CSE 165 Discussion 2

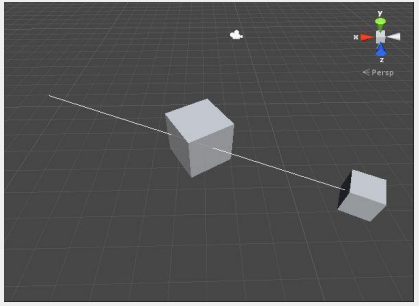
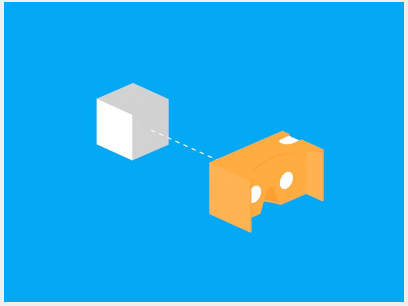
Haoqi Wu

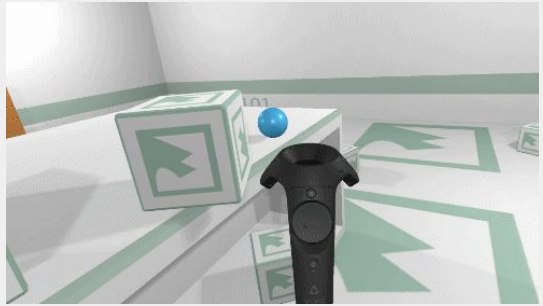
Topics

- Raycasting and Gaze
- Project 1

Raycasting

- In 3D space (and in VR), we often need to interact with distant objects
- We can do this through a process called **Raycasting**
- Raycasting involves projecting a 3D ray from a point in a direction
- Once the ray hits something, it returns information about what it hits
- In VR, this is often used with either the HMD or the controller objects
  - Raycasting is the base of **gaze interaction**: Looking at objects to interact
  - With controllers, allows the user to select using rays, or “lasers”



Raycasting

- The function for raycasting in Unity is Physics.Raycast
  - https://docs.unity3d.com/ScriptReference/Physics.Raycast.html
  - **The object you want to hit must have a collider**
- Give an initial position and direction, checks if the ray hits anything
  - Returns “true” if so, “false” otherwise
  - Stores the hit result in “out RaycastHit hitInfo”

```
void Update() {

    // First, let's create a ray to start at the object's position and go forward.
    Ray myRay = new Ray(this.transform.position, this.transform.forward);

    // Next, a variable to store whatever our ray hits.
    RaycastHit hitObject;

    // Now for the actual Raycast:
    if (Physics.Raycast(myRay, out hitObject, Mathf.Infinity)) {

        // What do we do now?

    }
```

