복셀 기반 네트워크 게임 충돌처리 전략

유영천

http://megayuchi.com

tw:@dgtman

미리 공지

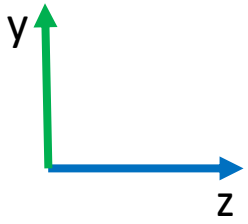
- 충돌처리 구현을 위한 기본적인 접근방법을 설명합니다.
- 수학은 물어보지 마세요. 수학공식은 직접 찾아보시기 바랍니다.

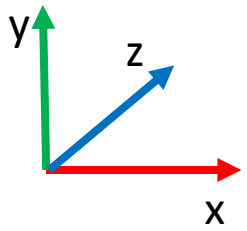
목표

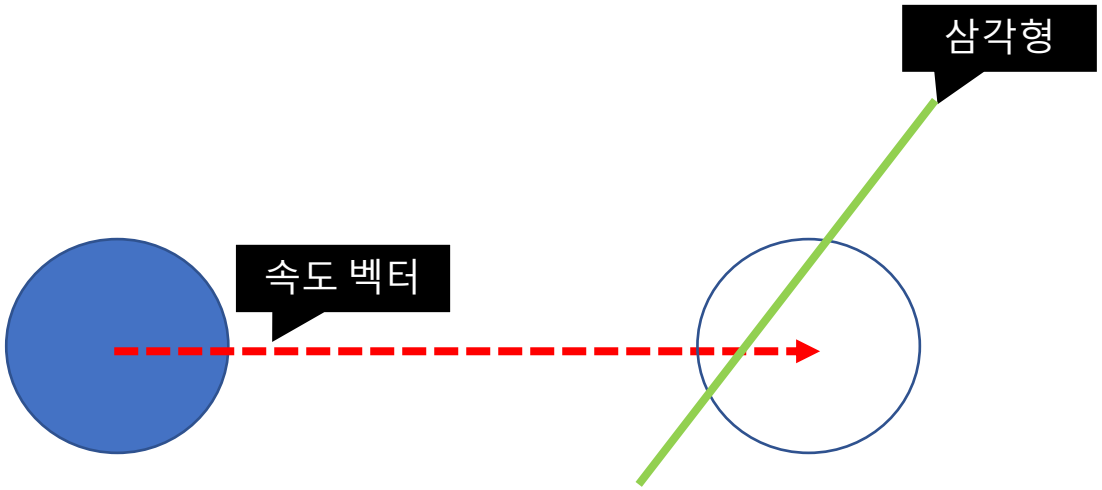
- 지형지물 위에서 적절히 상호작용할 수 있는 충돌처리 기능을 구현한다.
- 첫째도 성능 둘째도 성능
  - 서버에서도 사용한다.
  - 다수의 캐릭터의 이동처리에 사용한다.
- 정교한 충돌처리는 어차피 쓸모가 없다.
  - 네트워크 동기화 하려면 정교한 충돌처리는 어차피 쓰지도 못한다.
- 클라이언트/서버 공통 사용 가능할 것

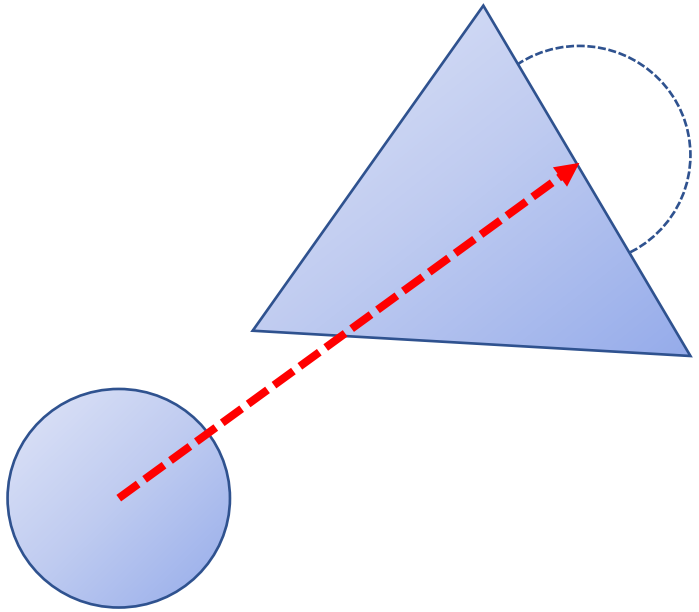
충돌처리 기본

움직이는 구 vs 삼각형 충돌처리

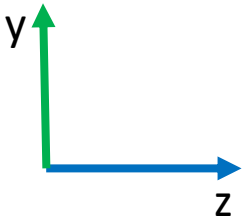


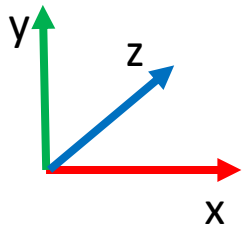




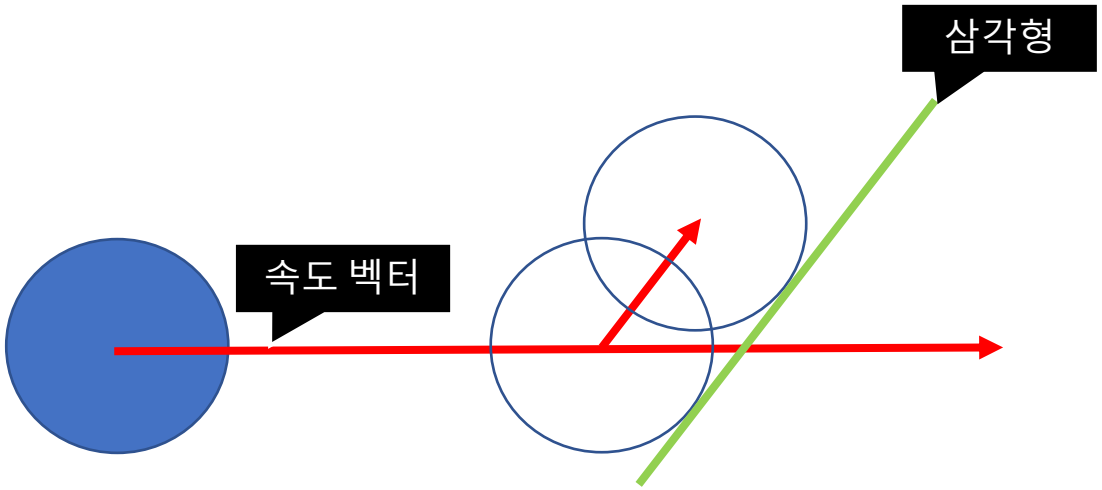


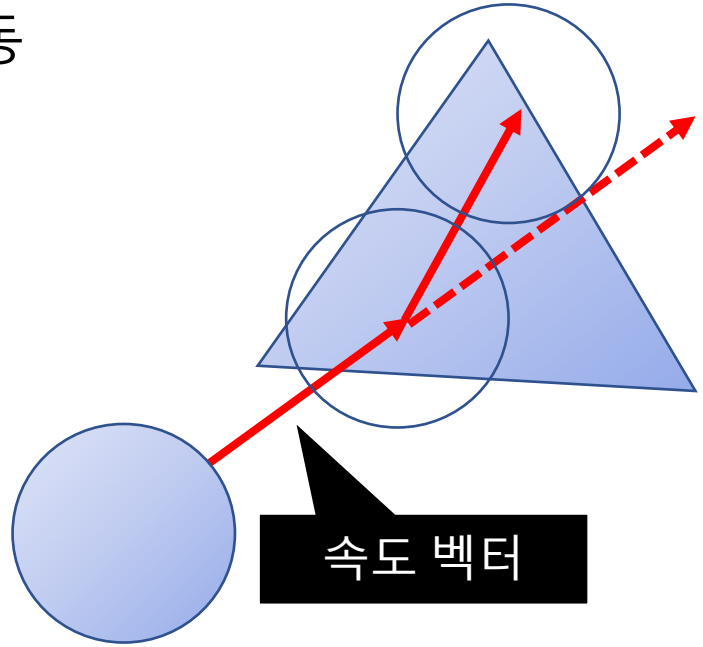
움직이는 구가 삼각형과 충돌했을 때



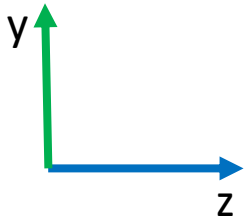


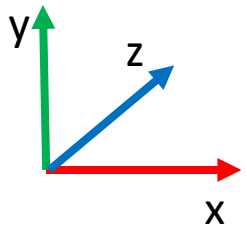
삼각형에 충돌 후 미끄러짐 벡터를 구하고 새로운 방향으로 이동

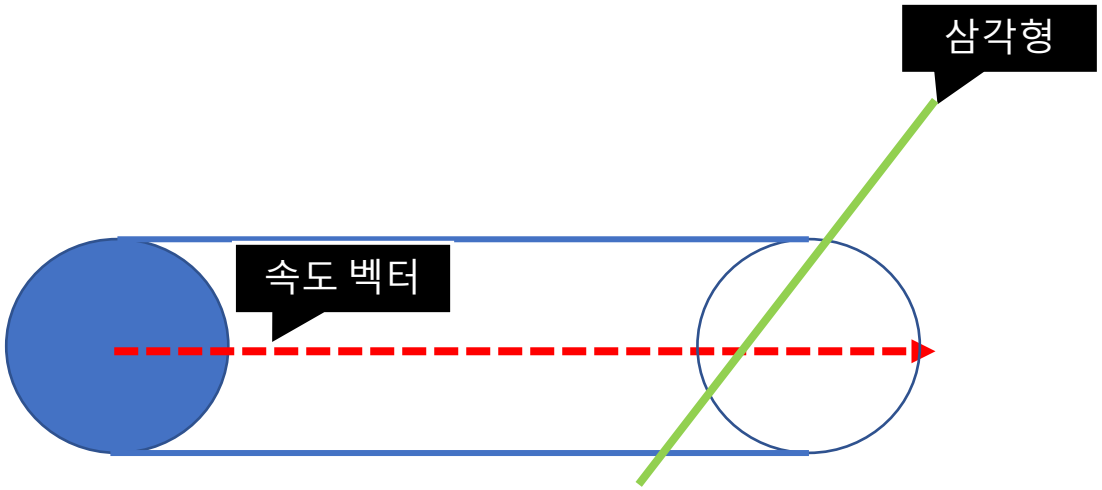


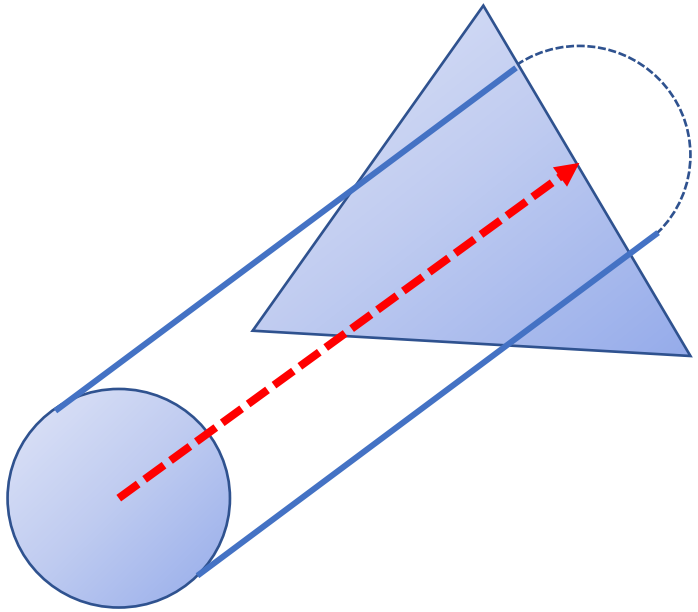


움직이는 구가 삼각형과 충돌했을 때 – 의도하는 결과







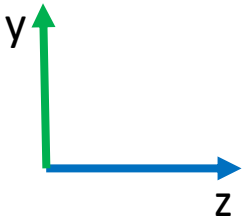


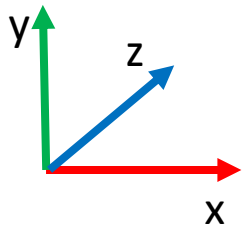
움직이는 구의 궤적은 캡슐이므로 '캡슐 vs 삼각형' 충돌처리가 필요하다.

