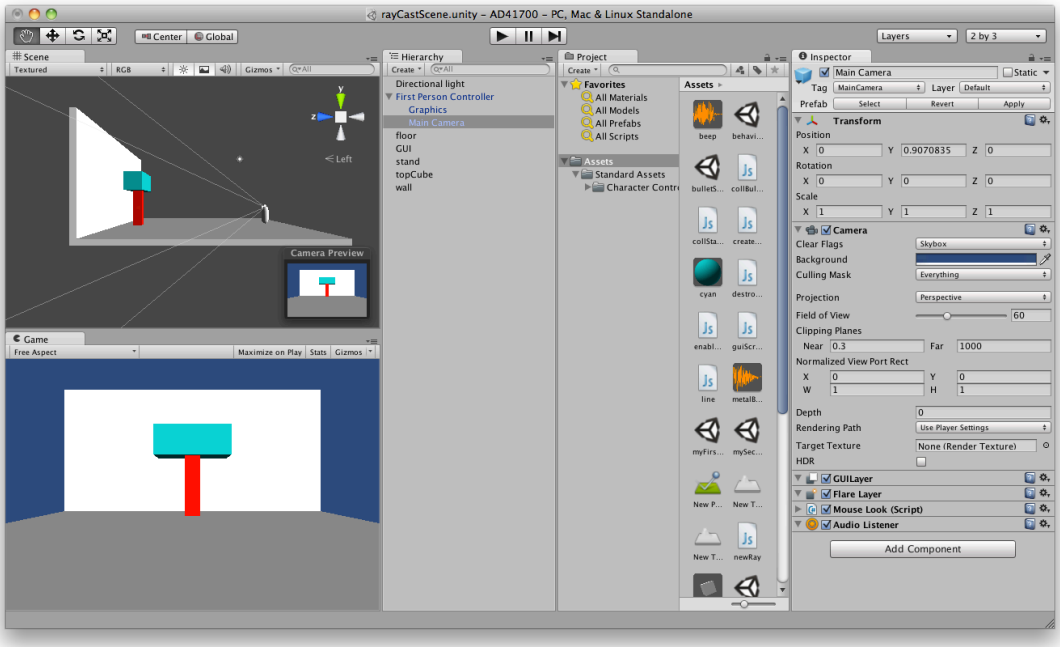
AD41700 Computer Games
Prof. Fabian Winkler
Fall 2013

Raycasting in Unity3D (vers. 4.2)

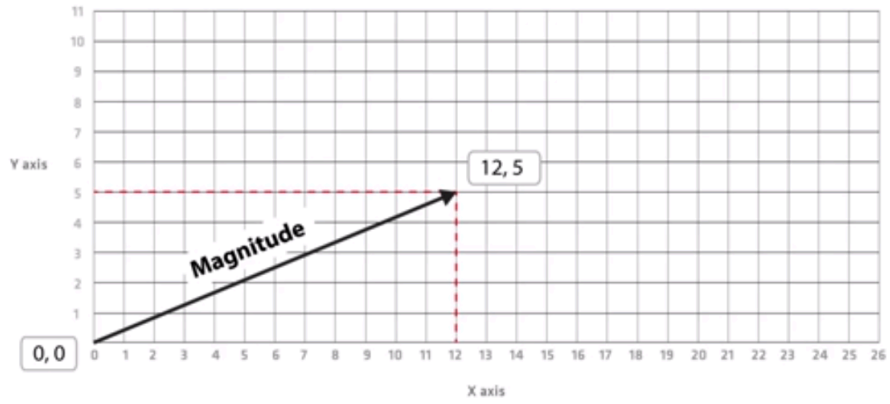
Raycasting is casting a line (or vector) from one point to another in a 3D world and checking if it intersects other colliders or game objects. It is very helpful to figure out which game object your camera is looking at (without running into it), determining distances between game objects, as a means of aiming at something (see the game example "Tag" – https://www.digipen.edu/?id=1170&proj=1506) or as a component of a basic AI/following system (see Yaman's workshop on 10/14).