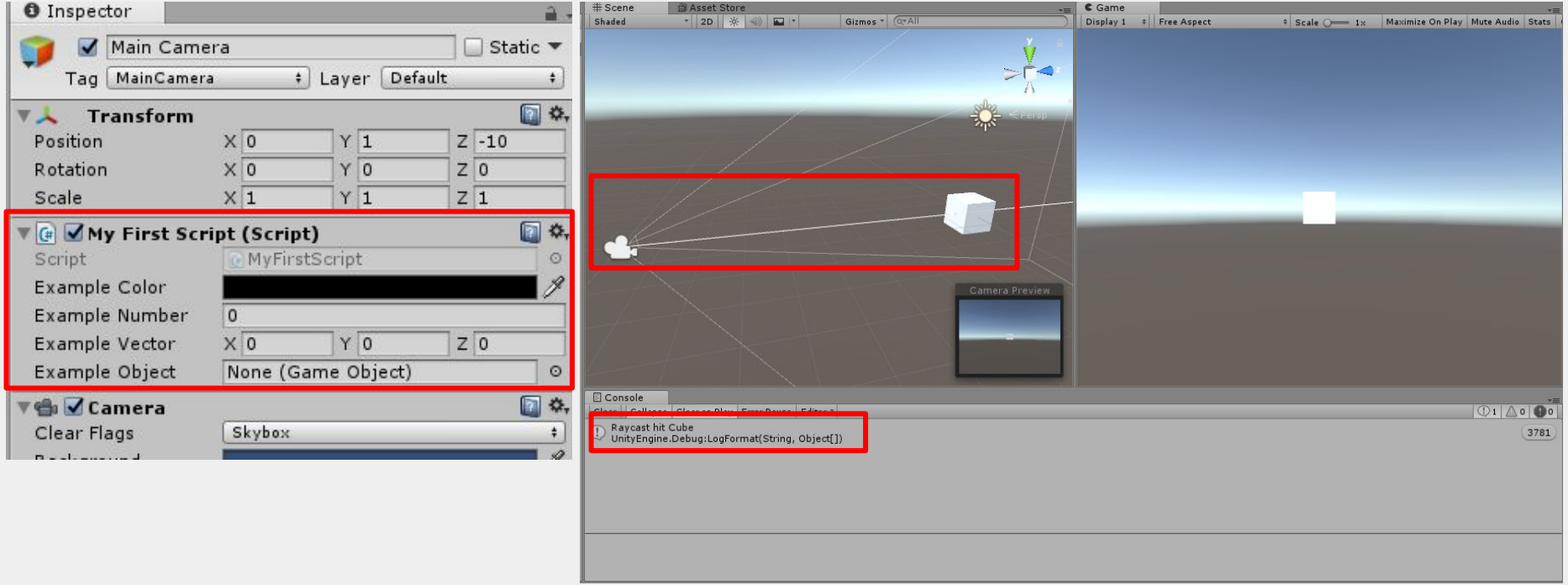
Raycasting

- Let’s try just a really basic raycast from the user’s camera! (Gaze)
  - “Debug.DrawRay” will show a visible ray in the Scene view
- For this one, just print out whatever the ray hits as an example
  - Note we access the object through hitObject’s **collider**

```
void Update () {

    // First, let's create a ray to start at the object's position and go forward.
    Ray myRay = new Ray(this.transform.position, this.transform.forward);
    Debug.DrawRay(myRay.origin, myRay.direction * 1000.0f);

    // Next, a variable to store whatever our ray hits.
    RaycastHit hitObject;

    // Now, for the actual raycast.
    if (Physics.Raycast(myRay, out hitObject, Mathf.Infinity))
    {
        // If it hits an object, print out what it hits
        Debug.LogFormat("Raycast hit {0}", hitObject.collider.gameObject.name);
    }
}
```

Raycasting

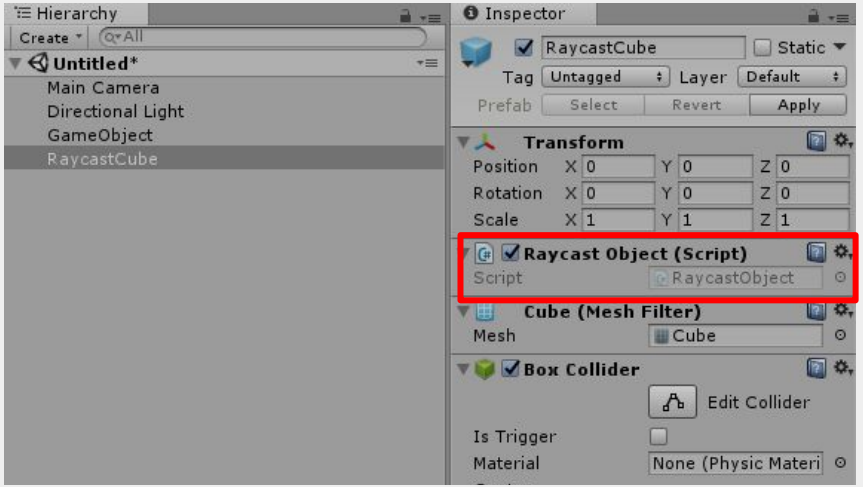
- Now just add this script to the MainCamera and test!