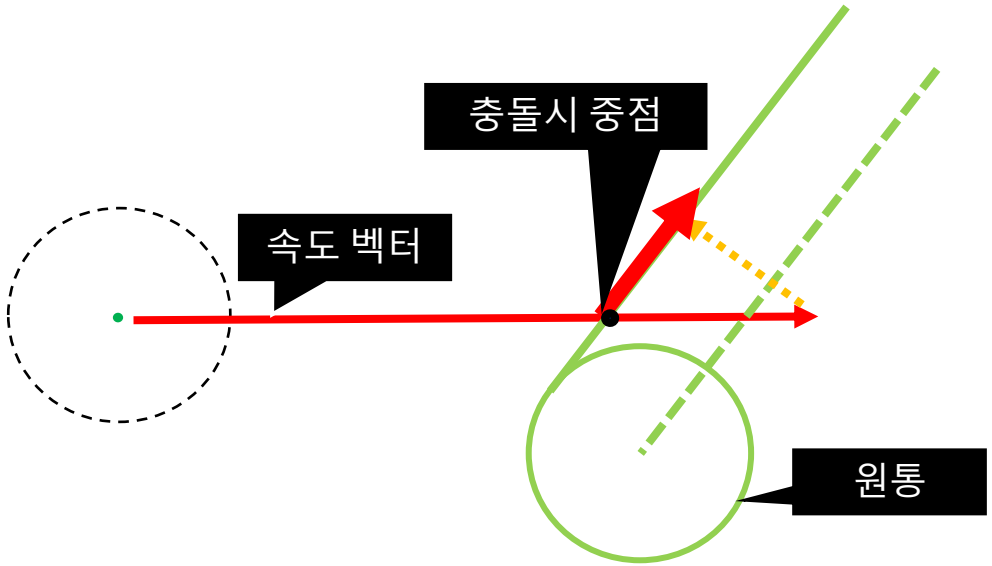
움직이는 구 vs 삼각형 – 간단히 만들기

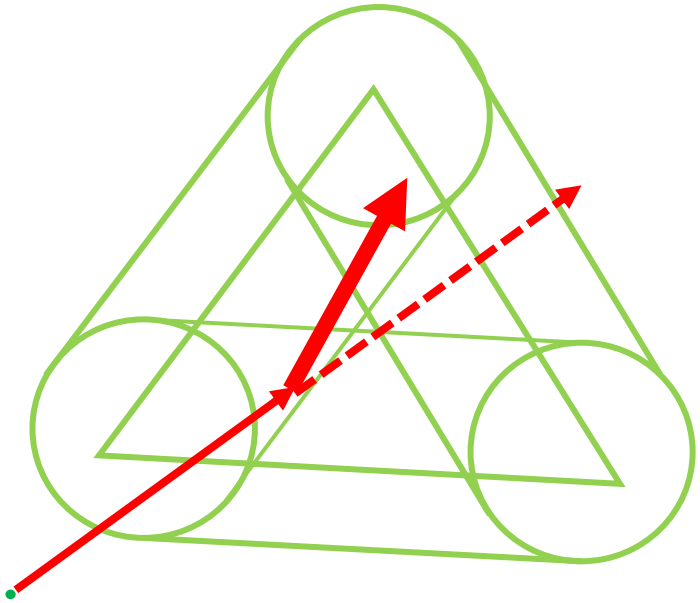
- 구의 궤적은 캡슐이 된다.
- 캡슐은 다루기 까다롭다.
- 구의 반지름을 0으로 만든다.
- 구의 반지름만큼 삼각형을 부풀린다.
- 부풀린 삼각형은
  - 꼭지점 3개 -> 구 3개
  - 모서리(변) 3개 -> 원통 3개
  - 삼각형 면 -> 삼각형의 up벡터 방향으로 반지름만큼 이동한 삼각형

캡슐 vs 삼각형 충돌처리 == ray vs (구x3, 원통x3, 삼각형) 충돌처리



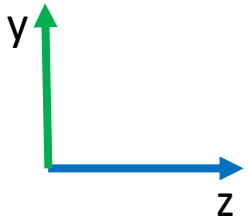


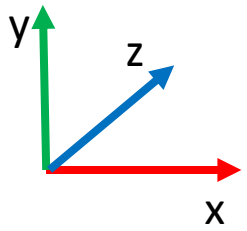


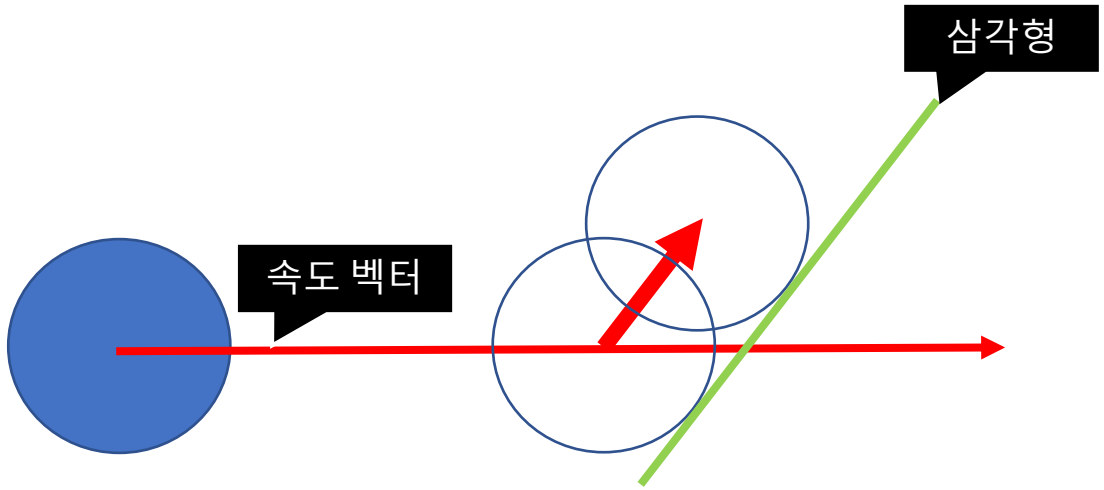


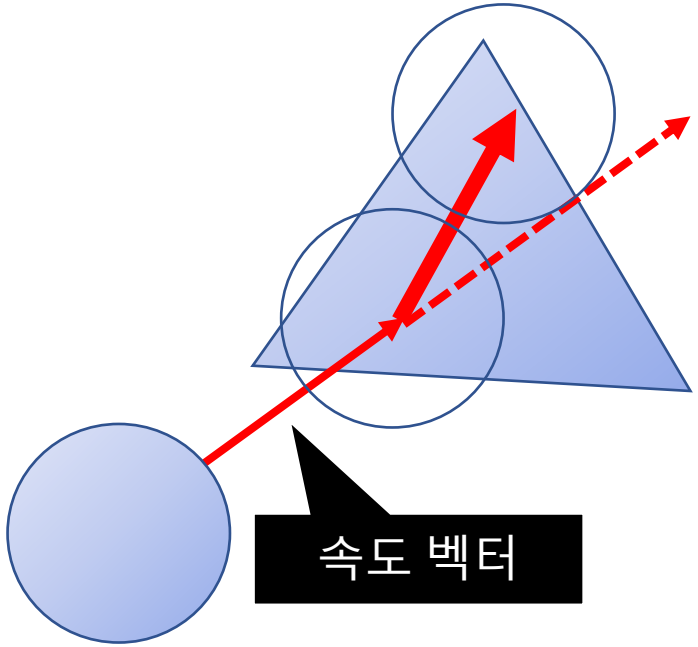
움직이는 구는 점으로, 대상 삼각형을 구의 반지름만큼 부풀린다.

