



# Transform

---

<https://docs.unity3d.com/ScriptReference/Transform.html>  
<https://docs.unity3d.com/ScriptReference/Vector3.html>  
<https://docs.unity3d.com/ScriptReference/Quaternion.html>

3D Computer Game Programming



# Transform

---

- The **Transform** component determines the **Position**, **Rotation**, **Scale** and parenting state of each object in the scene.
- Every **GameObject** has a **Transform**.
  - A **GameObject** will always have a **Transform** component attached.
  - It is not possible to remove a **Transform** or to create a **GameObject** without one.

3D Computer Game Programming



## Transform Properties

Transform			
Position	X	-3.539483	Y -9.493628 Z 0.5715332
Rotation	X	0	Y 0 Z 0
Scale	X	1	Y 1 Z 1

<b>Position</b>	Position of the Transform in X, Y, and Z coordinates.
<b>Rotation</b>	Rotation of the Transform around the X, Y, and Z axes, measured <b>in degrees</b> .
<b>Scale</b>	Scale of the Transform along X, Y, and Z axes. Value "1" is the original size (size at which the object was imported).



## Transform Properties

Transform			
Position	X	-3.539483	Y -9.493628 Z 0.5715332
Rotation	X	0	Y 0 Z 0
Scale	X	1	Y 1 Z 1

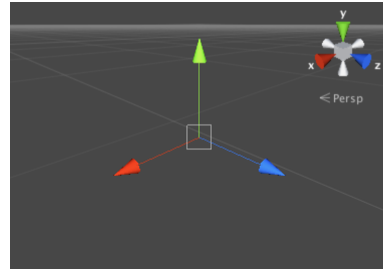
The position, rotation and scale values of a Transform are measured **relative to the Transform's parent**.

If the Transform has no parent, the properties are measured **in world space**.



## Axes in 3D

- Transforms are manipulated in 3D space in the X, Y, and Z axes.
- In Unity, these axes are represented by the colors **red (X)**, **green (Y)**, and **blue (Z)** respectively.



3D Computer Game Programming



## Parenting

- When a GameObject is a **Parent** of another GameObject, the **Child** GameObject will move, rotate, and scale exactly as its Parent does.
- Any object can have multiple children, but only one parent.
- The **Transform** values **in the Inspector** for any child GameObject are displayed relative to the Parent's Transform values. These values are referred to as **local coordinates**.
- In gameplay it is often necessary to find their exact position in **world space** or **global coordinates**.

3D Computer Game Programming



## Vector3 Static Variables

<u>back</u>	Shorthand for writing Vector3(0, 0, -1).
<u>down</u>	Shorthand for writing Vector3(0, -1, 0).
<u>forward</u>	Shorthand for writing Vector3(0, 0, 1).
<u>left</u>	Shorthand for writing Vector3(-1, 0, 0).
<u>negativeInfinity</u>	Shorthand for writing Vector3(float.NegativeInfinity, float.NegativeInfinity, float.NegativeInfinity).
<u>one</u>	Shorthand for writing Vector3(1, 1, 1).
<u>positiveInfinity</u>	Shorthand for writing Vector3(float.PositiveInfinity, float.PositiveInfinity, float.PositiveInfinity).
<u>right</u>	Shorthand for writing Vector3(1, 0, 0).
<u>up</u>	Shorthand for writing Vector3(0, 1, 0).
<u>zero</u>	Shorthand for writing Vector3(0, 0, 0).

3D Computer Game Programming



```
using UnityEngine; using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        transform.position +=
            Vector3.forward * Time.deltaTime;
    }
}
```

3D Computer Game Programming



## Vector3 Variables

<u>magnitude</u>	Returns the length of this vector (Read Only).
<u>normalized</u>	Returns this vector with a magnitude of 1 (Read Only).
<u>sqrMagnitude</u>	Returns the squared length of this vector (Read Only).
<u>this[int]</u>	Access the x, y, z components using [0], [1], [2] respectively.
<u>x</u>	X component of the vector.
<u>y</u>	Y component of the vector.
<u>z</u>	Z component of the vector.