Raycasting

- You may not want every single object to respond to raycasting, though
- It may help to make an **InteractableObject** or **RaycastObject** script
  - Then any object that should respond to your raycast can extend it!
  - Make sure to add this script to the object you want to respond

```
public class RaycastObject : MonoBehaviour {

    public virtual void OnRaycastEnter(RaycastHit hitInfo)
    {
        Debug.LogFormat("Raycast entered on {0}", gameObject.name);
    }

    public virtual void OnRaycast()
    {
        Debug.LogFormat("Raycast stayed on {1}", gameObject.name);
    }

    public virtual void OnRaycastExit()
    {
        Debug.LogFormat("Raycast exited on {2}", gameObject.name);
    }
```



Raycasting

- Now, just need to call these RaycastObject functions in our raycast
  - To do this, we also need to keep track of the **last raycast object**
  - Let's start by looking at a skeleton of all the different conditions

```
// Keeps track of the last raycasted object, if any.
private RaycastObject lastRaycastObject;
```

```
// Now, for the actual raycast.
if (Physics.Raycast(myRay, out hitObject, Mathf.Infinity))
{
    // Try to get the raycast script from the hit object.
    RaycastObject raycastHitObject = hitObject.collider.GetComponent<RaycastObject>();

    // If the hit object actually had the script, handle it.
    if (raycastHitObject != null)...

    // If the object didn't have the script on it and there is a last raycast object, deactivate it.
    else if (lastRaycastObject != null)...
}

//  If there's no object being looked at, and there's a last raycast object, deactivate it.
else if (lastRaycastObject != null)...
```

Raycasting

- The first case is when we look at a NEW object for the first time
  - In other words, this object wasn't being looked at last frame

```
// If this is a NEW object, call Exit on the old object, and Enter on the new one.
if (raycastHitObject != lastRaycastObject)
{
    if (lastRaycastObject != null)
    {
        lastRaycastObject.OnRaycastExit();
    }

    raycastHitObject.OnRaycastEnter(hitObject);
    lastRaycastObject = raycastHitObject;
}
```

- Otherwise, if it isn't a new object, just call OnRaycast()
  - OnRaycast should run every frame the object is being looked at/raycasted

```
// If this isn't a new object, just call OnRaycast on the same object.
else
{
    raycastHitObject.OnRaycast();
}
```

Raycasting

- Finally, we need cases for when to call OnRaycastExit()
  - This should be called as soon as an object that was raycasted isn't anymore
  - In other words, if the CURRENT hit object is null or not a raycast object

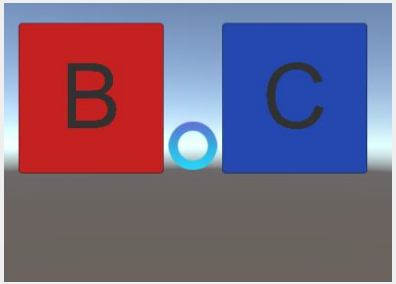
```
    // If the object didn't have the script on it and there is a last raycast object, deactivate it.
    else if (lastRaycastObject != null)
    {
        lastRaycastObject.OnRaycastExit();
        lastRaycastObject = null;
    }
}
//  If there's no object being looked at, and there's a last raycast object, deactivate it.
else if (lastRaycastObject != null)
{
    lastRaycastObject.OnRaycastExit();
    lastRaycastObject = null;
}
```

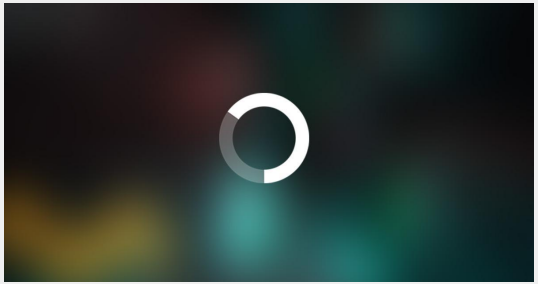
```
// First, let's create a ray to start at the object's position and go forward.
Ray myRay = new Ray(this.transform.position, this.transform.forward);
Debug.DrawRay(myRay.origin, myRay.direction * 1000.0f);

// Next, a variable to store whatever our ray hits.
RaycastHit hitObject;

// Now, for the actual raycast.
if (Physics.Raycast(myRay, out hitObject, Mathf.Infinity))
{
    // Try to get the raycast script from the hit object.
    RaycastObject raycastHitObject = hitObject.collider.GetComponent<RaycastObject>();

    // If the hit object actually had the script, handle it.
    if (raycastHitObject != null)
    {

        // If this is a NEW object, call Exit on the old object, and Enter on the new one.
        if (raycastHitObject != lastRaycastObject)
        {
            if (lastRaycastObject != null)
            {
                lastRaycastObject.OnRaycastExit();
            }

            raycastHitObject.OnRaycastEnter(hitObject);
            lastRaycastObject = raycastHitObject;
        }
        // If this isn't a new object, just call OnRaycast on the same object.
        else
        {
            raycastHitObject.OnRaycast(hitObject);
        }
    }

    // If the object didn't have the script on it and there is a last raycast object, deactivate it.
    else if (lastRaycastObject != null)
    {
        lastRaycastObject.OnRaycastExit();
        lastRaycastObject = null;
    }
}

//  If there's no object being looked at, and there's a last raycast object, deactivate it.
else if (lastRaycastObject != null)
{
    lastRaycastObject.OnRaycastExit();
    lastRaycastObject = null;
}
```