In this workshop I am using a collection of tutorials to setup a very simple raycasting system in the last scene of our previous workshop. This scene already has some game objects in it and features a first person controller. The aim of this example is to use raycasting to figure out what objects the camera is looking at, to determine the distance between the camera and the object that it is looking at and finally to draw the vector of the raycast as a debugging function but also in the actual game.

We start with the scene from the previous tutorial:



Before we begin it is a good idea to review the concept of a vector in Unity3D: http://unity3d.com/learn/tutorials/modules/scripting/lessons/vector-maths-dot-cross-products:



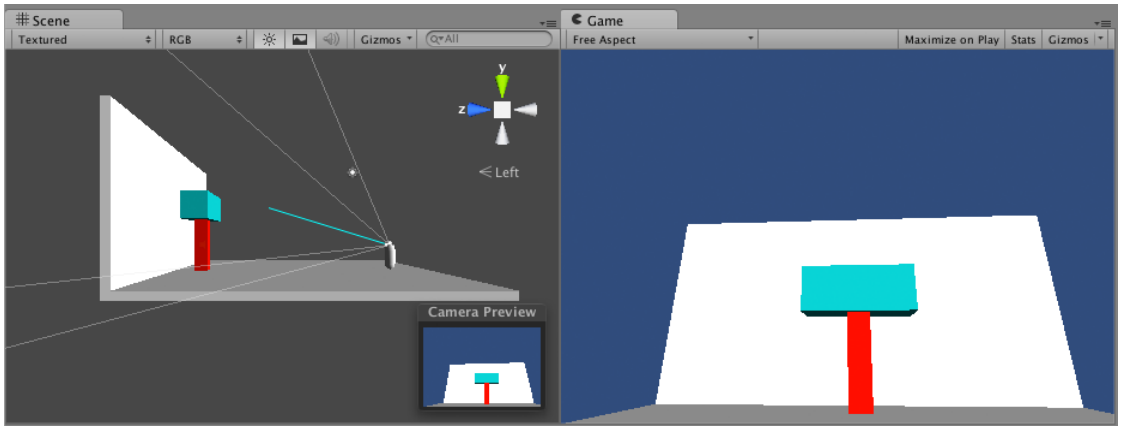
We can also check the Unity3D tutorial on Unity3D.com and find some helpful examples using raycasting to get us started:

http://docs.unity3d.com/Documentation/ScriptReference/Physics.Raycast.html

Specifically I am interested to get the ray from the camera of the first person controller (origin) to the 2D point on the screen defined by the mouse location (direction). The function `Camera.main.ScreenPointToRay` returns a ray going from camera through a 2D point on the screen. If we define the 2D point as the current position of the mouse we can use the mouse pointer to aim at different objects in the 3D scene:

```
function Update() {

      var ray = Camera.main.ScreenPointToRay (Input.mousePosition);
      Debug.DrawRay(ray.origin, ray.direction * 10, Color.cyan);

}
```

Attaching the script to the camera of the first person controller it should render the following scene:

