Collision Detection and Response

KOREA UNIVERSITY

KUCG

In This Lecture

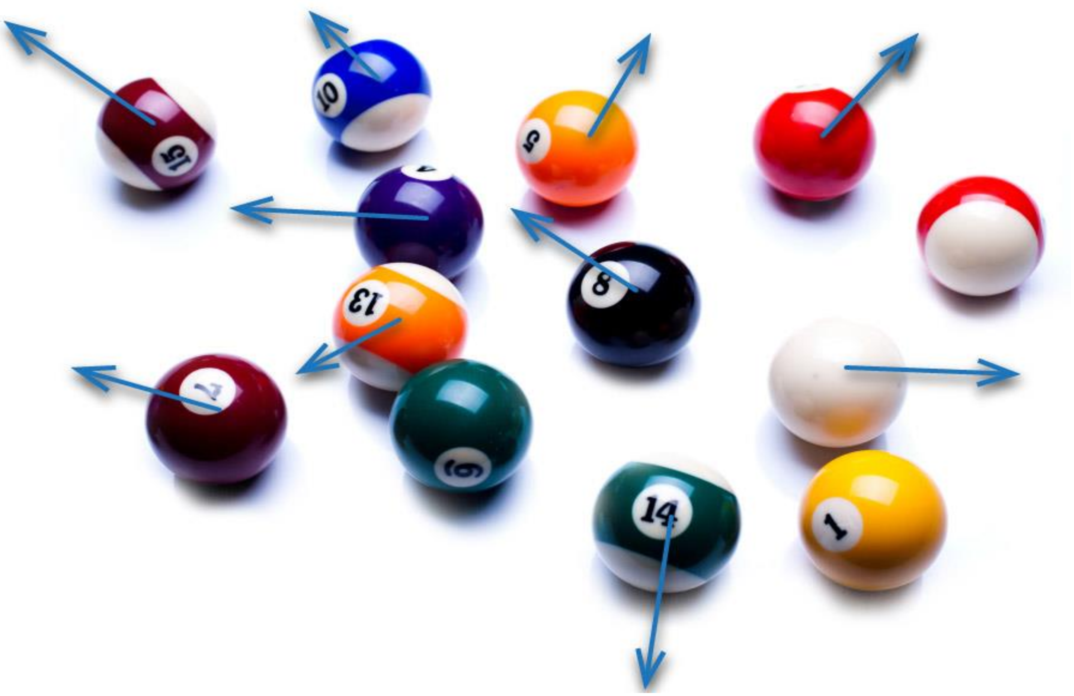
- Introduce of Collision and Conservation

- Collision Detection and Response
  - Particle-Particle
  - Particle-Plane
  - Restitution

- Hash Grid
  - Algorithm
  - Generate Hash Table
  - Get Neighbor Particles

What is a Collision?

- An interaction between two or more bodies in motion is a collision



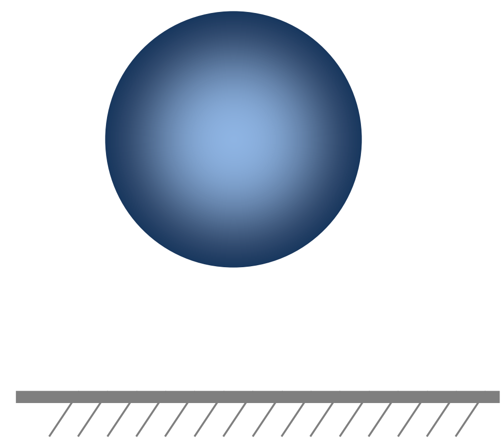
Pool balls bouncing off of each other is one example.

Types of Collisions

Particle-Particle

Collision

Particle-Plane



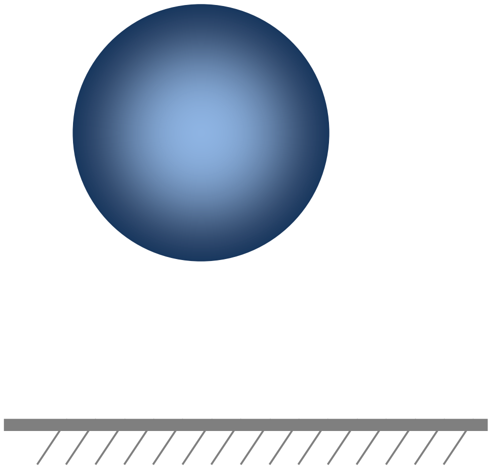
Collision

Collision Response

Particle-Particle

Collision->Response

Particle-Plane

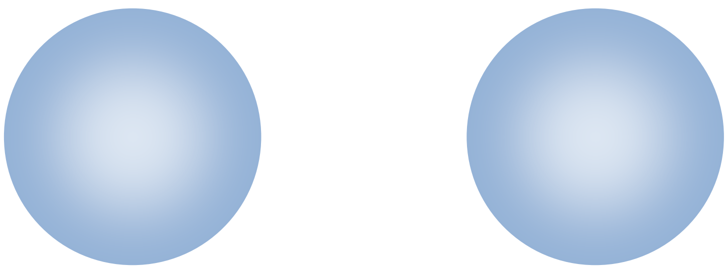


Collision->Response

Restitution

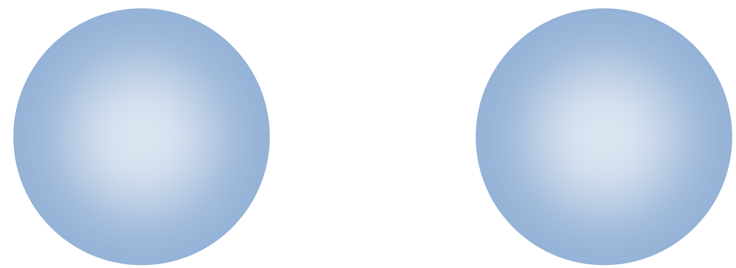
- When two objects collide
  - Their speeds after the collision depend on the their **material**
- **How to denote the material?**
  - Using **Coefficient of Restitution**

Perfectly elastic collision



Coefficient of Restitution=1

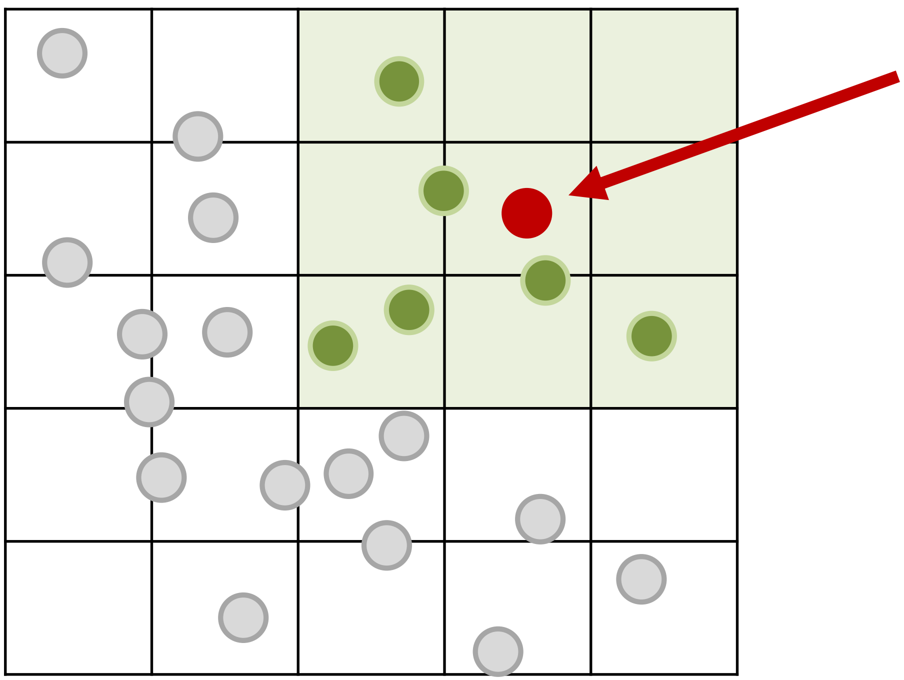
Perfectly inelastic collision



Coefficient of Restitution=0

Advanced Algorithm: Hash Grid

- Hash Grid is an accelerate method
  - Efficiently for collision detection between objects
- Commonly used for large set of particles



If check the collision for the red particle

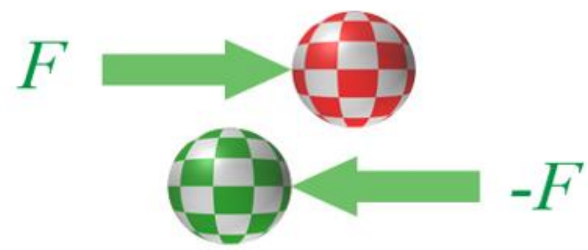
➢ Only check the neighbor particles(green) of it

Introduce of Collision and Conservation

What Happens in a Collsion?

- What is happening in this collision between two balls?
- What might happen next?
  - Velocity change

How Do the Velocities Change?

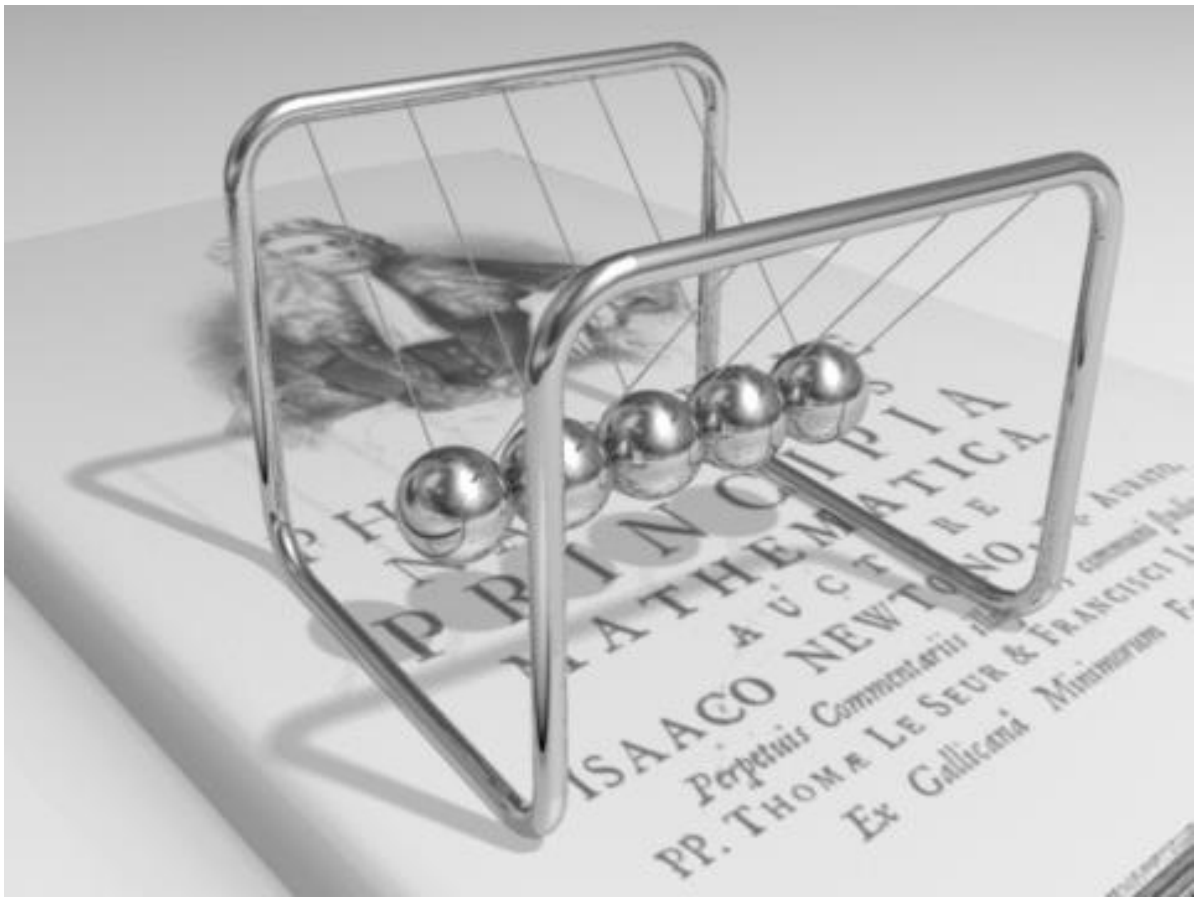


- Every collision involved forces
  - Ex. In the collision pictured above, the green ball exerts a force F on the red ball. By Newton's third law, the red ball exerts an equal and opposite force on the green ball.
- No external forces act on the system, and the internal forces cancel each other.

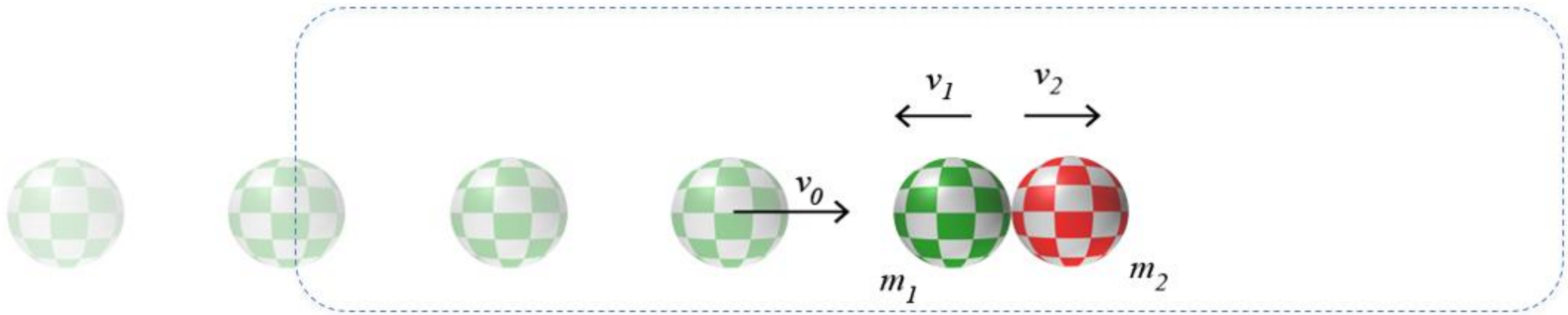
So what is conserved?

What is Conserved?

