Пусть частица задается N точками координаты которых $\boldsymbol{X}_k, k \in \{1, N\}$ и скорости в этих точках $\boldsymbol{U}_k, k \in \{1, N\}$, также центр частицы $\boldsymbol{X}_{center}$ и его скорость $\boldsymbol{U}_{center}$.

Для каждой частицы создается объект класса $SurfaceVelocity$. У объекта класса $SurfaceVelocity$ есть приватные поля $force$ – равнодействующая сила, с которой жидкость влияет на частицу, $torque$ – общий момент сил, $center$ – (x, y) координаты центра частицы, $center\_velocity$ – скорость движения центра частицы, $omega$ – угловая скорость частицы, m – масса частицы, rad и $rad2$ радиус частицы и его квадрат.

Этот объект должен иметь переопределенный оператор () – это требование функции instantiateImmersedWallData из Palabos. Оператор () принимает координаты точки на границе частицы (x, y) и возвращает скорость в этой точке.

$$M\frac{d\boldsymbol{U}(t)}{dt} = \boldsymbol{F}(t)$$

$$\frac{1}{2}MR^2 \cdot \frac{d\omega}{dt} = T_z$$

$$\boldsymbol{U}_k(t) = \boldsymbol{U}_{center}(t) + \{-\boldsymbol{r}_y \cdot \omega, \boldsymbol{r}_x \cdot \omega\}$$

Где $\boldsymbol{r} = \boldsymbol{X}_k(t) - \boldsymbol{X}_{center}(t)$ радиус-вектор.

```
Array<T,2> operator()(Array<T,2> const& pos)
{
    Array<T, 2> r = pos - center;
    return center_velocity + Array<T, 2>({-r[1] * omega, r[0] * omega});
}
```

Главный цикл в программе делает следующее:

1. LBM step
2. move particles

```
    for particle in particles
       particle.recalc_velocities()
       particle.move_center()
       for point in particle.points
           //add to particle's coordinates its velocity
           point += particle.SurfaceVelocity(point)
       endfor
       particle.force = 0
       particle.torque = 0
    endfor
3. IBM
    for particle in particles
        //Calculate acting force using IBM
        particle.force = IBM.force
        particle.torque = IBM.torque
    endfor
```

Сейчас координаты я меняю, используя явный метод Эйлера ($\Delta t = 1$).

То есть, простейший подход - итерационный. Делается так $u_j(t+1) = \alpha u_{j-1}(t+1)+(1-\alpha)u(t)+$правая часть, где $u_0(t+1) = u(t)$ - и делаются 3-4 итерации, пока например $(u_j(t+1)-u_{j-1}(t+1))/u_{j-1}(t+1)$ не будет менее $1e-3$.

Чтобы рассчитать скорости $\{vx, vy\}(t+1)$ и $omega(t+1)$ для центра частицы, использую формулы 1,2 :

$$\{vx, vy\}_j(t+1) = \alpha\{vx, vy\}_{j-1}(t+1)+(1-\alpha)\{vx, vy\}(t)+g(\{vx, vy\}(t), t)/m \quad (1)$$

Критерий остановки $||\{vx, vy\}_j - \{vx, vy\}_{j-1}||/||\{vx, vy\}_{j-1}|| < 1e-3$

$$w_j(t+1) = \alpha w_{j-1}(t+1) + (1-\alpha)w(t) + 2torque(t)/m/R^2 \qquad (2)$$

Критерий остановки $|w_j - w_{j-1}|/|w_{j-1}| < 1e-3$

Эти формулы реализованы в функции recalc_velocities. Переменные, которые не меняют своих значений во время итераций: $\{vx, vy\}(t)$ – $center\_velocity$, $g(\{vx, vy\}(t), t)$ – $force$. Я сделала ограничение на количество итераций (20), вдруг условие $eps < 1e-3$ не сработает. Останавливается примерно на 9-10. Сами начальные скорости и жидкости, и частицы примерно $1e-5$.

```
void recalc_velocities(){
    Array<T, 2> velocity_j_1 = {center_velocity[0], center_velocity[1]};
    Array<T, 2> velocity_j = {0., 0.};
    T a = 0.5;
    for(int i = 0; i < 20; i++  ){
        velocity_j = a * velocity_j_1 +
                    (1 - a) * center_velocity + force / m;
        Array<T, 2> d = velocity_j_1 - velocity_j;

        T diff = std::sqrt(d[0]*d[0] + d[1]*d[1])/
                 std::sqrt(velocity_j_1[0]*velocity_j_1[0]+
                    velocity_j_1[1]*velocity_j_1[1]);
        if(diff < 1e-3){
            pcout << "iterations velocity " << diff
                  << " it " << i << std::endl;
            break;
        }
        velocity_j_1 = velocity_j;
    }
    center_velocity = velocity_j;
```

```
    T omega_j_1 = omega;
    T omega_j = 0.;

    for(int i = 0; i < 20; i++  ){
        omega_j = a * omega_j_1 + (1 - a) * omega
                 + 2 * torque / m / rad2;
        if(fabs(omega_j_1 - omega_j)/fabs(omega_j_1) < 1e-3){
            pcout << "iterations omega" <<
                fabs(omega_j_1 - omega_j)/fabs(omega_j_1)
                << " it " << i << std::endl;
            break;
        }
        omega_j_1 = omega_j;
    }
    omega = omega_j;
}
```