3D Computer Game Programming

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Transform other;
    public float closeDistance = 5.0F;
    void Update() {
        if (other) {
            Vector3 offset = other.position
                - transform.position;
            float sqrLen = offset.sqrMagnitude;
            if (sqrLen < closeDistance * closeDistance)
                print("The other transform is close to me!");
        }
    }
}
```

What does this program do?

3D Computer Game Programming



## Vector3 Static Functions

<a href="#">Angle</a>	Returns the angle in degrees between from and to.
<a href="#">ClampMagnitude</a>	Returns a copy of vector with its magnitude clamped to maxLength.
<a href="#">Cross</a>	Cross Product of two vectors.
<a href="#">Distance</a>	Returns the distance between a and b.
<a href="#">Dot</a>	<b>Dot Product of two vectors.</b>
<a href="#">Lerp</a>	Linearly interpolates between two vectors.
<a href="#">LerpUnclamped</a>	Linearly interpolates between two vectors.
<a href="#">Max</a>	Returns a vector that is made from the largest components of two vectors.
<a href="#">Min</a>	Returns a vector that is made from the smallest components of two vectors.
<a href="#">MoveTowards</a>	Moves a point current in a straight line towards a target point.
<a href="#">Normalize</a>	Makes this vector have a magnitude of 1.

3D Computer Game Programming



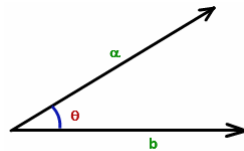
## Dot Product of Two Vectors

$$\mathbf{a} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}$$
$$\mathbf{a} \cdot \mathbf{b} = x_1 x_2 + y_1 y_2 + z_1 z_2$$
$$\mathbf{a} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} 3 \\ 5 \\ 7 \end{pmatrix}$$
$$\mathbf{a} \cdot \mathbf{b} = 2(3) + 4(5) + 6(7)$$
$$= 68$$

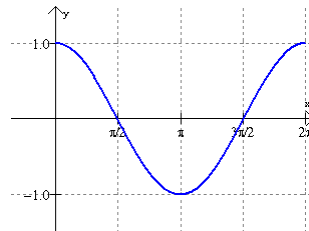
3D Computer Game Programming



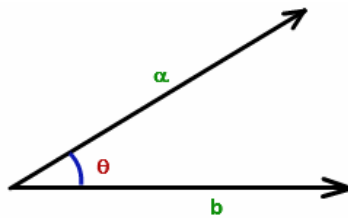
# Dot Product of Two Vectors



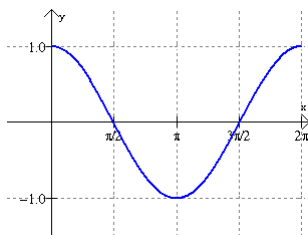
$$a \cdot b = |a| |b| \cos \theta$$



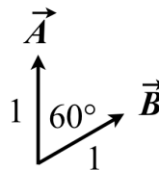
3D Computer Game Programming



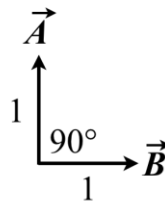
$$a \cdot b = |a| |b| \cos \theta$$



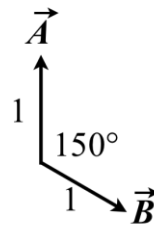
$$\vec{A} \cdot \vec{B} = 1$$



$$\vec{A} \cdot \vec{B} = 0.5$$



$$\vec{A} \cdot \vec{B} = 0$$



$$\vec{A} \cdot \vec{B} = -0.5$$

3D Computer Game Programming

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Transform other;
    void Update() {
        if (other) {
            Vector3 forward =
                transform.TransformDirection(Vector3.forward);
            Vector3 toOther =
                other.position - transform.position;
            if (Vector3.Dot(forward, toOther) < 0)
                print("The other transform is behind me!");
        }
    }
}
```

What does this program do?