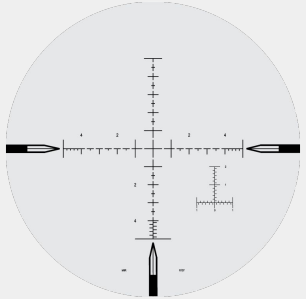
```
public class RaycastObject : MonoBehaviour {

    public virtual void OnRaycastEnter(RaycastHit hitInfo)
    {
        Debug.LogFormat("Raycast entered on {0}", gameObject.name);
    }

    public virtual void OnRaycast(RaycastHit hitInfo)
    {
        Debug.LogFormat("Raycast stayed on {0}", gameObject.name);
    }

    public virtual void OnRaycastExit()
    {
        Debug.LogFormat("Raycast exited on {0}", gameObject.name);
    }
```

Gaze Interaction

- Raycasting is a critical component of **gaze interaction**
  - Gaze interaction involves modifying objects just by looking at them!
  - Most useful for VR devices that don't have controllers
- For gaze interaction, you generally also want a **gaze cursor**
  - This will signify what the user is looking at at any given time
  - Can also fill/change depending on how long the user is looking at something
  - The concept of this delay before activating is called "dwelling"







Gaze Cursors

- Create a UI -> Image object and use whatever image you want!
  - A canvas object should be automatically created
  - Set the canvas Render Mode to “World Space”
- Attach and place the cursor in front of the camera
  - This can be done by parenting the canvas to the camera, or using a script
- To always render the cursor over everything, you need a custom shader
  - This shader should go on the actual cursor object (specifically, it’s material)
  - https://answers.unity.com/questions/878667/world-space-canvas-on-top-of-everything.html

Project Outline

- Scene
  - Player position
  - Spawn locations
  - De-spawn locations
- Player
  - First-person camera control
  - Gaze
- Game Manager
  - Timer
  - Distance check (game over condition)
  - Restart
- Person
  - Self-movement
  - Mask
  - Colliders
- UI
  - Crosshair
  - Game over text

First Person Camera

- A nice video on first person camera control (4:10 - 10:40) https://www.youtube.com/watch?v=_QajrabyTJc
- Get mouse input:

```
float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity * Time.deltaTime;
float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity * Time.deltaTime;
```

- Hide cursor:

```
Cursor.visible = false;
```

- Gaze as ray:

```
Ray ray = new Ray(cam.position, cam.forward);
```

Game Manager

- Pause game (game over):
  Time.timeScale = 0;

- Resume game:
  Time.timeScale = 1;

- Restart game (reload scene):

```
using UnityEngine.SceneManagement;
```

```
SceneManager.LoadScene("Main");
```

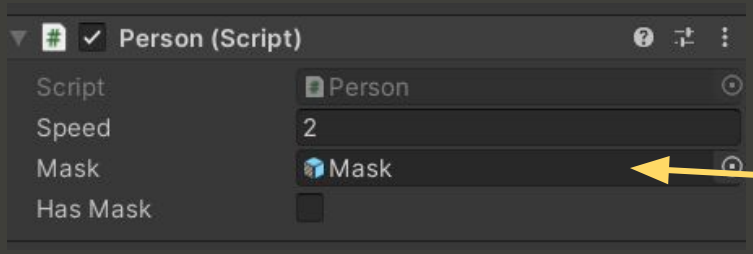
Timer

- Use Time.deltaTime to update timer

```
public float threshold = 1;
private float timer = 0;

void Update()
{
    timer += Time.deltaTime;
    if(timer > threshold)
    {
        // do something
    }
}
```

Person

- Movement:
  transform.Translate();
- Mask
  - You can have a public object for mask and then simply drag and drop:
    public GameObject mask;



Thank you