Newton's Cradle



Conservation Laws



- Momentum:
  - The momentum before the collision equals the momentum after the collision
- Energy:
  - The energy before the collision equals the energy after the collision. But the energy may be transformed
- Momentum if conserved in all three types(**Perfectly inelastic, Inelastic, Elastic**) of collisions

Collision Detection and Response:

Particle-Particle

Particle-particle Collision and Response

Collision Algorithm

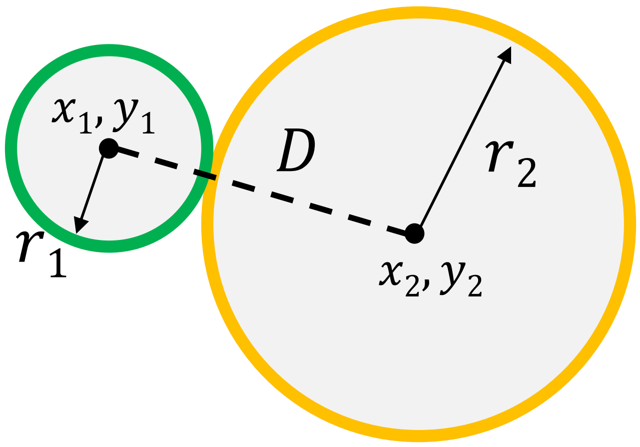
- **For each particle i**
  - **For each other particle j**
    - **if** ((radius_i + radius_j)- distance(i , j)<0)

      {
      - Collision has detected
      - Compute the velocity vectors after collision

      }

Detect that a Collision Occurred

- If the distance between two particles is less than the sum of their radii
  - checking ***(r1+r2) – D < 0***, where
    - **D = sqrt($(x_1$-$x_2)^2$ + $(y_1$-$y_2)^2$)**



Response Algorithm

- **if** (Collision has detected)

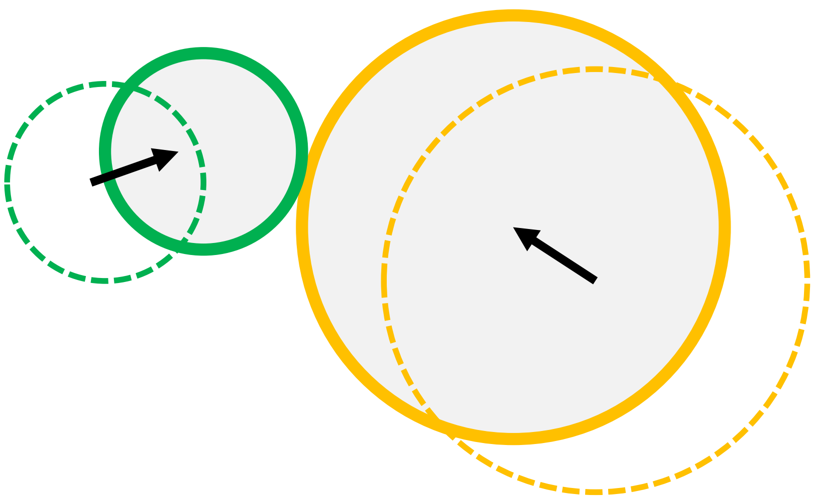
{

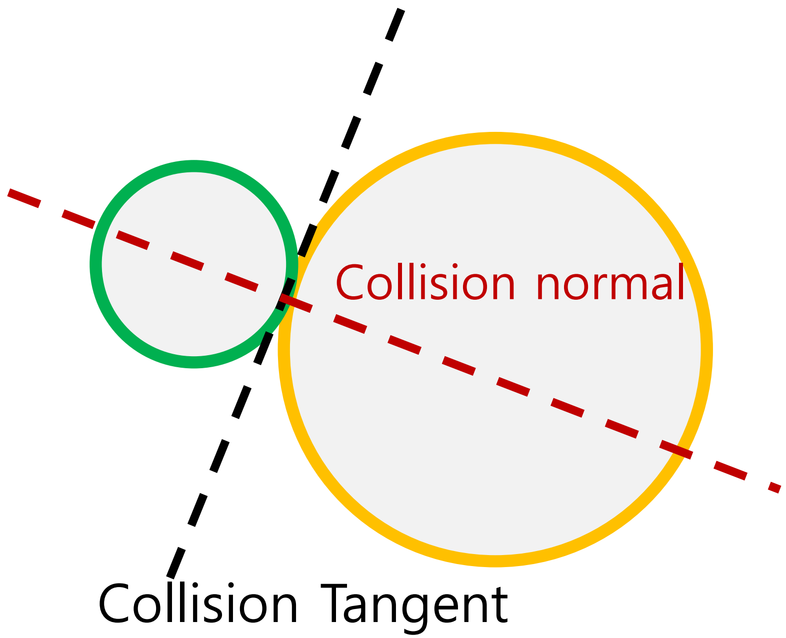
  - Collision Normal
    - Compute the difference between the particle's centers, then normalization
  - Velocity Change
    - Compute the response velocity
  - Solve the Trap Problem
    - Solve the problem if particle trapped each other
  - Types of Restitution
    - Decision the collision is elastic or inelastic

}

Collision Normal & Tangent

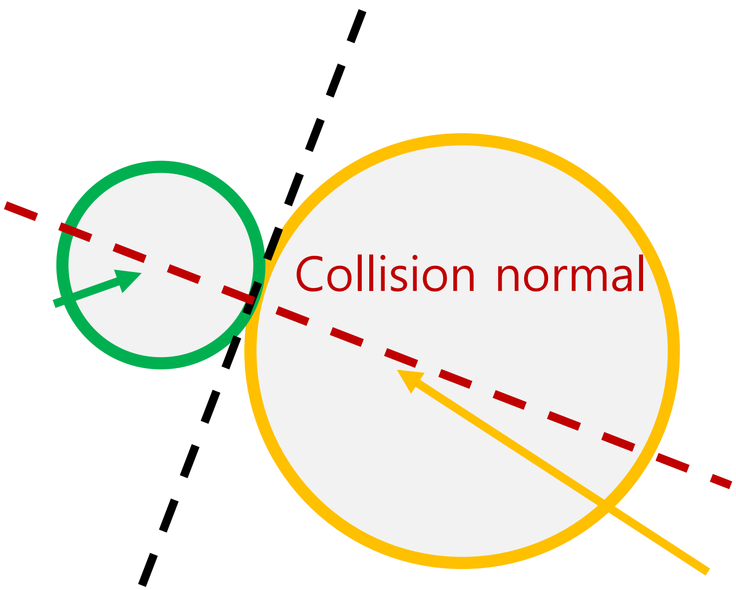
- Determine the collision normal
  - Bisects the centers of the two Particles through the colliding intersection



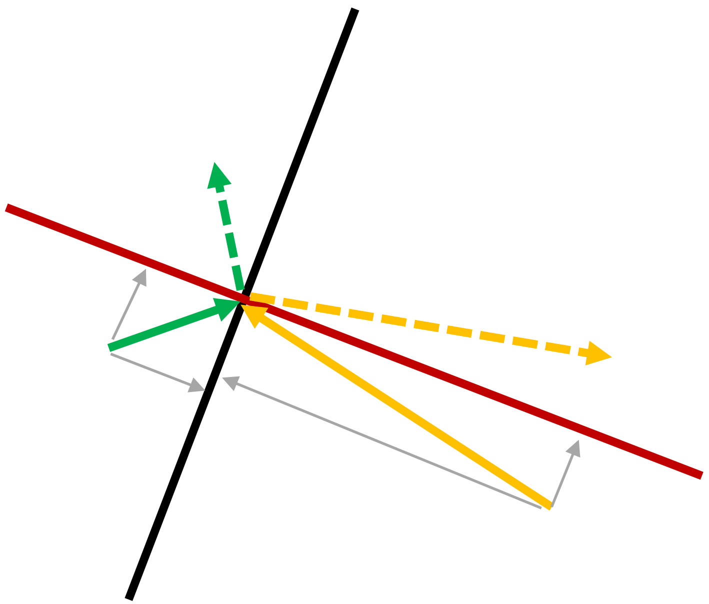


Response Velocity

- Velocity change:
  - Change of velocity reflect against the collision normal



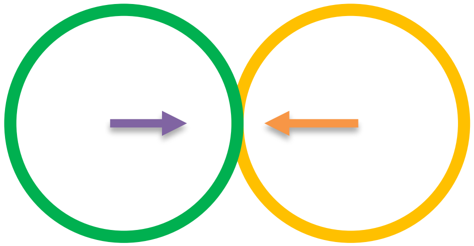
Collision Tangent



Response Velocity Vectors

Head on Collision Response

- Determine the velocity
  - assume elastic, no friction,
  - **head on collision**



- Conservation of Momentum (mass * velocity):
  - $m_1\mathbf{v}_1 + m_2\mathbf{v}_2 = m_1\mathbf{v}'_1 + m_2\mathbf{v}'_2$
- Conservation of Energy (Kinetic Energy):
  - $m_1{\mathbf{v}_1}^2 + m_2{\mathbf{v}_2}^2 = m_1{\mathbf{v}'_1}^2 + m_2{\mathbf{v}'_2}^2$

- **Final Velocities**
  - $\mathbf{v}_1' = \frac{2m_2\mathbf{v}_2 + (m_1 - m_2)\mathbf{v}_1}{m_1 + m_2}$
  - $\mathbf{v}_2' = \frac{2m_1\mathbf{v}_1 + (m_2 - m_1)\mathbf{v}_2}{m_1 + m_2}$

질량이 같은 경우에는 v'1 = v2, v'2 = v1이 됨. 계산해보셈

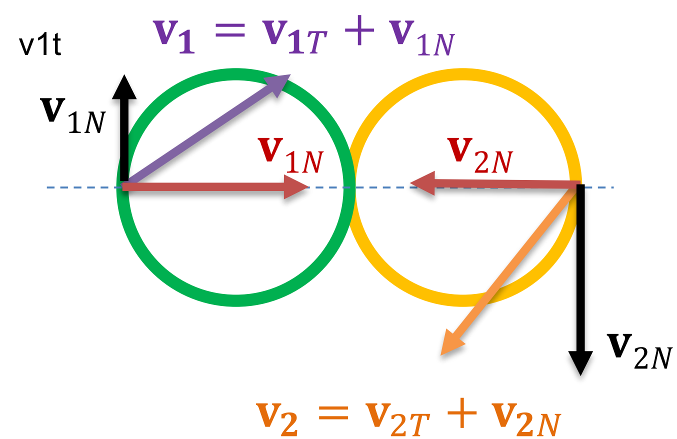
Detail of Compute Final velocity

- (1): $m_1\mathbf{v}_1 + m_2\mathbf{v}_2 = m_1\mathbf{v'}_1 + m_2\mathbf{v'}_2$
- (2): $m_1\mathbf{v}_1{}^2 + m_2\mathbf{v}_2{}^2 = m_1\mathbf{v'}_1{}^2 + m_2\mathbf{v'}_2{}^2$
- Because (1) =>
  - (3): $m_1(\mathbf{v}_1 - \mathbf{v'}_1) = m_2(\mathbf{v}_2 - \mathbf{v'}_2)$
- Because (2) =>
  - (4): $m_1\left(\mathrm{v}_1{}^2 - \mathrm{v'}_1{}^2\right) = m_2\left(\mathrm{v'}_2{}^2 - \mathrm{v}_2{}^2\right)$
- Let x^2-$y^2 = (x-y)(x+y)$
  - $m_1(\mathbf{v}_1 - \mathbf{v'}_1)\ (\mathbf{v}_1 + \mathbf{v'}_1) = m_2(\mathbf{v'}_2 - \mathbf{v}_2)\ (\mathbf{v'}_2 + \mathbf{v}_2)$
- Let $\frac{(3)}{(4)}$ =>
  - (5): $\mathrm{v'}_1 = \mathrm{v'}_2 + \mathrm{v}_2 - \mathrm{v}_1$
  - (6): $\mathrm{v'}_2 = \mathrm{v}_1 + \mathrm{v'}_1 - \mathrm{v}_2$
- (5)(6)번 식을 (1)번식에 대입 :
  - $\mathbf{v'}_1 = \frac{2m_2\mathbf{v}_2 + (m_1 - m_2)\mathbf{v}_1}{m_1 + m_2}$
  - $\mathbf{v'}_2 = \frac{2m_1\mathbf{v}_1 + (m_2 - m_1)\mathbf{v}_2}{m_1 + m_2}$

Arbitrary Collision Response

- Determine the velocity
  - assume elastic, no friction
  - **arbitrary collision**

vn = 두 파티클의 센터를 빼면 나오고
vt = v-vn



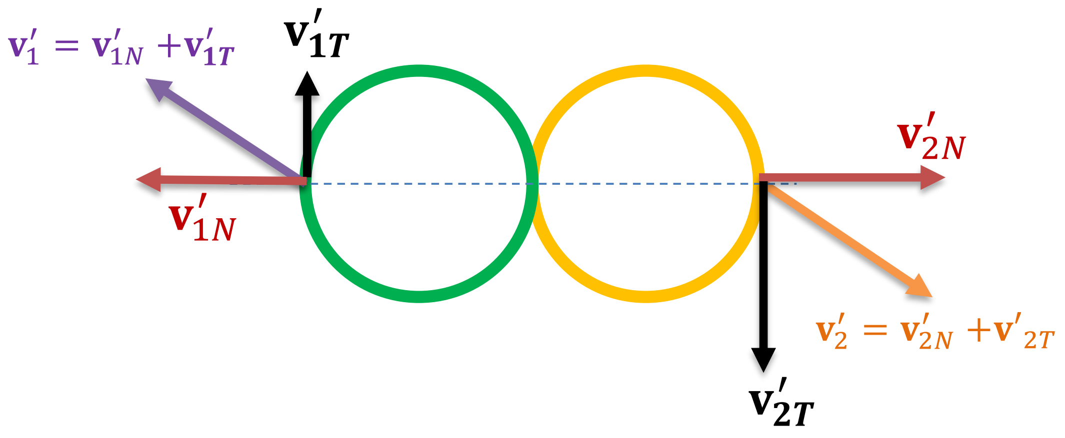
- **Velocity Decomposition**
  - $\mathbf{v}_N = (\mathbf{N} \cdot \mathbf{v})\mathbf{N}$
    - **N is collision normal.**
  - $\mathbf{v}_T = (\mathbf{T} \cdot \mathbf{v})\mathbf{T}$ **or** $\mathbf{v} - \mathbf{v}_N$
    - **T is collision tangent.**

Arbitrary Collision Response

Final Velocities

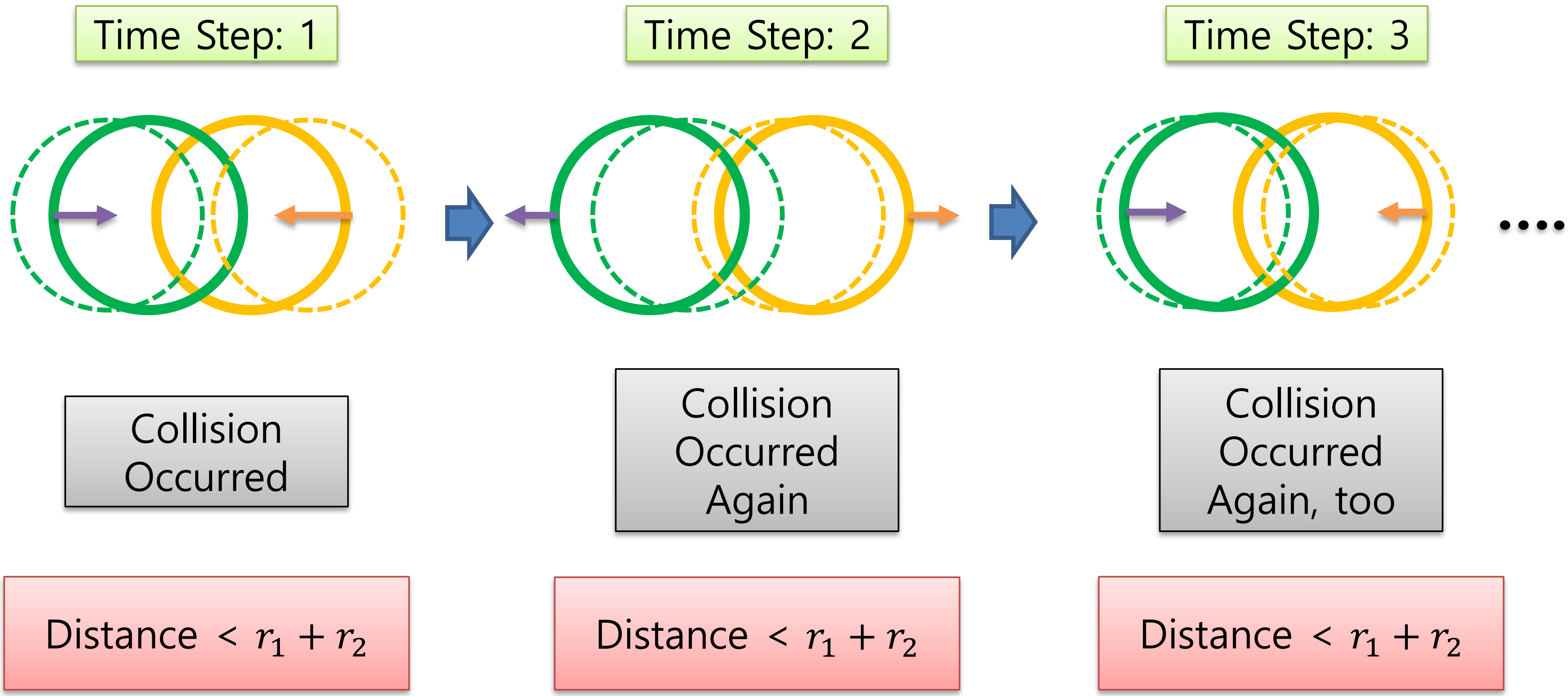
- Final Velocities
  - $\mathbf{v}'_{1N} = \frac{2m_2\mathbf{v}_{2N}+(m_1-m_2)\mathbf{v}_{1N}}{m_1+m_2}$
  - $\mathbf{v}'_{2N} = \frac{2m_1\mathbf{v}_{1N}+(m_2-m_1)\mathbf{v}_{2N}}{m_1+m_2}$
  - $\mathbf{v}'_{1T} = \mathbf{v}_{1T}$
  - $\mathbf{v}'_{2T} = \mathbf{v}_{2T}$