1. 꼭지점 -> 구
2. 모서리 -> 원통
3. 삼각형면 > 면의 up 벡터 방향으로 이동



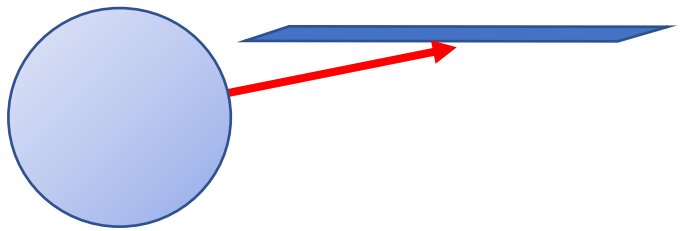


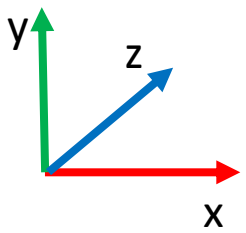


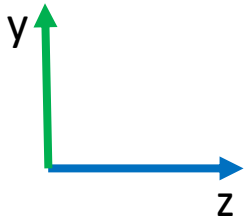


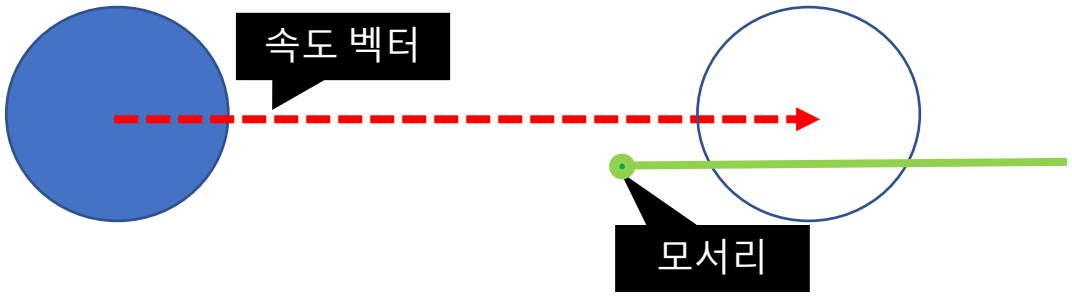
캡슐 vs 삼각형 == ray vs 부풀린 삼각형

구가 삼각형 모서리에 충돌할때

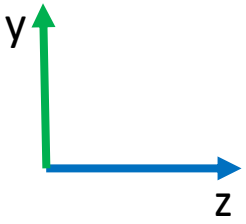


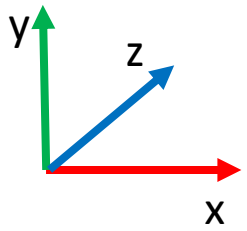


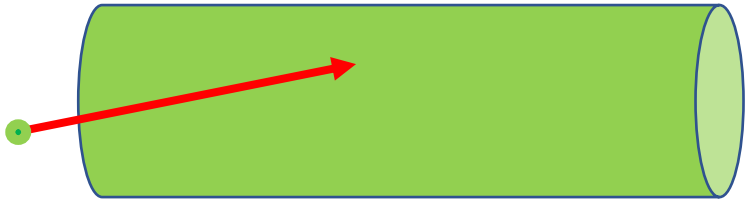


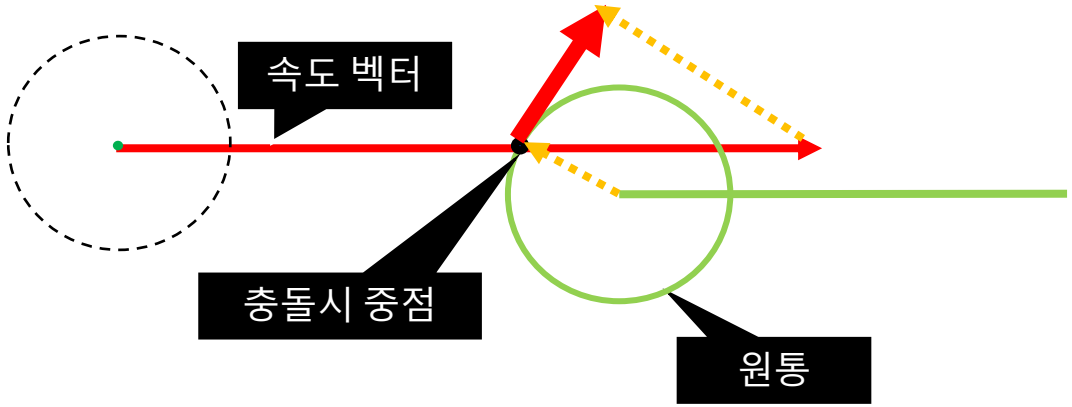


움직이는 구가 모서리와 충돌했을 때

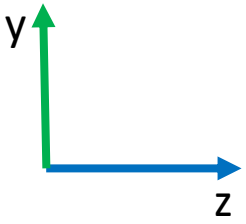


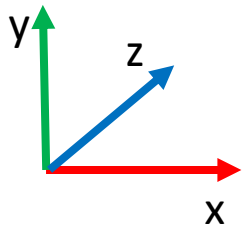


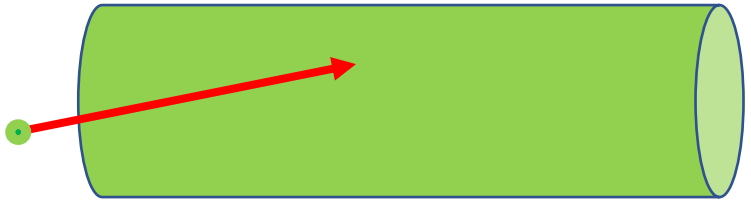


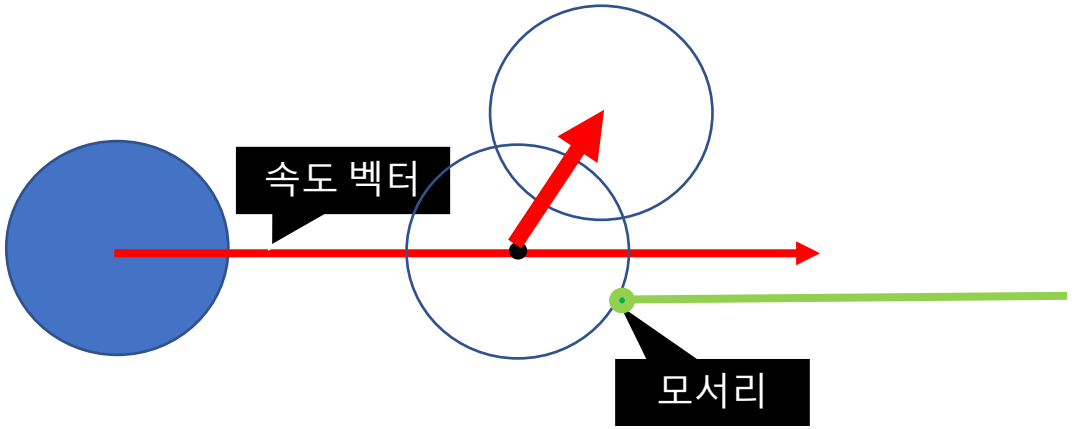


움직이는 구가 모서리와 충돌했을 때



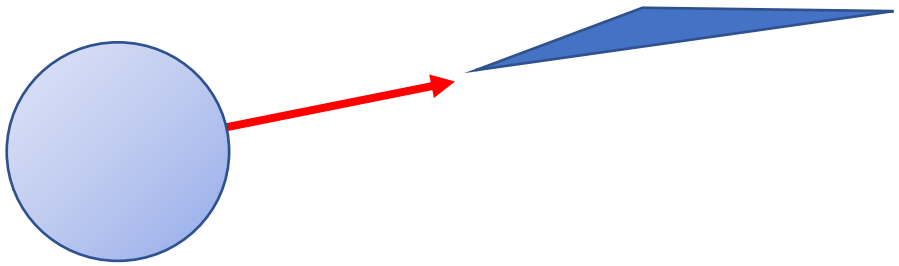


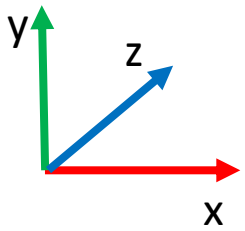


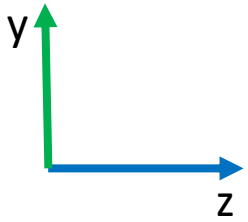


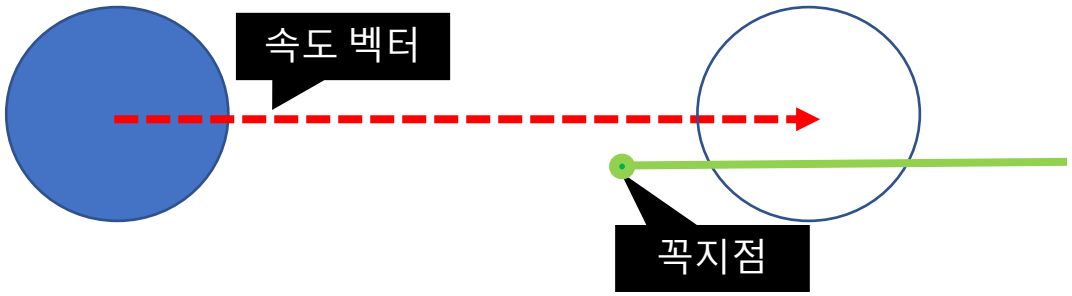
움직이는 구가 모서리와 충돌했을 때

구가 삼각형 꼭지점에 충돌할때

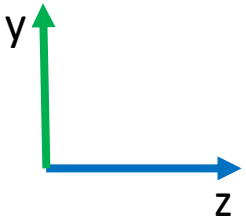


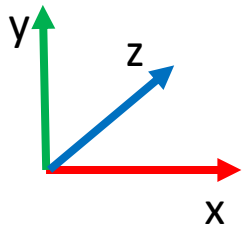


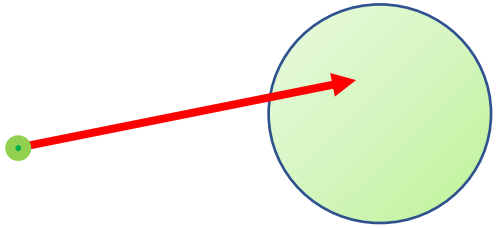


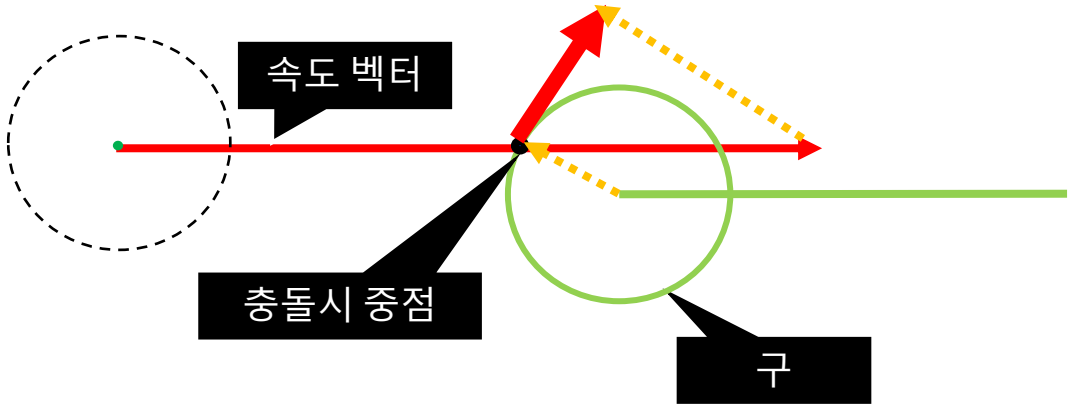


움직이는 구가 꼭지점과 충돌했을 때

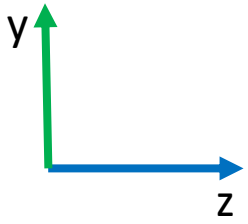


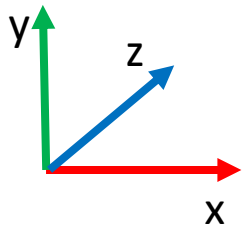


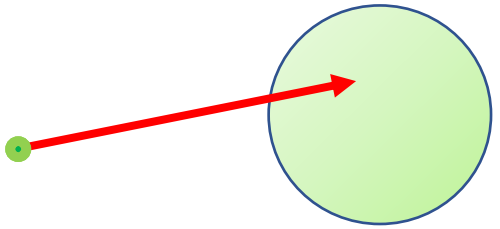


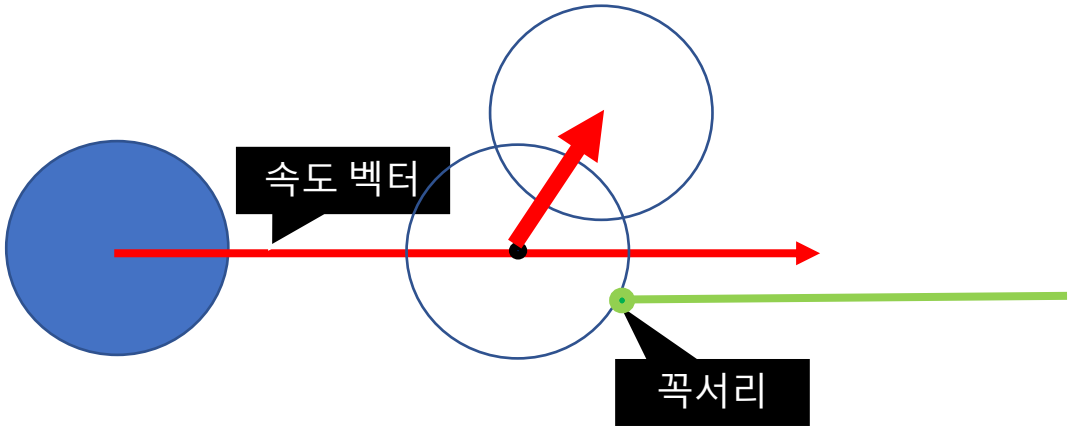


움직이는 구가 꼭지점과 충돌했을 때









움직이는 구가 꼭지점과 충돌했을 때

충돌처리 수순

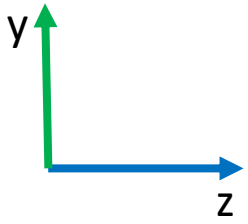
1. 움직이는 구와 충돌할 가능성이 높은 삼각형 선별
2. 움직이는 구를 ray로 변환
3. 대상 삼각형을 부풀린 삼각형으로 변환 (구x3 , 원통x3, 면)
4. Ray vs 변환된 도형(구x3 , 원통x3, 면) 충돌 검사
5. (구x3 , 원통x3, 면) 중 충돌점이 가장 가까운 (t값이 가장 작은) 도형에 대해서 미끄러짐 벡터 계산
6. 초기 속도벡터와 결과 속도벡터간의 각도, 마찰계수등을 고려해 방향 벡터의 크기를 적당히 축소
7. 새로운 벡터로 방향을 설정.

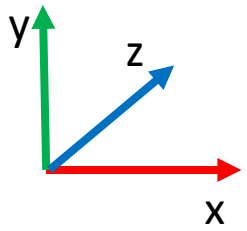
속도 벡터의 크기가 0에 가까워질때까지 반복

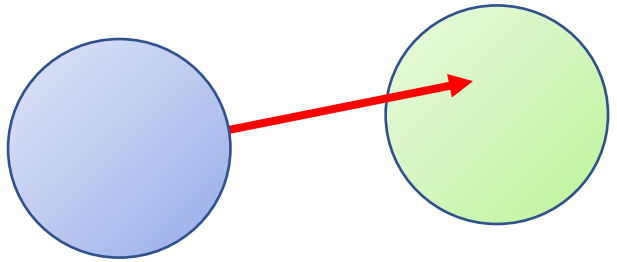
캐릭터 vs 캐릭터 충돌처리

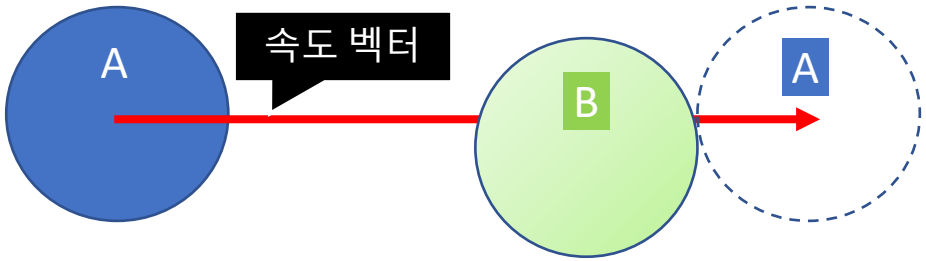
움직이는 구 vs 정지한 구

- 캡슐 vs 삼각형 충돌처리와 기본적으로는 같다.
- 움직이는 구의 반지름을 0으로 만든다.
- 움직이는 구의 반지름을 정지한 구의 반지름에 합산한다.
- 움직이는 구는 ray로, 정지한 구는 더 커진 구가 된다.
- ray vs 구로 충돌처리를 진행한다.

