Cloth Simulation 기술문서

목차

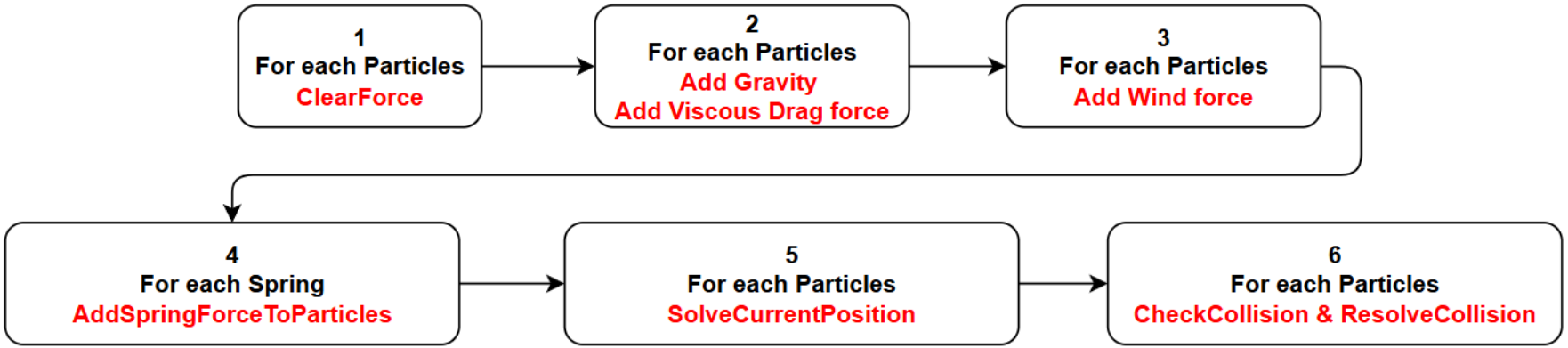
개요

- **개발 언어 :** **C++**
- **라이브러리 :** **WinAPI, DirectX11**
- **제작자 :** **김동현**
- **Video :** https://youtu.be/Ysl-bq_JyEQ
- **Github :** https://github.com/DongHyun96/DX3D_ClothSimulation