  - $\mathbf{v}'_1 = \mathbf{v}'_{1N} + \mathbf{v}'_{1T}$
  - $\mathbf{v}'_2 = \mathbf{v}'_{2N} + \mathbf{v}'_{2T}$



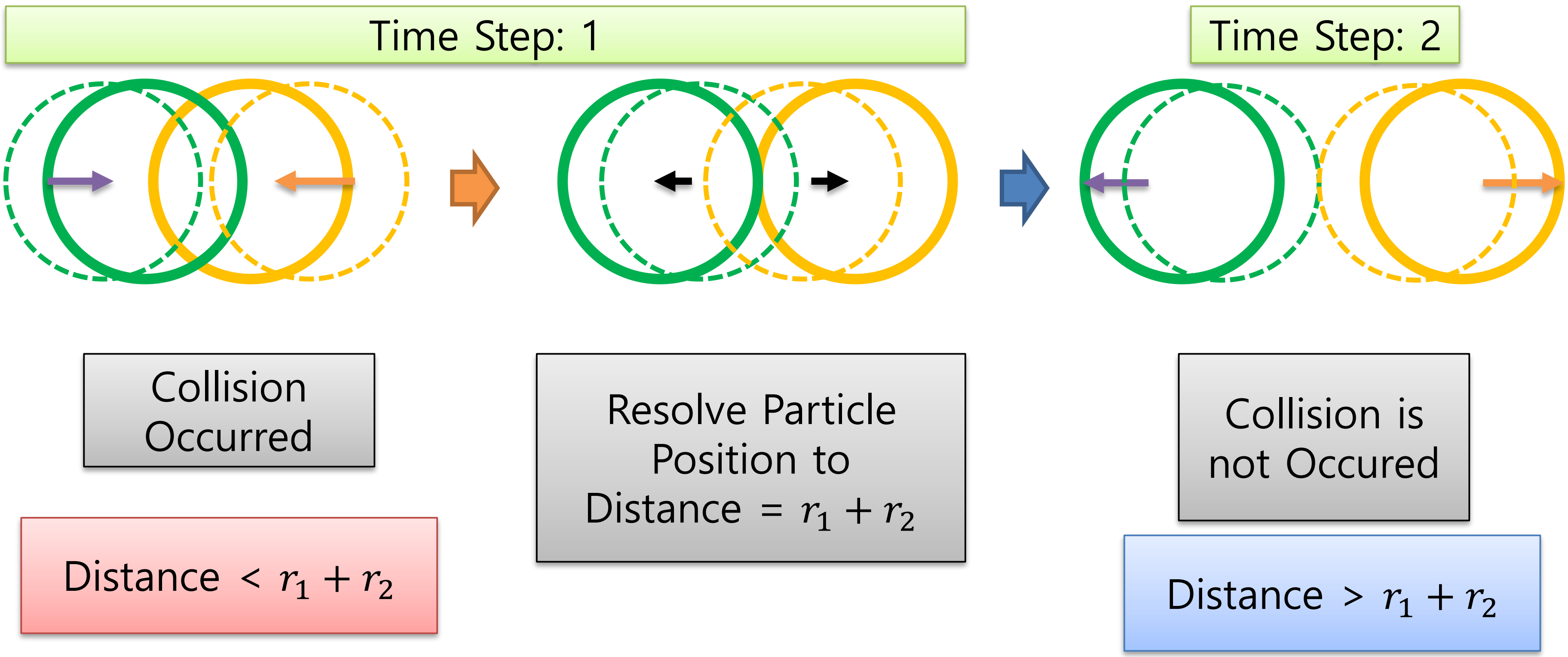
Trapped Problem

- When Collision has occurred, both particles can be trapped each other



Trapped Problem: Solution 1

- When Collision has occurred, resolve particle position



- This Solution is similar to real world, but it's difficult way to resolve position with **many collisions occurred at the same time.**

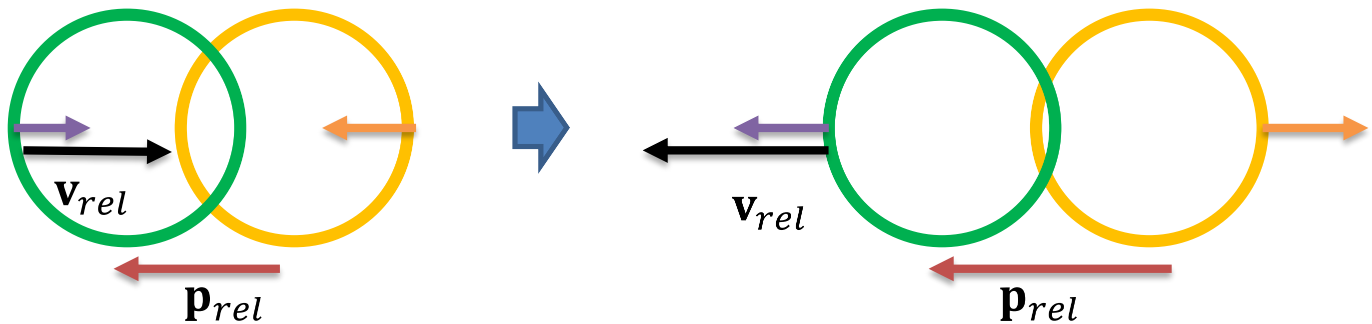
Trapped Problem: Solution 2

- Add one more state check for collision detection
  - $\mathbf{p}_{rel} \cdot \mathbf{v}_{rel} < 0$
    - $\mathbf{p}_{rel}$ is related position $\mathbf{p}_{rel} = \mathbf{p}_i - \mathbf{p}_j$
    - $\mathbf{v}_{rel}$ is related velocity $\mathbf{v}_{rel} = \mathbf{v}_i - \mathbf{v}_j$

We will use this Solution.

Time Step: 1

Time Step: 2



Collision Occurred

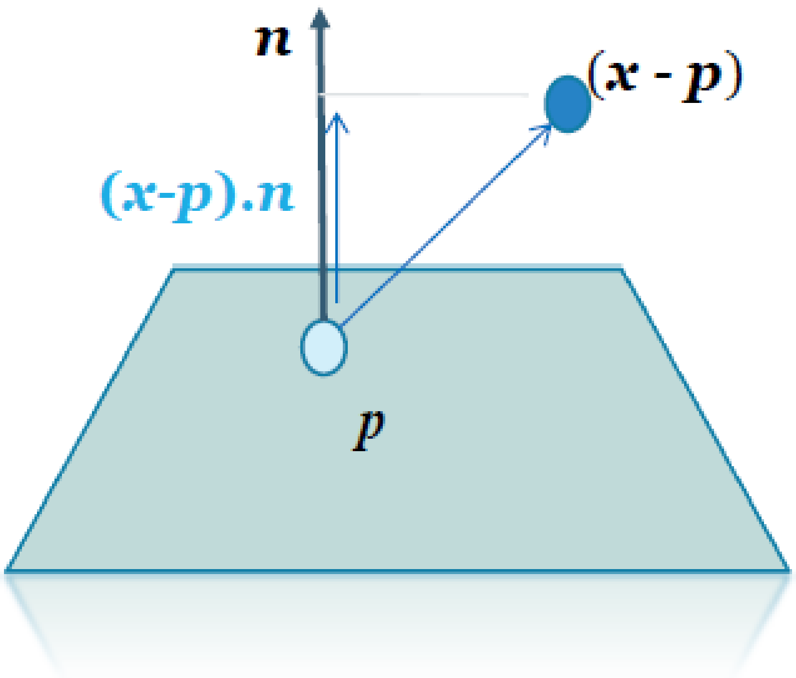
Collision is **not** Occurred

Distance $< r_1 + r_2$
&
$\mathbf{p}_{rel} \cdot \mathbf{v}_{rel} < 0$

Distance $< r_1 + r_2$
But, $\mathbf{p}_{rel} \cdot \mathbf{v}_{rel} > 0$

Collision Detection and Response:
Particle-Plane

Particle Position



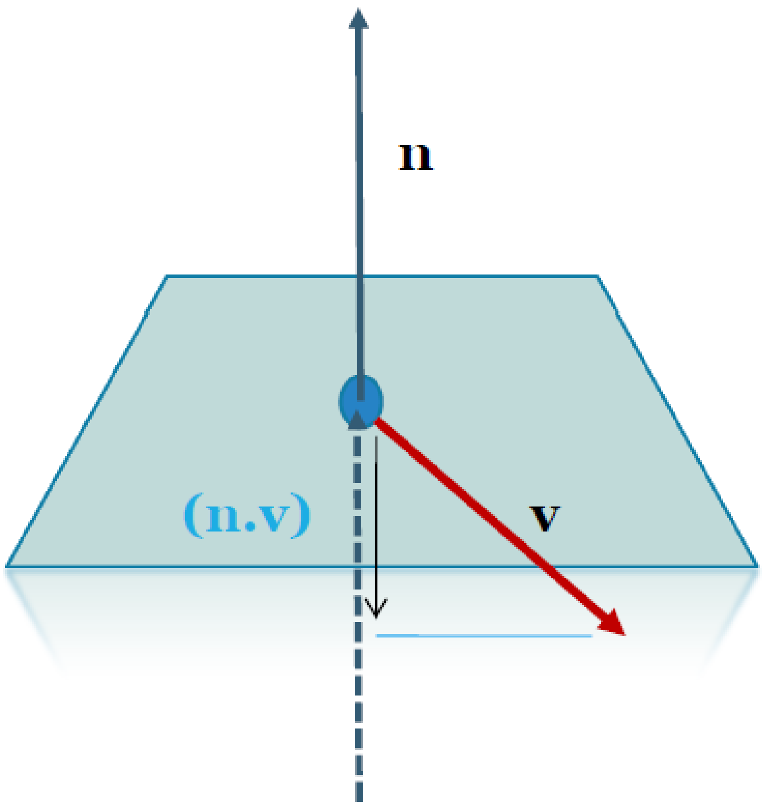
- Given normal $\boldsymbol{n}$ and any point $\boldsymbol{p}$ on plane
- Particle is on the "inside" of the plane (i.e. intersection)

$$\text{If } (x - p) \cdot n < r$$

- In practice we take a threshold distance ε i.e. Particle is intersecting

$$\text{If } (x - p) \cdot n < \varepsilon$$

Particle Velocity

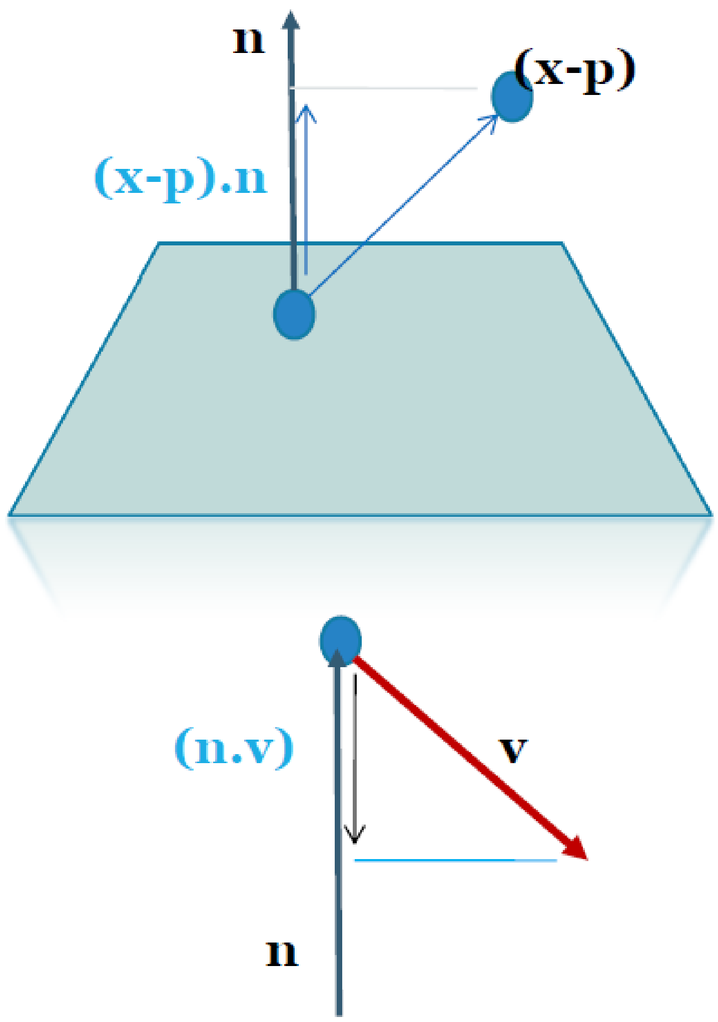


- If intersecting we should also check whether the particle is moving further into the plane

$$\text{If } (n \cdot v) < 0$$

  - Particle is heading deeper into plane

Particle-Plane Collision Detection

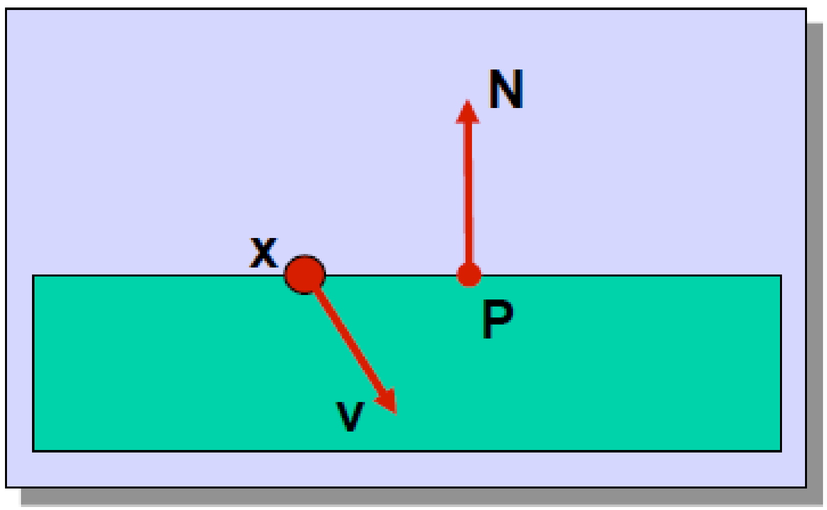


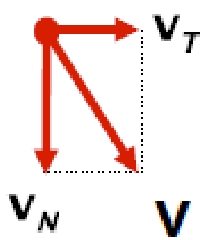
- Given normal n and any point p on plane

$$\text{If } ((x-p)\cdot n < r \mathbin{\&\&} (n \cdot v < 0))$$

Collision Response:
Normal and Tangential Velocity

- To compute the collision response, we need to consider the normal and tangential components of a particle's velocity

