0456024 蔡佩珊

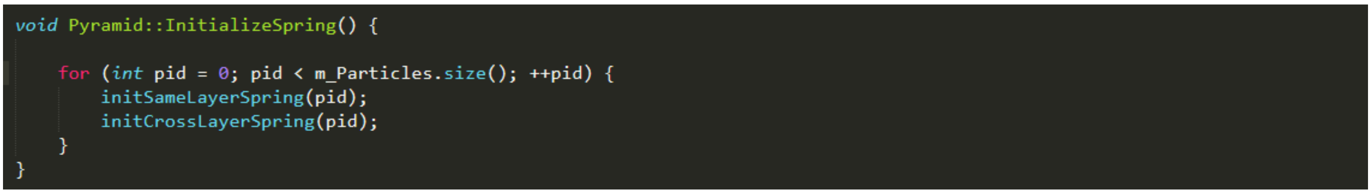
Computer Animation Project Assignment 1

♦ Topic

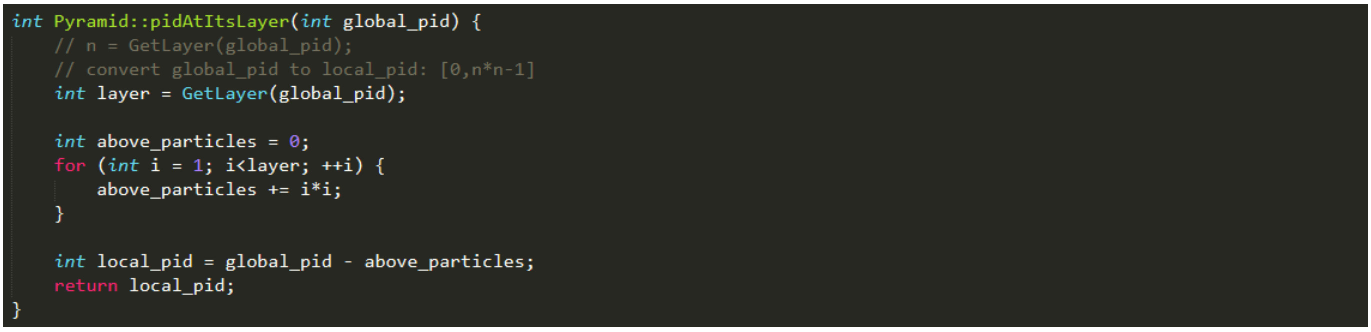
Particle System – Soft Body Simulation

♦ Implementation (see commands in code)

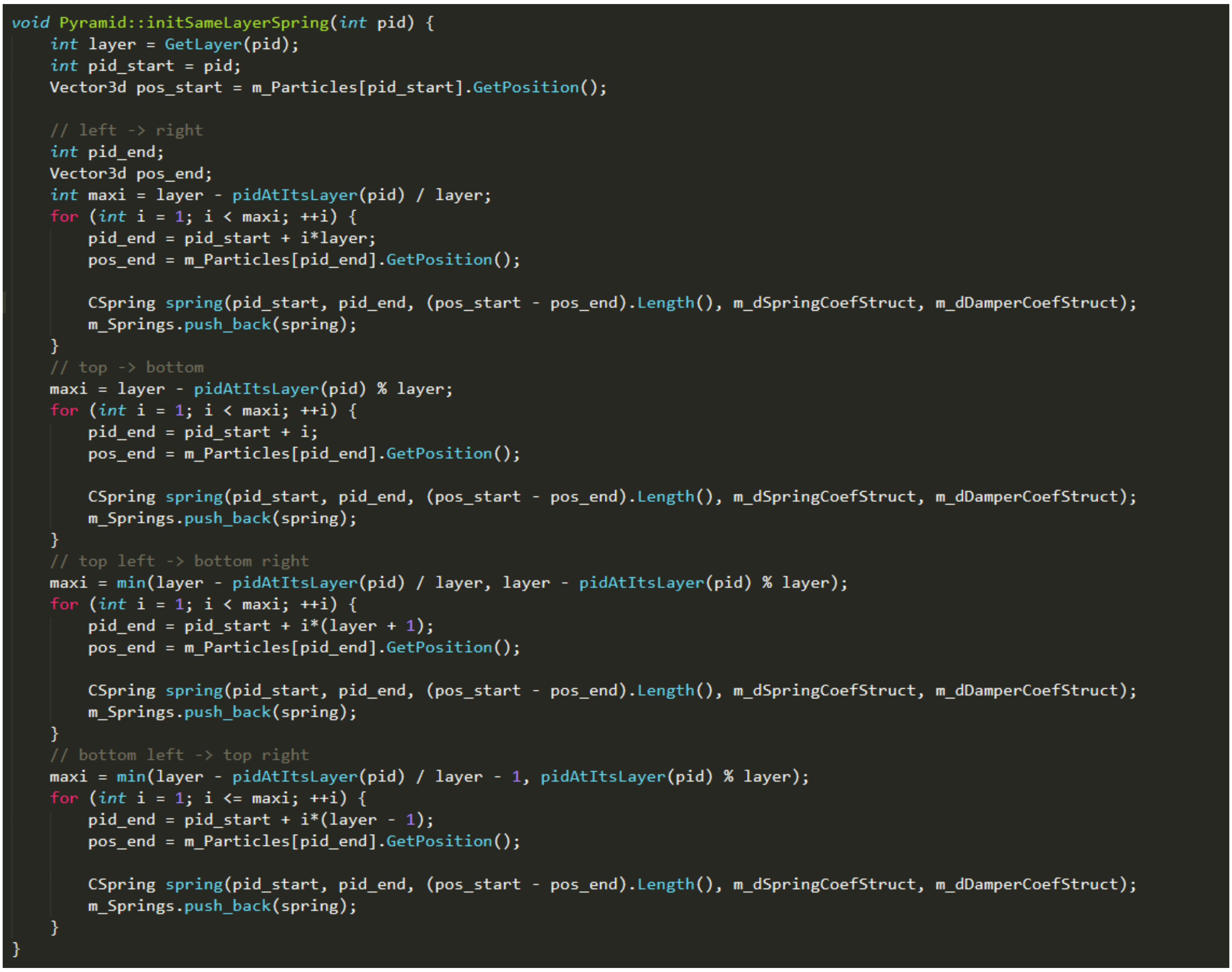
Pyramid::InitializeSpring()



