컴퓨터 그래픽스_레이캐스팅

이준

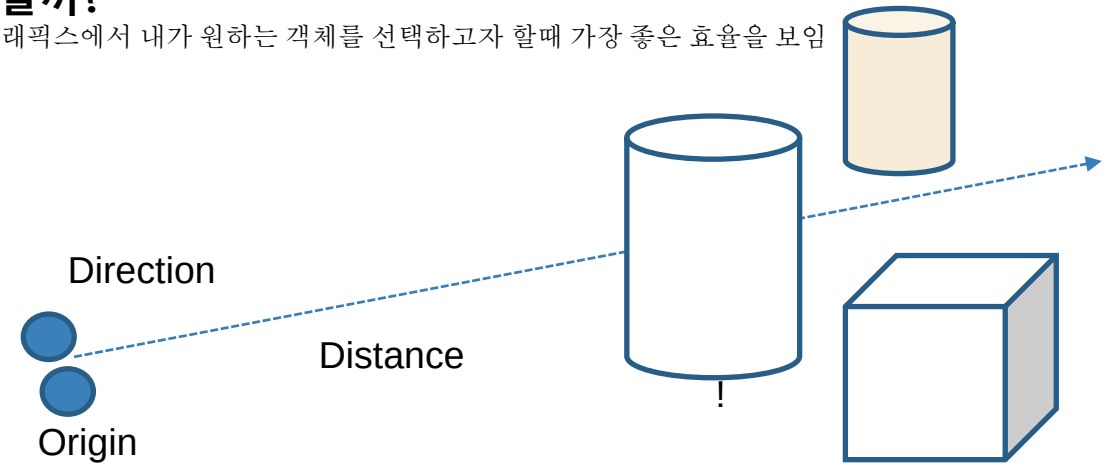
학습 내용

- 레이캐스팅의 원리 및 사용 방법에 대해서 알아보기

레이 캐스팅이란?

- **레이 캐스팅 (Ray Casting)**
  - 3차원 공간에서 어느 한점에서 시작해서 Ray를 정해진 방향으로 이동하여 Ray와 충돌하는 객체를 구하는 방법

- **왜 사용할까?**
  - 3차원 그래픽스에서 내가 원하는 객체를 선택하고자 할때 가장 좋은 효율을 보임



Physics.Raycast(Vector3 origin, Vector3 directyion, RaycastHit hitinfo, float distance, int LayerMask);

레이 캐스팅이란?

- **레이 캐스팅 (Ray Casting)**
  - 3차원 공간에서 어느 한점에서 시작해서 Ray를 정해진 방향으로 이동하여 Ray와 충돌하는 객체를 구하는 방법
- **왜 사용할까?**
  - 3차원 그래픽스에서 내가 원하는 객체를 선택하고자 할때 가장 좋은 효율을 보임

레이 캐스팅이란?

유니티에서 레이 캐스팅 적용

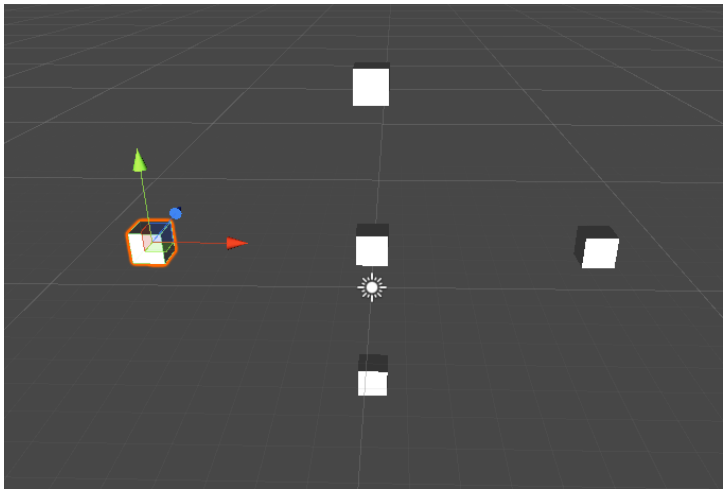
- **Ray (struct)**
  - Ray Cast를 위한 재료중 가장 중요한 Ray의 정보를 담음
  - Origin : Ray가 시작되는 지점을 설정
  - Direction : Ray가 시작 지점(origin) 에서 쏘여지는 방향을 설정