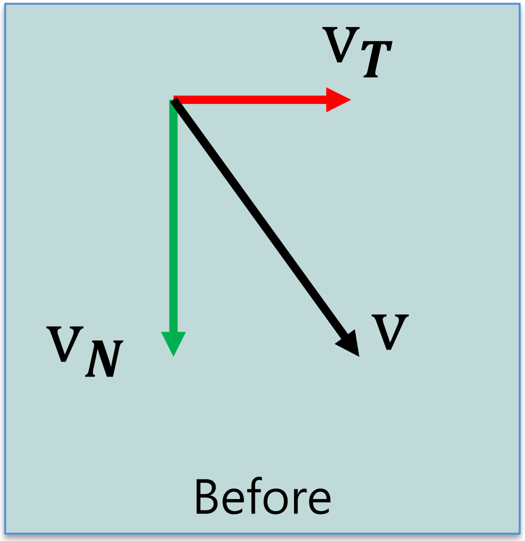


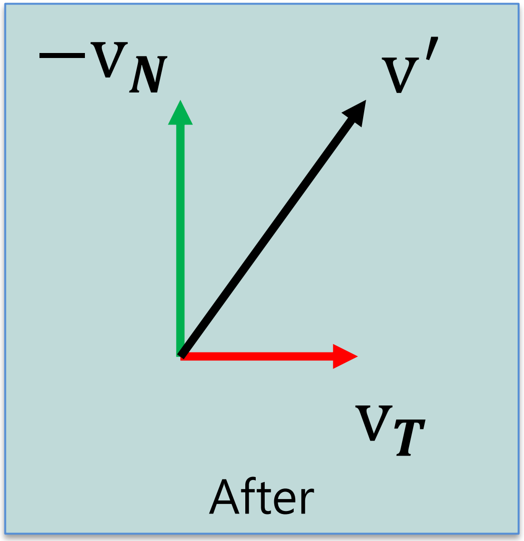


$$\mathbf{v}_N = (\mathbf{N} \cdot \mathbf{v})\mathbf{N}$$

$$\mathbf{v}_T = \mathbf{v} - \mathbf{v}_N$$

Collision Response





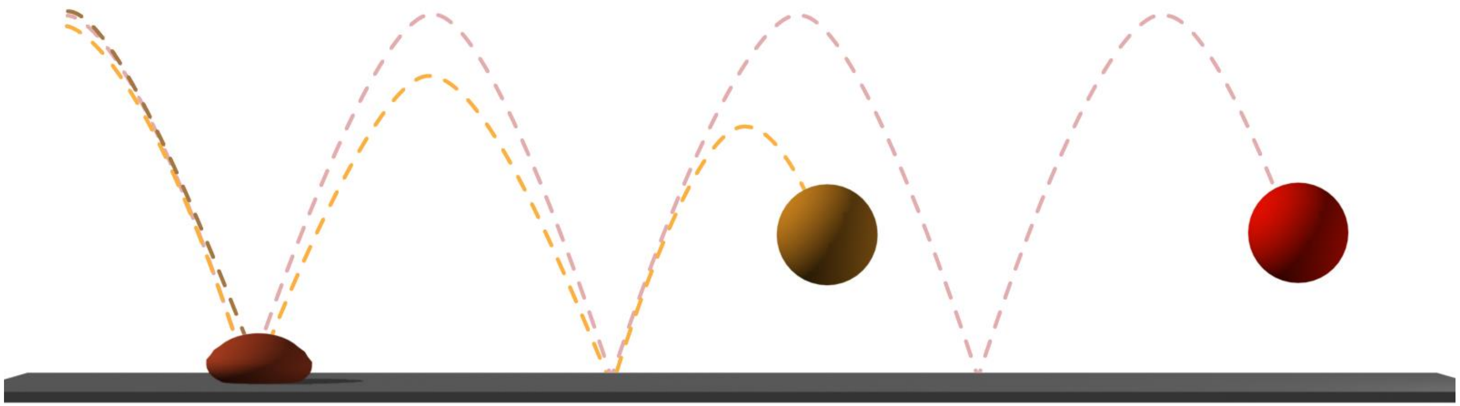
- The response to collision is then to immediately replace the current velocity with a new velocity:

$$V' = V_T - V_N$$

- The particle will then move according to this velocity in the next timestep.

Types of Restitution

Three Types of Collisions



Perfectly inelastic | Inelastic | Elastic

Coefficient of Restitution $k=0$	Coefficient of Restitution $0<k<1$	Coefficient of Restitution $k=1$

- **Perfectly** inelastic collision: The objects **stick together**.
- **Inelastic** collision: These collisions are **somewhat bouncy**.
- **Elastic** collisions: These collisions are "**perfectly**" **bouncy**.

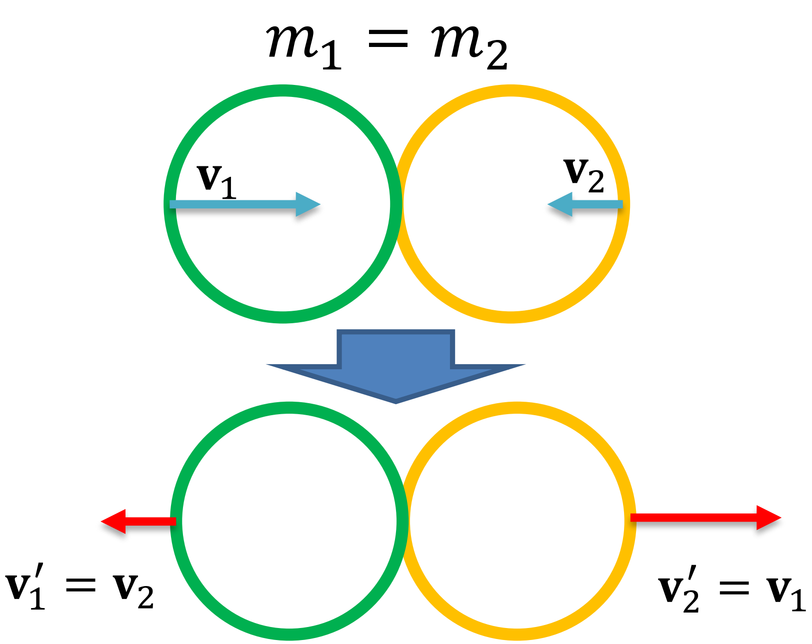
Coefficient of Restitution

- Ratio of the final to initial relative velocity after *collision*.
  - $k_{restitution} = -\frac{\mathbf{v}_1' - \mathbf{v}_2'}{\mathbf{v}_1 - \mathbf{v}_2}$
  - $0 \le k_{restitution} \le 1$
- $k_{restitution} = 1$: Called (Perfectly) Elastic Collision
- $0 < k_{restitution} < 1$: Called Inelastic Collision
- $k_{restitution} = 0$: Called Perfectly Inelastic Collision

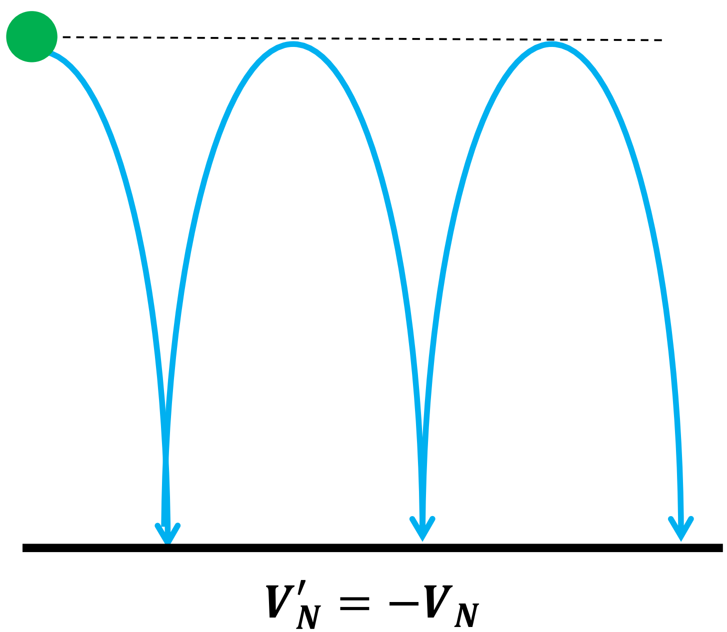
Elastic Collision

- $k_{restitution} = 1$: (Perfectly) Elastic Collision
  - Energy is conserved
  - Momentum is conserved

Particle-Particle



Particle-Plane

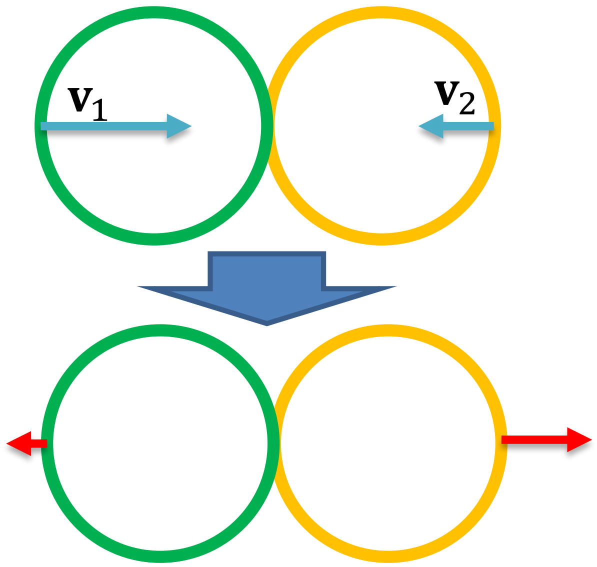


Inelastic Collision(particle-particle)

- $0 < k_{restitution} < 1$: Inelastic Collision
  - Energy is **not** conserved
  - Momentum is conserved

Particle-Particle

$m_1 = m_2$

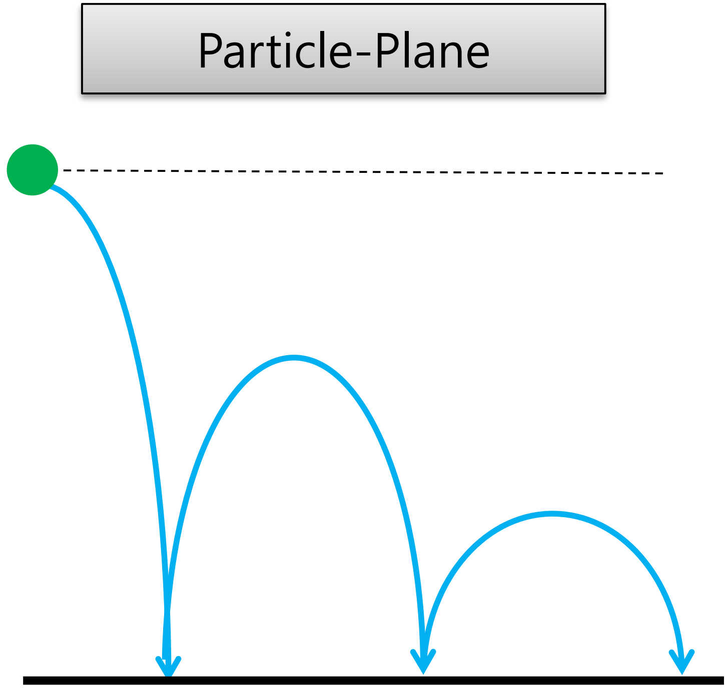


$$\mathbf{v}_1' = \frac{(1+k)\mathbf{v}_2 + (1-k)\mathbf{v}_1}{2}$$

$$\mathbf{v}_2' = \frac{(1+k)\mathbf{v}_1 + (1-k)\mathbf{v}_2}{2}$$

Inelastic Collision(particle-plane)

- $0 < k_{restitution} < 1$: Inelastic Collision
  - Energy is **not** conserved
  - Momentum is conserved



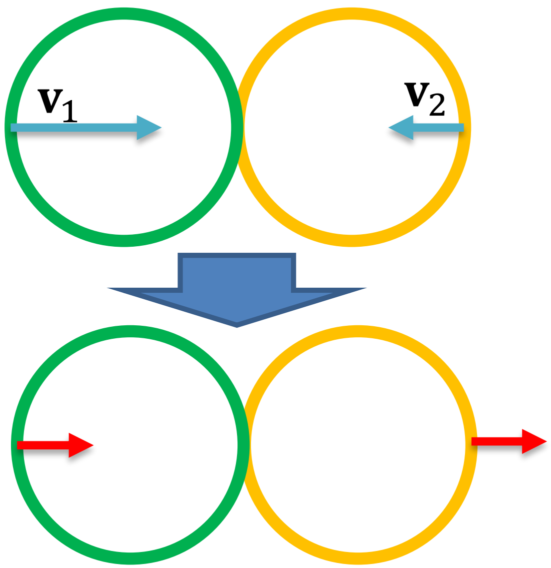
$$V'_N = -kV_N$$

Perfectly Inelastic Collision

- $k_{restitution} = 0$: Perfectly Inelastic Collision
  - Energy is **not** conserved
  - Momentum is conserved

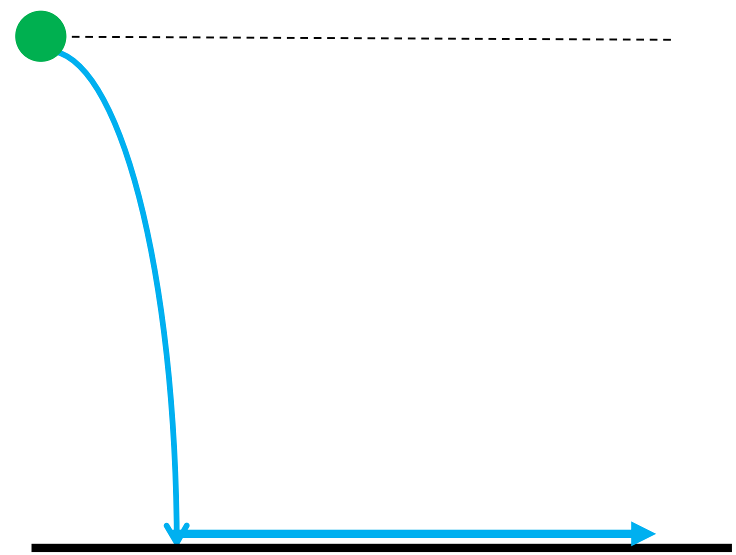
Particle-Particle

$$m_1 = m_2$$



$$\mathbf{v}_1' = \mathbf{v}_2' = \frac{\mathbf{v}_1 + \mathbf{v}_2}{2}$$

Particle-Plane



$$\boldsymbol{V_N' = 0}$$

Code skeleton: Collision()

Main

Initialize( )

OpenGL 기본 설정
- DisplayMode 설정
- Window 생성

ParticleSystem_Setting( )