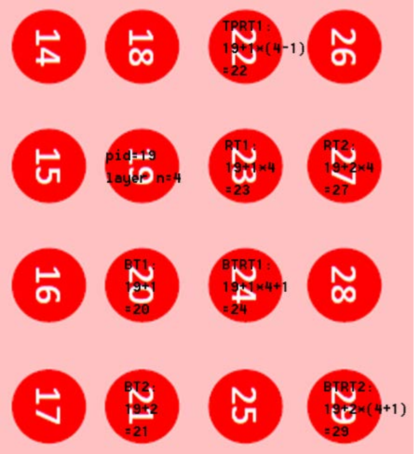
```cpp
void Pyramid::InitializeSpring() {

    for (int pid = 0; pid < m_Particles.size(); ++pid) {
        initSameLayerSpring(pid);
        initCrossLayerSpring(pid);
    }
}
```

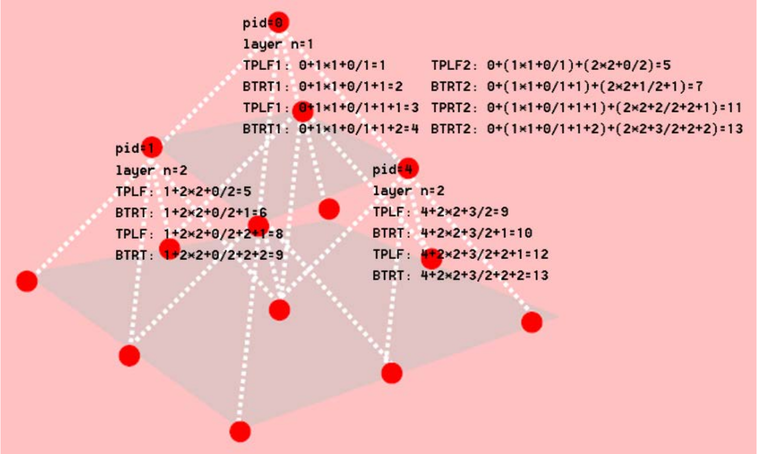


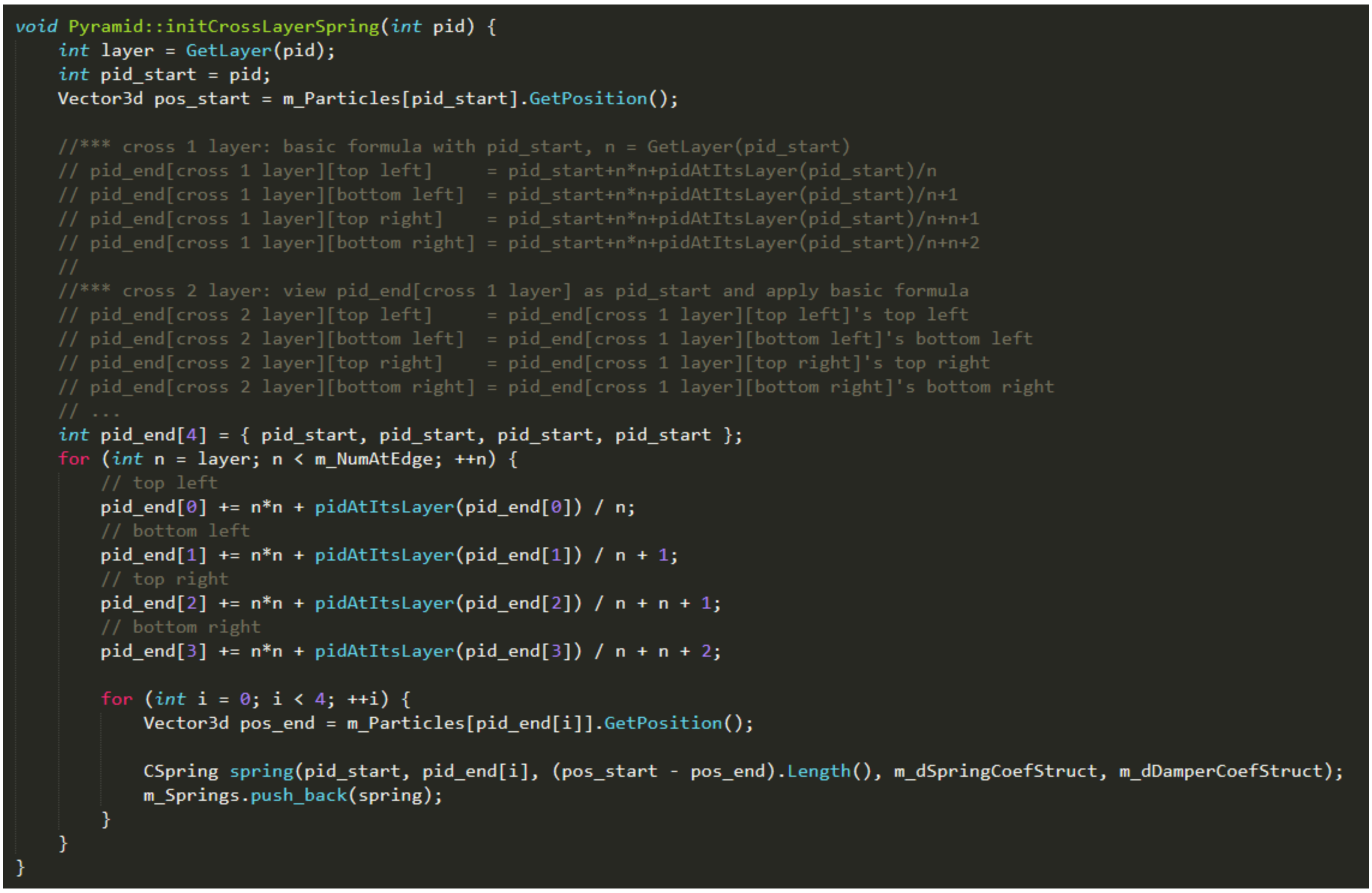
```cpp
int Pyramid::pidAtItsLayer(int global_pid) {
    // n = GetLayer(global_pid);
    // convert global_pid to local_pid: [0,n*n-1]
    int layer = GetLayer(global_pid);

    int above_particles = 0;
    for (int i = 1; i<layer; ++i) {
        above_particles += i*i;
    }

    int local_pid = global_pid - above_particles;
    return local_pid;
}
```



```cpp
void Pyramid::initSameLayerSpring(int pid) {
    int layer = GetLayer(pid);
    int pid_start = pid;
    Vector3d pos_start = m_Particles[pid_start].GetPosition();

    // left -> right
    int pid_end;
    Vector3d pos_end;
    int maxi = layer - pidAtItsLayer(pid) / layer;
    for (int i = 1; i < maxi; ++i) {
        pid_end = pid_start + i*layer;
        pos_end = m_Particles[pid_end].GetPosition();

        CSpring spring(pid_start, pid_end, (pos_start - pos_end).Length(), m_dSpringCoefStruct, m_dDamperCoefStruct);
        m_Springs.push_back(spring);
    }
    // top -> bottom
    maxi = layer - pidAtItsLayer(pid) % layer;
    for (int i = 1; i < maxi; ++i) {
        pid_end = pid_start + i;
        pos_end = m_Particles[pid_end].GetPosition();

        CSpring spring(pid_start, pid_end, (pos_start - pos_end).Length(), m_dSpringCoefStruct, m_dDamperCoefStruct);
        m_Springs.push_back(spring);
    }
    // top left -> bottom right
    maxi = min(layer - pidAtItsLayer(pid) / layer, layer - pidAtItsLayer(pid) % layer);
    for (int i = 1; i < maxi; ++i) {
        pid_end = pid_start + i*(layer + 1);
        pos_end = m_Particles[pid_end].GetPosition();

        CSpring spring(pid_start, pid_end, (pos_start - pos_end).Length(), m_dSpringCoefStruct, m_dDamperCoefStruct);
        m_Springs.push_back(spring);
    }
    // bottom left -> top right
    maxi = min(layer - pidAtItsLayer(pid) / layer - 1, pidAtItsLayer(pid) % layer);
    for (int i = 1; i <= maxi; ++i) {
        pid_end = pid_start + i*(layer - 1);
        pos_end = m_Particles[pid_end].GetPosition();

        CSpring spring(pid_start, pid_end, (pos_start - pos_end).Length(), m_dSpringCoefStruct, m_dDamperCoefStruct);
        m_Springs.push_back(spring);
    }
}
```

