〈유니티 cast 정리〉

cast는 종류가 많다. 간단하게 정리해보자면

Raycast: 길이 없는 선을 쏘아 충돌체를 검사. 가장 가볍고 자주 사용됨

|

SphereCast: 구를 쭉 쓸어보는 느낌 구를 이동시켜서 닿는 첫 충돌을 구함 (얕게는 '두께 있는 레이')

|

CapsuleCast: 캡슐(원통 + 반구)을 쓸어봄 사람형(직립) 충돌 체크에 자주 사용

|

BoxCast: 직육면체(박스)를 쭉 쓸어보는 검사 크기·회전 가짐. 벽/박스 형태 충돌에 유리

|

CircleCast (2D) / BoxCast (2D): 2D 전용 버전들(Physics2D). CircleCast = 2D에서의 SphereCast 역할

이러한 역할들 다음장부터 상세히 설명한다.

[RayCast(3D)]

|

{개념}

1. 광선(무한히 얇은 선)을 한 방향으로 쏘아 충돌체가 있나 검사
2. 가장 가볍고 빠름. “시선(line of sight)” 검사, 조준선, 총알 궤적 등에 사용

|

{주요 파라미터}

1. origin (Vector3): 시작점
2. direction (Vector3): 쏘는 방향(정규화 권장)
3. maxDistance (float): 검사 거리
4. layerMask: 검사할 레이어
5. QueryTriggerInteraction: 트리거 검사 여부
6. out RaycastHit hit: 충돌 정보

|

{주의사항}

1. 두께가 전혀 없어서 얇은 장애물을 놓칠 수 있음
2. Rigidbody 연속 충돌 방지와 다름 그래서 실제 이동엔 SweepTest 고려
3. 정규화된 방향 필수 그래서 distance와 혼동 방지

|

{예제}

```
Vector3 origin = transform.position;
Vector3 dir = transform.forward;

if (Physics.Raycast(origin, dir, out RaycastHit hit, 100f, LayerMask.GetMask("Default")))
{
    Debug.Log($"Hit {hit.collider.name} at {hit.point}");
}
```

[SphereCast(3D)]

|

{개념}

1. 반지름을 가진 구 모양을 특정 방향으로 쓸어 검사
2. Raycast에 “두께”를 준 것
3. 총알/투사체가 빠르게 이동할 때 벽을 뚫는 현상(터널링)을 방지하는 데 유용

|

{주요 파라미터}

1. origin (Vector3): 시작 구의 중심
2. radius (float): 구 반지름
3. direction (Vector3): 이동 방향
4. maxDistance (float): 검사 거리
5. orientation 없음 (구는 회전 불필요)
6. out RaycastHit hit: 충돌 정보

|

{주의사항}

1. 시작 위치가 이미 겹쳐 있으면 distance=0 히트
2. Ray보다 연산 비용이 조금 더 큼
3. radius가 크면 얕은 경사면도 충돌로 잡힘

|

{예제}

```
float radius = 0.5f;
Vector3 origin = transform.position;
Vector3 dir = transform.forward;

if (Physics.SphereCast(origin, radius, dir, out RaycastHit hit, 5f))
{
    Debug.Log($"Sphere hit {hit.collider.name}");
}
```

[CapsuleCast(3D)]

|

{개념}

1. 캡슐 모양(원통 + 반구) 을 스윕 검사
2. 주로 사람형 캐릭터(세로로 긴 충돌체)에 유용

|

{주요 파라미터}

1. point1, point2 (Vector3): 캡슐 양 끝 구의 중심
2. radius (float): 캡슐 반지름
3. direction (Vector3): 검사 방향
4. maxDistance (float): 검사 거리
5. out RaycastHit hit: 충돌 정보

|

{주의사항}

1. Ray나 Sphere보다 연산 비용 큼
2. 캐릭터 충돌체 기준으로 point1, point2를 잘 잡아야 정확
3. 시작 겹침 주의

|

{예제}

```
Vector3 point1 = transform.position + Vector3.up * 0.5f;
Vector3 point2 = transform.position + Vector3.up * 1.5f;
float radius = 0.5f;
Vector3 dir = transform.forward;

if (Physics.CapsuleCast(point1, point2, radius, dir, out RaycastHit hit, 2f))
{
    Debug.Log($"Capsule hit {hit.collider.name}");
}
```

[BoxCast(3D)]

|

{개념}

1. 직육면체 모양을 지정한 방향으로 쭉 스윕하며 충돌 검사
2. 박스 형태의 오브젝트가 이동할 때, 앞으로 나아가면 무엇과 부딪힐지 미리 예측할 때 사용

|

{주요 파라미터}

1. center (Vector3): 박스의 시작 중심 위치
2. halfExtents (Vector3): 박스의 반사이즈(각 축의 절반 길이). 주의: halfExtents를 주는 것을 잊지 말자(전체 크기 아님)
3. direction (Vector3): 이동 방향(보통 정규화된 벡터 권장)
4. orientation (Quaternion): 박스의 회전(박스가 어느 쪽을 향하는지)
5. maxDistance (float): 최대 스윕 거리(얼마만큼 밀어볼지)
6. layerMask (int) / QueryTriggerInteraction: 검사 대상 레이어나 트리거를 포함할지 여부
7. out RaycastHit hit: 충돌 정보를 받을 수 있음(맞은 콜라이더, 거리, 노말 등)

|

{주의사항}

1. halfExtents와 full size 혼동: halfExtents는

각 축의 반길이임

2. 시작부터 콜라이더와 겹치는 경우: 겹쳐 있으면 거리가 0으로 감지되거나 즉시 히트가 발생함, 시작중첩을 확인하려면 Physics.OverlapBox로 사전 검사