Particle System Initialize
- Particle system 생성
- Gravity 설정

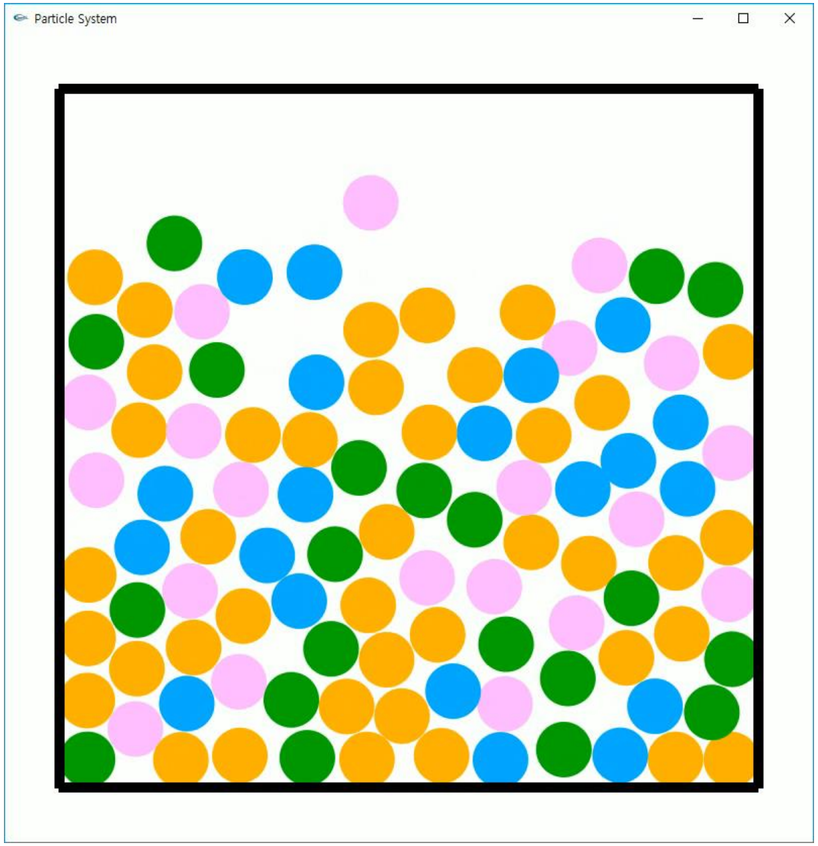
ParticleSystem_Movement( )

Particle System
- Particle position update
- Particle velocity update
- **Collision**

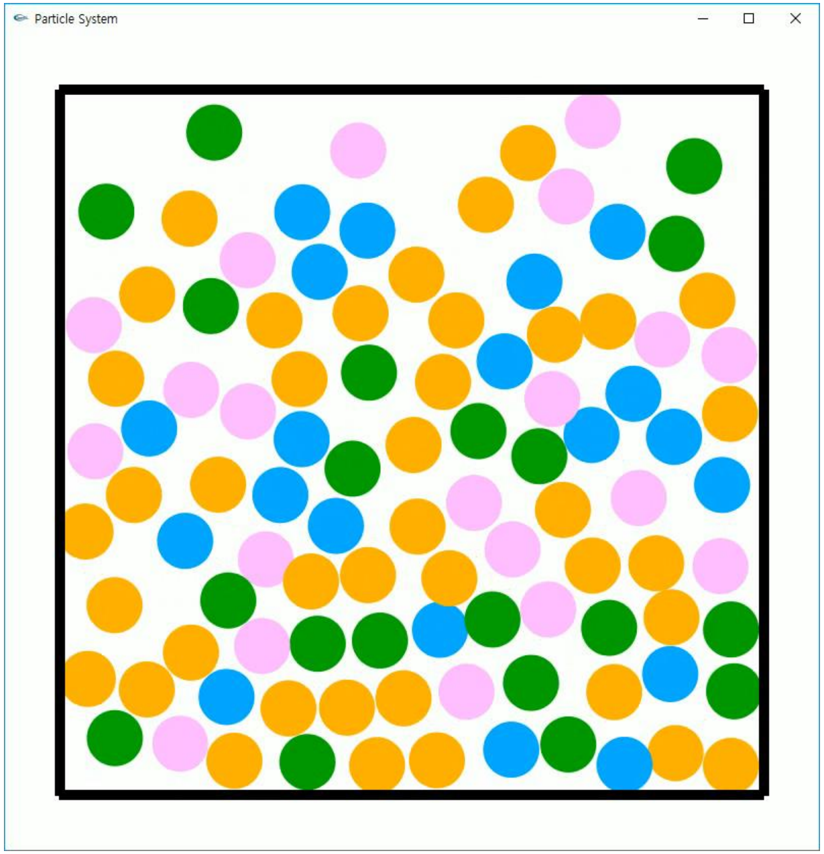
Rendering( )

Draw the Scene
Particle system그리기

Demo: Detection and Response



- Particle Number: 100
- Particle Radius: 4
- **COR: 0.5**



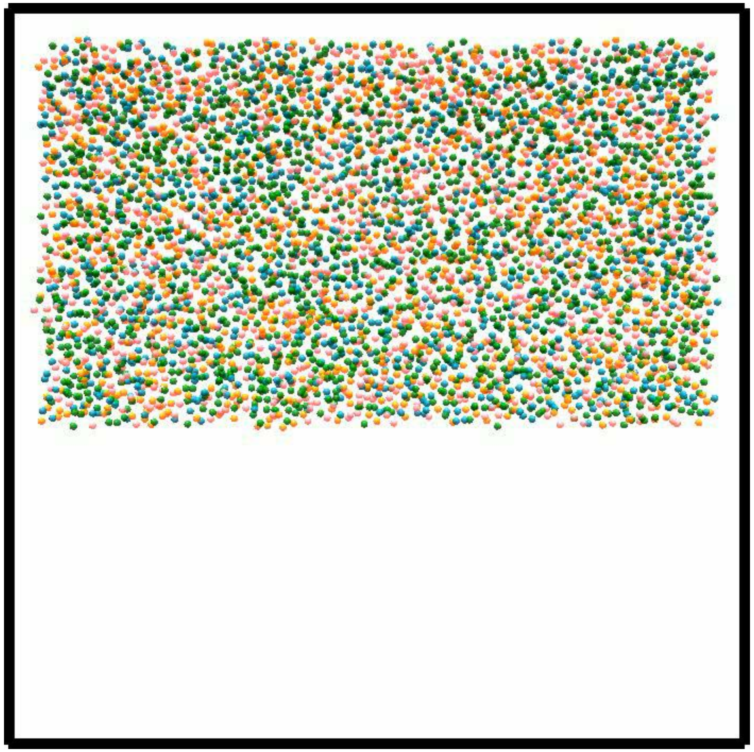
- Particle Number: 100
- Particle Radius: 4
- **COR: 1.0**

Particle collision detection and response

Hash Grid

Why Hash Grid

- Current Time complexity: O(n^2)
  - For all particle i Collide Detect
    - For all particle j
- You can reduce time complexity: O(n)
  - With Hash Grid
  - For all particle i Collide Detect
    - For neighbor particle j
      - # of neighbor Particles is much less than # of total particles



Particle number: 5000

Hash Grid Algorithm

- **Generate Hash table**
  - Divide space into grid table
  - Find grid index for each particle
    - [Position of Particle] is key of hash table
    - [Particle Position to Grid Index] is Hash Function
  - Each grid stores particle indices which are located in
    - [Particle indices located in grid] are buckets

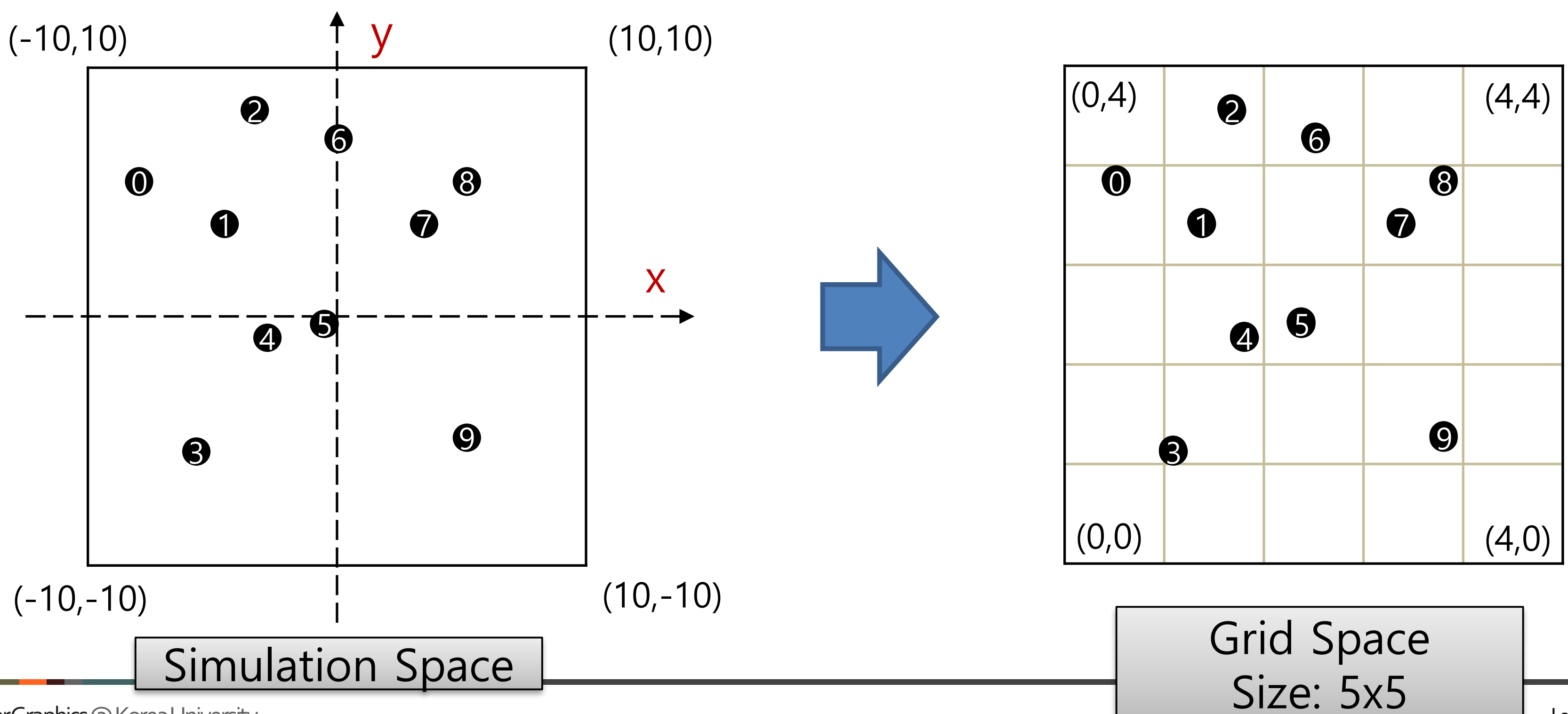
- **Get Neighbor Particles**
  - Find all neighbor grid indices for particle
  - Find the particles stored in neighbor grids
    - Get particle Indices stored in Neighbor Grids Hash

Coding Scheme

- **Generate Hash table**
  - Divide space into grid table
    - vector<int> hash[GRID_DEPTH][GRID_HEIGHT][GRID_WIDTH]
  - Find grid index for each particle
    - Convert Particle position to Grid index.
      - Get (grid_x, grid_y, grid_z)
  - Each grid stores particle indices which are located in
    - hash[grid_x][grid_y][grid_z].pushback(particleIndex)
- **Get Neighbor Particles**
  - Find all neighbor grid indices for particle
    - Get Neighbor grid Index (nGrid_x, nGrid_y, nGrid_z)
  - Find the particles stored in neighbor grids
    - Get particle Indices stored in Neighbor Grids Hash
      - hash[nGrid_x][nGrid_y][nGrid_z]
- **Collision Detection**
  - Compute the distance of the particle and it's neighbor particles
    - Distance=dist(p.position-neighbor.position)
    - If p.radius+neighbor.radius-Distance<0, collision detected

Grid Partition

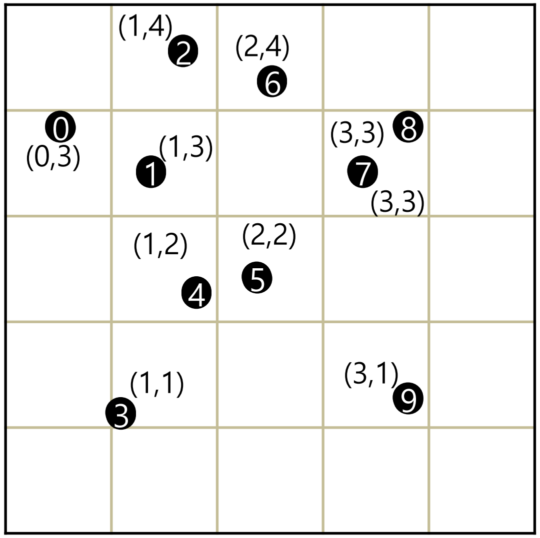
- **Divide space into grid table**
- Find grid index for each particle
- Each grid stores particle indices which are located in



Hash Indexing

- Divide space into grid table
- **Find grid index for each particle**
- Each grid stores particle indices which are located in

- Grid Index is Computed From Particle Position
  - Simulation Space Pos→ Grid Space Index
    1. Normalize Position to (0,1)
    2. Multiply and Clamp Grid Size
    3. Round down after the decimal point
       - Make Integer

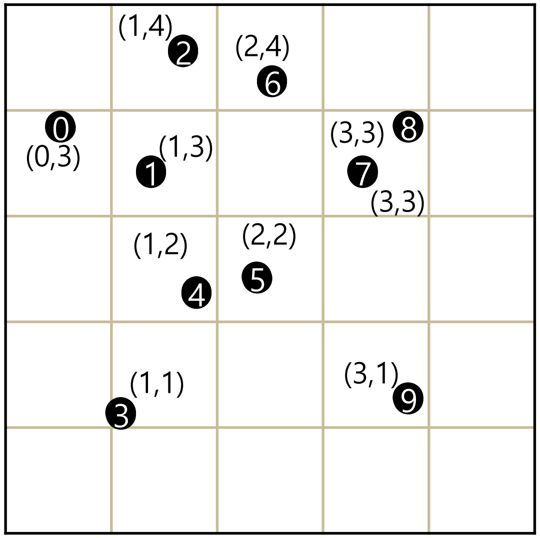


Indexing Example

- Grid Index is Computed From Particle Position
  - Simulation Space Pos→ Grid Space Index
  - For this Example, Particle Number 0
    - Particle Position: (-8, 5)
    - 1. Normalize Position to (0,1)
      - Simulation Space: (-10,-10)~(10,10)
      - ((-8, 5) –(-10,-10))/(20,20) = (0.1, 0.75)
      - Grid Space Size: 5x5 (25 Cells)

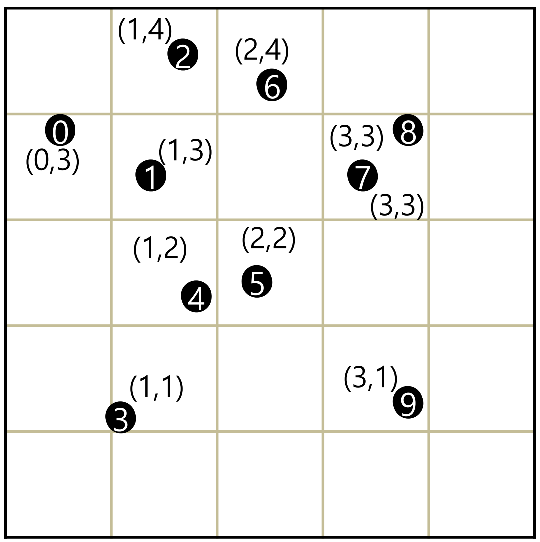
    2. Multiply and Clamp Grid Size
      - Grid Space Size: 5x5 (25 Cells)
      - (0.1, 0.75)x(5,5) = (0.5, 3.75)

    3. Round down after the decimal point
      - (0.5, 3.75) → (0,3)



Store Particles into Grid

- Divide space into grid table
- Find grid index for each particle
- **Each grid stores particle indices which are located in**