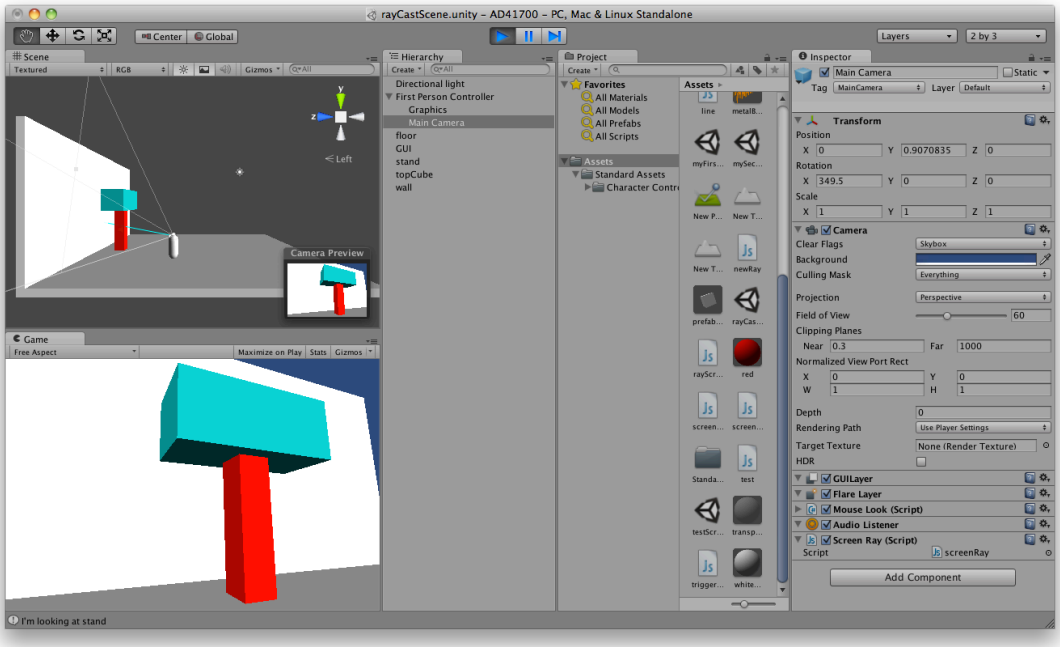


The `Debug.Draw` function draws a ray/vector from the camera in the direction we are looking. However it only shows up in the Scene window and not in the Game window (this somewhat makes sense since you need to imagine that you are looking directly through this ray in the first person perspective of the first person controller.

Let's add a few more lines of code to this script to give us information about which game object we are looking at:

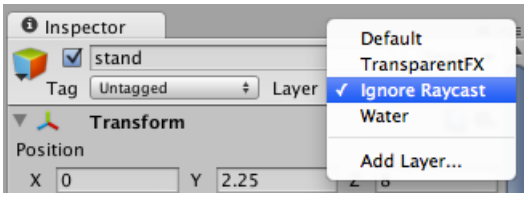
```
function Update() {

      var ray = Camera.main.ScreenPointToRay (Input.mousePosition);
      Debug.DrawRay(ray.origin, ray.direction * 10, Color.cyan);

      var hit : RaycastHit;

      if (Physics.Raycast (ray, hit)){
                  print ("I'm looking at " + hit.transform.name);
            } else {
                  print ("I'm looking at nothing!");
            }
}
```

This prints the name of the game object we are looking at into the console window. If there is no game object in the path of the ray emitted from the camera it prints: "I'm looking at nothing."