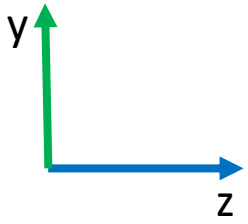


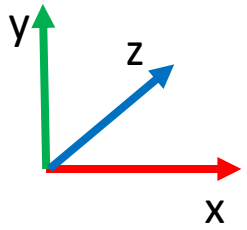


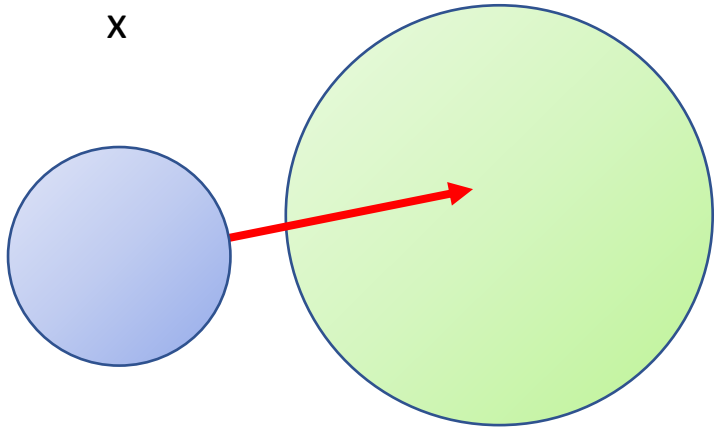


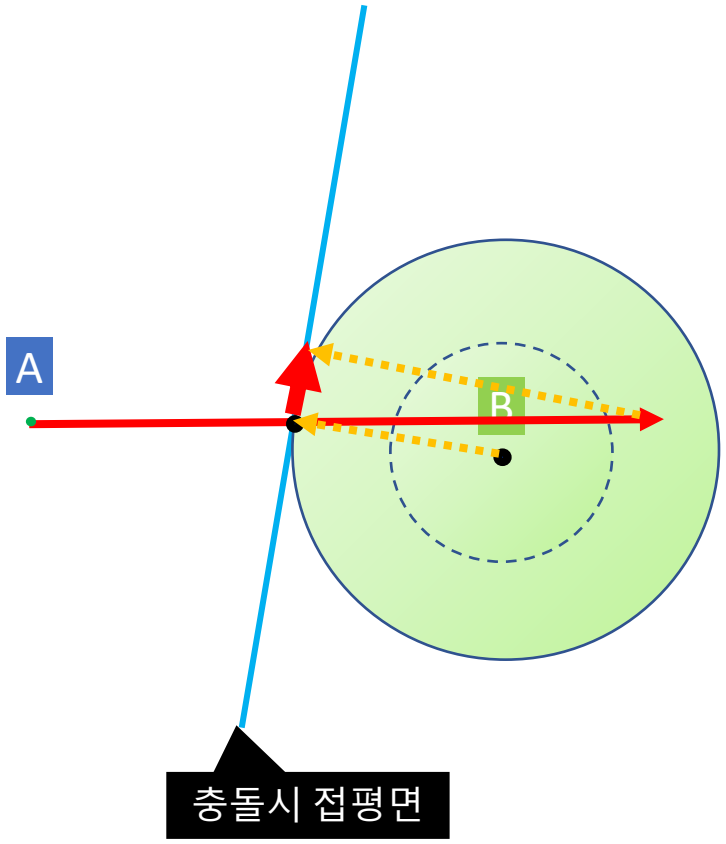


움직이는 구가 정지한 구와 충돌했을 때

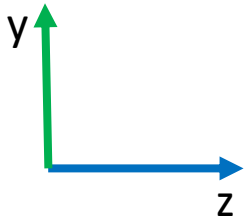


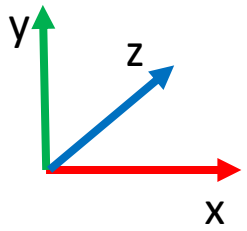


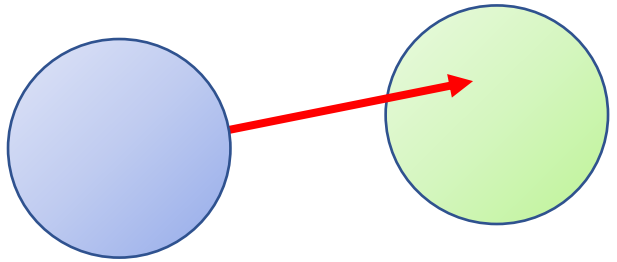


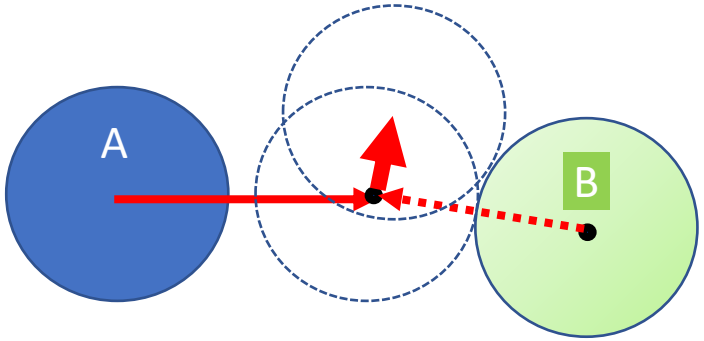


움직이는 구가 정지한 구와 충돌했을 때





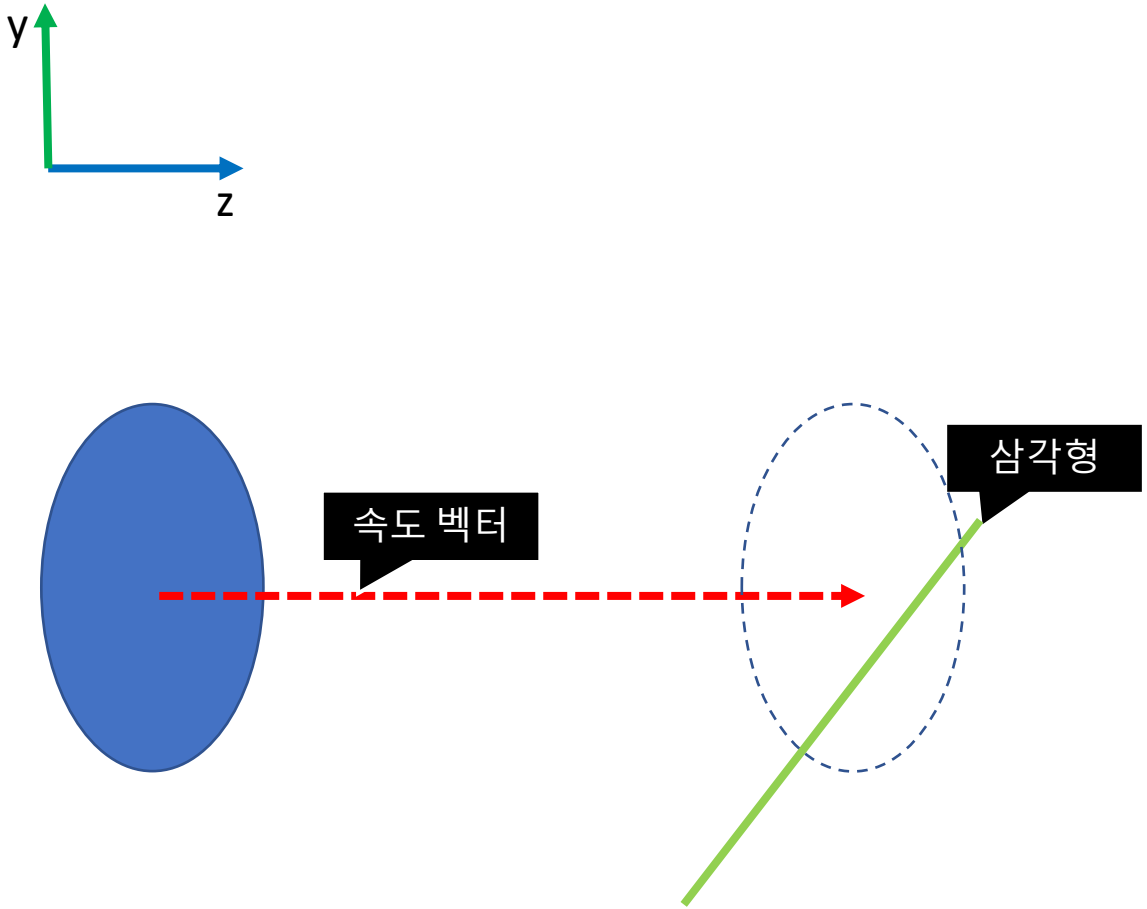


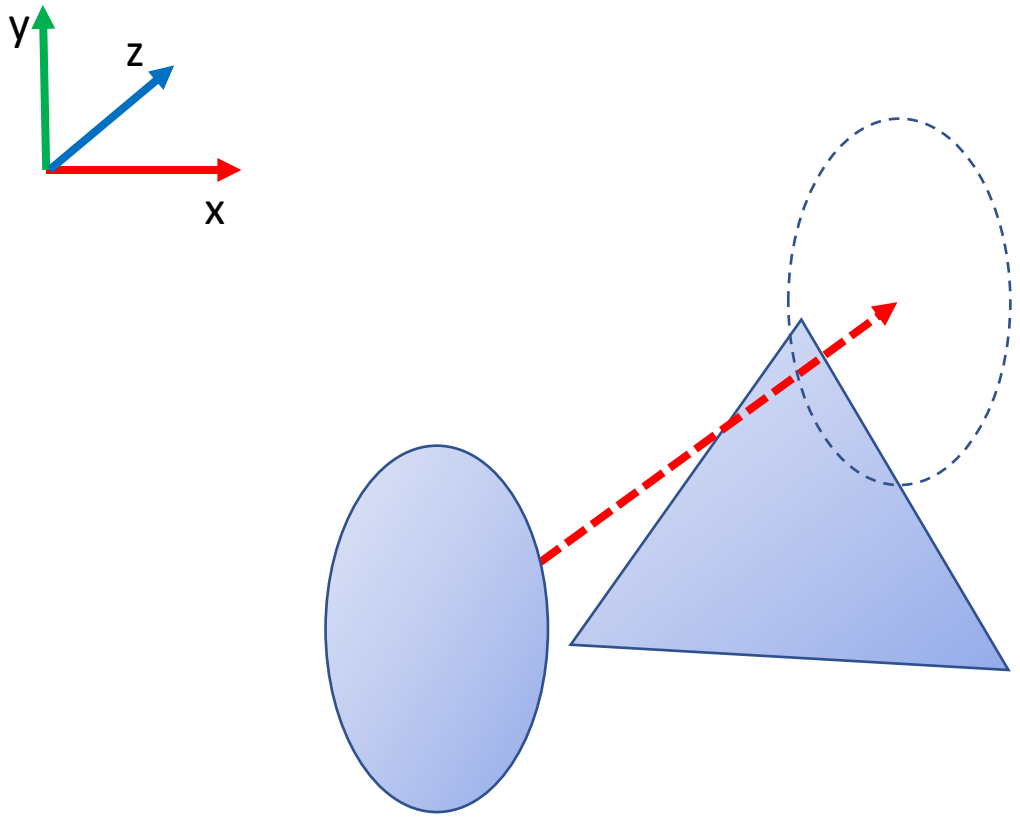


움직이는 구가 정지한 구와 충돌했을 때

타원체 모델

- 캐릭터는 어지간하면 타원체 하나로 표현 가능.
- 구도 타원체로 표현 가능.
- 타원체는 스케일된 구로 표현 가능.





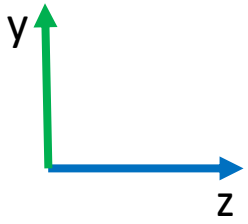
움직이는 타원체가 삼각형과 충돌했을 때

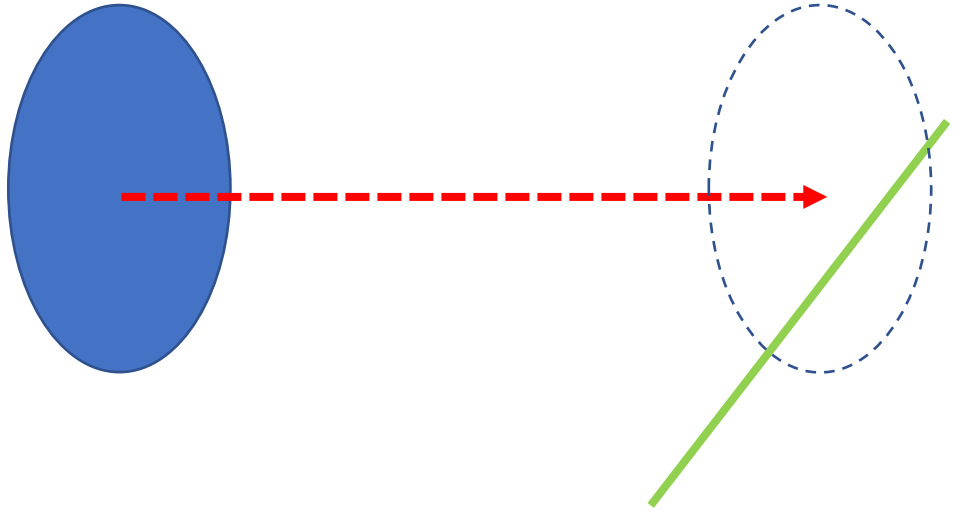
타원체 모델 간단히 하기

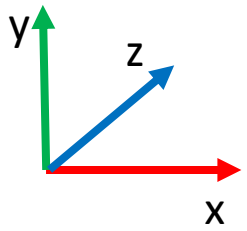
- 공간을 y축으로 일그러뜨리면 구로 타원체를 구현 가능
- 구로 타원체를 표현할 수 있으므로
  - 캐릭터 vs 캐릭터 -> 구 vs 구 로 처리
  - 캐릭터 vs 지형 -> 구 vs 삼각형으로 처리

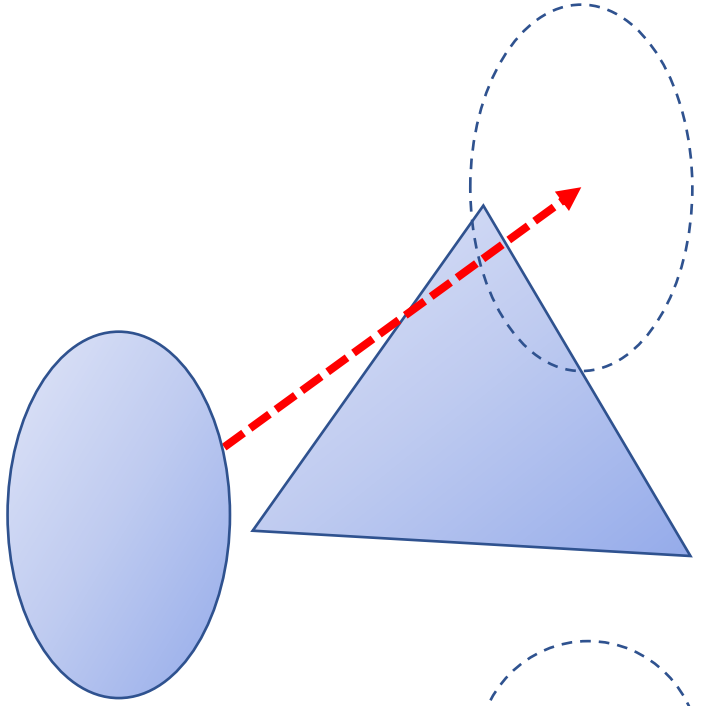
타원체 vs 삼각형

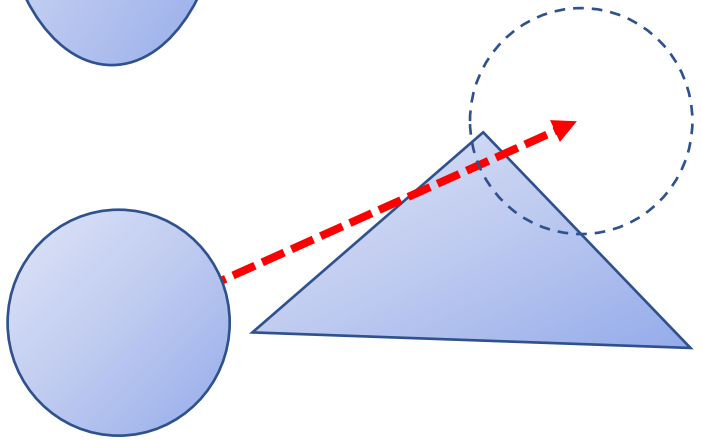
1. 타원체의 짧은 축이 반지름이 되는 구로 변환
2. 단축 / 장축을 스케일 값으로 대상 삼각형들을 y축으로 스케일
3. 구 vs 삼각형 충돌처리
4. 결과물로 나온 최종 속도 벡터, 최종 위치들을 y축으로 역스케일(y축으로 일그러졌던 월드를 복구)


y
z




y
z
x



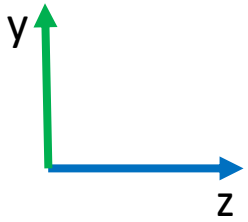


움직이는 타원체 vs 타원체

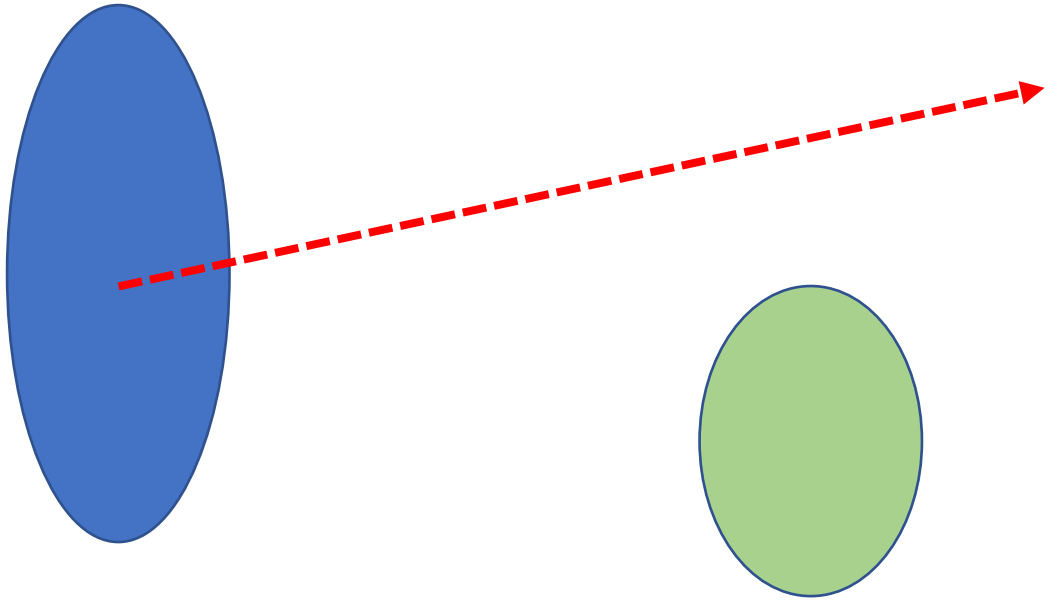
- 캐릭터 충돌처리를 위해서 충돌처리를 하는 그 순간은 현재 계산중인 캐릭터 외에는 모두 멈춰있다고 가정한다.
- 캐릭터는 타원체 하나로 표현한다.
- 움직이는 타원체 vs 다수의 정지한 타원체의 충돌처리가 된다.