Grid Space
Size: 5x5

	2	6		
0	1		7, 8	
	4	5		
	3		9	

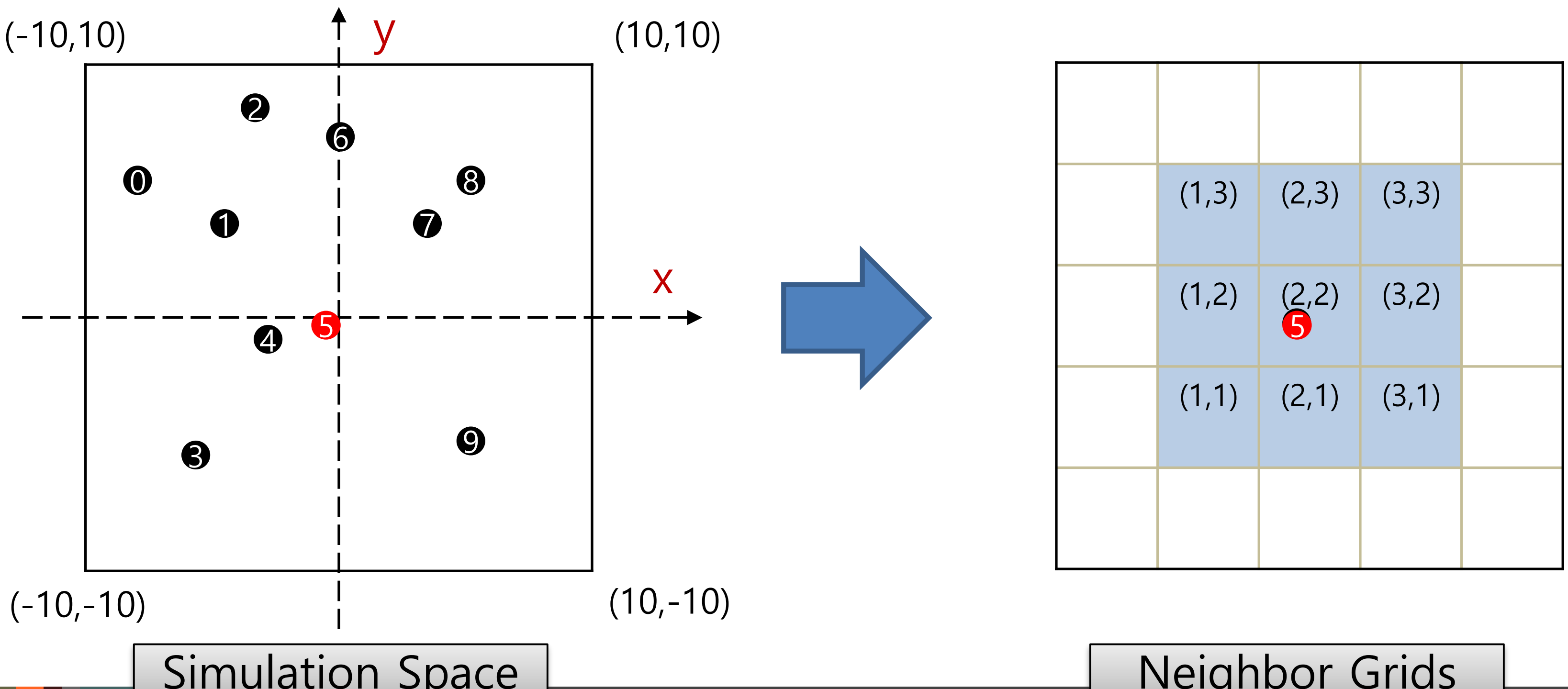
Particle indices
Stored in Hash

Get Neighbor Particles

- Find all neighbor grid indices for particle
- Find the particles stored in neighbor grids

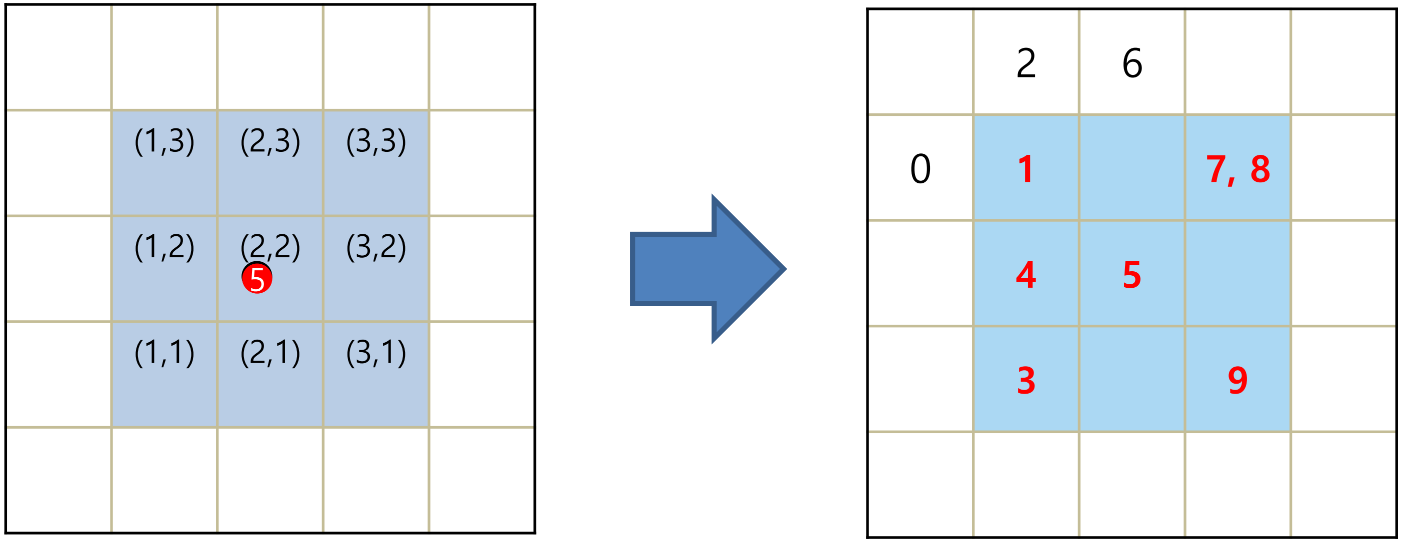
Get Grid Neighborhoods

- **Find all neighbor grid indices for particle**
- Find the particles stored in neighbor grids



Search Particles in N_Grid

- Find all neighbor grid indices for particle
- **Find the particles stored in neighbor grids**

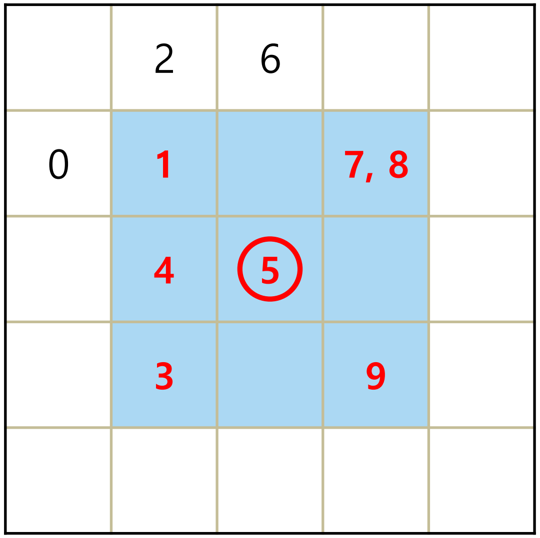


Neighbor Grids

Neighbor Particle indices Stored in Hash

Collision Check

- For particle 5
  - Check the neighbor particles(1,3,4,7,8,9)
    - For each neighbor particle
      - Compute the distance of particle 5 and the neighbor particle
      - If **distance**<=Collide Distance
        - Collision is true
      - Else if **distance**>Collide Distance
        - Collision is false



Code skeleton: Hash_Grid()

Main

Initialize( )

OpenGL 기본 설정
- DisplayMode 설정
- Window 생성

ParticleSystem_Setting( )

Particle System Initialize
- Particle system 생성

ParticleSystem_Movement( )

Particle System
- Particle position update
- Particle velocity update
- **Collision Detection**

↑

Hash Grid

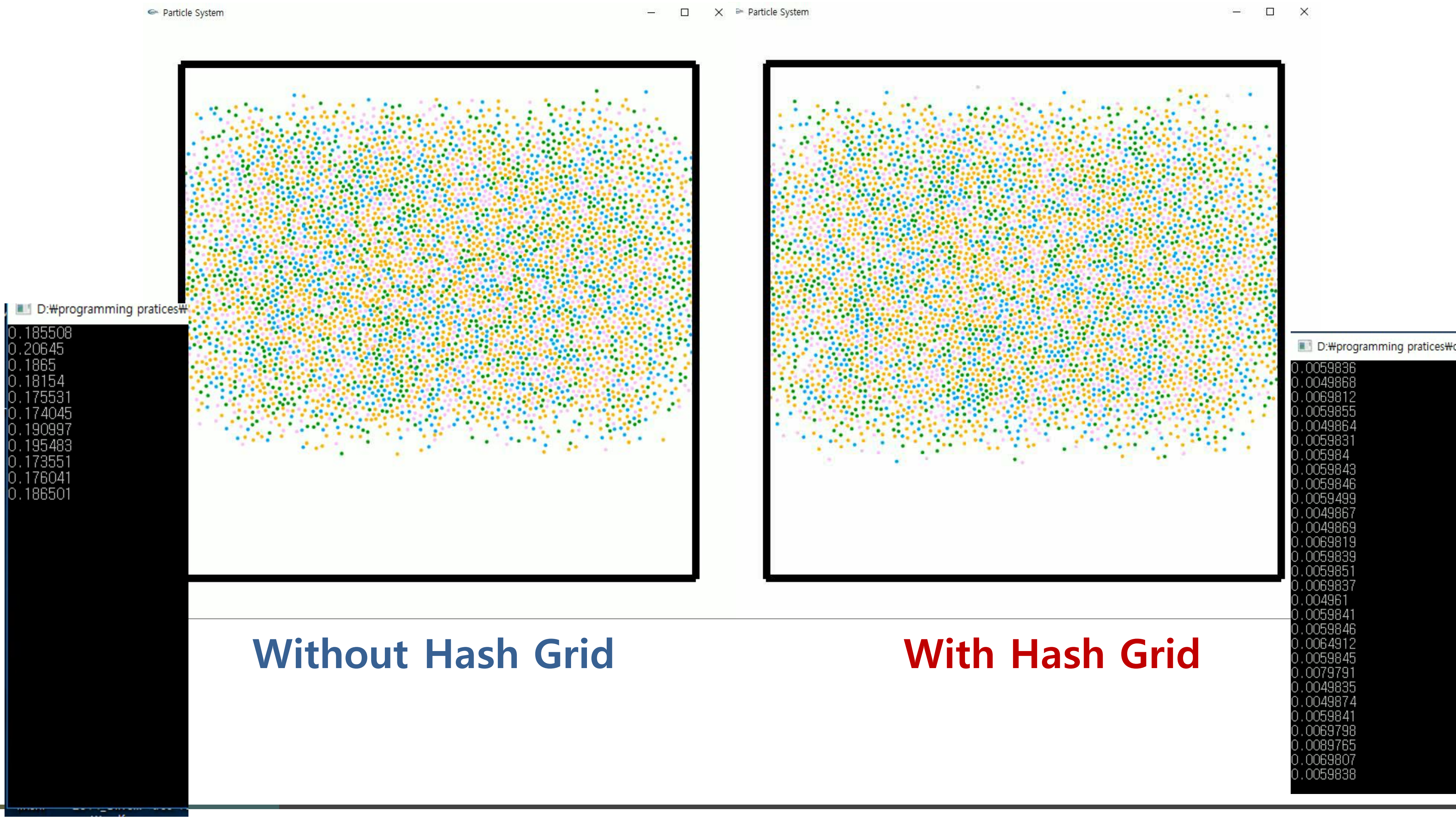
Rendering( )

Draw the Scene

Particle system그리기

Demo
Hash Grid vs. w/o Hash

- **Particle number**: 5000, **Radius**:1, **Simulation Space**: (-100,100), **HashSize**(100*100)



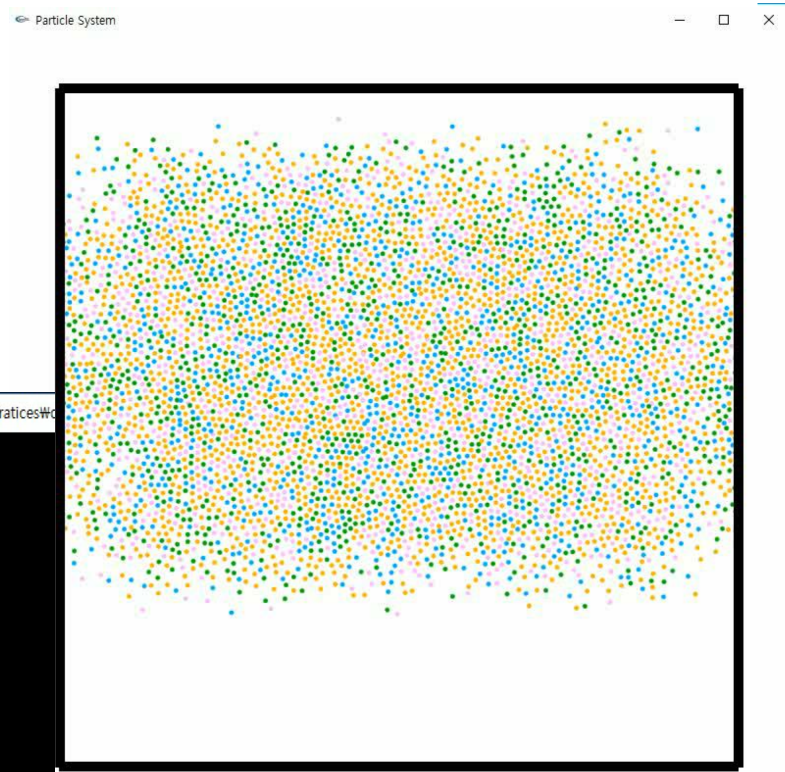
Without Hash Grid

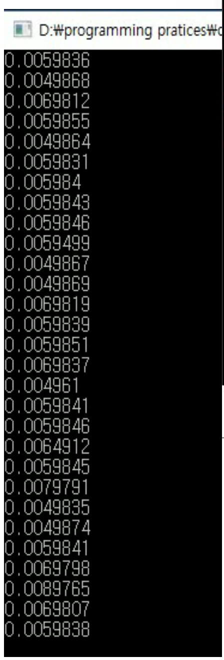
With Hash Grid

Demo

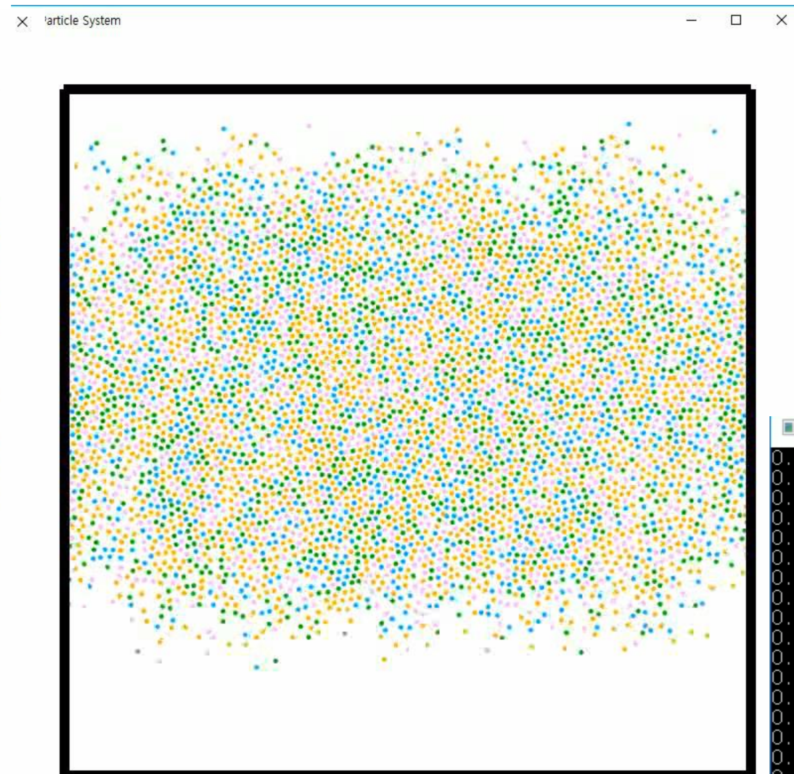
Various Size of Hash Grid

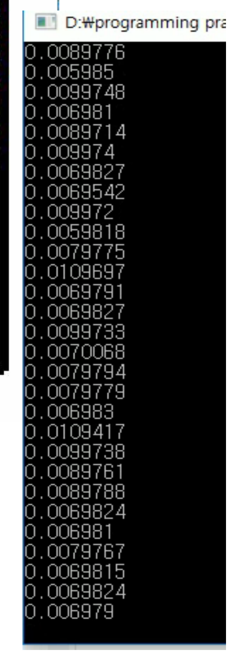
- **Particle number**: 5000, **Radius**:1, **Simulation Space**: (-100,100)