3D Computer Game Programming



## Position & Translation

3D Computer Game Programming





## Transform.localPosition Variable

- Position of the transform relative to the parent transform. **Transform.localPosition will get scaled by the scale of all ancestors.**

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        transform.localPosition = new Vector3(0, 0, 0);
        print(transform.localPosition.x);
    }
}
```

3D Computer Game Programming



## Transform.position Variable

- The position of the transform **in world space.**

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        transform.position = new Vector3(0, 0, 0);
        print(transform.position.x);
    }
}
```

3D Computer Game Programming



## Transform.Translate Function

- `public void Translate(Vector3 translation, Space relativeTo = Space.Self);`
- Moves the transform in the direction and distance of translation.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.Translate(Vector3.forward * Time.deltaTime);
        transform.Translate(Vector3.up * Time.deltaTime,
                           Space.World);
    }
}
```

**Space.Self** - the movement is applied relative to the transform's local axes (default)  
**Space.World** - the movement is applied relative to the world coordinate system

3D Computer Game Programming



## Transform.TransformPoint Function

- ```
public Vector3 TransformPoint(Vector3 position);
```
- Transforms position from local space to world space.

```
using UnityEngine;
using System.Collections;

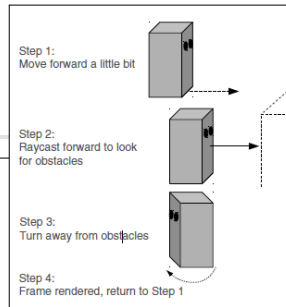
public class ExampleClass : MonoBehaviour {
    public GameObject someObject;
    public Vector3 thePosition;

    void Start() {
        // Instantiate an object to the right of the current object
        thePosition = transform.TransformPoint(2,0,0);
        Instantiate(someObject, thePosition,
                   someObject.transform.rotation);
    }
}
```

3D Computer Game Programming

## WanderingAI.cs

```
using UnityEngine;
using System.Collections;
public class WanderingAI : MonoBehaviour {
    public float speed = 3.0f;
    public float obstacleRange = 5.0f;
    void Update() {
        //step 1
        transform.Translate(0, 0, speed * Time.deltaTime);
        //step 2
        Ray ray = new Ray(transform.position, transform.forward);
        RaycastHit hit;
        if (Physics.SphereCast(ray, 0.75f, out hit)) {
            if (hit.distance < obstacleRange) {
                // step 3
                float angle = Random.Range(-110, 110);
                transform.Rotate(0, angle, 0);
            }
        }
    }
}
```



## Modify WanderingAI.cs

```
...
[SerializeField] private GameObject fireballPrefab;
private GameObject _fireball;
...
if (Physics.SphereCast(ray, 0.75f, out hit)) {
    GameObject hitObject = hit.transform.gameObject;
    if (hitObject.GetComponent<PlayerCharacter>()) {
        if (_fireball == null) {
            _fireball = Instantiate(fireballPrefab) as GameObject;
            _fireball.transform.position =
                transform.TransformPoint(Vector3.forward * 1.5f);
            _fireball.transform.rotation = transform.rotation;
        }
    }
    else if (hit.distance < obstacleRange) {
        float angle = Random.Range(-110, 110);
        transform.Rotate(0, angle, 0);
    }
}
...

```


WanderingAI additions for emitting fireballs



## Get Axes

---

3D Computer Game Programming



## Transform.forward Variable

---

- The blue (Z) axis of the transform in world space.