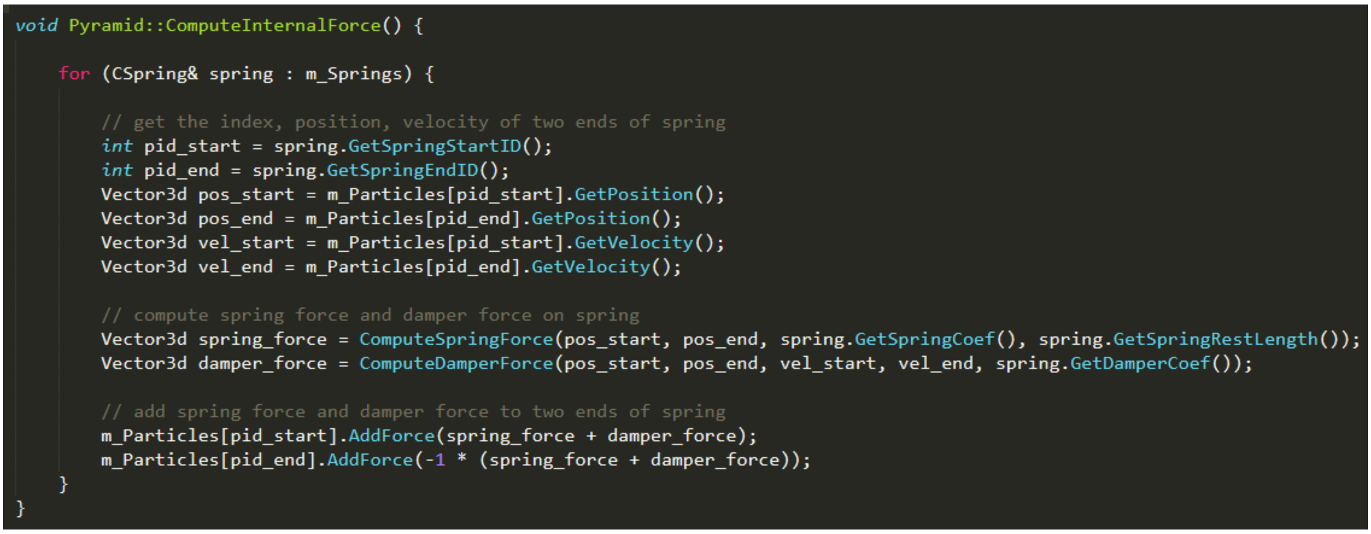






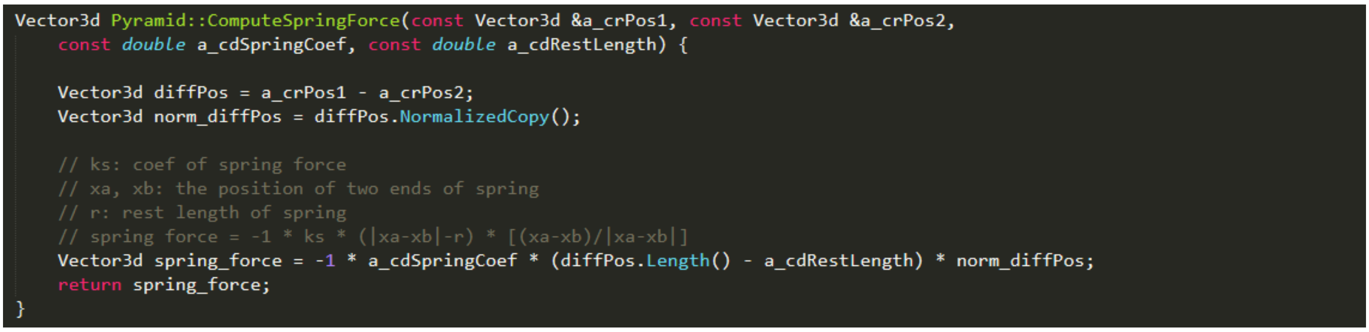
```
void Pyramid::initCrossLayerSpring(int pid) {
    int layer = GetLayer(pid);
    int pid_start = pid;
    Vector3d pos_start = m_Particles[pid_start].GetPosition();

    //*** cross 1 layer: basic formula with pid_start, n = GetLayer(pid_start)
    // pid_end[cross 1 layer][top left]     = pid_start+n*n+pidAtItsLayer(pid_start)/n
    // pid_end[cross 1 layer][bottom left]  = pid_start+n*n+pidAtItsLayer(pid_start)/n+1
    // pid_end[cross 1 layer][top right]    = pid_start+n*n+pidAtItsLayer(pid_start)/n+n+1
    // pid_end[cross 1 layer][bottom right] = pid_start+n*n+pidAtItsLayer(pid_start)/n+n+2
    //
    //*** cross 2 layer: view pid_end[cross 1 layer] as pid_start and apply basic formula
    // pid_end[cross 2 layer][top left]     = pid_end[cross 1 layer][top left]'s top left
    // pid_end[cross 2 layer][bottom left]  = pid_end[cross 1 layer][bottom left]'s bottom left
    // pid_end[cross 2 layer][top right]    = pid_end[cross 1 layer][top right]'s top right
    // pid_end[cross 2 layer][bottom right] = pid_end[cross 1 layer][bottom right]'s bottom right
    // ...
    int pid_end[4] = { pid_start, pid_start, pid_start, pid_start };
    for (int n = layer; n < m_NumAtEdge; ++n) {
        // top left
        pid_end[0] += n*n + pidAtItsLayer(pid_end[0]) / n;
        // bottom left
        pid_end[1] += n*n + pidAtItsLayer(pid_end[1]) / n + 1;
        // top right
        pid_end[2] += n*n + pidAtItsLayer(pid_end[2]) / n + n + 1;
        // bottom right
        pid_end[3] += n*n + pidAtItsLayer(pid_end[3]) / n + n + 2;

        for (int i = 0; i < 4; ++i) {
            Vector3d pos_end = m_Particles[pid_end[i]].GetPosition();

            CSpring spring(pid_start, pid_end[i], (pos_start - pos_end).Length(), m_dSpringCoefStruct, m_dDamperCoefStruct);
            m_Springs.push_back(spring);
        }
    }
}
```