HashSize(100*100)

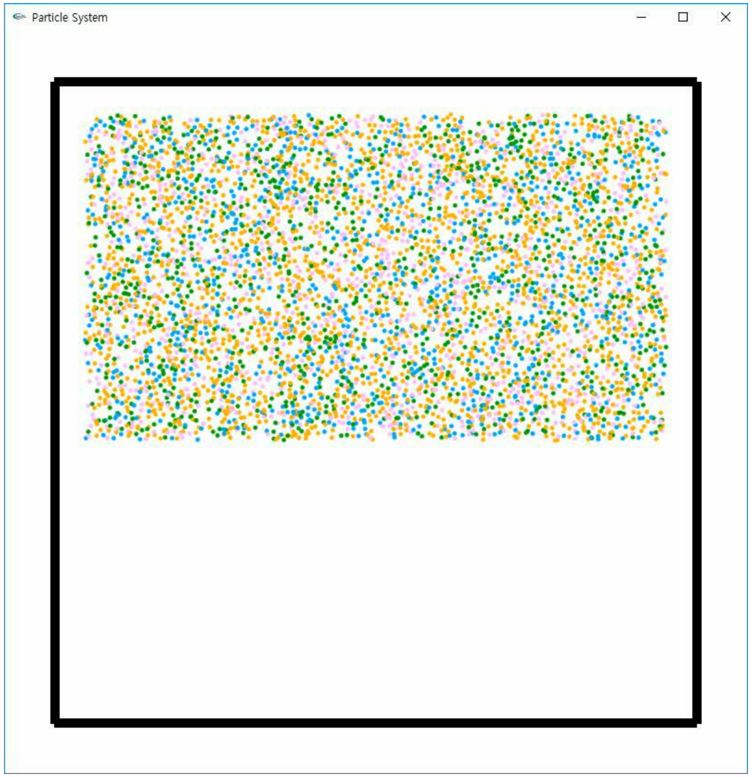




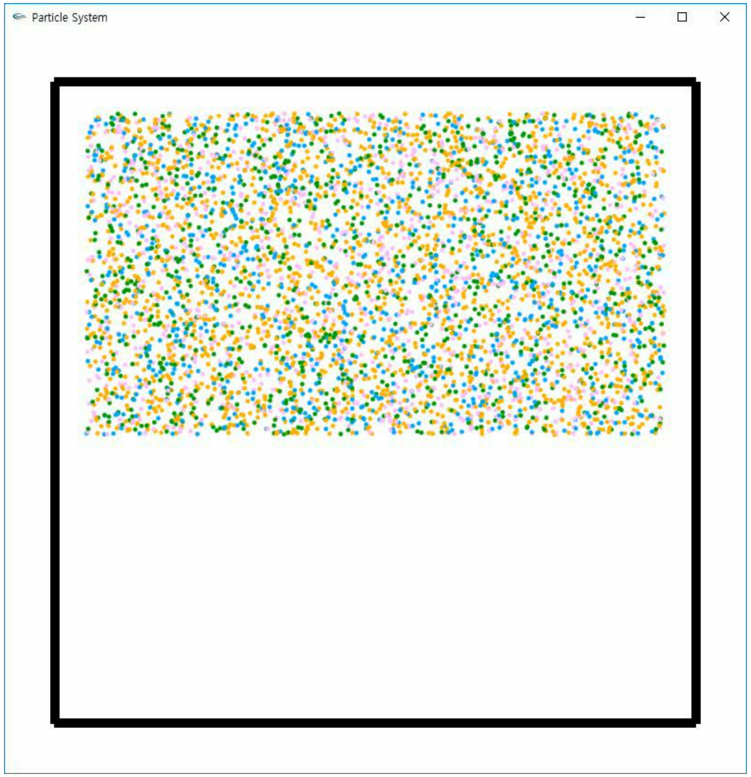
HashSize(50*50)

Demo
Neighboring radius of Hash Grid

- **Particle number**: 5000, **Radius**:1&3, **Simulation Space**: (-100,100)



Radius 1



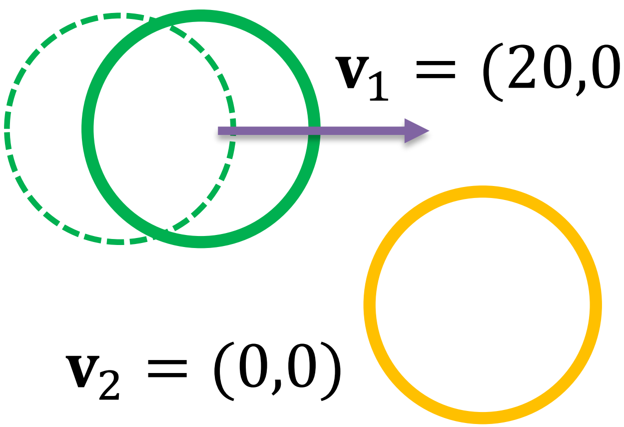
Radius 3

Particle to Particle collision response

Example

Collision Response Example#1

Dynamic-Static Case: Collision Normal & Tangent



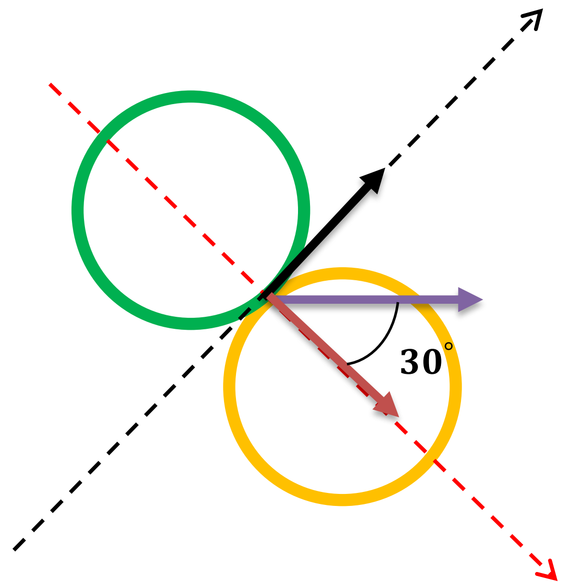
$$\mathbf{v}_1 = (20,0)$$

$$\mathbf{v}_2 = (0,0)$$

$$m_1 = m_2$$

Collision Tangent

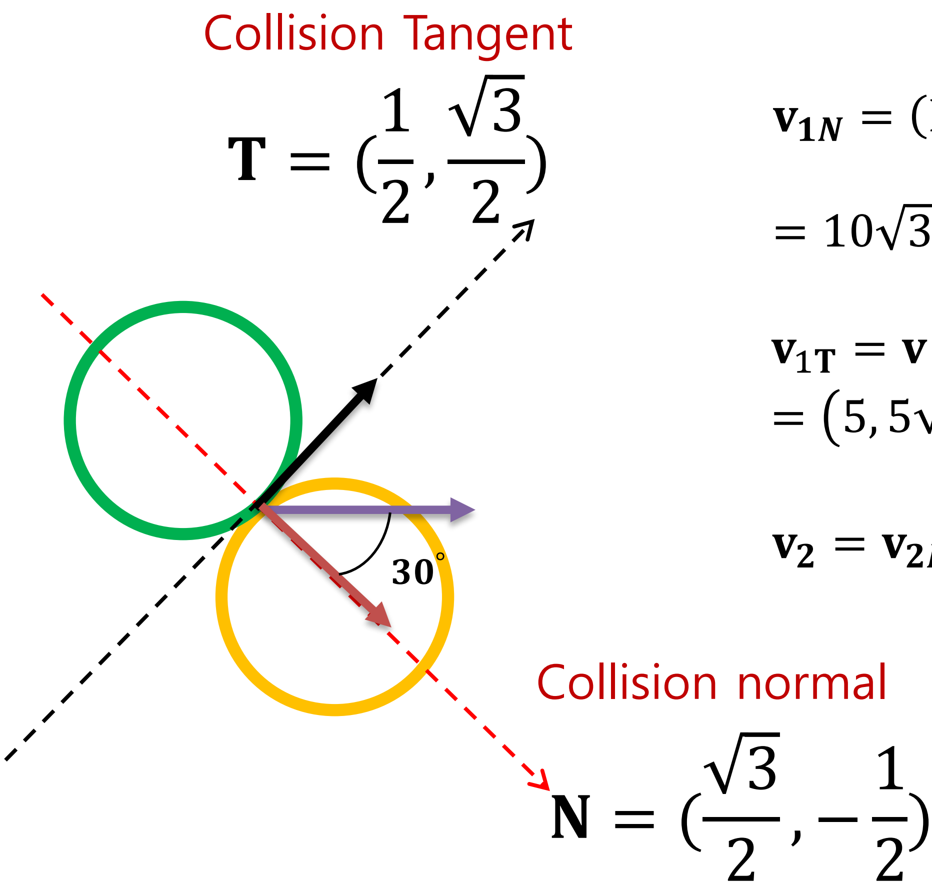
$$\mathbf{T} = (-\sin(-30), \cos(-30)) = (\frac{\sqrt{3}}{2}, -\frac{1}{2})$$



Collision normal

$$\mathbf{N} = (\cos(-30)\,, \sin(-30)) = (\frac{\sqrt{3}}{2}, -\frac{1}{2})$$

Dynamic-Static: Velocity Decomposition



$$\mathbf{v}_{1N} = (\mathbf{N} \cdot \mathbf{v}_1)\mathbf{N} = \left(\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right) \cdot (20,0)\right)\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right)$$

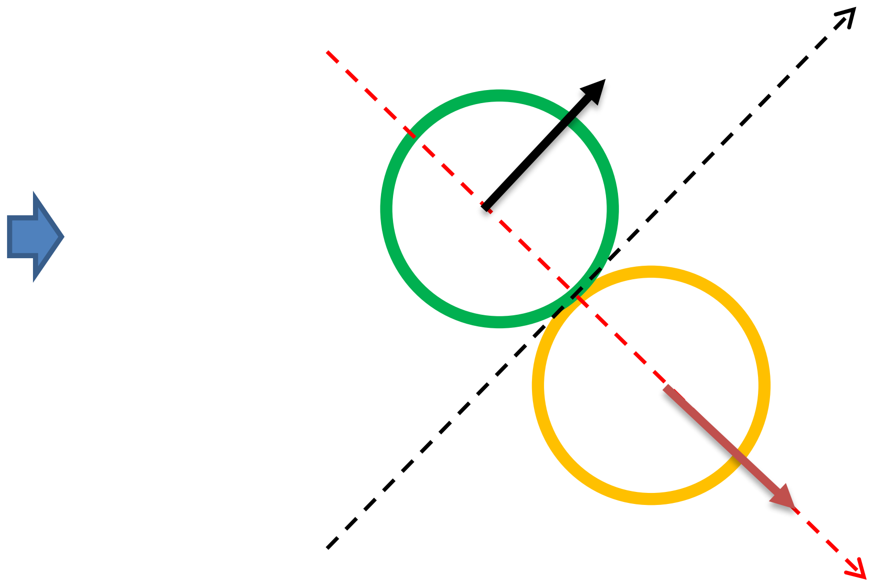
$$= 10\sqrt{3}\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right) = (15, -5\sqrt{3})$$

$$\mathbf{v}_{1\mathbf{T}} = \mathbf{v} - \mathbf{v}_{1N} = (20,0) - \left(15, -5\sqrt{3}\right)$$

$$= \left(5, 5\sqrt{3}\right)$$

$$\mathbf{v}_2 = \mathbf{v}_{2N} = \mathbf{v}_{2T} = (\mathbf{0}, \mathbf{0})$$

Collision Response Example#1
Dynamic-Static: Determine Velocity after Collision



$$\mathbf{v}'_{1N} = \frac{2m_2\mathbf{v}_{2N}+(m_1-m_2)\mathbf{v}_{1N}}{m_1+m_2} = \frac{2m_1 v_{2N}+(m_1-m_1)v_{1N}}{2m_1} = (0,0)$$

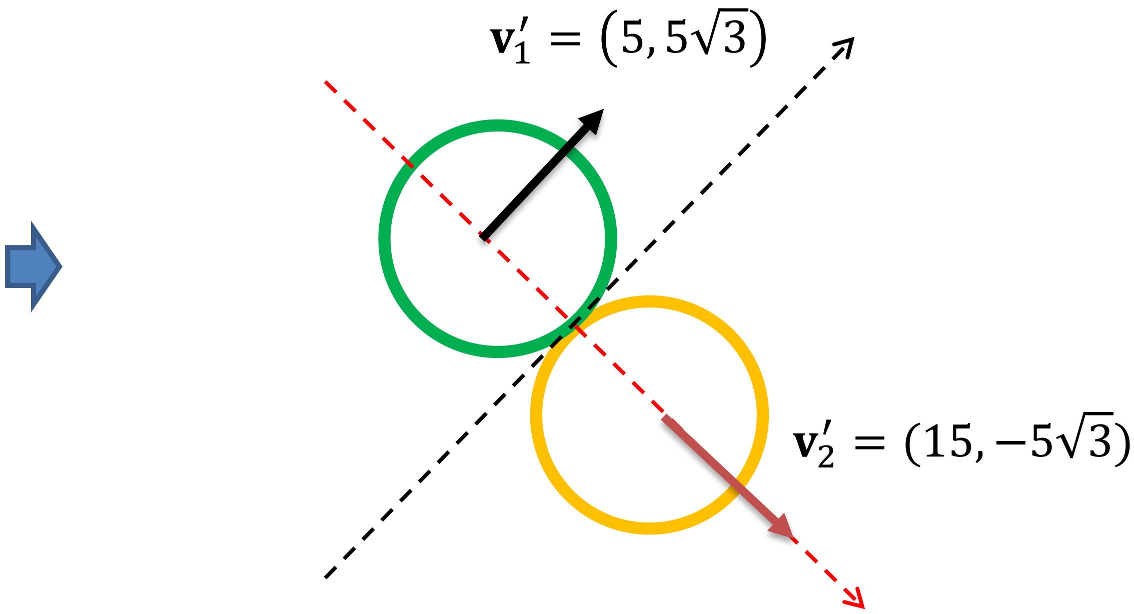
$$\mathbf{v}'_{2N} = \frac{2m_1\mathbf{v}_{1N}+(m_2-m_1)\mathbf{v}_{2N}}{m_1+m_2} = (15, -5\sqrt{3})$$

$$\mathbf{v}'_{\mathbf{1}\boldsymbol{T}} = \mathbf{v}_{1T} = \left(5, 5\sqrt{3}\right)$$

$$\mathbf{v}'_{\mathbf{2}\boldsymbol{T}} = \mathbf{v}_{2T} = (\mathbf{0}, \mathbf{0})$$

