600086 Lab Book

Week 6 – Lab F

Date: 16th Mar 2022

Q1. Colliding Particles

Questions:

a) Modify the particle object so that it can collide with other particles
b) Add a scoped thread pool that handles the collisions of particles and counts them using a local variable

Solution:

a) Modified the particle struct top contain a collision function that takes another particle as a parameter

```
impl Particle
{
    pub fn new( _id:u32, xdir : f32, ydir :f32, x:f32 , y: f32,
vel:f32,radius:f32) -> Particle
    {
        Particle
        {
            id       : _id,
            x_dir    : xdir,
            y_dir    : ydir,
            x_pos    : x,
            y_pos    : y,
            velocity : vel,
            size     : radius,
        }
    }
    pub fn move_particle(&mut self, enclosure_width :f32,
enclosure_height:f32)
    {
        let x2 = self.x_dir * self.velocity;
        let y2 = self.y_dir * self.velocity;
        let test = self.x_pos + x2;
        if test < 0.0 || test > enclosure_width
        {
            self.x_dir *= -1.0;
        }

        let test = self.y_pos + y2;
        if test < 0.0 || test > enclosure_height
        {
            self.y_dir *= -1.0;
        }
```

```rust
        self.x_pos += x2;
        self.y_pos += y2;
    }
    pub fn collide(&mut self, p : &Particle) -> bool
    {
        let p2 = p;
        if self.id == p2.id
        {
            return false;
        }
        let delta_x = self.x_pos - p2.x_pos;
        let delta_y = self.y_pos - p2.y_pos;
        let delta_h_between_centers =f32::sqrt(f32::powf(delta_x,2.0) +
f32::powf(delta_y,2.0));
        let delta_h = delta_h_between_centers - self.size - p2.size;
        if delta_h < 0.0
        {
            return true;
        }
        return false;
    }
```

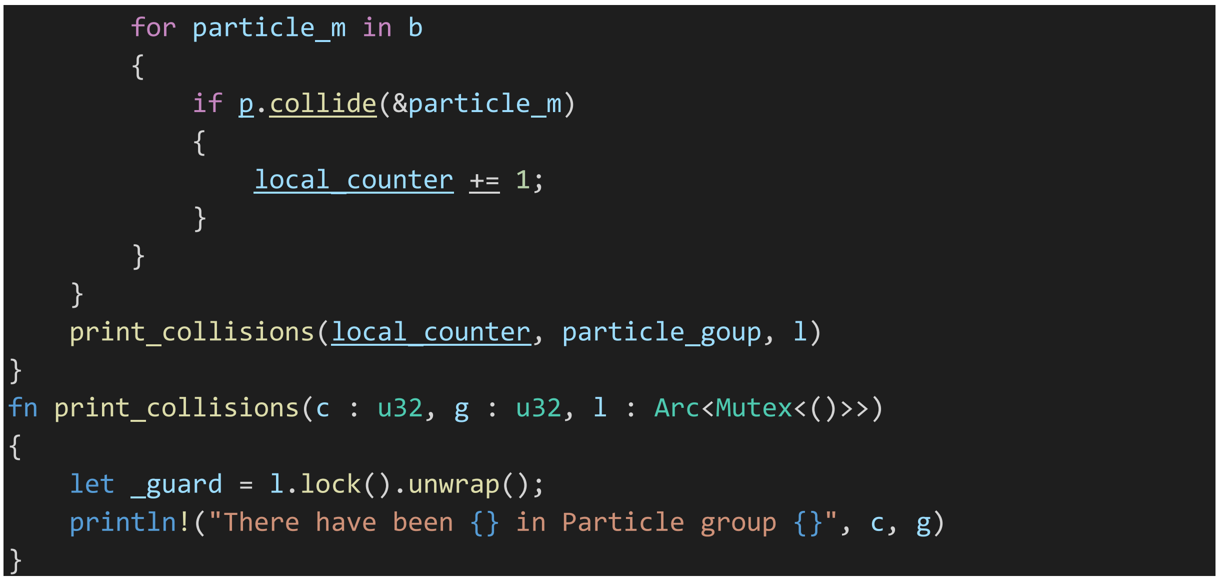
It then calculates the distance between the centres of each particle and subtracts the radius of each from the distance if the resulting value is less then 0 then the particles would have collided it then returns true, otherwise it returns false, I also added an id attribute to the particle class so that the particles can identify itself and ignore the collision check in this circumstance.

b) Added a second thread main function that takes the chunk of particles to check and a master list of particles to check against and uses a local counter to keep track

```rust
pool.scoped(|scope| {
    let mut i = 1;
    for slice in ref p.chunks_mut(PARTICLES_PER_THREAD) {
        let m = Clone::clone(&master_list);
        let l = global_lock.clone();
         scope.execute(move || collision_thread_main(i,slice,&m,l));
         i += 1;
    }
});
```

…

```rust
fn collision_thread_main(particle_goup : u32, list: &mut [Particle],
master_list:&Vec<Particle>, l : Arc<Mutex<()>>)
{
    let b = master_list;
    let a = list.iter_mut();
    let mut local_counter : u32 = 0;
    for particle_a in a
    {
        let p = particle_a;
```