Pyramid::ComputeInternalForce()



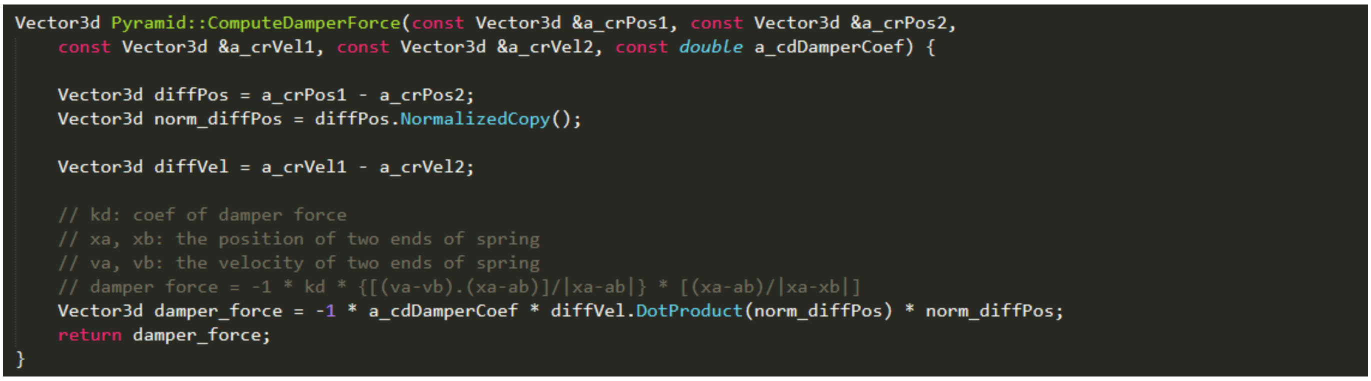
```
void Pyramid::ComputeInternalForce() {

    for (CSpring& spring : m_Springs) {

        // get the index, position, velocity of two ends of spring
        int pid_start = spring.GetSpringStartID();
        int pid_end = spring.GetSpringEndID();
        Vector3d pos_start = m_Particles[pid_start].GetPosition();
        Vector3d pos_end = m_Particles[pid_end].GetPosition();
        Vector3d vel_start = m_Particles[pid_start].GetVelocity();
        Vector3d vel_end = m_Particles[pid_end].GetVelocity();

        // compute spring force and damper force on spring
        Vector3d spring_force = ComputeSpringForce(pos_start, pos_end, spring.GetSpringCoef(), spring.GetSpringRestLength());
        Vector3d damper_force = ComputeDamperForce(pos_start, pos_end, vel_start, vel_end, spring.GetDamperCoef());

        // add spring force and damper force to two ends of spring
        m_Particles[pid_start].AddForce(spring_force + damper_force);
        m_Particles[pid_end].AddForce(-1 * (spring_force + damper_force));
    }
}
```

Pyramid::ComputeSpringForce()