Collision Response Example#1

Dynamic-Static: Velocity Composition

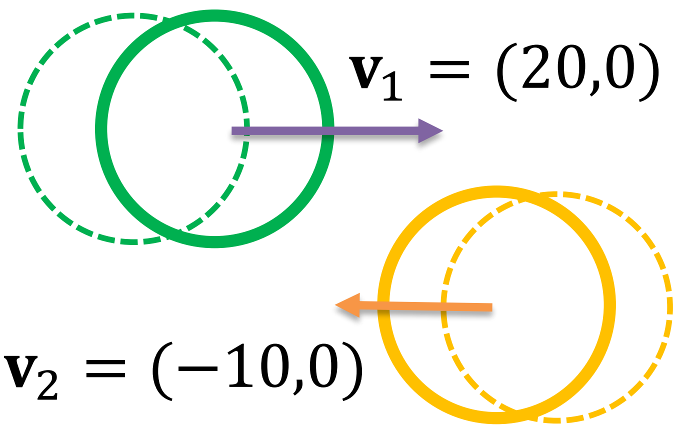


$$\boxed{\mathbf{v}_1' = \mathbf{v}_{1\mathbf{N}}' + \mathbf{v}_{1\mathbf{T}}' = (5, 5\sqrt{3})}$$

$$\boxed{\mathbf{v}_2' = \mathbf{v}_{2\mathbf{N}}' + \mathbf{v}_{2\mathbf{T}}' = (15, -5\sqrt{3})}$$

Collision Response Example#2

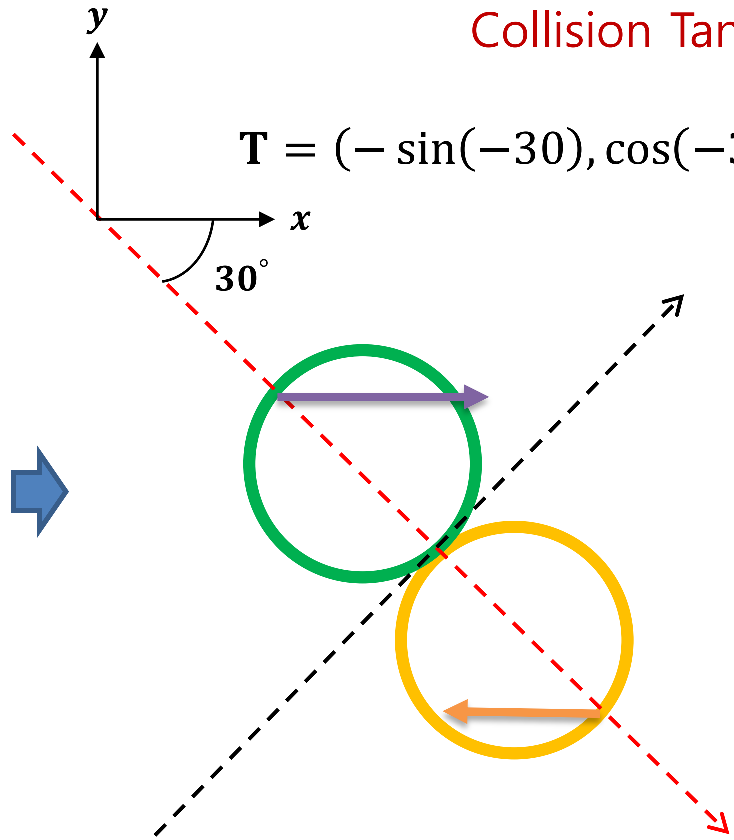
Dynamic-Dynamic Case: Collision Normal & Tangent



$$\mathbf{v}_1 = (20,0)$$

$$\mathbf{v}_2 = (-10,0)$$

$$m_1 = \frac{1}{2} m_2$$



Collision Tangent

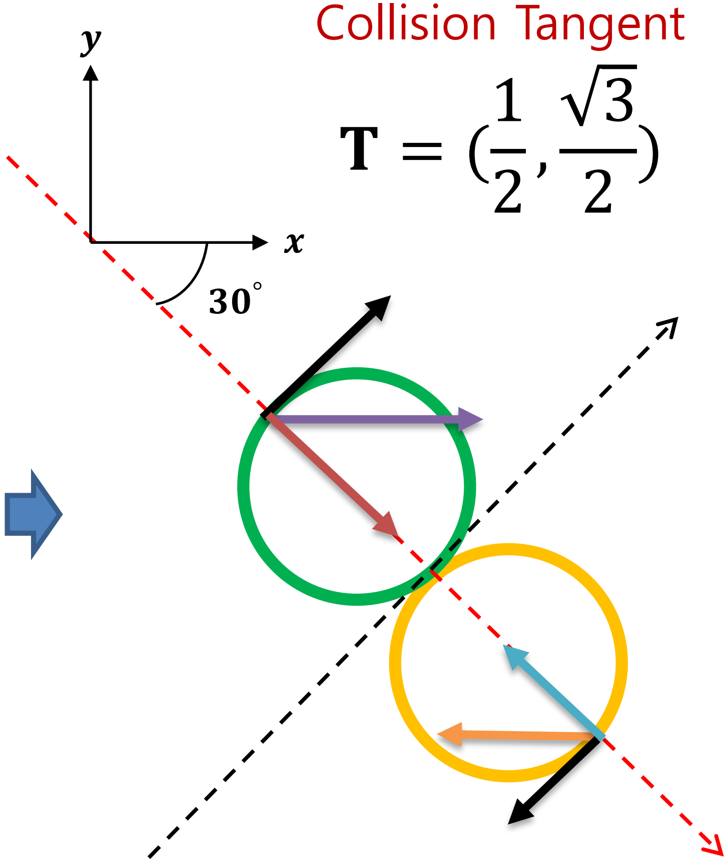
$$\mathbf{T} = (-\sin(-30), \cos(-30)) = (\frac{\sqrt{3}}{2}, -\frac{1}{2})$$

Collision normal

$$\mathbf{N} = (\cos(-30)\,, \sin(-30)) = (\frac{\sqrt{3}}{2}, -\frac{1}{2})$$

Collision Response Example#2

Dynamic-Dynamic: Velocity Decomposition



Collision Tangent

$$\mathbf{T} = \left(\frac{1}{2}, \frac{\sqrt{3}}{2}\right)$$

Collision normal

$$\mathbf{N} = \left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right)$$

$$\mathbf{v}_{1N} = (\mathbf{N}\cdot\mathbf{v}_1)\mathbf{N} = \left(\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right)\cdot(20,0)\right)\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right)$$

$$= 10\sqrt{3}\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right) = (15, -5\sqrt{3})$$

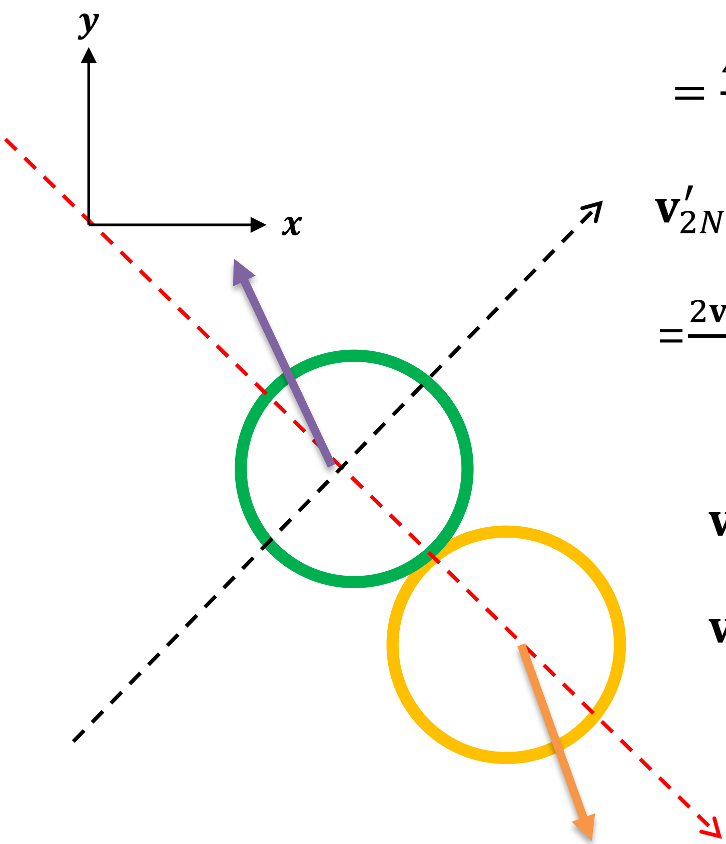
$$\mathbf{v}_{1\mathbf{T}} = \mathbf{v} - \mathbf{v}_{1N} = (20,0) - (15, -5\sqrt{3}) = (5, 5\sqrt{3})$$

$$\mathbf{v}_{2N} = (\mathbf{N}\cdot\mathbf{v}_2)\mathbf{N} = \left(\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right)\cdot(-10,0)\right)\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right)$$

$$= -5\sqrt{3}\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right) = \left(-\frac{\mathbf{15}}{\mathbf{2}}, \frac{\mathbf{5\sqrt{3}}}{\mathbf{2}}\right)$$

$$\mathbf{v}_{2\mathbf{T}} = \mathbf{v} - \mathbf{v}_{2N} = (-10,0) - \left(-\frac{15}{2}, \frac{5\sqrt{3}}{2}\right) = \left(-\frac{5}{2}, -\frac{5\sqrt{3}}{2}\right)$$

Collision Response Example#2
Dynamic-Dynamic: Determine Velocity after Collision



$$\mathbf{v}'_{1N} = \frac{2m_2\mathbf{v}_{2N}+(m_1-m_2)\mathbf{v}_{1N}}{m_1+m_2} = \frac{4m_1 v_{2N}+(m_1-2m_1)v_{1N}}{3m_1}$$

$$= \frac{4\mathbf{v}_{2N}-v_{1N}}{3} = \frac{4\left(-\frac{15}{2},\frac{5\sqrt{3}}{2}\right)-(15,-5\sqrt{3})}{3} = (-15, 5\sqrt{3})$$

$$\mathbf{v}'_{2N} = \frac{2m_1\mathbf{v}_{1N}+(m_2-m_1)\mathbf{v}_{2N}}{m_1+m_2} = \frac{2m_1\mathbf{v}_{1N}+(2m_1-m_1)\mathbf{v}_{2N}}{3m_1}$$

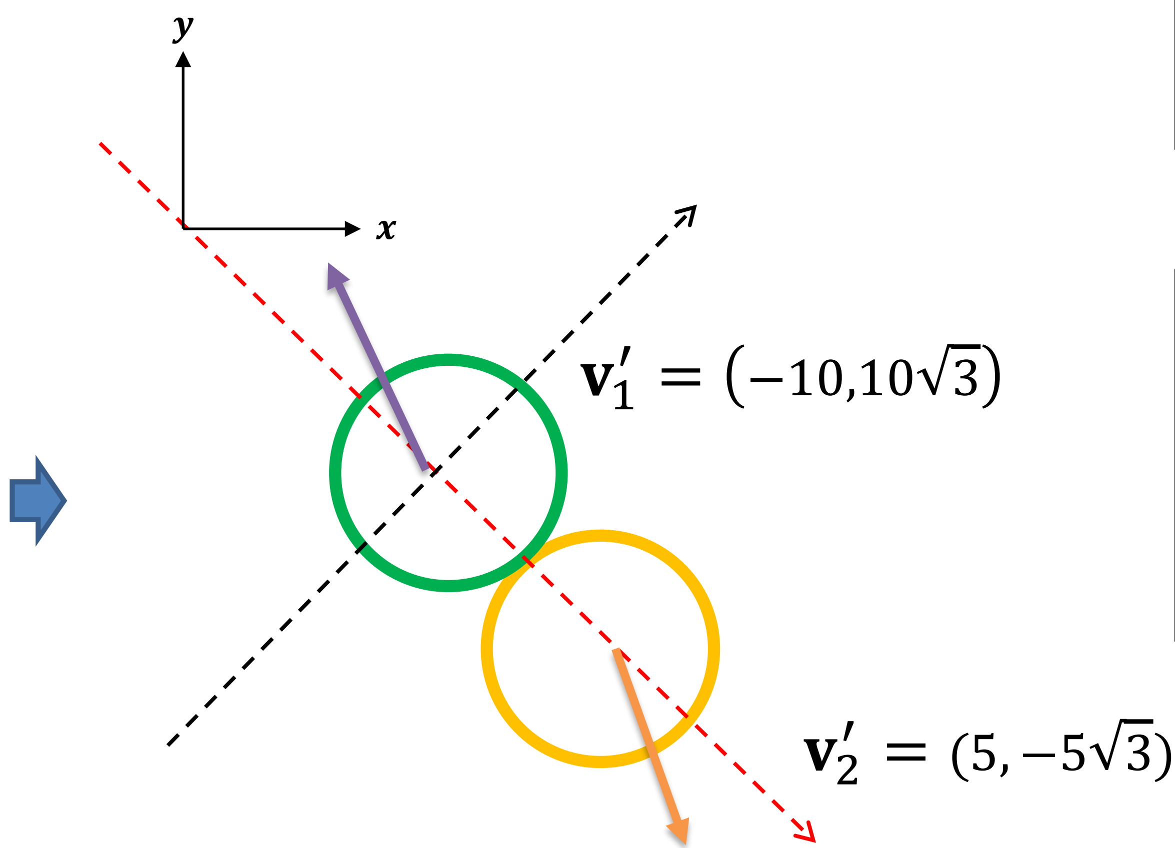
$$= \frac{2\mathbf{v}_{1N}+v_{2N}}{3} = \frac{2(15,-5\sqrt{3})+\left(-\frac{15}{2},\frac{5\sqrt{3}}{2}\right)}{3} = \left(\frac{15}{2}, -\frac{5\sqrt{3}}{2}\right)$$

$$\mathbf{v}'_{1T} = \mathbf{v}_{1T} = (5, 5\sqrt{3})$$

$$\mathbf{v}'_{2T} = \mathbf{v}_{2T} = \left(-\frac{5}{2}, -\frac{5\sqrt{3}}{2}\right)$$

Collision Response Example#2

Dynamic-Dynamic: Velocity Composition



$$\mathbf{v}_{1\mathrm{N}}' = (-15, 5\sqrt{3})$$
$$\mathbf{v}_{1\mathrm{T}}' = (5, 5\sqrt{3})$$
$$\mathbf{v}_1' = \mathbf{v}_{1\mathrm{N}}' + \mathbf{v}_{1\mathrm{T}}' = (-10, 10\sqrt{3})$$

$$\mathbf{v}_{2\mathrm{N}}' = (\frac{15}{2}, -\frac{5\sqrt{3}}{2})$$
$$\mathbf{v}_{2\mathrm{T}}' = \left(-\frac{5}{2}, -\frac{5\sqrt{3}}{2}\right)$$
$$\mathbf{v}_2' = \mathbf{v}_{2\mathrm{N}}' + \mathbf{v}_{2\mathrm{T}}' = (5, -5\sqrt{3})$$