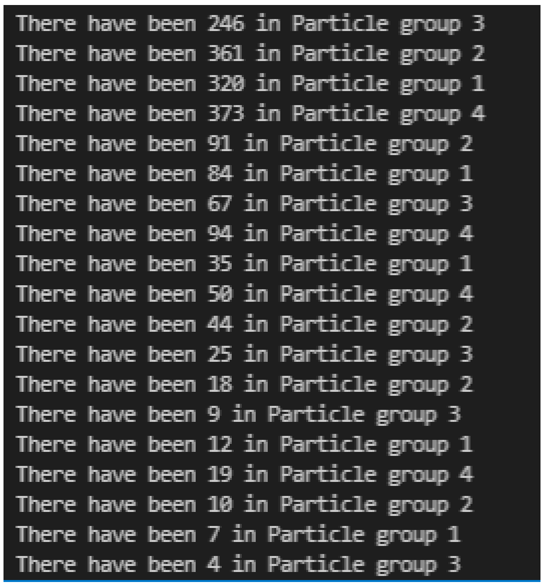
```rust
        for particle_m in b
        {
            if p.collide(&particle_m)
            {
                local_counter += 1;
            }
        }
    }
    print_collisions(local_counter, particle_goup, l)
}
fn print_collisions(c : u32, g : u32, l : Arc<Mutex<()>>)
{
    let _guard = l.lock().unwrap();
    println!("There have been {} in Particle group {}", c, g)
}
```

I also added a print function that will print the local thread count for each chunk , I implemented a guard to avoid a race condition.

Test data:

N/A

Sample output:



```
There have been 246 in Particle group 3
There have been 361 in Particle group 2
There have been 320 in Particle group 1
There have been 373 in Particle group 4
There have been 91 in Particle group 2
There have been 84 in Particle group 1
There have been 67 in Particle group 3
There have been 94 in Particle group 4
There have been 35 in Particle group 1
There have been 50 in Particle group 4
There have been 44 in Particle group 2
There have been 25 in Particle group 3
There have been 18 in Particle group 2
There have been 9 in Particle group 3
There have been 12 in Particle group 1
There have been 19 in Particle group 4
There have been 10 in Particle group 2
There have been 7 in Particle group 1
There have been 4 in Particle group 3
```

Reflection:

Had more difficulty implementing the local counter than the atomic counter.

Metadata:

Further information:

N/A

Q2. Recording collisions using an Atomic

Question:

Replace the local counter with an atomic counter to measure the number of collisions across all threads. This counter should be stored only once in the Particle System class.

Solution:

Modified the particle system class to include the Atomic reference counter

```
struct ParticleSystem
{
    collision_counter: Arc<Mutex<u32>>,
    particles: Vec<Particle>,
}
impl ParticleSystem
{
    pub fn new() -> ParticleSystem
    {
        ParticleSystem
        {
            collision_counter : Arc::new(Mutex::new(0)),
            particles: Vec::new(),
        }
    }
    pub fn run_system(&mut self,enclosure_width :f32, enclosure_height:f32)
    {
        for particle in self.particles.iter_mut()
        {
            let x = enclosure_width;
            let y = enclosure_height;
            let p = particle;
            p.move_particle(x,y);
        }
    }
}
```

This is then passed into the scoped thread main for the collision checker

```
        let master_list = Clone::clone(&p.particles);
        ref p = & mut p; // p represents my particle system
 pool.scoped(|scope| {
            for slice in ref p.particles.chunks_mut(PARTICLES_PER_THREAD) {
                let m = Clone::clone(&master_list);
                let l = ref p.collision_counter.clone();
                scope.execute(move || collision_thread_main(slice,&m,l));
            }
        });
```

The collision thread main has been modified to the following

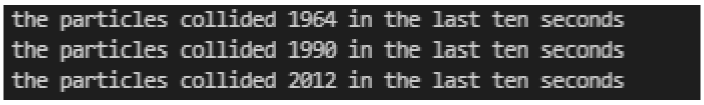
```
fn collision_thread_main(list: &mut [Particle], master_list:&Vec<Particle>,
atomic_counter : Arc<Mutex<u32>>)
{
    let b = master_list;
    let a = list.iter_mut();

    for particle_a in a
    {
        let p = particle_a;
        for particle_m in b
        {
            if p.collide(&particle_m)
            {
                *atomic_counter.lock().unwrap() += 1;
            }
        }
    }
}
```

The atomic_counter locks while it is updated then the next thread can also update the same value I have set the code up to print the running total of collisions every 10 seconds.

Test data:

N/A

Sample output:



```
the particles collided 1964 in the last ten seconds
the particles collided 1990 in the last ten seconds
the particles collided 2012 in the last ten seconds
```

Reflection:

The particle collisions decrease exponentially at first this is likely due to the blooming of the p[articles as they all begin in the same location and bloom out in all direction as they spread the collisions become less and less frequent.

Metadata:

OpenGL

Further information:

N/A