir\_x \* ball\_speed;  
    ball\_pos\_y += ball\_dir\_y \* ball\_speed;  
      
    // hit by left racket?  
    if (ball\_pos\_x < racket\_left\_x + racket\_width &&   
        ball\_pos\_x > racket\_left\_x &&  
        ball\_pos\_y < racket\_left\_y + racket\_height &&  
        ball\_pos\_y > racket\_left\_y) {  
        // set angle depending where it hit the rocket  
        // (t is 0.5 if hit at top, 0 at center, -0.5 at bottom)  
        float t = ((ball\_pos\_y - racket\_left\_y) / racket\_height) - 0.5f;  
        ball\_dir\_x = fabs(ball\_dir\_x); // force it to be positive  
        ball\_dir\_y = t;  
    }

Now that I’ve got this done let’s go to the collider in the game engine and see how thing differ. The first thing that has to change is the number of axes as it is a 3 dimensional object we have to introduce a 3 rd. axes Z and for the cubes to collide we need all 3 axes X Y Z to be crossed.

bool CubeCollider::collidesWith(Collider\* other) {

if (other == NULL) {

return false;

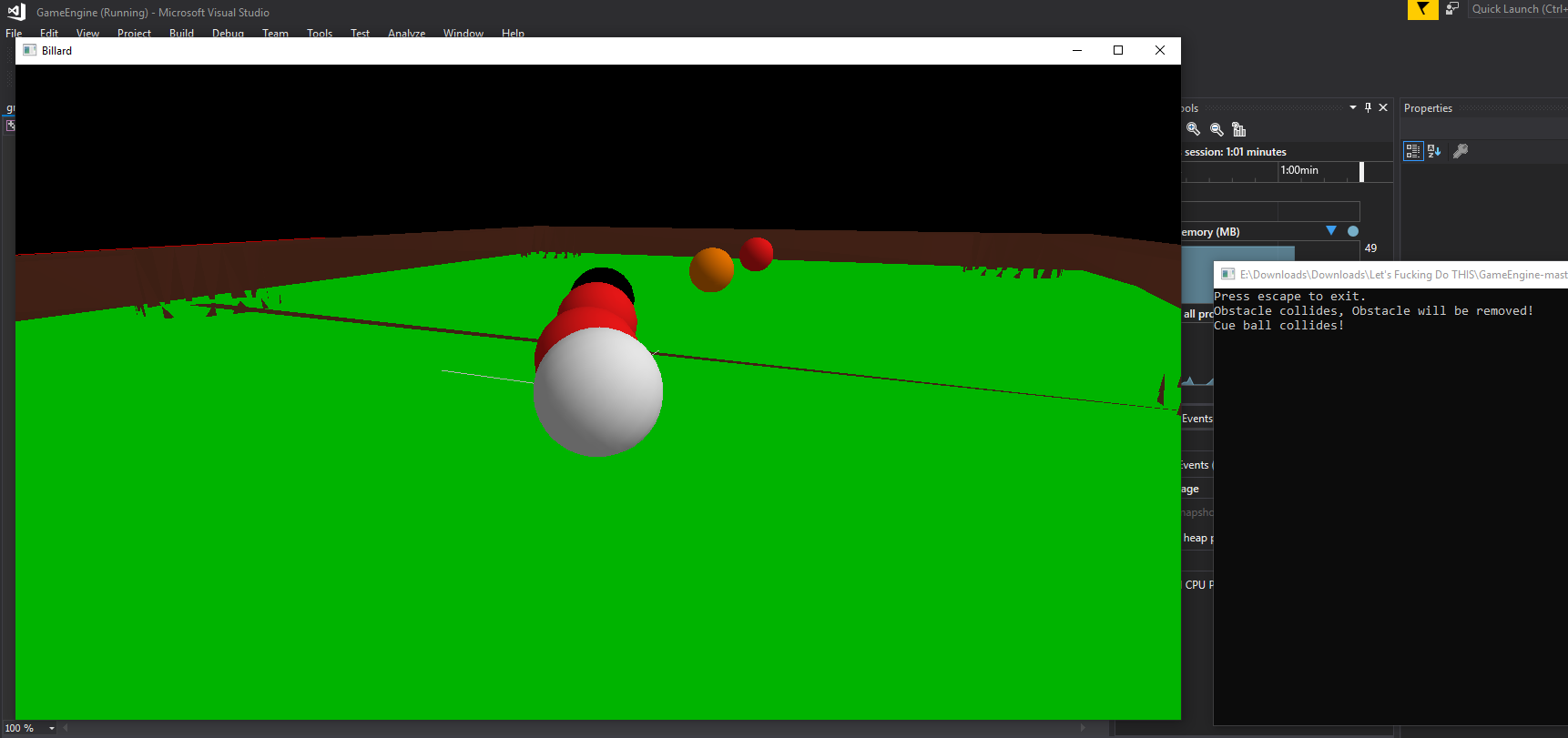
}

bool overlapX = other->minX() <= this->maxX() && other->maxX() >= this->minX();

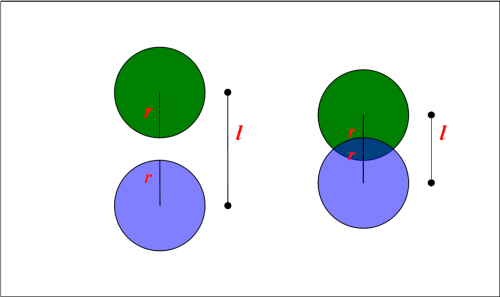
bool overlapY = other->minY() <= this->maxY() && other->maxY() >= this->minY();

bool overlapZ = other->minZ() <= this->maxZ() && other->maxZ() >= this->minZ();

return overlapX && overlapY && overlapZ;

We check all the axes and return the values, so we know whether we collider or not. Once collision has been detected between 2 or more objects we test who these objects were and take action in this case deleting the object.

From this point on I started working on the sphere collider and sphere vs sphere collision. To detect collision the most important value we need to know is the diameter and radius. If two spheres are colliding we take the radiuses and compare it to the length from center of the spheres. If the length is smaller than the sum of radiuses there is collision.

f(P,S)=Sradius>=(Px-Sx)2+(Py-Sy)2+(Pz-Sz)2  
  
However I did not manage to integrate a sphere   
collider and sphere vs sphere collision without  
the use of a vertex library so I would leave it here.

References:   
uroborostudio.com

Noobtuts.com

Youtube.com/thecherno