```
using UnityEngine;
using System.Collections;

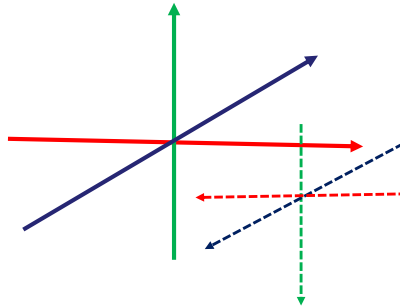
public class ExampleClass : MonoBehaviour {
    public float angleBetween = 0.0F;
    public Transform target;
    void Update() {
        Vector3 targetDir =
            target.position - transform.position;
        angleBetween =
            Vector3.Angle(transform.forward, targetDir);
    }
}
```

3D Computer Game Programming



## Transform.up Variables

- Transform.up - the green axis of the transform in world space.

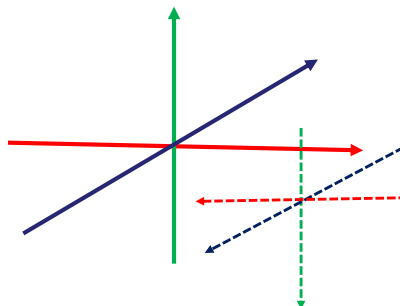


3D Computer Game Programming



## Transform.right Variables

- Transform.right - the red axis of the transform in world space.



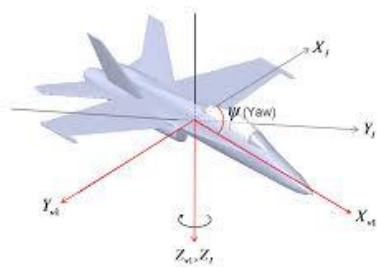
3D Computer Game Programming

# Rotation & Orientation

3D Computer Game Programming

## Euler Angles

- Euler angles have a simpler representation, that being three angle values for X, Y and Z that are applied sequentially.
- To apply a Euler rotation to a particular object, each rotation value is applied in turn, as a rotation around its corresponding axis.



3D Computer Game Programming



## Euler Angles (2)

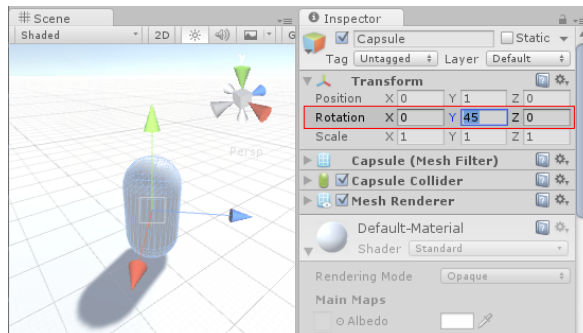
- **Benefit:**
  - Euler angles have an intuitive “human readable” format, consisting of three angles.
  - Euler angles can represent the rotation from one orientation to another through a turn of more than 180 degrees
- **Limitation:** Euler angles suffer from Gimbal Lock.
  - When applying the three rotations in turn, it is possible for the first or second rotation to result in the third axis pointing in the same direction as one of the previous axes. This means a “degree of freedom” has been lost, because the third rotation value cannot be applied around a unique axis.

3D Computer Game Programming



## Transform Inspector in Unity

- In the Transform Inspector, Unity displays the rotation using Euler angles, because this is more easily understood and edited.



3D Computer Game Programming



## Quaternions

- Quaternions can be used to represent the orientation or rotation of an object.
- This representation internally consists of four numbers (referenced in Unity as x, y, z & w) however these numbers don't represent angles or axes and you never normally need to access them directly.
- Unless you are particularly interested in delving into the mathematics of Quaternions, you only really need to know that a Quaternion represents a rotation in 3D space and you will never normally need to know or modify the x, y z, & w properties.

3D Computer Game Programming



## Quaternions (cont'd)

- **Benefit:** Quaternion rotations do not suffer from Gimbal Lock.
- **Limitations:**
  - The numeric representation of a Quaternion is not intuitively understandable.
  - A single quaternion cannot represent a rotation exceeding 180 degrees in any direction.

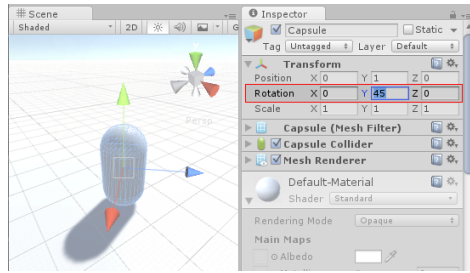
3D Computer Game Programming





## Rotation in Unity

- In Unity all Game Object rotations are stored internally as Quaternions, because the benefits outweigh the limitations.
- In the Transform Inspector, the rotation is displayed in Euler angles, because this is more easily understood and edited.
- Any values entered into the inspector for the rotation of a Game Object are converted “under the hood” into a new Quaternion rotation value for the object.



3D Computer Game Programming



## Creating Quaternion

- When dealing with handling rotations in your scripts, sometimes it is unavoidable to use the **Quaternion** class and its functions.