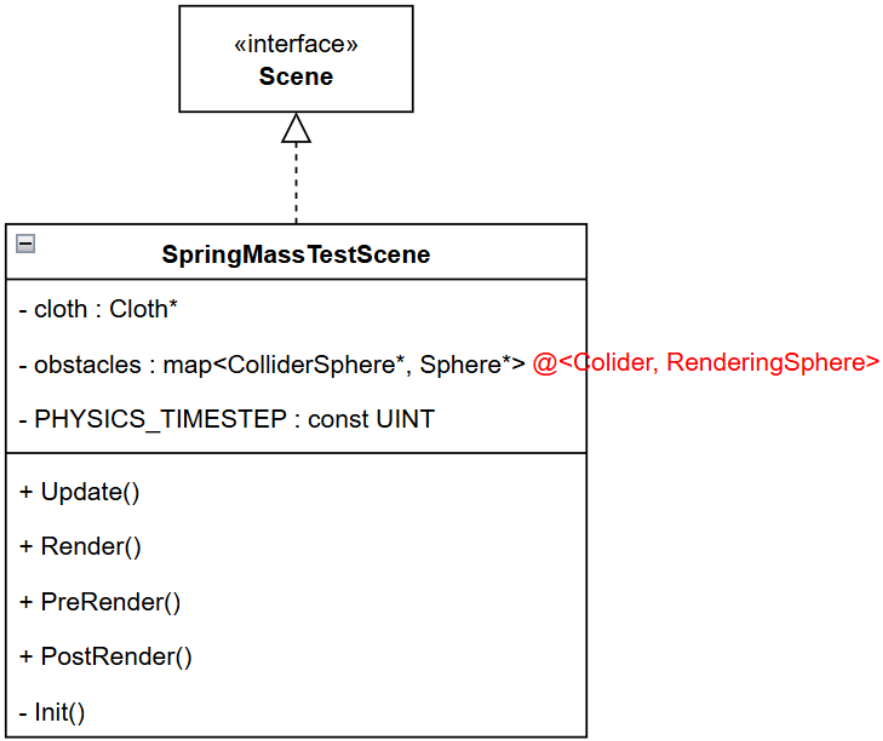
Update Tick Pipeline & Architecture

Update Tick Pipeline

한 Update Tick 내에서 다음과 같은 일련의 과정을 실행

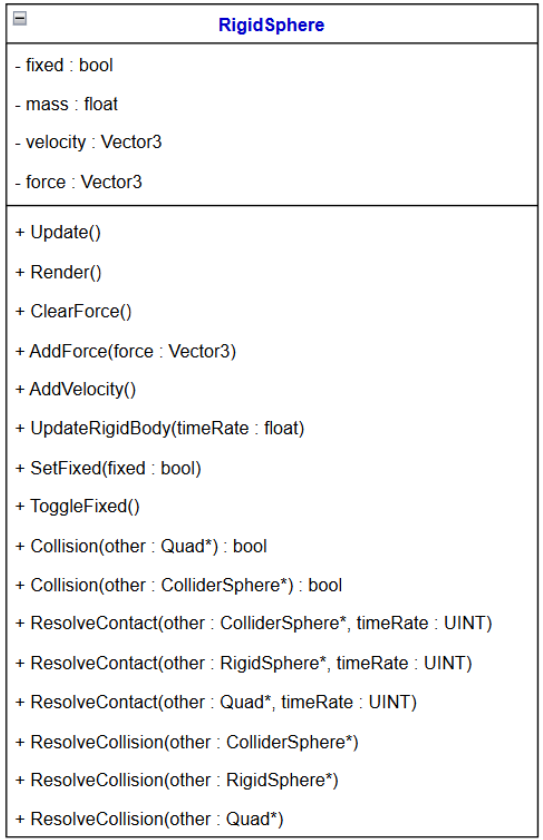


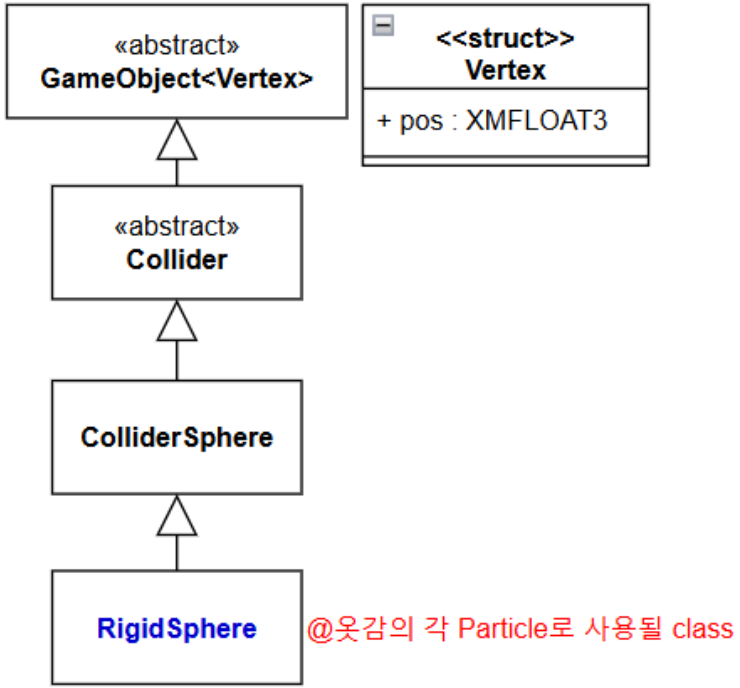
Architecture



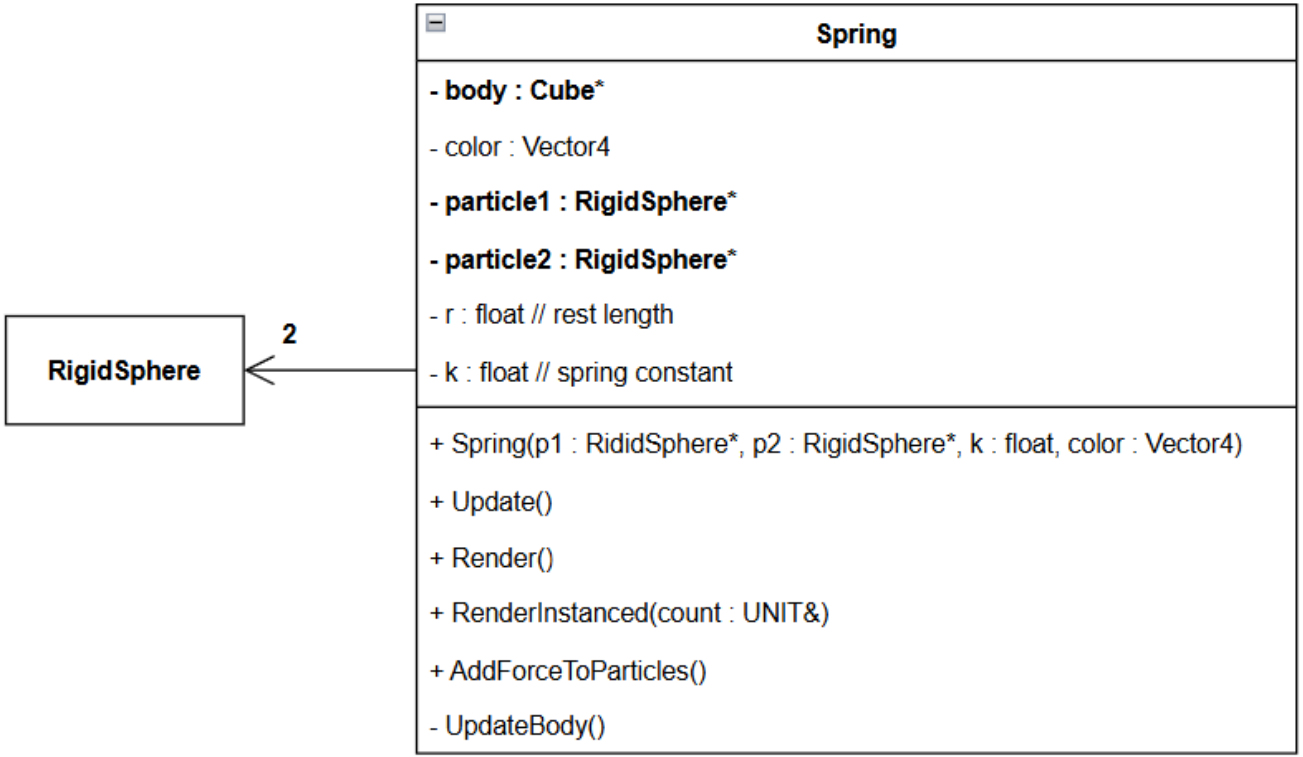
- **Main Scene**

Architecture

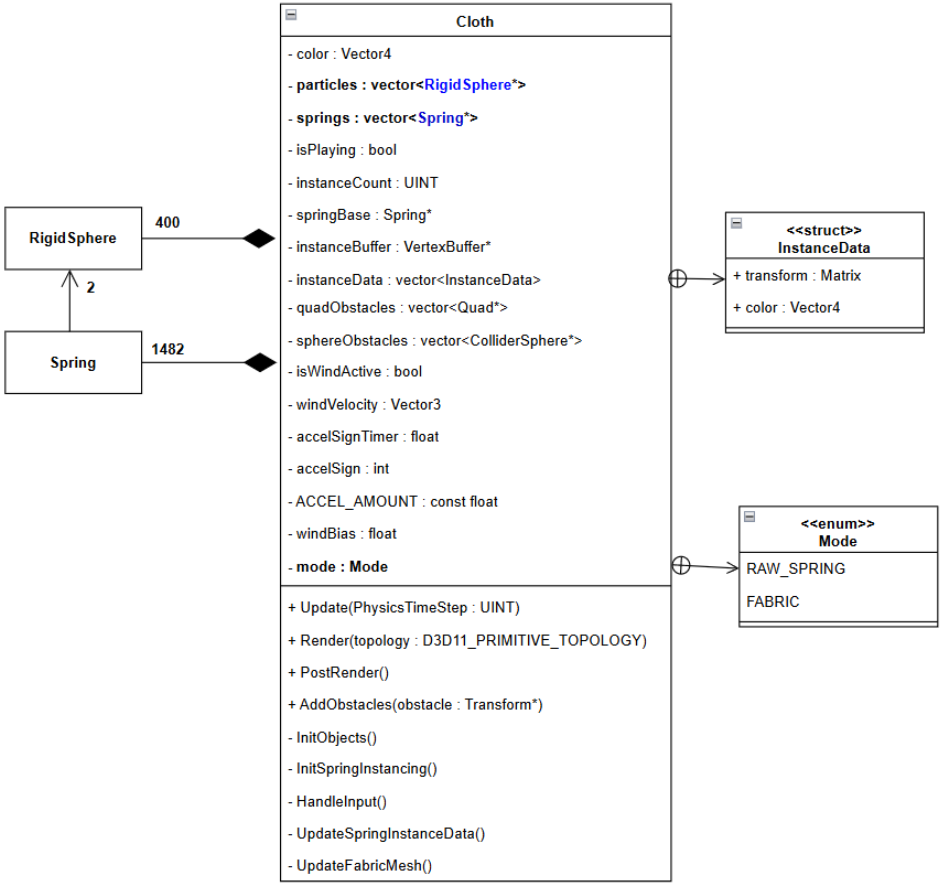




Architecture

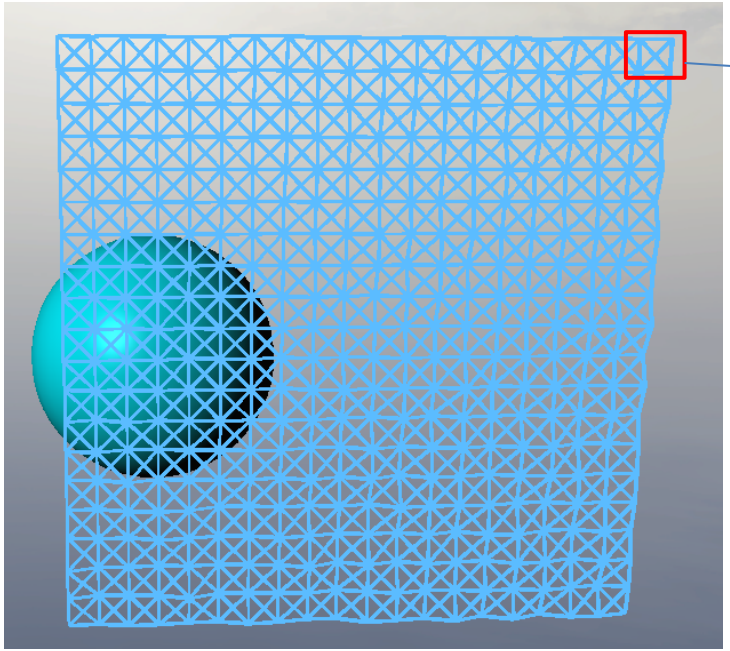


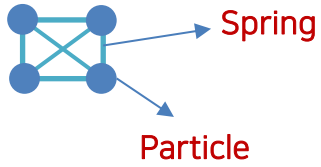
Architecture



Spring Hooke’s Law

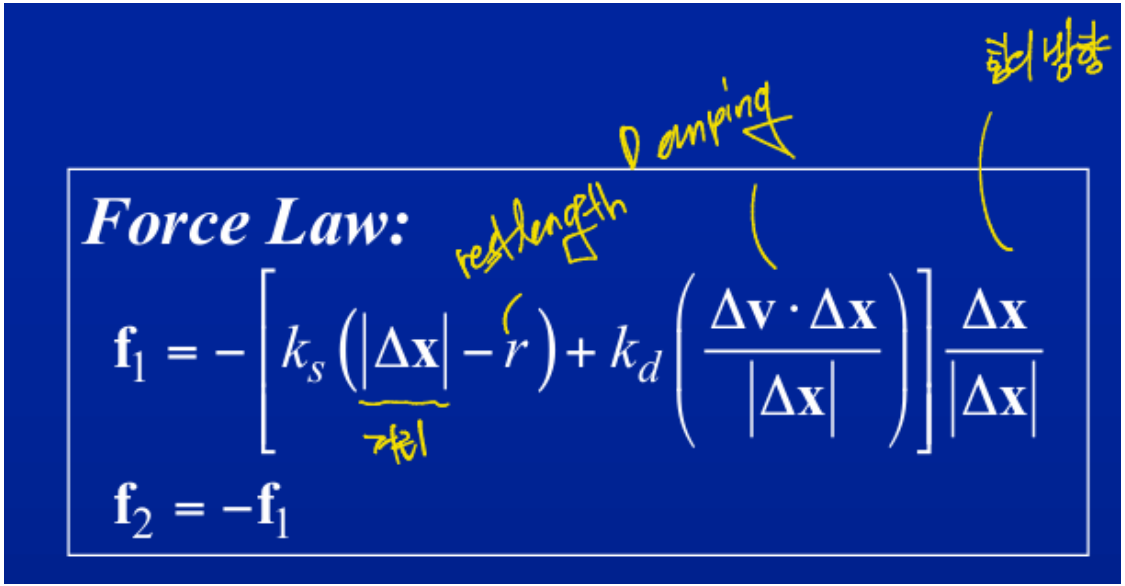
Cloth's springs and particles





- 2개의 옷감 Particle을 연결하는 spring으로 그림과 같은 옷감 구조를 생성
- Hooke's Law를 사용하여 각 Spring은 두 입자의 매 update tick에 spring force를 더함

Using Damped Spring Force Law



Spring Hooke's Law

Implementation