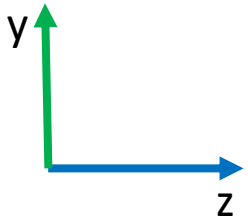
움직이는 타원체 vs 타원체

1. 움직이는 타원체를 A , 정지한 타원체를 B라 한다. A 타원체를 구로 변환.
2. A 타원체의 스케일값으로 B타원체를 스케일
3. 구A의 반지름을 B타원체의 장축/단축 길이에 합산.
4. 이제 ray A와 B타원체간의 충돌처리가 된다.
5. Ray A와 B타원체의 교차 테스트 –> t값을 구한다.
6. t값으로 충돌시 타원체 A의 중점 위치를 구한다.
7. 타원체 B로부터 타원체 A의 중점 위치까지의 접평면을 구한다.

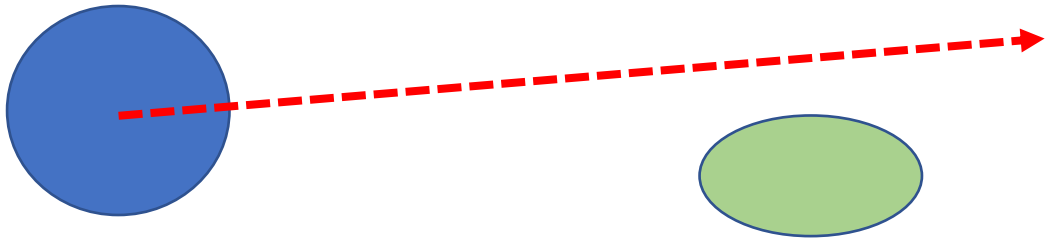


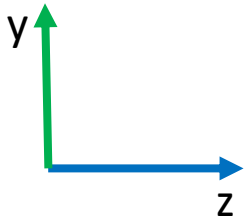
어딘가에서 충돌하여 멈출텐데 이때 타원체의 중점을 구해야한다.
멈췄을때의 중점 위치 = 현재 중점 위치 + 속도벡터 * t

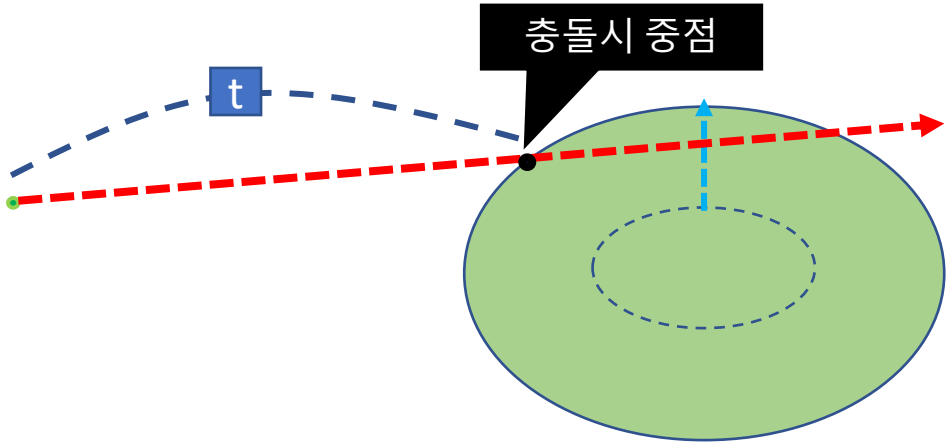




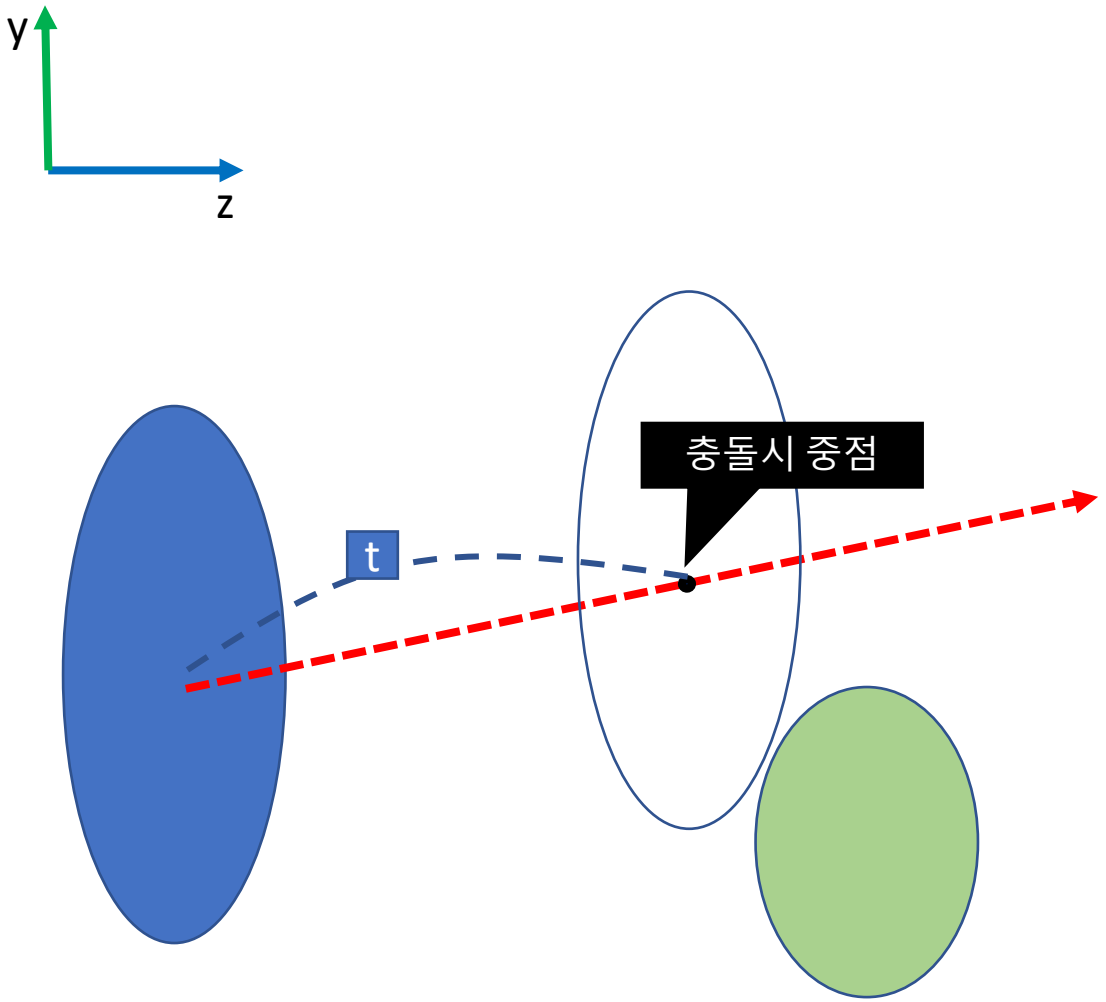
어딘가에서 충돌하여 멈출텐데 이때 타원체의 중점을 구해야한다.
멈췄을때의 중점 위치 = 현재 중점 위치 + 속도벡터 * t

