# Session 15

Materials and Lights  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Materials

Materials are (in conjunction with shaders) used to control and define the appearance of your scene. Materials contain a certain shader and are an instance of that shader. You can change whatever values that are on the shader. For instance, in a standard shader, you can change things like the texture, colour, smoothness et cetera.

When you change a material’s values every object with that material will update in real time. This can lead to certain problems. Say if you want to duplicate an object but you want it to have a different colour than the original, you might just want to change the colour of the material. However, that will change the colour of that material overall, not just for that one object. Thus, it would also change the original object’s colour as well.

To get around this issue, you can give your duplicated object a new material, but make it have all the same values as the old material. Once you have done this you can edit the new material to your hearts content.

To duplicate a new material based on an old material you can use this.

**Material newMaterial = new Material(testMaterialTwo);**

‘testMaterialTwo’ is the name of the old material that you want to duplicate the properties from.

It is simply not enough to be able to change that material. You must apply it to your object. This process will change slightly depending on what the object is. But in most cases an object will have a MeshRenderer that renders the shape and the material. You can see what type of renderer the object uses by finding the renderer component in the inspector.

**MeshRenderer meshRenderer = gameObject.GetComponent<MeshRenderer>();**

This will find the MeshRenderer component of the game object and assign it to a variable with the MeshRenderer type.

Now that we have the MeshRenderer we can finally change the material of the object. To access the material of the object you can do the following:

**meshRenderer.material = newMaterial;**

This will change the game object’s material to be that of the new material.

|  |
| --- |
| **Example 1** |
| ////////////////////////////////////////////////////////////////////  // Changing the material of an object  // Duplicate the test object for demonstration purposes  GameObject newGameObject = Instantiate(testGameObject);  // Setting the objects name  newGameObject.name = "Test Object 2";  // Getting the mesh renderer component which contains the object's materials  MeshRenderer meshRenderer = newGameObject.GetComponent<MeshRenderer>();    // The main material of the object, we are changing it to test material two  meshRenderer.material = testMaterialTwo;  // Changing the game object's position  newGameObject.transform.position = new Vector3(3, 0, 0);  ////////////////////////////////////////////////////////////////////  // Modifying the values of a material of an object  // Duplicate the test object for demonstration purposes  GameObject newGameObject2 = Instantiate(testGameObject);  // Setting the objects name  newGameObject2.name = "Test Object 3";  // Getting the mesh renderer component which contains the object's materials  MeshRenderer meshRenderer2 = newGameObject2.GetComponent<MeshRenderer>();  // Creating a new material based on the old material's properties.  Material newMaterial = new Material(testMaterialTwo);    // Changing the colour of the material  newMaterial.color = new Color(0.6f, 0.3f, 0f);  // Changing the material of the renderer to be our new material  meshRenderer2.material = newMaterial;  // Changing the game objects position  newGameObject2.transform.position = new Vector3(6, 0, 0); |