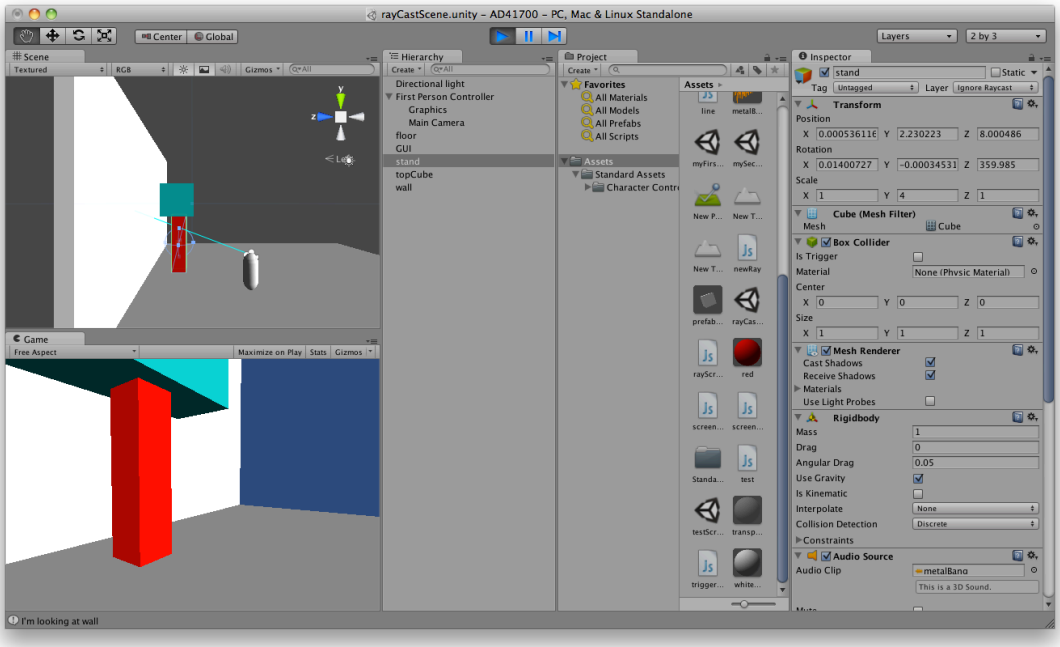


Excluding Game Objects from Raycasting

Unity3D makes it easy to exclude game objects from raycast detection. Simply select the game object you would like to exclude in the Hierarchy window and then in the Inspector in the Layer property select “Ignore Raycast.”



Applying this to the stand game object makes it disappear from the list of objects detected, instead the raycast detects the wall behind it in this situation:

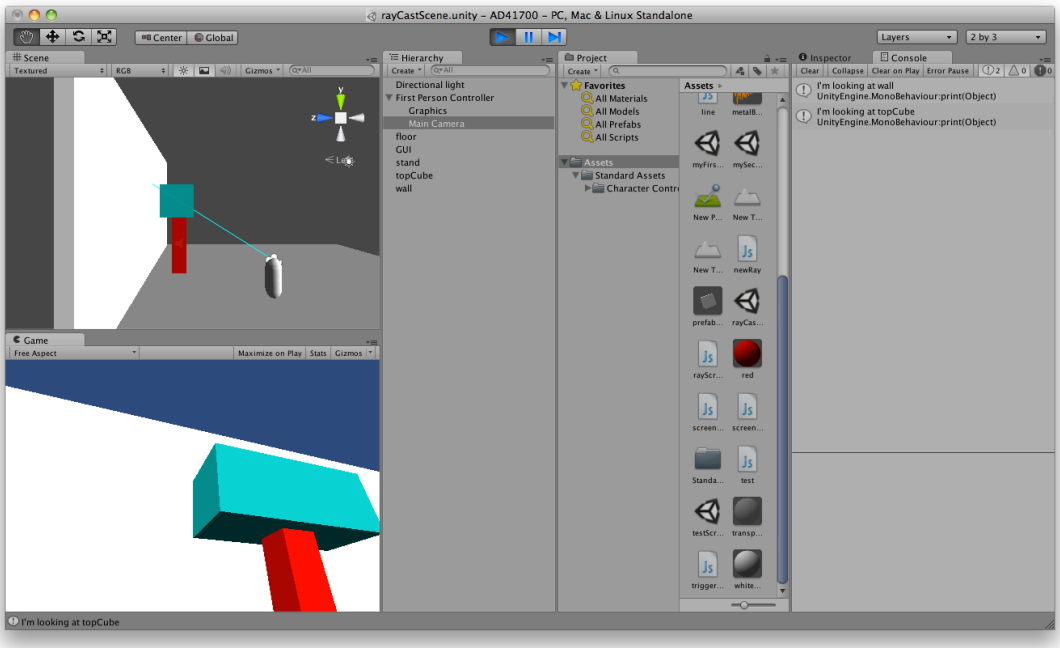


Detecting all Objects in the Path of a Ray

Since the ray we are casting is potentially infinite (if we do not limit it's length) we can detect all the game objects it intersects with, rather than just the first one it hits. This script uses an array to store all the references to the detected game objects (see this mini tutorial on Arrays if you are not familiar with this data type: http://unity3d.com/learn/tutorials/modules/beginner/scripting/arrays):

```
function Update() {
      var ray = Camera.main.ScreenPointToRay (Input.mousePosition);

      //get all gameobjects in the ray:
      var hit : RaycastHit[];

      hit = Physics.RaycastAll(ray);

      //only output when left mouse button clicked:
      if(Input.GetMouseButtonDown(0)){
        if (hit.length > 0) {
            for (var i : int = 0; i < hit.Length; i++) {
            print ("I'm looking at " +  hit[i].transform.name);
        }
      }
}

   // draw the line in the scene window only
   Debug.DrawRay(ray.origin, ray.direction * 10, Color.cyan);
}
```