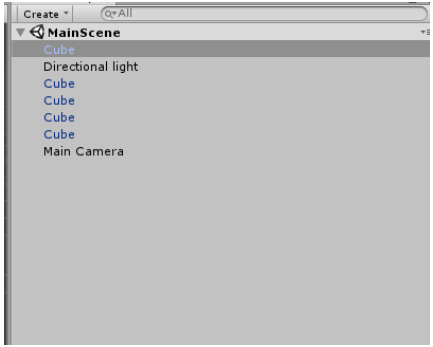
- **Physics.Raycast (Physics class, Raycast Method)**
  - Ray Cast를 실행하여 Ray와 객체가 충돌하는 지 체크하는 메소드
  - 객체와 충돌 되는 경우 true 값을 리턴

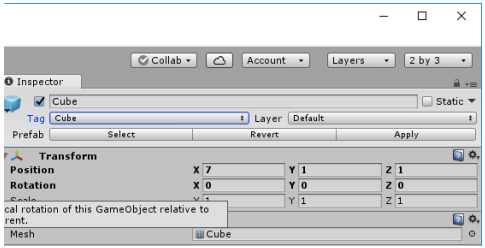
- **RaycastHit (struct)**
  - Physics.Raycast 메서드의 파라미터로 값을 할당(out) 하여 Ray에 충돌된 객체의 정보를 담음

유니티에서 레이 캐스팅 적용

- 유니티를 실행 시키고 다음과 같이 큐브들을 추가하고 배치
- Cube 태그 등록 하고 Cube들을 설정







유니티에서 레이 캐스팅 적용

- **MainSceneScript 생성**
  - 사용자가 화면에서 마우스 버튼을 클릭하면 카메라의 시점을 Origin으로 하여 클릭한 위치 방향으로 Ray를 발사하여 Ray Casting을 수행

```
public Camera MainCamera;
```

멤버 변수로 추가하고 MainCamera를 설정해야 함

```
void Update () {

        if (Input.GetMouseButtonUp(0)) {
            // Ray 객체 생성
            Ray ray = MainCamera.ScreenPointToRay(Input.mousePosition);

            // rayCasting 실행
            rayCasting(ray);
        }
}
```

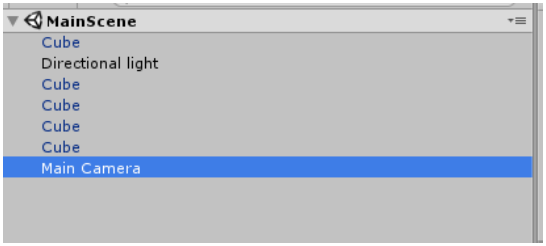
유니티에서 레이 캐스팅 적용

- **rayCasting 함수**
  - 레이 캐스팅을 수행함
  - 히트된 오브젝트의 태그가 "Cube"인 경우 Cube 객체를 얻어와서 Hit 되었다는 정보를 알려줌

```
void rayCasting(Ray ray) {
        RaycastHit hitObj;
        if (Physics.Raycast(ray, out hitObj, Mathf.Infinity)) {
            if (hitObj.transform.tag.Equals("Cube")) {
                CubeScript cubeScript = hitObj.transform.GetComponent<CubeScript>();
                if (null != cubeScript) {
                    cubeScript.Hit();
                }
            }
        }
    }
```

유니티에서 레

- 씬에서 메인 카메라 설정



Main Camera
Static
Tag MainCamera
Layer Default

Transform

Position	X	0	Y	1	Z -10
Rotation	X	0	Y	0	Z 0
Scale	X	1	Y	1	Z 1

Camera

Clear Flags: Skybox
Background
Culling Mask: Everything
Projection: Perspective
Field of View: 60
Clipping Planes: Near 0.3, Far 1000
Viewport Rect: X 0, Y 0, W 1, H 1
Depth: -1
Rendering Path: Use Graphics Settings
Target Texture: None (Render Texture)
Occlusion Culling
Allow HDR
Allow MSAA
Allow Dynamic Resolution

MSAA is requested by the camera but not enabled in quality settings. This camera will render without MSAA buffers. If you want MSAA enable it in the quality settings.

