600086 Lab Book

# Week 6 – Lab F

Date: 16th Mar 2022

## Q1. Colliding Particles

### Questions:

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

### Solution:

1. 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;

        }

        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.

1. 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

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;

}

});

…

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;

        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:

Text

Description automatically generated

### 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:



### 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