```
void Spring::AddForceToParticles()
{
    // Using spring force law
    Vector3 p1_to_p2 = particle2->GetGlobalPosition() - particle1->GetGlobalPosition();

    Vector3 F = -(k * (p1_to_p2.Length() - r) +
                K_D * Vector3::Dot(particle2->GetVelocity() - particle1->GetVelocity(), p1_to_p2.GetNormalized())) *
                (p1_to_p2).GetNormalized();

    particle1->AddForce(-F);
    particle2->AddForce(F);
}
```

Differential Equation Solver

Euler Method / Runge-Kutta Method

Differential Equation : f = ma

- A Newtonian Particle
- f = ma 미분방정식을 통해 충돌검사 및 처리를 하기 전, 현재 update tick 시점의 옷감 particle의 velocity 및 position을 구함
- Method 1 : Euler Method Integration
- Method 2 : Runge-Kutta Integration

Method1 : Euler Method

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t\,\mathbf{f}(\mathbf{x}, t)$$

- x(t) = t 시점에서의 position
- f(x, t) = t 시점에서의 velocity

Problem : Descrete한 delta time으로 인한 오차 발생
DeltaTime이 클수록 오차가 커지는 문제점이 있음

Euler Method 오차 해결방안

- Euler Method with better stability
- Update 한 tick 내에서 TimeStep으로 쪼갠 DeltaTime을 사용

```
void RigidSphere::SolveCurrentPosition(const UINT& timeStep)
{
    if (fixed)
    {
        velocity = Vector3();
        return;
    }

    Vector3 forcePerMass = force / mass;
    float dt             = DELTA_TIME / timeStep;

    // Euler integration - translation(t + dt) = translation(t) + curVelocity * dt;
    // dt를 최대한 줄임으로써(timeStep으로) 오차 줄이기 시도
    velocity    += forcePerMass * dt;
    translation += velocity     * dt;