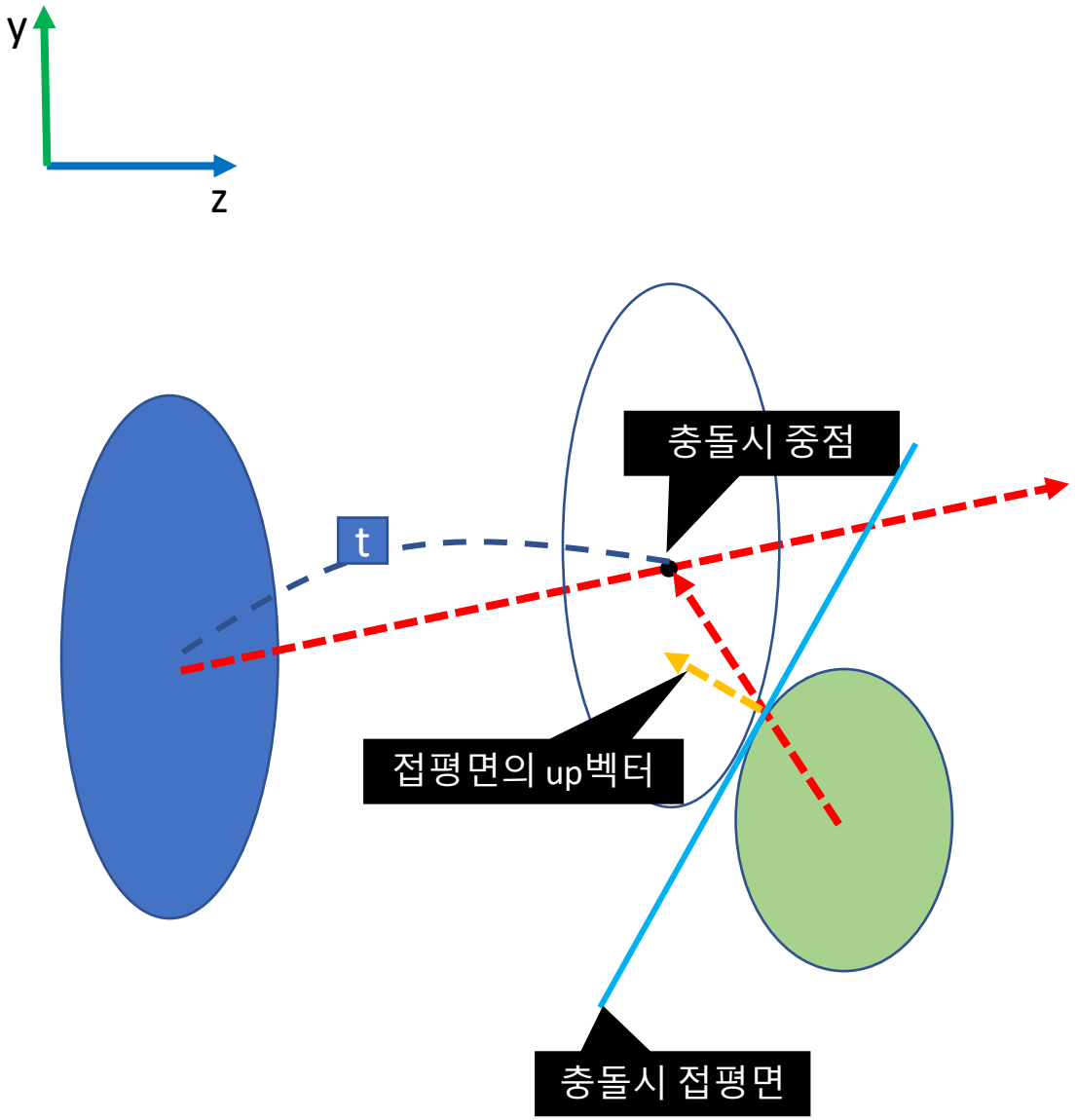


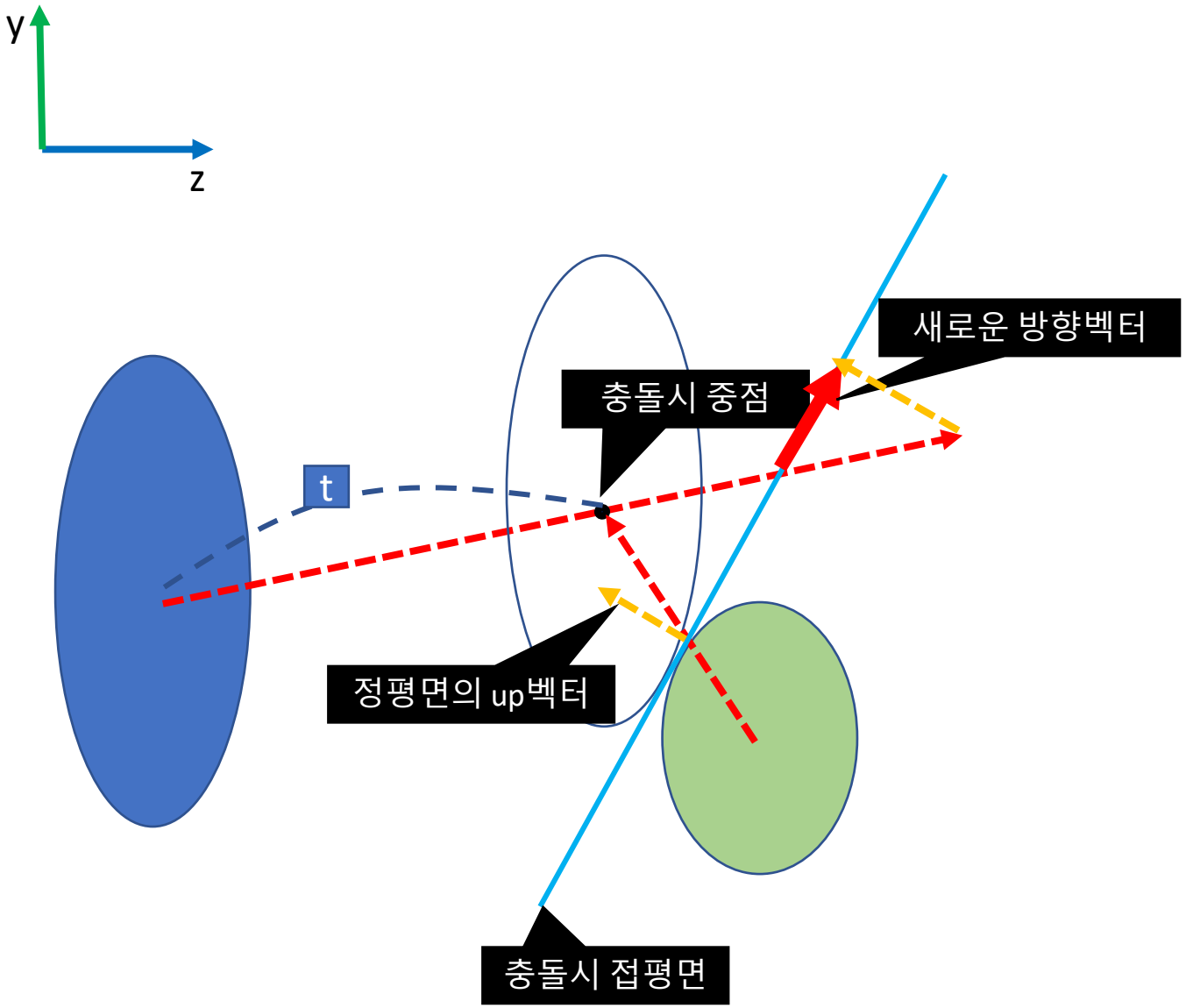




t값 계산 완료


y
z
충돌시 중점
t


y
z
t
충돌시 중점
접평면의 up벡터
충돌시 접평면


y
z
t
새로운 방향벡터
충돌시 중점
정평면의 up벡터
충돌시 접평면

구현시 주의사항

- 이론은 간단하다.
- 직접 만들어보면 간단하지 않다.
- 오차 문제가 크다
  - 결코 수학식대로 깔끔하게 떨어지지 않는다.
  - 꼭지점이나 모서리에 충돌할 때 뚫고 지나가는 경우가 꽤 나온다.
- 속도 문제
  - 한 땀 한 땀 신경 써서 작성해야 한다.
  - 간단한 기하구조를 이용해서 움직이는 타원체와 충돌처리가 필요한 사각형/타원체만 걸러내야 한다.

복셀월드에서 충돌처리

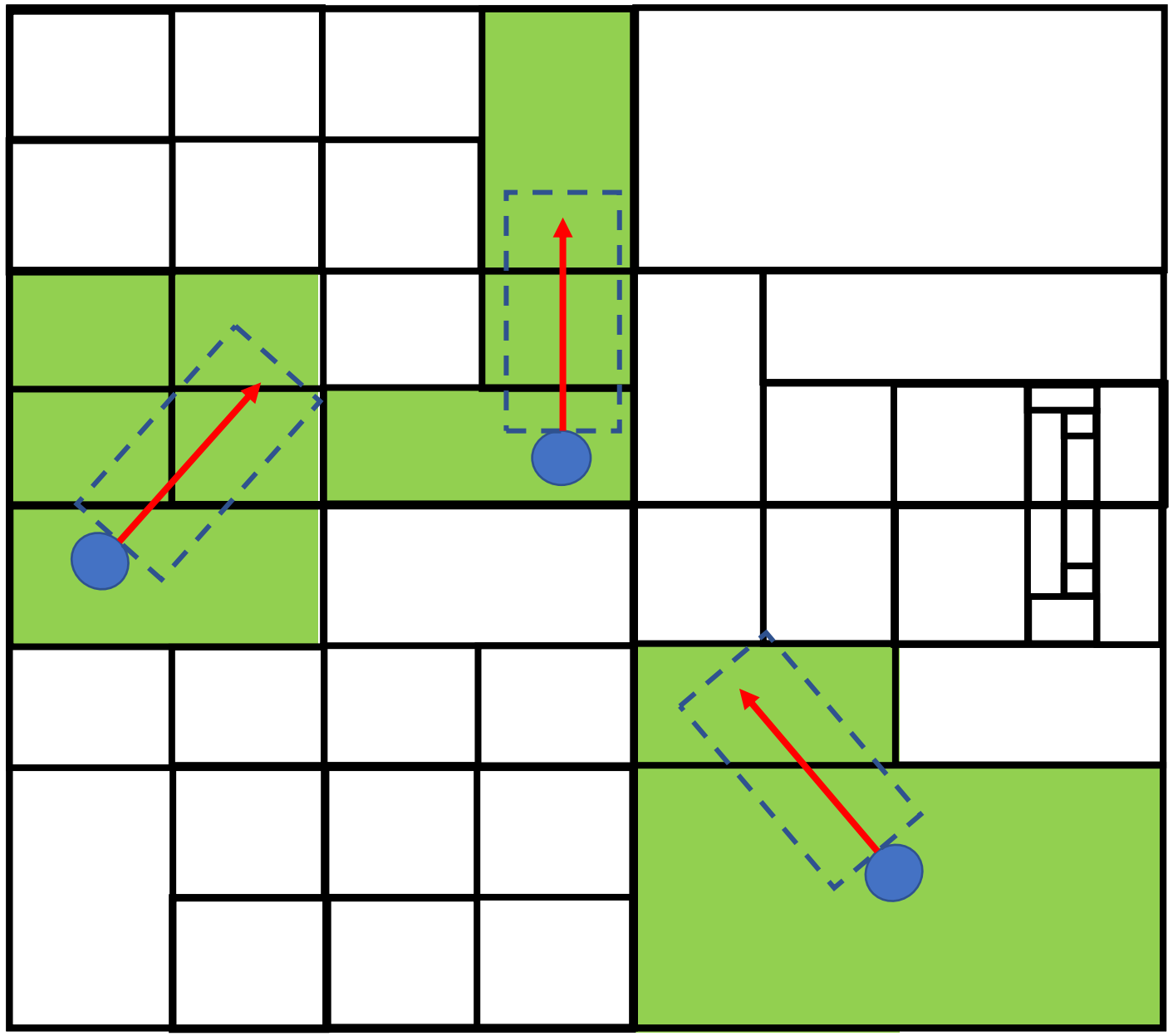
- 복셀맵을 삼각형으로 변환하면 삼각형이 엄청나게 많이 나온다.
- 복셀 한칸을 AABB로 놓고 충돌처리를 하면 더 빠를 것 같지만 AABB 개수가 어마무지하게 나온다.
- 복셀 지형은 실시간으로 변형되므로 충돌처리 전용 매시를 미리 만들어 둘 수 없다.
- 기존 타원체 vs 삼각형 충돌처리를 사용하되 복셀 오브젝트가 변형될 때마다 즉시 최적화된 충돌처리용 삼각형 매시를 갱신한다.

공간 자료구조

- 복셀 오브젝트 – KD Tree
- 캐릭터 – 3D Grid, 메모리 절약을 위한 on demand 할당 테이블

충돌처리를 위한 복셀 오브젝트 및 삼각형 선별

Tree에서 복셀 충돌처리를 위한 오브젝트 선별



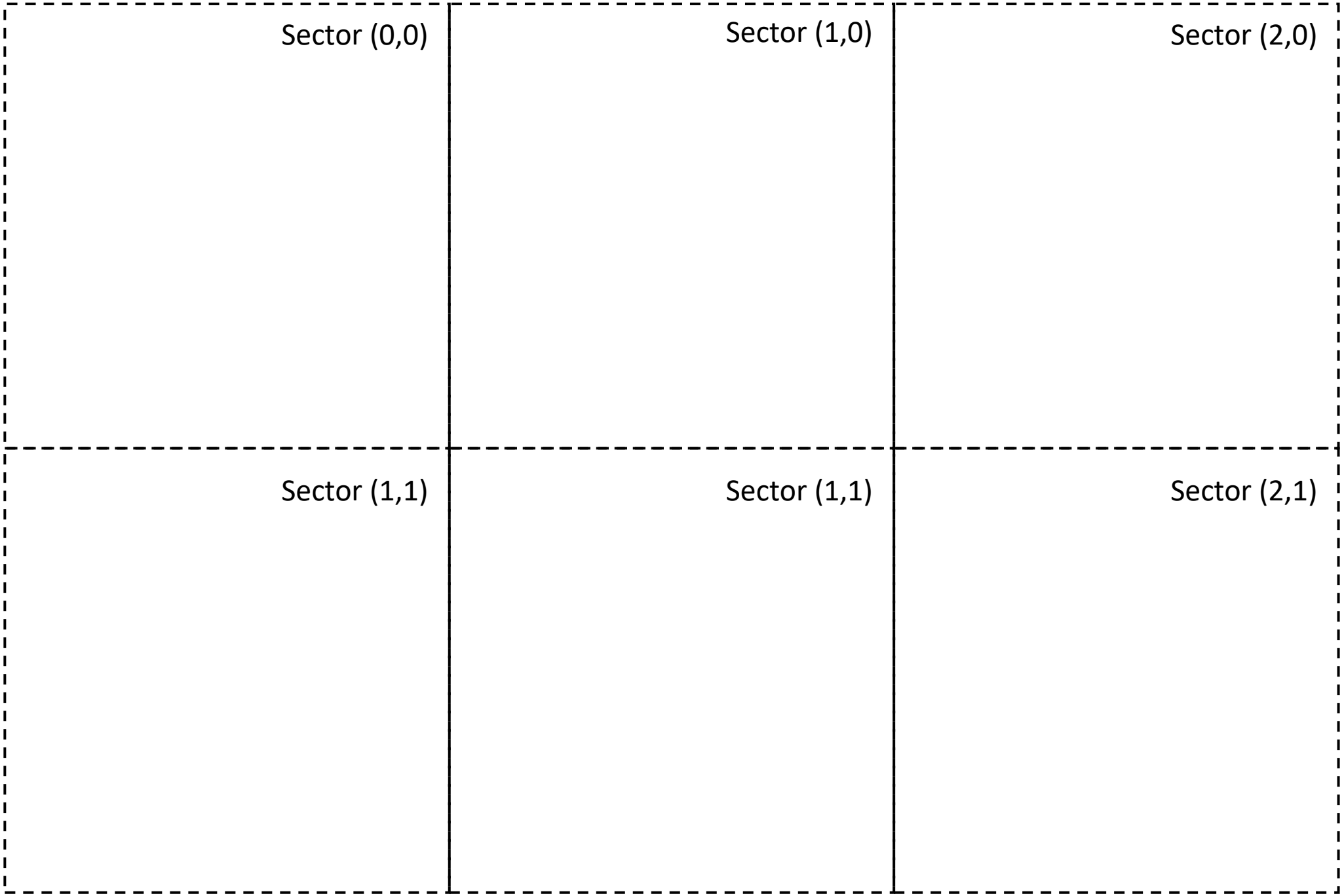
1. 타원의 위치와 반지름 속도 벡터를 이용하여 6면 frustum을 만든다.
2. 6면 frustum에 교차하는 복셀 오브젝트를 선별
3. 복셀 오브젝트로부터 삼각형 추출. 이때 속도벡터에 대해 뒤집힌 삼각형 제외.
4. 멀티 스레드로 작업
   - 충돌처리 코드가 멀티스레드로 돌아가고 이 때 대상 삼각형을 선별하므로 결과적으로 멀티스레드로 돌아간다.
   - 즉 충돌처리 삼각형을 선별하는 코드는 thread safe해야한다.

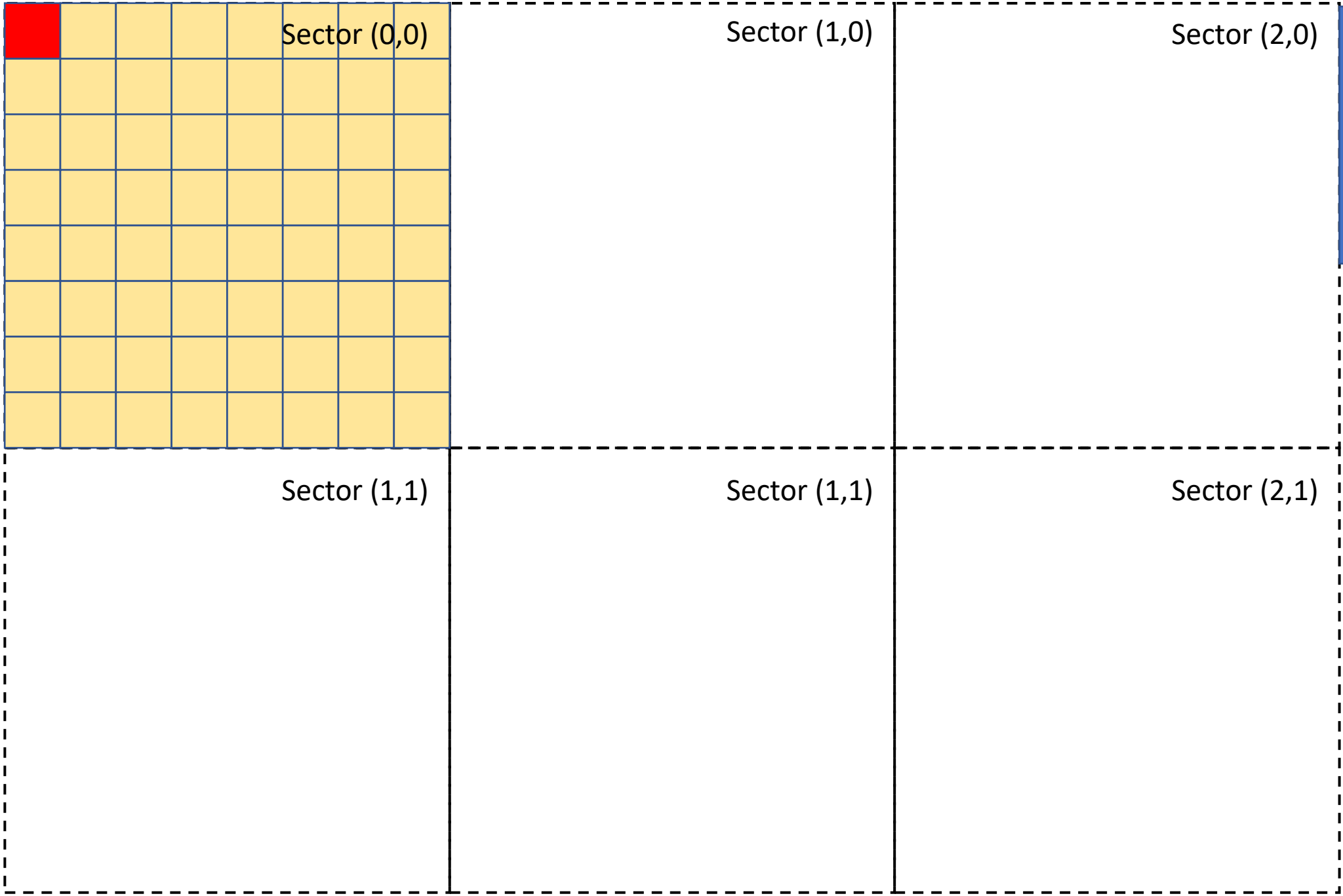
대상 타원체 검출(캐릭터 vs 캐릭터)