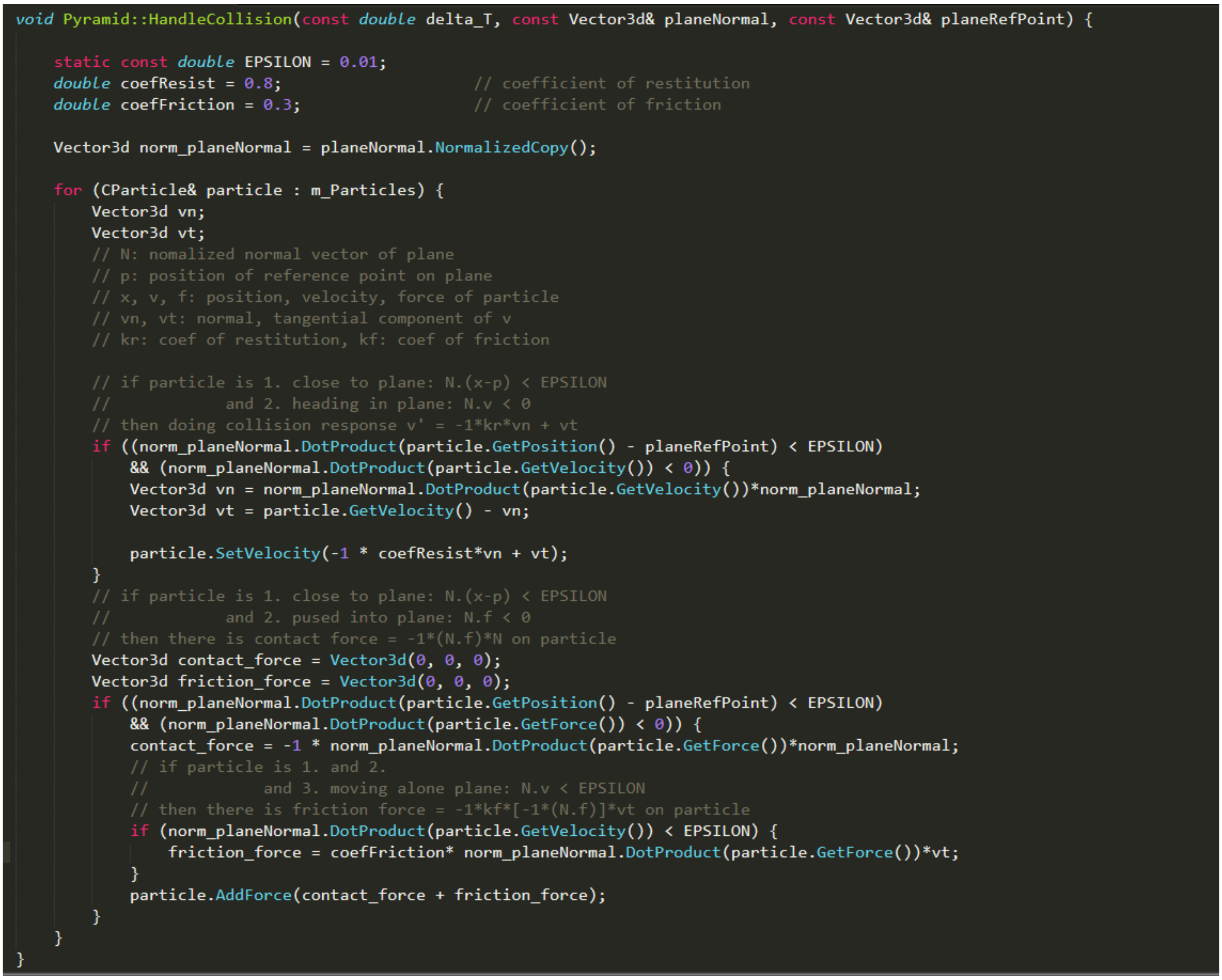
```
Vector3d Pyramid::ComputeSpringForce(const Vector3d &a_crPos1, const Vector3d &a_crPos2,
    const double a_cdSpringCoef, const double a_cdRestLength) {

    Vector3d diffPos = a_crPos1 - a_crPos2;
    Vector3d norm_diffPos = diffPos.NormalizedCopy();

    // ks: coef of spring force
    // xa, xb: the position of two ends of spring
    // r: rest length of spring
    // spring force = -1 * ks * (|xa-xb|-r) * [(xa-xb)/|xa-xb|]
    Vector3d spring_force = -1 * a_cdSpringCoef * (diffPos.Length() - a_cdRestLength) * norm_diffPos;
    return spring_force;
}
```

Pyramid::ComputeDamperForce()



```
Vector3d Pyramid::ComputeDamperForce(const Vector3d &a_crPos1, const Vector3d &a_crPos2,
    const Vector3d &a_crVel1, const Vector3d &a_crVel2, const double a_cdDamperCoef) {

    Vector3d diffPos = a_crPos1 - a_crPos2;
    Vector3d norm_diffPos = diffPos.NormalizedCopy();

    Vector3d diffVel = a_crVel1 - a_crVel2;

    // kd: coef of damper force
    // xa, xb: the position of two ends of spring
    // va, vb: the velocity of two ends of spring
    // damper force = -1 * kd * {[(va-vb).(xa-ab)]/|xa-ab|} * [(xa-ab)/|xa-xb|]
    Vector3d damper_force = -1 * a_cdDamperCoef * diffVel.DotProduct(norm_diffPos) * norm_diffPos;
    return damper_force;
}
```

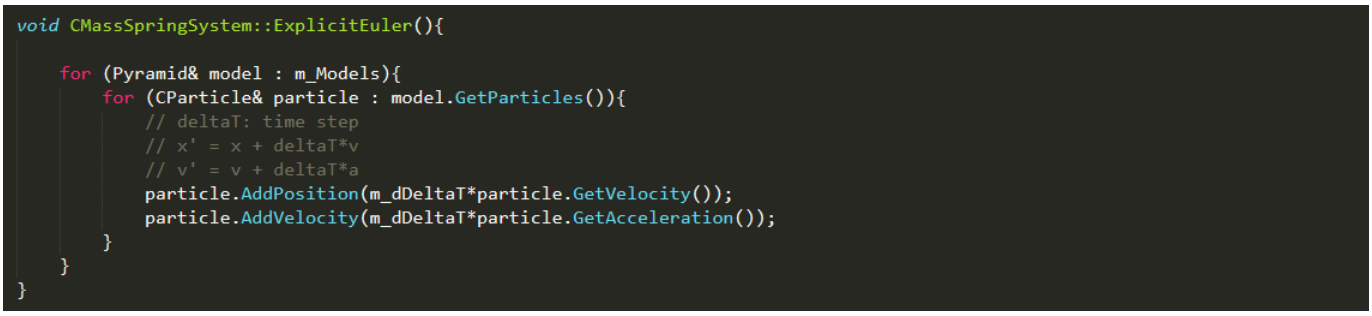
Pyramid::HandleCollision()



```
void Pyramid::HandleCollision(const double delta_T, const Vector3d& planeNormal, const Vector3d& planeRefPoint) {

    static const double EPSILON = 0.01;
    double coefResist = 0.8;                    // coefficient of restitution
    double coefFriction = 0.3;                  // coefficient of friction

    Vector3d norm_planeNormal = planeNormal.NormalizedCopy();

    for (CParticle& particle : m_Particles) {
        Vector3d vn;
        Vector3d vt;