Target Display: Display 1

GUI Layer (Deprecated)

This component is part of the legacy UI system and will be removed in a future release.

Flare Layer

Audio Listener

Main Scene Script (Script)

Script: MainSceneScript
Main Camera: Main Camera (Camera)

유니티에서 레이 캐스팅 적용

- **CubeScript 생성**
  - 사용자로 부터 RayCasting 을 통해 Hit를 당한 경우 액션을 취함 (회전하는 효과)

```
enum State {
        Idle,
        Hit
    }
State currentState;
```

Cube 에서는 2개의 상태를 가지므로 enum형 변수를 선언해 줘야 함

유니티에서 레이 캐스팅 적용

- **CubeScript 생성**
  - 사용자로 부터 RayCasting 을 통해 Hit를 당한 경우 액션을 취함 (회전하는 효과)

```
 enum State {
        Idle,
        Hit
    }
State currentState;
```

Cube 에서는 2개의 상태를 가지므로 enum형 변수를 선언해 줘야 함

```
void Start () {
        currentState = State.Idle;
        NextState();
}
```

Start 함수에서는 상태를 초기상태로 초기화를 해주고 상태 체크를 해주는 함수를 호출

유니티에서 레이 캐스팅 적용

- **NextState**
  - Cube의 상태에 따라서 처리를 다르게 해줌
  - 코루틴을 호출함

```
void NextState() {
        switch (currentState) {
            case State.Idle:
                StartCoroutine(IdleState());
                break;
            case State.Hit:
                StartCoroutine(HitState());
                break;
        }
    }
```

유니티에서 레이 캐스팅 적용

- **NextState**
  - Cube의 상태에 따라서 처리를 다르게 해줌
  - 코루틴을 호출함

```
void NextState() {
        switch (currentState) {
            case State.Idle:
                StartCoroutine(IdleState());
                break;
            case State.Hit:
                StartCoroutine(HitState());
                break;
        }
    }
```

유니티에서 레이 캐스팅 적용

- **IdleState**
  - 아직 사용자에게 Hit이 안된 상태, Cube를 회전하지 않고 행렬의 기본 값을 쿼터니언으로 넣어준다.

```
IEnumerator IdleState() {
        this.gameObject.transform.rotation = Quaternion.identity;

        while (currentState == State.Idle) {
            yield return null;
        }

        NextState();
    }
```

유니티에서 레이 캐스팅 적용

```
public void Hit() {
        currentState = State.Hit;
    }
```

유니티에서 레이 캐스팅 적용

- **HitState**
  - 사용자에 의해서 Hit가 된 상태, 랜덤한 각도로 회전을 수행한다.

```
IEnumerator HitState() {
        float angle = Random.Range(270, 360);
        float hitTime = 0.5f;

        while (currentState == State.Hit) {
            yield return null;

            this.gameObject.transform.Rotate(Time.deltaTime * angle * Vector3.one);

            if (hitTime <= 0) {
                this.currentState = State.Idle;
            } hitTime -= Time.deltaTime;
        }

        NextState();
    }
```

코루틴이란?

- **협동해서 문제를 해결하는 코드들**
  - Update 함수에서 매번 호출해서 사용하는 것이 아니라 필요할때만 호출함으로 성능 향상을 할 수 있음
    - 특정 조건 하에 계산 로직이 들어 가도록 되어 있다면 (Ex: 몬스터 AI) update 함수에 있던 문장들을 코루틴으로 처리 함으로써 연산의 낭비를 최소한으로 줄일 수 있음
  - 병렬 처리 관점에서 쓰레드를 사용하지 않고도 멀티 코어의 퍼포먼스를 낼 수 있음