}
```

<문제점>

- 안정성은 확보, 하지만 프레임 방어가 어려운 등 현실적으로 사용하기 어려웠음
- timeStep값을 너무 높이면 오히려 frame drop이 생겨 dt가 더 증가하여 오차가 커질 수 있음

Method2 : 4th order Runge-Kutta Method

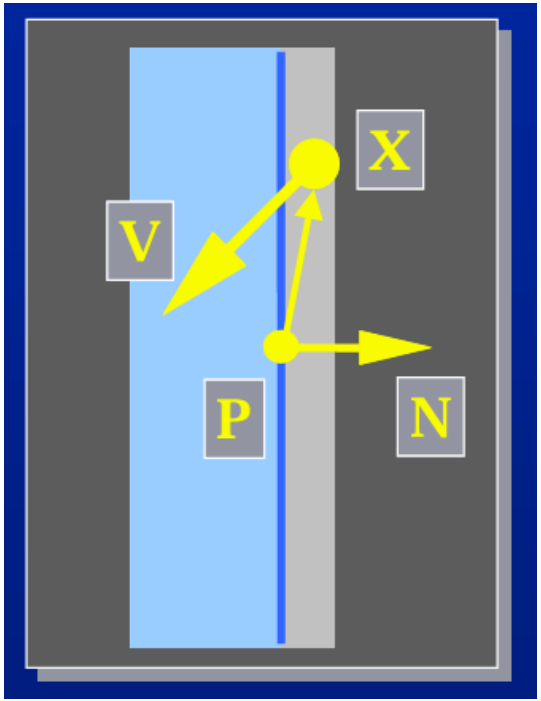
```
void RigidSphere::SolveCurrentPosition(const UINT& timeStep)
{
    if (fixed)
    {
        velocity = Vector3();
        return;
    }

    Vector3 forcePerMass = force / mass;
    float dt             = DELTA_TIME / timeStep;

    // Runge-Kutta integration
    Vector3 k1 = forcePerMass * dt;
    Vector3 k2 = (forcePerMass + 0.5f * k1) * dt;
    Vector3 k3 = (forcePerMass + 0.5f * k2) * dt;
    Vector3 k4 = (forcePerMass + k3) * dt;

    velocity    += (k1 + 2.f * k2 + 2.f * k3 + k4) / 6.f;
    translation += velocity * dt;

}
```