        // N: nomalized normal vector of plane
        // p: position of reference point on plane
        // x, v, f: position, velocity, force of particle
        // vn, vt: normal, tangential component of v
        // kr: coef of restitution, kf: coef of friction

        // if particle is 1. close to plane: N.(x-p) < EPSILON
        //            and 2. heading in plane: N.v < 0
        // then doing collision response v' = -1*kr*vn + vt
        if ((norm_planeNormal.DotProduct(particle.GetPosition() - planeRefPoint) < EPSILON)
            && (norm_planeNormal.DotProduct(particle.GetVelocity()) < 0)) {
            Vector3d vn = norm_planeNormal.DotProduct(particle.GetVelocity())*norm_planeNormal;
            Vector3d vt = particle.GetVelocity() - vn;

            particle.SetVelocity(-1 * coefResist*vn + vt);
        }
        // if particle is 1. close to plane: N.(x-p) < EPSILON
        //            and 2. pused into plane: N.f < 0
        // then there is contact force = -1*(N.f)*N on particle
        Vector3d contact_force = Vector3d(0, 0, 0);
        Vector3d friction_force = Vector3d(0, 0, 0);
        if ((norm_planeNormal.DotProduct(particle.GetPosition() - planeRefPoint) < EPSILON)
            && (norm_planeNormal.DotProduct(particle.GetForce()) < 0)) {
            contact_force = -1 * norm_planeNormal.DotProduct(particle.GetForce())*norm_planeNormal;
            // if particle is 1. and 2.
            //            and 3. moving alone plane: N.v < EPSILON
            // then there is friction force = -1*kf*[-1*(N.f)]*vt on particle
            if (norm_planeNormal.DotProduct(particle.GetVelocity()) < EPSILON) {
                friction_force = coefFriction* norm_planeNormal.DotProduct(particle.GetForce())*vt;
            }
            particle.AddForce(contact_force + friction_force);
        }
    }
}
```