코루틴과 Update 비교

```
IEnumerator StartRutine()
{
do {
        if (Input.GetMouseButtonUp (0)) {
          //왼쪽버튼 누르면 스타트버튼으로 체킹
         StartButton.GetComponent<Button> ().Select ();
        } else if (Input.GetMouseButtonUp (1)) {
          //오른쪽버튼 누르면 나가기버튼으로 체킹
         QuitButton.GetComponent<Button> ().Select ();
        }
        yield return null;
    } while(inMainRutine);
    yield return null;
}
```

```
void Update()
{
   if (Input.GetMouseButtonUp (0)) {
     //왼쪽버튼 누르면 스타트버튼으로 체킹
       StartButton.GetComponent<Button>().Select ();
    } else if (Input.GetMouseButtonUp (1)) {
    //오른쪽버튼 누르면 나가기버튼으로 체킹
    QuitButton.GetComponent<Button>().Select ();
    }
}
```

동일한 로직 이지만, update는 메인 쓰레드에서 체크를 매번함,
코루틴의 경우 메인 쓰레드외의 여유 코어 쓰레드에서 작업을 처리

코루틴 함수들

- **코루틴에서 사용 하는 함수들**

`yield return null`	다음 프레임까지 대기
`yield return new WaitForSeconds(float)`	지정된 초 만큼 대기
`yield return new WaitForFixedUpdate()`	다음 물리 프레임까지 대기
`yield return new WaitForEndOfFrame()`	모든 렌더링작업이 끝날 때까지 대기
`yield return StartCoRoutine(string)`	다른 코루틴이 끝날 때까지 대기
`yield return new WWW(string)`	웹 통신 작업이 끝날 때까지 대기
`yield return new AsyncOperation`	비동기 작업이 끝날 때까지 대기 (씬로딩)

쿼터니언이란?

- **사원수(Quaternion)**
  - 3차원 그래픽스에서 객체의 회전을 표현 할때, 행렬 대신 사용하는 수학적 개념으로 4개의 값으로 이루어진 복소수 (Complex) 체계

- **사원수는 다음과 같은 4차원 공간에 표시됨**

$$q = \langle w, x, y, z \rangle = w + xi + yj + zk$$

쿼터니언이란?

- **사원수는 물체의 3개 축에 대비가 됨**

- 쿼터니언의 크기 (단위 쿼터니언을 쓴다.)

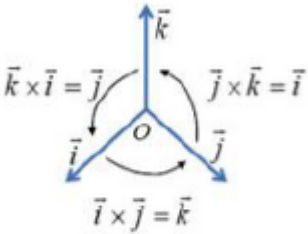
$$\| q \| = \text{Norm}(q) = \text{sqrt}(w^2 + x^2 + y^2 + z^2)$$

- 단위 쿼터니언은 다음과 같은 속성을 지닌다.

$$w^2 + x^2 + y^2 + z^2 = 1$$

- 쿼터니언의 정규화 (단위 쿼터니언을 쓴다.)

$$q = q / \| q \| = q / \text{sqrt}(w^2 + x^2 + y^2 + z^2)$$



$$\vec{i}\times\vec{j} = (|\vec{i}||\vec{j}|\sin 90^\circ)\vec{k} = \vec{k}$$

쿼터니언이란?

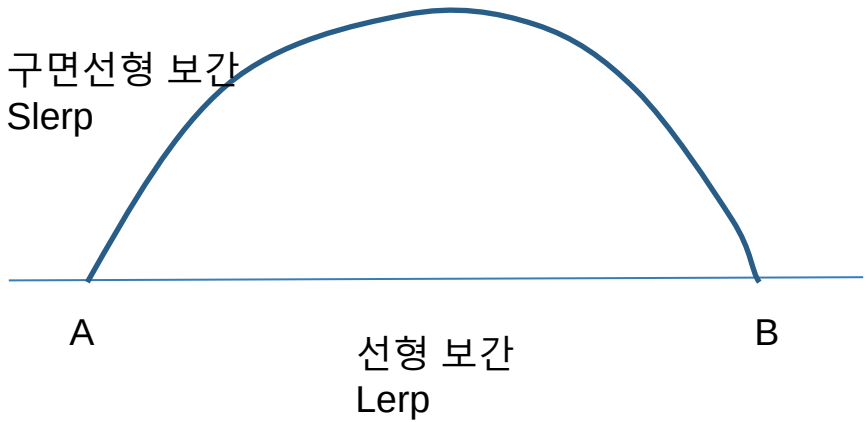
- **사원수를 객체 행렬에 사용하는 이유?**
  - 오일러 각을 사용하는 경우 발생하는 짐벌락 현상을 해결할 수 있음
  - 회전 행렬 계산에 비해서 연산 속도가 빠름
  - 회전 행렬 계산에 비해서 차지하는 메모리 양도 적음
  - 결과가 오류가 날 확률이 적어짐