```
public static Quaternion identity;  
public Vector3 eulerAngles;  
    Returns the euler angle representation of the rotation.  
public static Quaternion Lerp(Quaternion a, Quaternion b, float t);  
    Interpolates between a and b by t and normalizes the result afterwards. The parameter t is clamped to the range [0, 1].  
public static Quaternion Slerp(Quaternion a, Quaternion b, float t);  
    Spherically interpolates between a and b by t. The parameter t is clamped to the range [0, 1].  
public static Quaternion Euler(float x, float y, float z);  
    Returns a rotation that rotates z degrees around the z axis, x degrees around the x axis, and y degrees around the y axis (in that order).
```

- <https://docs.unity3d.com/ScriptReference/Quaternion.html>

3D Computer Game Programming



## Euler Angle and Quaternion

- It's convenient to use Euler angles in your scripts.
  - In this case it's important to note that you must keep your angles in variables, and only use them to *apply* them as Euler angles to your rotation.
- While it's possible to retrieve Euler angles *from* a quaternion, if you retrieve, modify and re-apply, problems might arise.

3D Computer Game Programming



## Transform.eulerAngles Variable

- The rotation as Euler angles in degrees.
- Only use this variable to read and set the angles to absolute values. Rotation about z-axis, x-axis and y-axis in that order.

```
using UnityEngine;
using System.Collections;
public class ExampleClass : MonoBehaviour {
    public float yRotation = 5.0F;
    void Update() {
        yRotation += Input.GetAxis("Horizontal");
        transform.eulerAngles = new Vector3(10, yRotation, 0);
    }
    void Example() {
        print(transform.eulerAngles.x);
        print(transform.eulerAngles.y);
        print(transform.eulerAngles.z);
    }
}
```

3D Computer Game Programming



## Transform.localRotation Variable

- The rotation of the transform **relative to the parent transform's rotation** stored as a **Quaternion**.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        transform.localRotation = Quaternion.identity;
    }
}
```

3D Computer Game Programming



## Transform.rotation Variable

- The rotation of the transform **in world space** stored as a **Quaternion**.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        transform.rotation = Quaternion.identity;
    }
}
```

3D Computer Game Programming



## Transform.rotation Variable

```
// Tilt the cube using the arrow keys. When the arrow keys are released
// the cube will be rotated back to the center using Slerp.
using UnityEngine;
using System.Collections;
public class ExampleClass : MonoBehaviour {
    public float smooth = 2.0F;
    public float tiltAngle = 30.0F;
    void Update() {
        float tiltAroundZ =
            Input.GetAxis("Horizontal") * tiltAngle;
        float tiltAroundX =
            Input.GetAxis("Vertical") * tiltAngle;
        Quaternion target =
            Quaternion.Euler(tiltAroundX, 0, tiltAroundZ);
        transform.rotation =
            Quaternion.Slerp(transform.rotation, target,
                Time.deltaTime * smooth);
    }
}
```

3D Computer Game Programming



## Transform.Rotate Function

```
public void Rotate(Vector3 eulerAngles, Space relativeTo = Space.Self);
```

- Applies a rotation of eulerAngles.z degrees around the z axis, eulerAngles.x degrees around the x axis, and eulerAngles.y degrees around the y axis (in that order).
- If relativeTo is not specified or set to **Space.Self** the rotation is applied around the transform's local axes.
- If relativeTo is set to **Space.World** the rotation is applied around the world x, y, z axes.

3D Computer Game Programming



## Transform.Rotate Function