- On demand 3D Table 자료구조

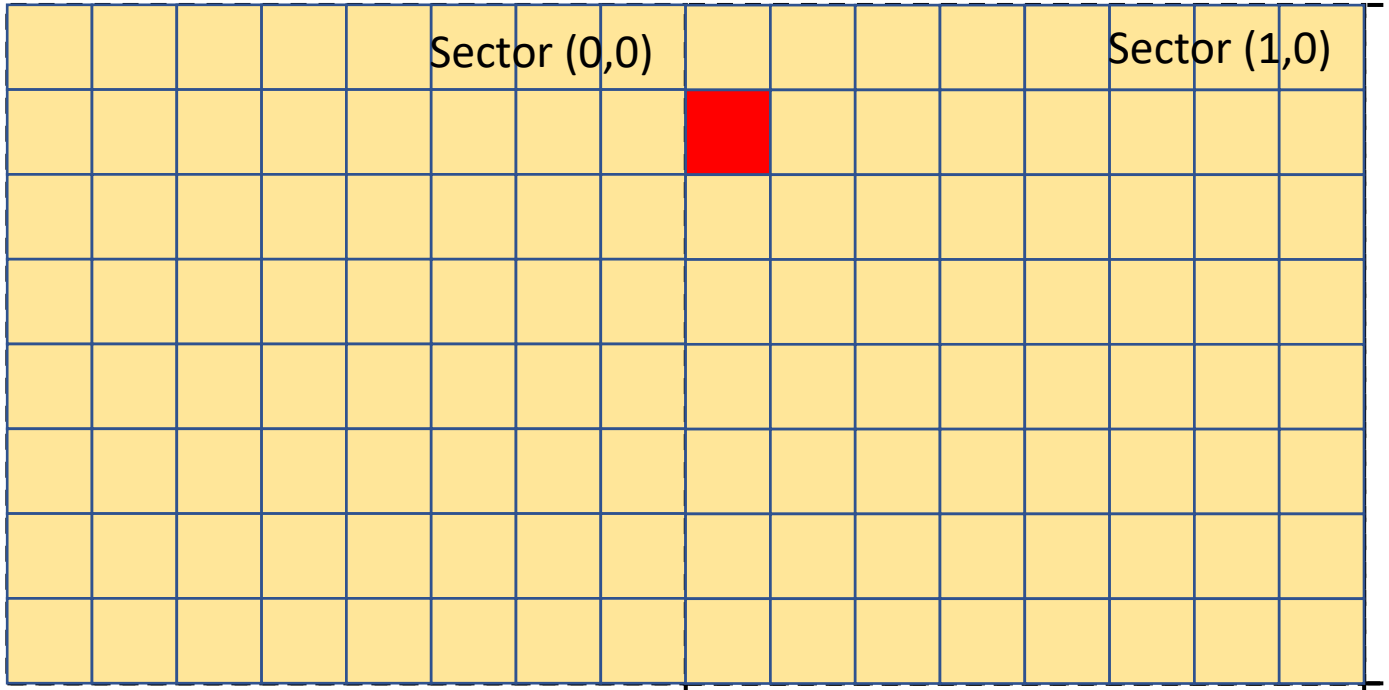
On demand 배열

- 거대한 배열이 필요할 때
  - 그러나 그 배열의 일부만 사용되고
  - 어느 영역이 사용될지 알 수 없다.
- NxNxN씩 잘라서 배열 그룹을 만든다.
- 각 배열그룹 내의 메모리는 할당하지 않는다
- Access(), Alloc()등의 함수로 요청이 들어오면 그때 할당하고 포인터를 돌려준다.
- Paging기법과 유사
- 월드 공간을 그리드로 구현할 때 유용함.


Sector (0,0)
Sector (1,0)
Sector (2,0)
Sector (1,1)
Sector (1,1)
Sector (2,1)

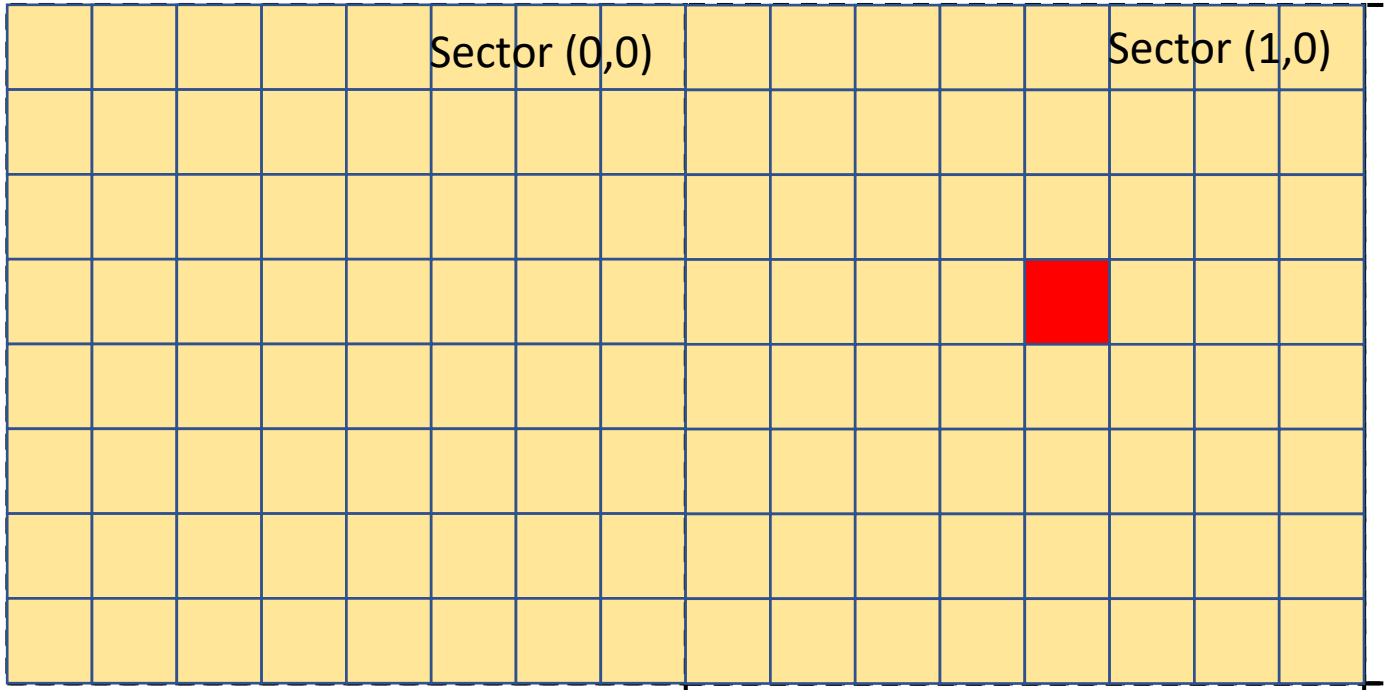


(0,0)에 억세스할때
ptr = AcquirePtr(0,0)
-> 8x8 메모리 할당
-> return sector (0,0)의 base
ptr + 0



(0,0)에 억세스할때
ptr = AcquirePtr(0,0)
-> 8x8 메모리 할당
-> return sector (0,0)의 base
ptr + 0

(8,1)에 억세스할때
ptr = AcquirePtr(8,1)
-> 8x8 메모리 할당
-> return sector (1,0)의 base
ptr + 8



Sector (2,0)

Sector (1,1)

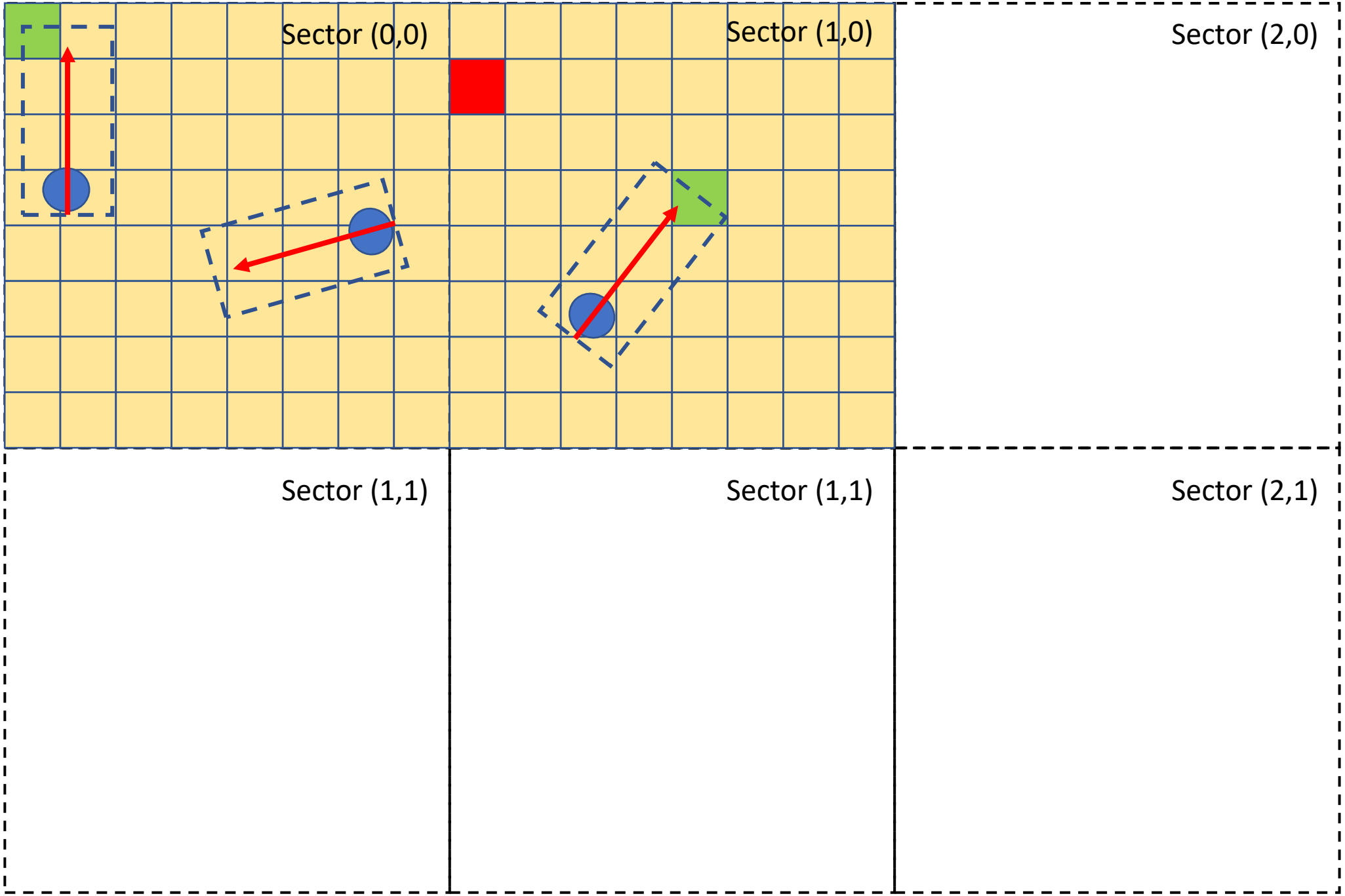
Sector (1,1)

Sector (2,1)

(0,0)에 억세스할때
ptr = AcquirePtr(0,0)
-> 8x8 메모리 할당
-> return sector (0,0)의 base
ptr + 0

(8,1)에 억세스할때
ptr = AcquirePtr(8,1)
-> 8x8 메모리 할당
-> return sector (1,0)의 base
ptr + 8

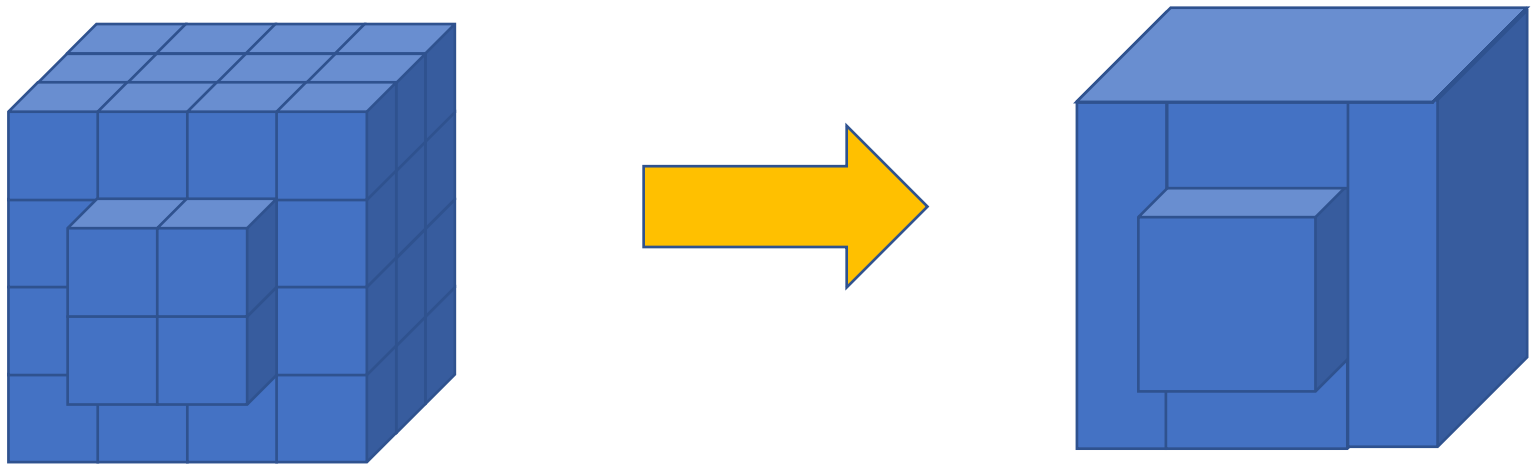
(12,3)에 억세스할때
ptr = AcquirePtr(8,1)
-> 8x8 메모리 할당
-> return sector (1,0)의 base
ptr + 8