쿼터니언을 사용한 보간

- **보간(interpolation)이란**
  - 처음과 끝의 값을 가지고 중간에 있는 값을 계산해 냄
  - 물체의 애니메이션을 수행할 때, 계산된 키프레임 사이의 중간 방향을 생성\
- **구면선형 보간 (spherical linear interpolation : slerp)이 필요한 이유**
  - 두개의 점들 사이를 이어주는 선형 직선의 방정식을 통해서 구하는 선형 보간의 경우, 물체가 호를 이루는 이동을 잘 하지 못한다는 단점이 있음

유니티에서 쿼터니언

- **Quaternion.Slerp(Quarternion a, Quarternion b, float t);**



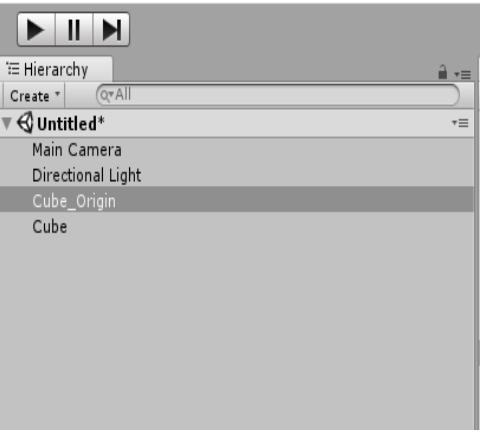
쿼터니언을 사용한 보간

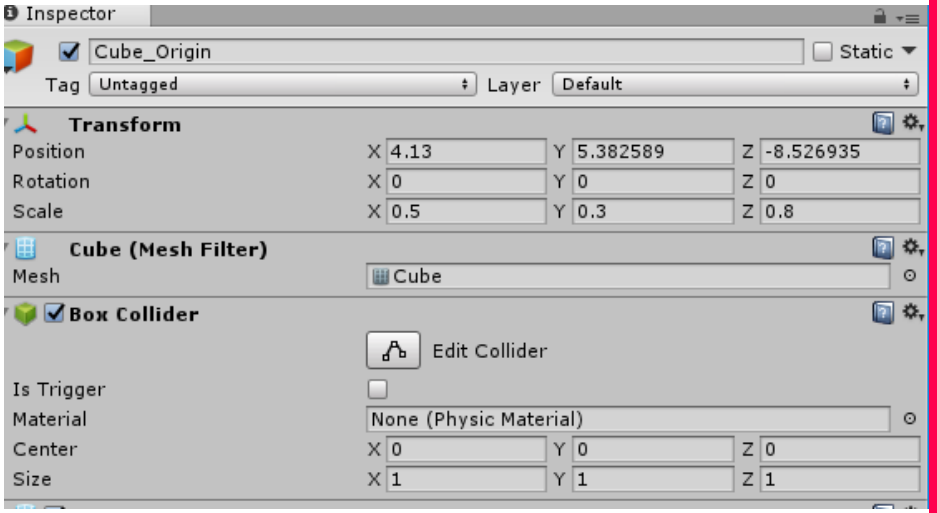
- **보간(interpolation)이란**
  - 처음과 끝의 값을 가지고 중간에 있는 값을 계산해 냄
  - 물체의 애니메이션을 수행할 때, 계산된 키프레임 사이의 중간 방향을 생성\
- **구면선형 보간 (spherical linear interpolation : slerp)이 필요한 이유**
  - 두개의 점들 사이를 이어주는 선형 직선의 방정식을 통해서 구하는 선형 보간의 경우, 물체가 호를 이루는 이동을 잘 하지 못한다는 단점이 있음

```
Quaternion.Slerp(gameObject.transform.rotation, target, 0.1f);
```