```
using UnityEngine;

public class ExampleClass : MonoBehaviour
{
    void Update()
    {
        // Rotate the object
        // around its local X axis
        transform.Rotate(Vector3.right * Time.deltaTime);

        // ...also rotate around the World's Y axis
        transform.Rotate(Vector3.up * Time.deltaTime,
                        Space.World);
    }
}
```

3D Computer Game Programming



## Transform.RotateAround Function

```
public void RotateAround(Vector3 point, Vector3 axis, float angle);
```

- Rotates the transform about axis passing through point **in world coordinates** by angle degrees.
- This modifies both the position and the rotation of the transform.

3D Computer Game Programming



## Transform.RotateAround Function

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.RotateAround(Vector3.zero,
                               Vector3.up,
                               20 * Time.deltaTime);
    }
}
```

3D Computer Game Programming



## Rotation Mistake #1

```
// rotation scripting mistake #1
// the mistake here is that we are modifying the x
// value of a quaternion
// this value does not represent an angle,
// and will not produce desired results

void Update () {

    var rot = transform.rotation;
    rot.x += Time.deltaTime * 10;
    transform.rotation = rot;

}
```

3D Computer Game Programming



## Rotation Mistake #2

```
// rotation scripting mistake #2
// the mistake here is that we are reading, modifying
// then writing the Euler values from a quaternion.
// Because these values calculated from a Quaternion,
// each new rotation may return very different Euler angles,
// which may suffer from gimbal lock.

void Update () {

    var angles = transform.rotation.eulerAngles;
    angles.x += Time.deltaTime * 10;
    transform.rotation = Quaternion.Euler(angles);

}
```

3D Computer Game Programming



## Correct Rotation

```
// rotation scripting with Euler angles correctly.
// here we store our Euler angle in a class variable,
// and only use it to apply it as a Euler angle,
// but we never rely on reading the Euler back.

float x;
void Update () {

    x += Time.deltaTime * 10;
    transform.rotation = Quaternion.Euler(x,0,0);

}
```

3D Computer Game Programming



## Transform.LookAt Function

```
public void LookAt(Transform target, Vector3 worldUp = Vector3.up);
```

- Rotates the transform so the forward vector points at the target's current position.

3D Computer Game Programming



## Transform.LookAt Function

```
// This script can be attached to a camera to make it  
// continuously point at another object.
```

```
using UnityEngine;  
using System.Collections;  
  
public class ExampleClass : MonoBehaviour {  
    public Transform target;  
  
    void Update() {  
        // Rotate the camera every frame  
        // so it keeps looking at the target  
        transform.LookAt(target);  
    }  
}
```

3D Computer Game Programming





## Scale & Others

---

3D Computer Game Programming



## Transform.localScale Variable

---

- The scale of the transform relative to the parent.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        // Widen the object by 0.1
        transform.localScale += new Vector3(0.1f, 0, 0);
    }
}
```

3D Computer Game Programming



## Transform.lossyScale Variable

- The global scale of the object (Read Only).

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Example() {
        print(transform.lossyScale);
    }
}
```

3D Computer Game Programming



## Transform.lossyScale Variable

- If you have a parent transform with scale and a child that is arbitrarily rotated, the scale will be skewed. Thus scale can not be represented correctly in a 3 component vector but only a 3x3 matrix.
- lossyScale is a convenience property that attempts to match the actual world scale as much as it can.
- If your objects are not skewed the value will be completely correct and most likely the value will not be very different if it contains skew too.

3D Computer Game Programming



## Transform.TransformDirection Function

```
public Vector3 TransformDirection(Vector3 direction);
```

- Transforms direction **from local space to world space**.
- This operation is not affected by scale or position of the transform. The returned vector has the same length as direction.

3D Computer Game Programming



## Transform.TransformVector Function

```
public Vector3 TransformVector(Vector3 vector);
```

- Transforms vector **from local space to world space**.
- This operation is not affected by position of the transform, but is affected by scale. The returned vector may have a different length than vector.

3D Computer Game Programming