- 가속도의 변화 패턴(또는 가속도를 미분한 속도값)의 변화 패턴을 단계별로 추정한 후, 이를 가중 평균하여 최종 해를 구하는 방식
- dt가 커도 오차를 최소화하는데 효과적
- 최종적으로 해당 방식으로 Position을 구함

Detecting Collision

Particle vs Quad / Particle vs Sphere

Particle vs Quad(바닥면) Collision Detection



$$(X - P) \cdot N < \varepsilon$$

$$N \cdot V < 0$$

- P = Quad의 Position
- N = Quad의 Normal vector
- X = Particle의 Position
- V = Particle의 Velocity

- 위의 두 조건을 동시에 만족했을 때 Quad와 Particle이 충돌했다고 판단

Particle vs Quad Implementation

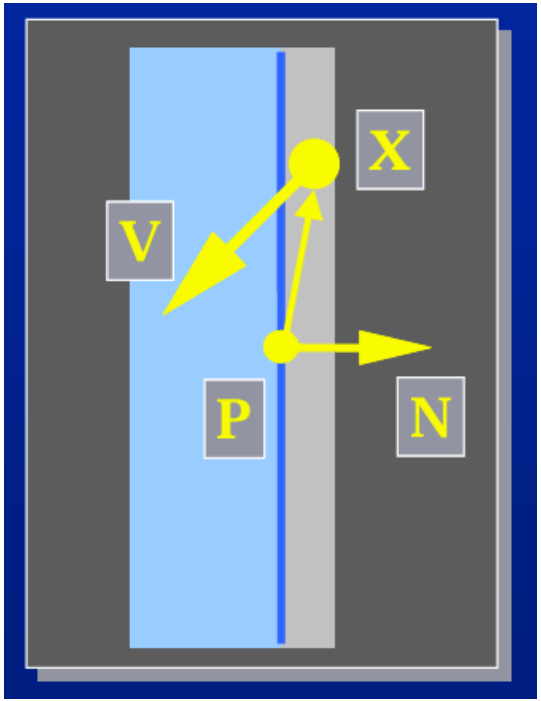
```
bool RigidSphere::Collision(const Quad* other)
{
    return Vector3::Dot(globalPosition - other->GetGlobalPosition(), other->GetNormal()) < 0.01f &&
           Vector3::Dot(velocity, other->GetNormal()) < 0;
}
```

N · V　　X − P　　N　　Epsilon

$$(X - P) \cdot N < \varepsilon$$

$$N \cdot V < 0$$

Particle vs Sphere Collision Detection



$$(X - P) \cdot N < \varepsilon$$

$$N \cdot V < 0$$

- P = Sphere 중점으로부터 Particle로 향하는 방향에 맞닿는 Sphere위의 점
- N = Sphere 중점으로부터 Particle로 향하는 방향 vector
- X = Particle의 Position
- V = Particle의 Velocity