3. 정규화된 방향 권장: 방향 벡터를 정규화(normalize)하지 않으면 maxDistance 해석이 혼동될 수 있음

4. 회전 고려: 박스 회전을 제대로 주지 않으면 의도와 다른 모양으로 검사됨(특히 비대칭 박스)

5. 성능: Raycast < SphereCast < BoxCast/CapsuleCast 순으로 비용이 커지는 경향. 빈번한 호출이면 레이어마스크 활용, NonAlloc 사용 권장

6. 실제 물리 시뮬레이션과 다름: BoxCast는 단순 쿼리. Rigidbody의 연속 충돌 감지(Continuous collision detection)와는 목적이 다름. Rigidbody 자체를 sweep하려면 Rigidbody.SweepTest 같은 걸 고려

|

{예제}

```
Vector3 center = transform.position;
Vector3 halfExtents = new Vector3(0.5f, 1.0f, 0.5f);
Vector3 dir = transform.forward;
float maxDist = 2f;
int layerMask = LayerMask.GetMask("Default");

if (Physics.BoxCast(center, halfExtents, dir, out RaycastHit hit,
                transform.rotation, maxDist, layerMask, QueryTriggerInteraction.Ignore))
{
    Debug.Log($"Hit {hit.collider.name} at distance {hit.distance}");
    // hit.point, hit.normal 등 사용 가능
}
else
{
    transform.position += dir.normalized * maxDist;
}
```

[CircleCast(2D)]

|

{개념}

1. 2D 물리(Physics2D) 에서 구의 스윕 검사
2. 3D의 SphereCast와 동일한 역할

|

{주요 파라미터}

1. origin (Vector2): 원의 중심
2. radius (float): 원 반지름
3. direction (Vector2): 검사 방향
4. distance (float): 검사 거리
5. layerMask: 검사할 레이어
6. out RaycastHit2D hit: 충돌 정보

|

{주의사항}

1. 시작 위치 겹침 시 distance=0 히트
2. Collider2D 전용
3. 3D 물리와 혼용 불가

|

{예제}

```
Vector2 origin = transform.position;
Vector2 dir = Vector2.right;
float radius = 0.5f;

RaycastHit2D hit = Physics2D.CircleCast(origin, radius, dir, 3f, LayerMask.GetMask("Default"));
if (hit.collider != null)
{
    Debug.Log($"CircleCast hit {hit.collider.name}");
}
```

[BoxCast(2D)]

|

{개념}

1. 2D 물리에서 직사각형(Box) 을 방향으로 스윕
2. 3D BoxCast의 2D 버전

|

{주요 파라미터}

1. origin (Vector2): 중심점
2. size (Vector2): 가로/세로 크기 (반사이즈 아님)
3. angle (float): 회전(도 단위)
4. direction (Vector2): 검사 방향
5. distance (float): 검사 거리
6. out RaycastHit2D hit: 충돌 정보

|

{주의사항}

1. size는 전체 크기. (3D는 halfExtents인데 다름)
2. Collider2D 전용
3. 3D BoxCast와 혼동 주의

|

{예제}

```
Vector2 origin = transform.position;
Vector2 size = new Vector2(1f, 2f);
float angle = transform.eulerAngles.z;
Vector2 dir = Vector2.up;

RaycastHit2D hit = Physics2D.BoxCast(origin, size, angle, dir, 2f, LayerMask.GetMask("Default"));
if (hit.collider != null)
{
    Debug.Log($"BoxCast2D hit {hit.collider.name}");
}
```

[디버깅 코드 모음]

{RayCast(3D)}

```
void OnDrawGizmos()
{
    Gizmos.color = Color.red;
    Gizmos.DrawRay(transform.position, transform.forward * 10f);
}
```

|

{SphereCast(3D)}

```
void OnDrawGizmos()
{
    Gizmos.color = Color.green;
    Gizmos.DrawWireSphere(transform.position, 0.5f);
}
```

|

{CapsuleCast(3D)}

```
void OnDrawGizmos()
{
    Gizmos.color = Color.cyan;
    Gizmos.DrawWireSphere(transform.position + Vector3.up * 0.5f,
0.5f);
    Gizmos.DrawWireSphere(transform.position + Vector3.up * 1.5f,
0.5f);
}
```

|

{BoxCast(3D)}

```
void DrawBoxGizmo(Vector3 center, Vector3 halfExtents, Quaternion
rot, Color col)
{
    Gizmos.color = col;
    Matrix4x4 old = Gizmos.matrix;
    Gizmos.matrix = Matrix4x4.TRS(center, rot, Vector3.one);
    Gizmos.DrawWireCube(Vector3.zero, halfExtents * 2f);
    Gizmos.matrix = old;
}
```

```
void OnDrawGizmosSelected()
{
    Vector3 center = transform.position;
    Vector3 halfExtents = new Vector3(0.5f,1f,0.5f);
    DrawBoxGizmo(center,      halfExtents,      transform.rotation,
Color.cyan);
    Vector3 endCenter = center + transform.forward * 2f;
    DrawBoxGizmo(endCenter,    halfExtents,    transform.rotation,
Color.yellow);
}
```

{CircleCast(2D)}

```
void OnDrawGizmos()
{
    Gizmos.color = Color.yellow;
    Gizmos.DrawWireSphere(transform.position, 0.5f);
}
```

{BoxCast(2D)}

```
void OnDrawGizmos()
{
    Gizmos.color = Color.magenta;
    Gizmos.matrix        =        Matrix4x4.TRS(transform.position,
Quaternion.Euler(0,0,transform.eulerAngles.z), Vector3.one);
    Gizmos.DrawWireCube(Vector3.zero, new Vector3(1f,2f,0f));
}
```