Sector (0,0)
Sector (1,0)
Sector (2,0)
Sector (1,1)
Sector (1,1)
Sector (2,1)

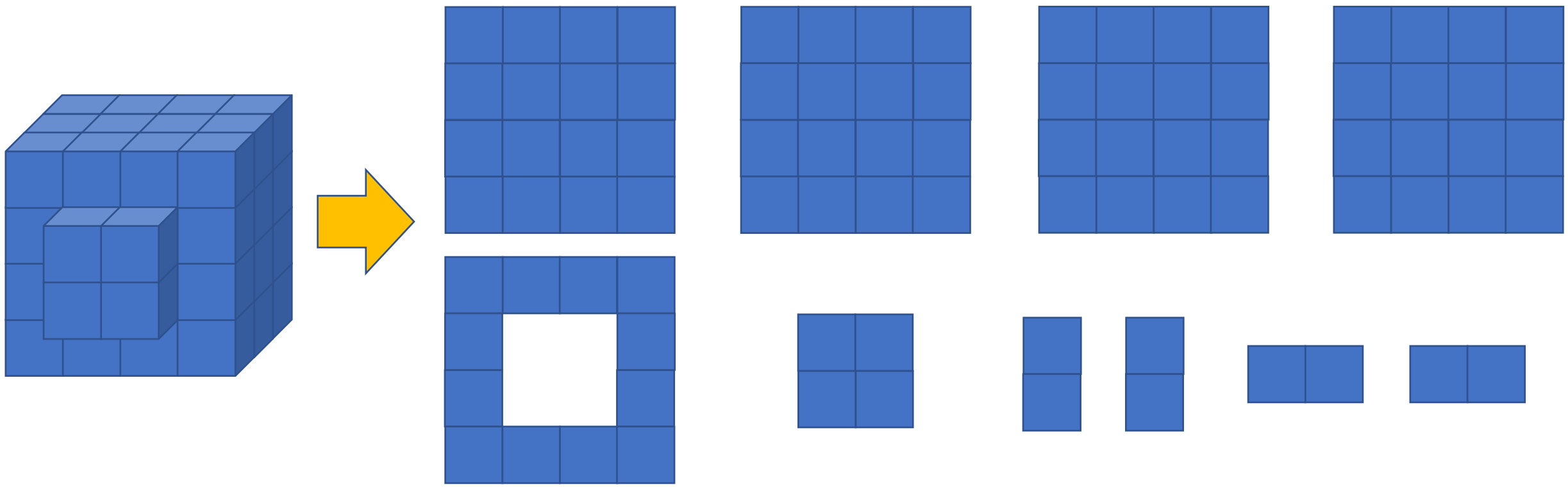
기하구조 -> 충돌처리용 삼각형리스트

- 인접한 복셀을 병합해서 면을 줄일 수 있다.
- 효율적인 충돌처리와 picking을 위해서 최대한 복셀을 병합하여 시스템 메모리 상에 유지한다.

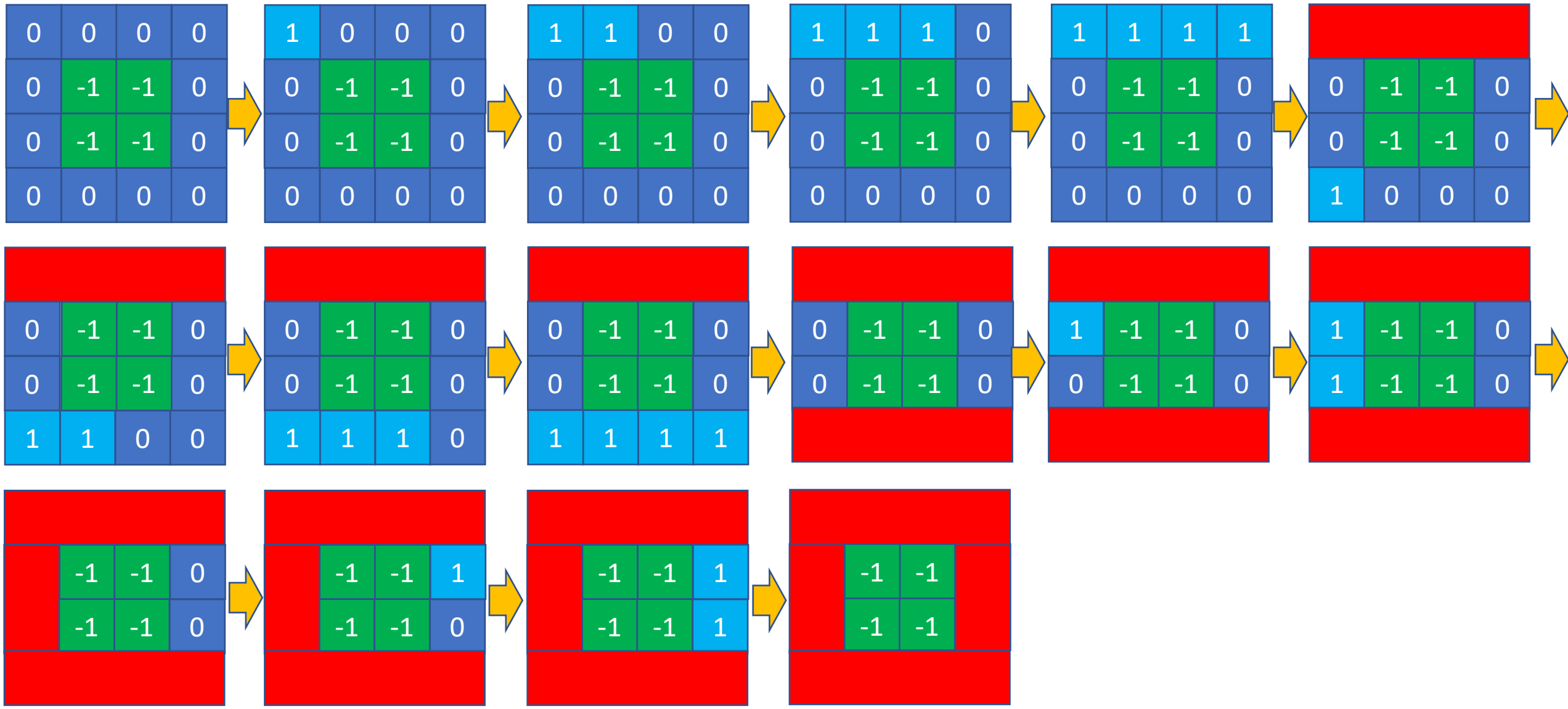
충돌처리를 위해 인접한 면들을 병합하여 최적화된 매시로 변환.



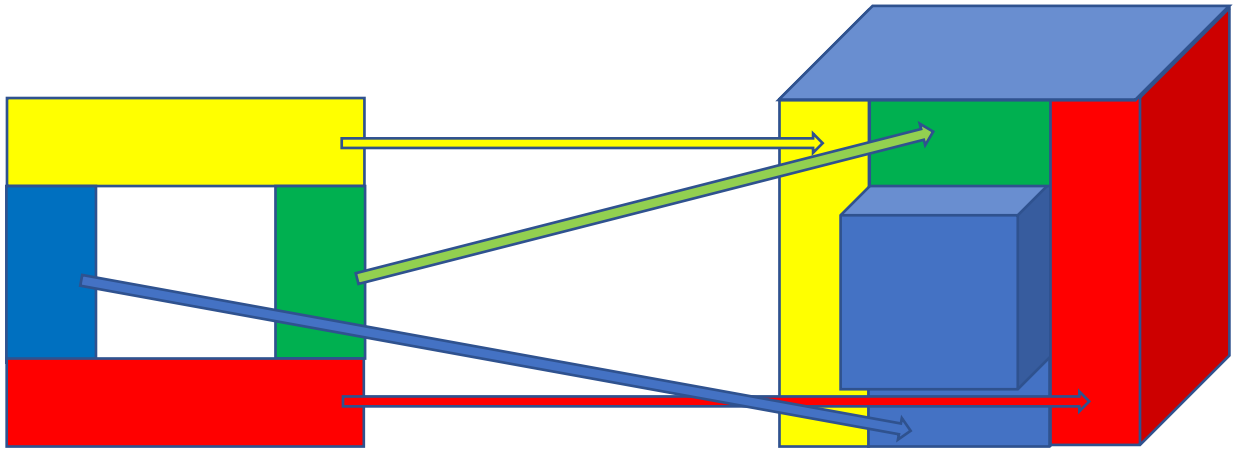
사각형들을 분해해서 같은 평면 방정식에 따라 그룹화



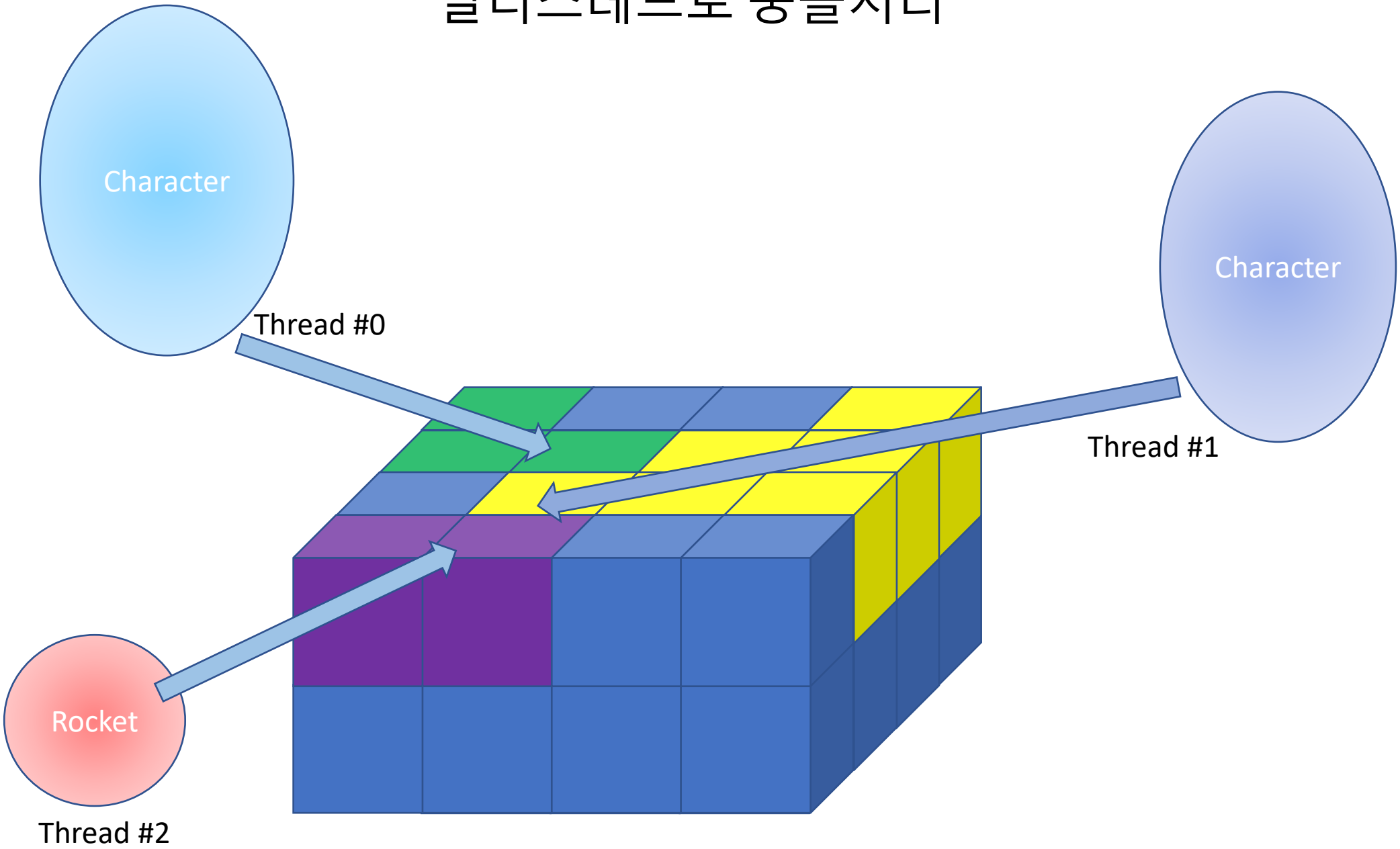
2D 비트맵상에서 x,y축으로 한칸씩 성장



오브젝트의 3D위치, 분해된 사각형 그룹의 면방정식이 있으므로 2D 비트맵을 다시 3D 사각형 리스트로 변환할 수 있다.



멀티스레드로 충돌처리



결론

- 이론적인 내용은 리얼타임 렌더링 책에 다 나와있습니다.
- 이론은 쉬운데 구현하려면 꽤 빡셔요.
- 멀티 스레드와 공간분할에 대한 연구는 따로 해야합니다.