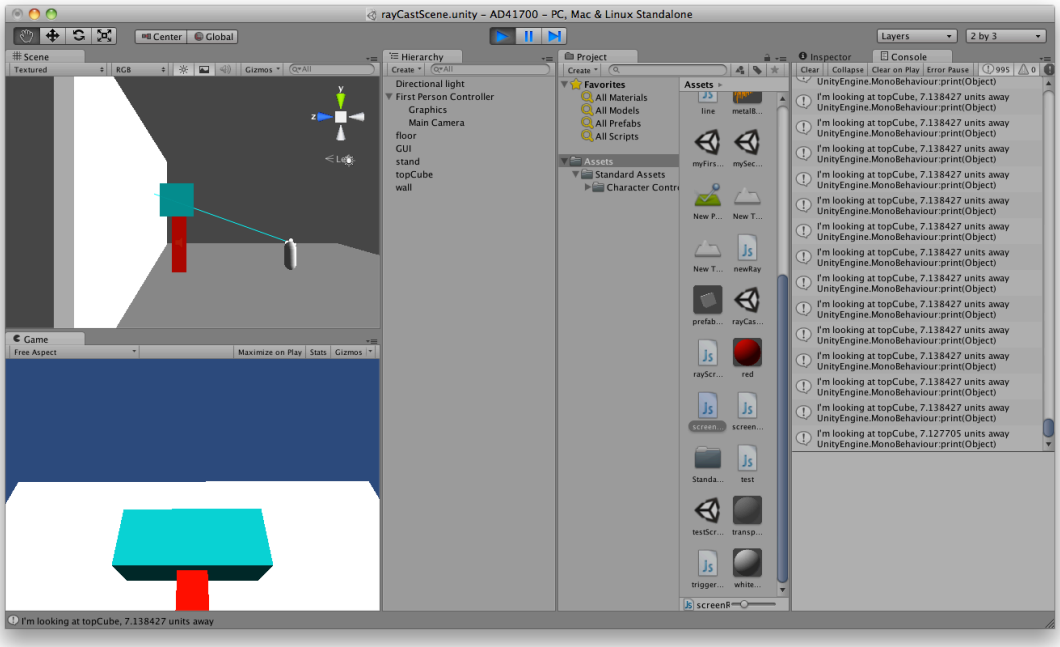
Rather than outputting the raycast hit information all the time, we use the Input.GetMouseButtonDown function to output only when we click on something. In this situation I only clicked once on the topCube:



Measuring Distance

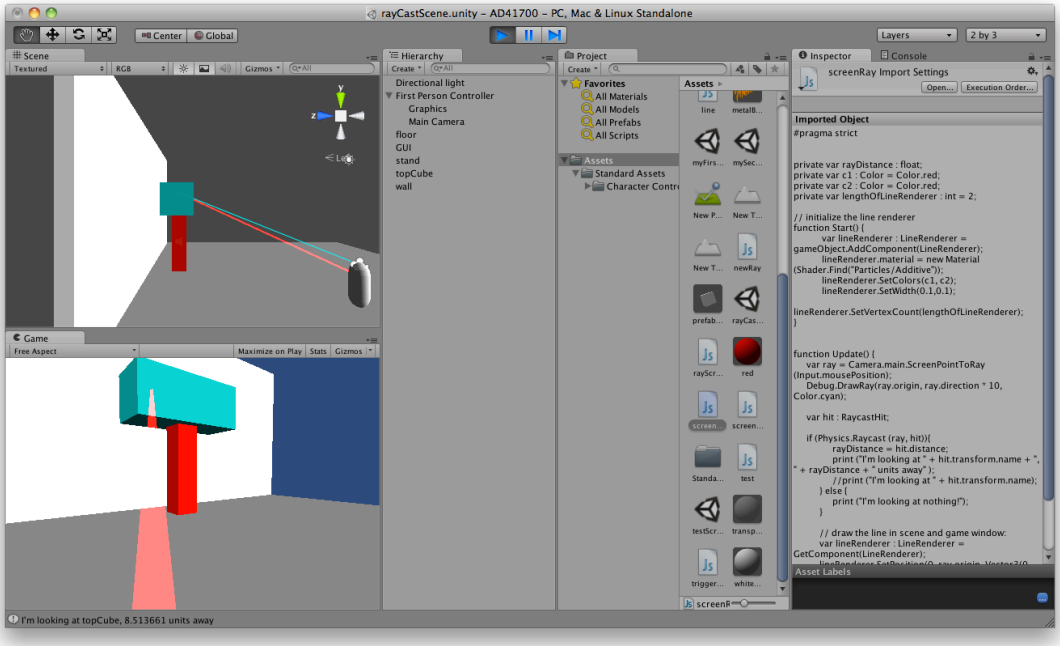
Let's go back to the script we used to detect only the first object in the path of the ray and add one line of code that returns the distance between the origin of the ray (first person controller camera) and the game object it intersects with:

```
var rayDistance : float;

function Update() {

  var ray = Camera.main.ScreenPointToRay (Input.mousePosition);
  Debug.DrawRay(ray.origin, ray.direction * 10, Color.cyan);

  var hit : RaycastHit;

  if (Physics.Raycast (ray, hit)){
      rayDistance = hit.distance;
      print ("I'm looking at " + hit.transform.name + ", " +
rayDistance + " units away" );
  } else {
      print ("I'm looking at nothing!");
  }
}
```



Drawing a Ray in the Game Window

Unfortunately, it is not as easy to draw a ray in the Game window as it is drawing it in the Scene window (using Debug.DrawRay). However we can use a LineRenderer Component which we can generate in the script to do the job for us. Here is what we need to add to our script (for more information about the LineRenderer Component see: http://docs.unity3d.com/Documentation/Components/class-LineRenderer.html



Here is the code:

```
#pragma strict

private var rayDistance : float;
private var c1 : Color = Color.red;
private var c2 : Color = Color.red;
private var lengthOfLineRenderer : int = 2;

// initialize the line renderer
function Start() {
  var lineRenderer : LineRenderer = gameObject.AddComponent(LineRenderer);
  lineRenderer.material = new Material (Shader.Find("Particles/Additive"));
  lineRenderer.SetColors(c1, c2);
  lineRenderer.SetWidth(0.1,0.1);
  lineRenderer.SetVertexCount(lengthOfLineRenderer);
}


function Update() {
       var ray = Camera.main.ScreenPointToRay (Input.mousePosition);
       Debug.DrawRay(ray.origin, ray.direction * 10, Color.cyan);

       var hit : RaycastHit;

       if (Physics.Raycast (ray, hit)){
        rayDistance = hit.distance;print ("Looking at " + hit.transform.name + ",
" + rayDistance + " m away");

       } else {
         print ("I'm looking at nothing!");
       }

 // draw the line in scene and game window:
 var lineRenderer : LineRenderer = GetComponent(LineRenderer);
 lineRenderer.SetPosition(0, ray.origin-Vector3(0, 0.5, 0));
 lineRenderer.SetPosition(1, ray.GetPoint(rayDistance));
}
```

Notice how the red line is just a little bit below the cyan line? I moved the origin of the red line a bit down so we can actually see it in the first person perspective. If I didn't move it down we would look right through the line (line of sight) and it wouldn't be visible in the Game window.

You can use the value of the variable "rayDistance" to dynamically set the length of the red "laserbeam" you are drawing to the distance between the camera and the game object it is intersecting with (rather than having a predefine length of the ray with `ray.GetPoint(rayDistance)`.

Further Raycasting Resources

Unity3D Tutorial:
http://unity3d.com/learn/tutorials/modules/beginner/physics/raycasting

Unity3Dstudent Tutorial:
http://www.unity3dstudent.com/2010/08/intermediate-i01-raycasting/