- 위의 두 조건을 동시에 만족했을 때 Sphere 와 Particle이 충돌했다고 판단

Detecting Collision

Particle vs Sphere Implementation

$$(X - P) \cdot N < \varepsilon$$

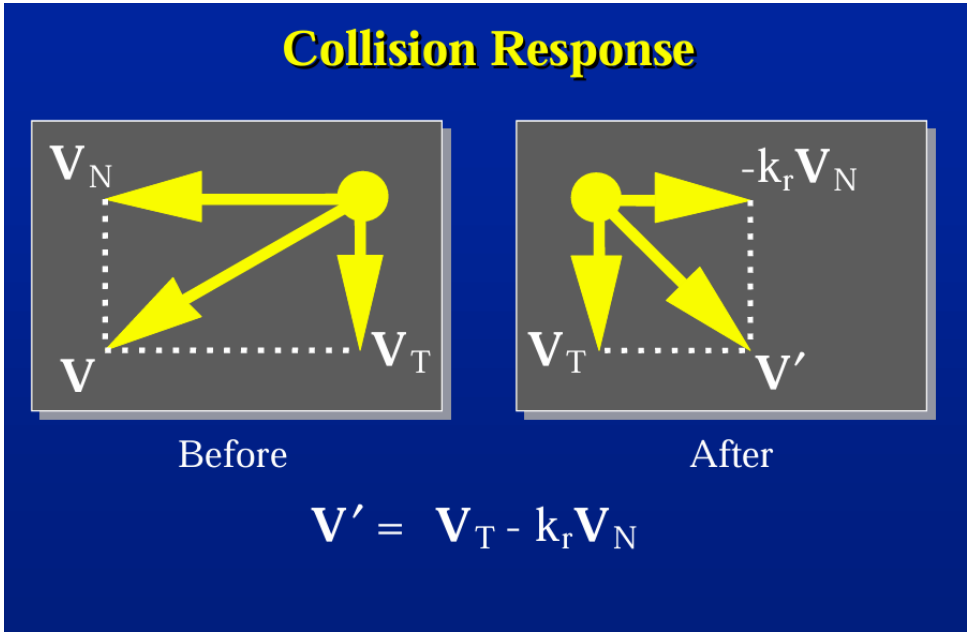
$$N \cdot V < 0$$

```
bool RigidSphere::Collision(const ColliderSphere* other)
{
    // z fighting 때문에 radius에 여유를 둠
    float otherRadius = other->Radius() + SPHERE_COLLISION_MARGIN;

    // pos = ColliderSphere로 부터 RigidSphere로 향하는 방향에 맞닿는 ColliderSphere위의 점
    Vector3 pos = other->GetGlobalPosition() + (this->globalPosition - other->GetGlobalPosition()).GetNormalized() * otherRadius;
    Vector3 n   = (this->globalPosition - other->GetGlobalPosition()).GetNormalized();

    return Vector3::Dot(this->globalPosition - pos, n) < 0.01f && Vector3::Dot(velocity, n) < 0;
}
```

Collision Handling

Particle vs Quad / Particle vs Sphere

Resolve Collision



- v′ = Update된 velocity
- kr = 반발계수

- 충돌처리는 Velocity Update와 particle이 충돌체를 관통하지 않도록 particle의 위치 조정을 함

Resolve Collision Implementation

- **Particle vs Quad ResolveCollision**

```
void RigidSphere::ResolveCollision(const Quad* other)
{
    // Velocity update
    Vector3 vN = Vector3::Dot(velocity, other->GetNormal()) * other->GetNormal();
    Vector3 vT = velocity - vN;
    velocity   = vT - vN * COR;

    // Translation 보정
    translation -= Vector3::Dot(translation - other->GetGlobalPosition(), other->GetNormal()) * other->GetNormal();
}
```

Coefficient of restitution(반발계수) = kr

Resolve Collision Implementation

- **Particle vs sphere ResolveCollision**

```
void RigidSphere::ResolveCollision(const ColliderSphere* other)
{
    // z fighting 문제로 실질적인 radius에 약간의 margin을 더함
    float otherRadius = other->Radius() + SPHERE_COLLISION_MARGIN;

    // other sphere 위치로부터 나 자신으로 향하는 방향
    Vector3 n = (this->globalPosition - other->GetGlobalPosition()).GetNormalized();

    // Velocity update
    Vector3 vN  = Vector3::Dot(velocity, n) * n;
    Vector3 vT  = velocity - vN;
    velocity    = vT - vN * COR;

    // Translation 보정
    Vector3 contactVec = otherRadius * n;
    Vector3 contactPos = other->GetGlobalPosition() + contactVec;
    this->translation -= Vector3::Dot(this->translation - contactPos, n) * n;
}
```