쿼터니언 예제

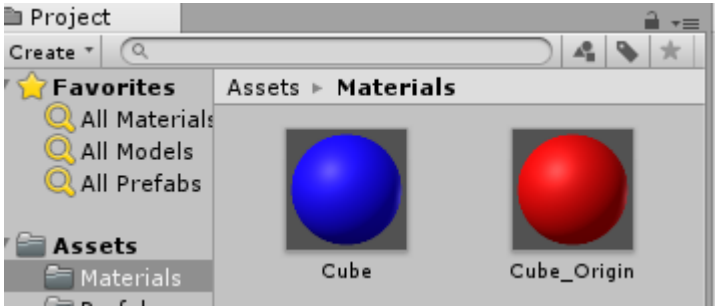
- 다음과 같이 새로 씬을 만들것





쿼터니언 예제

- 다음과 같이 새로 씬을 만들것



쿼터니언 예제

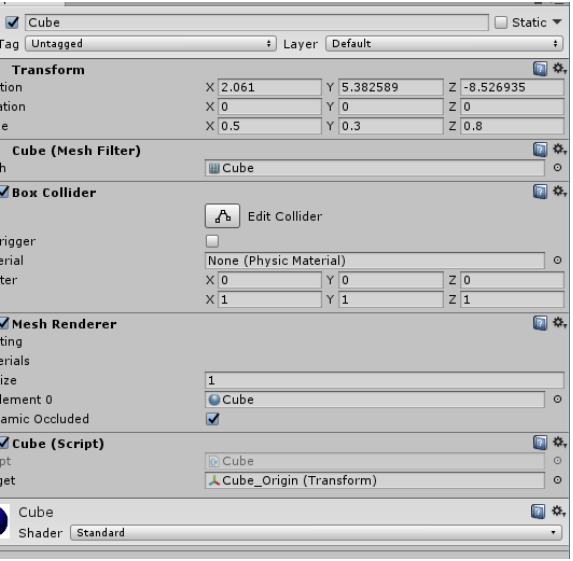
- **Cube 스크립트 생성**

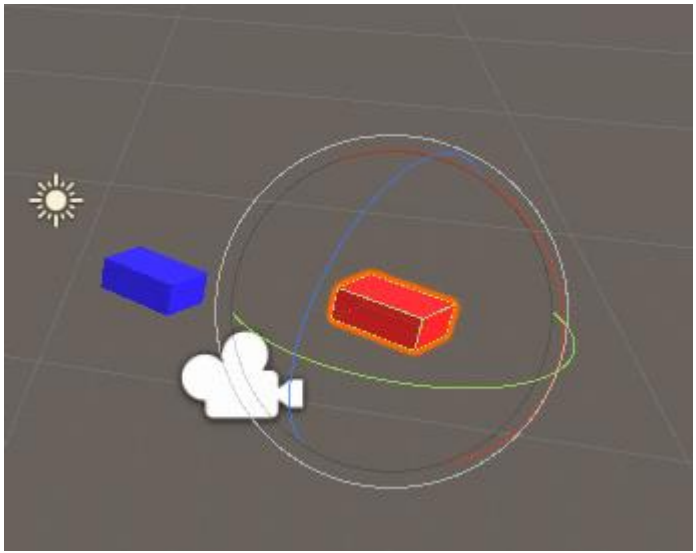
```
public class Cube : MonoBehaviour {
    public Transform target;
    // Use this for initialization
    void Start()
    {

    }
    // Update is called once per frame
    void Update()
    {
        transform.rotation = Quaternion.Lerp(transform.rotation,
target.rotation, 0.05f);
    }
}
```

쿼터니언 예제

- **실행후 원본을 돌려 보자!**
  - Quarternion.Lerp 와 비교하기





Thanks!

Any questions?

junlee@game.hoseo.edu