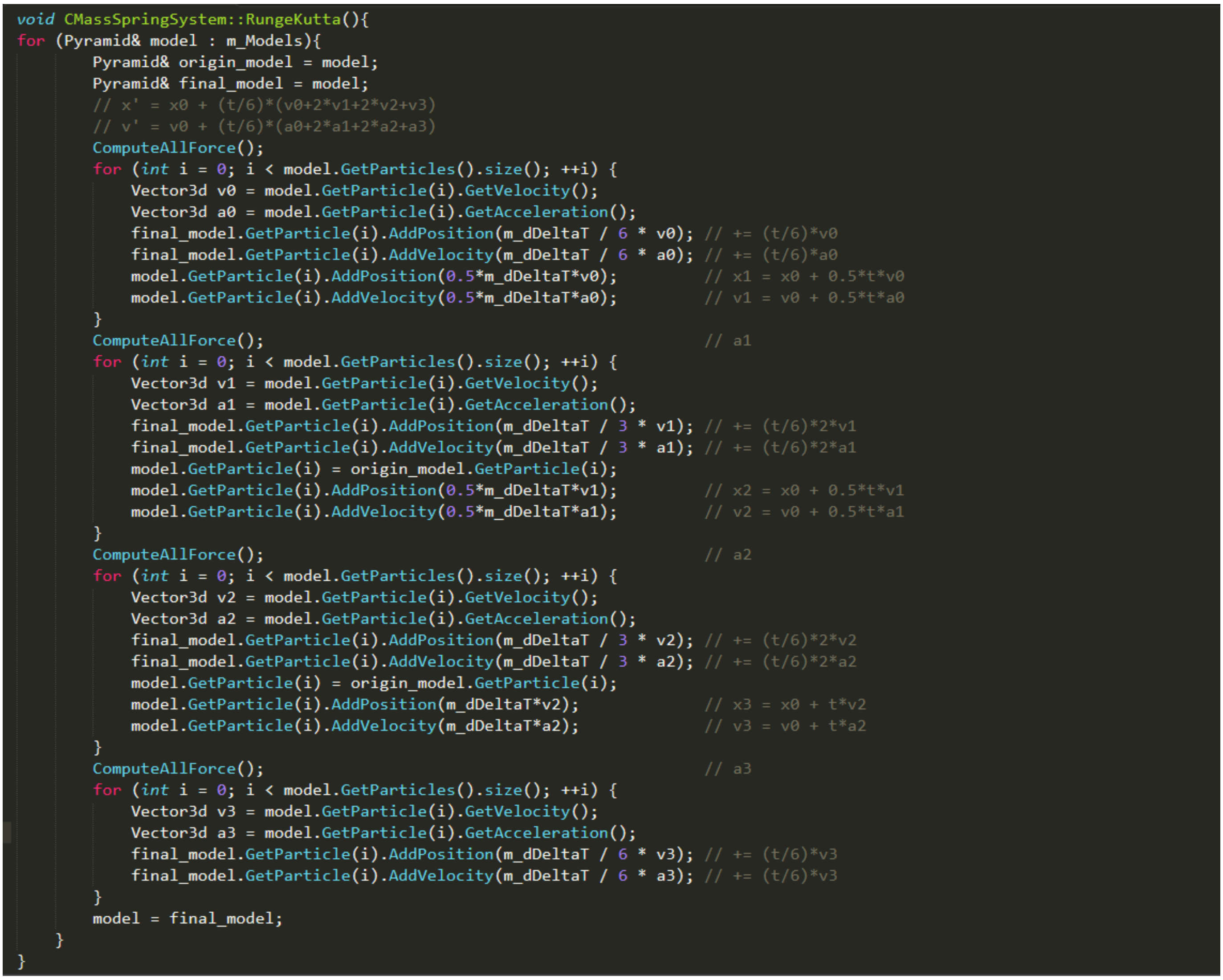
0456024 蔡佩珊

CMassSpringSystem::ExplicitEuler()



```
void CMassSpringSystem::ExplicitEuler(){

    for (Pyramid& model : m_Models){
        for (CParticle& particle : model.GetParticles()){
            // deltaT: time step
            // x' = x + deltaT*v
            // v' = v + deltaT*a
            particle.AddPosition(m_dDeltaT*particle.GetVelocity());
            particle.AddVelocity(m_dDeltaT*particle.GetAcceleration());
        }
    }
}
```

CMassSpringSystem::RungeKutta()



```
void CMassSpringSystem::RungeKutta(){
for (Pyramid& model : m_Models){
        Pyramid& origin_model = model;
        Pyramid& final_model = model;
        // x' = x0 + (t/6)*(v0+2*v1+2*v2+v3)
        // v' = v0 + (t/6)*(a0+2*a1+2*a2+a3)
        ComputeAllForce();
        for (int i = 0; i < model.GetParticles().size(); ++i) {
            Vector3d v0 = model.GetParticle(i).GetVelocity();
            Vector3d a0 = model.GetParticle(i).GetAcceleration();
            final_model.GetParticle(i).AddPosition(m_dDeltaT / 6 * v0); // += (t/6)*v0
            final_model.GetParticle(i).AddVelocity(m_dDeltaT / 6 * a0); // += (t/6)*a0
            model.GetParticle(i).AddPosition(0.5*m_dDeltaT*v0);         // x1 = x0 + 0.5*t*v0
            model.GetParticle(i).AddVelocity(0.5*m_dDeltaT*a0);         // v1 = v0 + 0.5*t*a0
        }
        ComputeAllForce();                                              // a1
        for (int i = 0; i < model.GetParticles().size(); ++i) {
            Vector3d v1 = model.GetParticle(i).GetVelocity();
            Vector3d a1 = model.GetParticle(i).GetAcceleration();
            final_model.GetParticle(i).AddPosition(m_dDeltaT / 3 * v1); // += (t/6)*2*v1
            final_model.GetParticle(i).AddVelocity(m_dDeltaT / 3 * a1); // += (t/6)*2*a1
            model.GetParticle(i) = origin_model.GetParticle(i);
            model.GetParticle(i).AddPosition(0.5*m_dDeltaT*v1);         // x2 = x0 + 0.5*t*v1
            model.GetParticle(i).AddVelocity(0.5*m_dDeltaT*a1);         // v2 = v0 + 0.5*t*a1
        }
        ComputeAllForce();                                              // a2
        for (int i = 0; i < model.GetParticles().size(); ++i) {
            Vector3d v2 = model.GetParticle(i).GetVelocity();
            Vector3d a2 = model.GetParticle(i).GetAcceleration();
            final_model.GetParticle(i).AddPosition(m_dDeltaT / 3 * v2); // += (t/6)*2*v2
            final_model.GetParticle(i).AddVelocity(m_dDeltaT / 3 * a2); // += (t/6)*2*a2
            model.GetParticle(i) = origin_model.GetParticle(i);
            model.GetParticle(i).AddPosition(m_dDeltaT*v2);             // x3 = x0 + t*v2
            model.GetParticle(i).AddVelocity(m_dDeltaT*a2);             // v3 = v0 + t*a2
        }
        ComputeAllForce();                                              // a3
        for (int i = 0; i < model.GetParticles().size(); ++i) {
            Vector3d v3 = model.GetParticle(i).GetVelocity();
            Vector3d a3 = model.GetParticle(i).GetAcceleration();
            final_model.GetParticle(i).AddPosition(m_dDeltaT / 6 * v3); // += (t/6)*v3
            final_model.GetParticle(i).AddVelocity(m_dDeltaT / 6 * a3); // += (t/6)*v3
        }
        model = final_model;
    }
}
```

♦ Result and Discussion

● the difference between Explicit Euler and RK4

Explicit Euler 的粒子位置改變量以單一速度決定，速度改變量以單一加速度決定

RK4 的粒子位置改變量以四個不同階段的速度決定，速度改變量以四個不同階段的加速度決定

➔ RK4 物體運動較順暢、模擬到最後物體會傾倒，因此真實性較高

➔ EE 計算量較小，適用於粒子數多且有即時性需求的情況

● effect of coefficient of spring force k_s

$F_s = -k_s \Delta x$

k_s 越大，彈簧越不容易形變，物體碰撞反彈較大

k_s 越小，彈簧越容易形變且難以恢復原狀，物體碰撞反彈較小

● effect of coefficient of damper force k_d

$F_d = -k_d \Delta v$

k_d 越大，彈簧形變後恢復較慢，物體碰撞反彈較小

k_d 越小，彈簧形變後恢復較快，物體碰撞反彈較大