```
using UnityEngine;
using System;

/// <summary>
/// Simply attach this script to a GameObject that is a child of metaioTracker.
/// The GameObject (or one of its children) must have a Collider component.
///
/// Drag, Pinch and Rotate gestures are supported and can be individually
/// enabled or disabled.
///
/// Rotate gesture is not supported on non-Touch inputs, i.e. mouse
/// </summary>
public class GestureHandler : MonoBehaviour
{
	[SerializeField]
	public bool enableDrag = true;

	[SerializeField]
	public bool enablePinch = true;

	[SerializeField]
	public bool enableRotate = true;

	/// <summary>
	/// Buffer to save ray hit information
	/// </summary>
	private RaycastHit mHit;

	/// <summary>
	/// true when this GameObject is selected
	/// </summary>
	private bool mSelected;

	/// <summary>
	/// Initial translation offset when the GameObject is selected
	/// </summary>
	private Vector3 mTranslationOffset;

	/// <summary>
	/// Last distance between two touch points
	/// </summary>
	private float mLastTouchDistance;

	/// <summary>
```

```
        /// Last angle between two touch points
        /// </summary>
        private float mLastTouchAngle;

        /// <summary>
        /// Tracked plane
        /// </summary>
        private GameObject mTrackerPlane;

        /// <summary>
        /// Reference to the last selected GameObject
        /// </summary>
        private static GameObject mLastSelectedGameObject = null;

        void Start()
        {
                // if the GameObject or its children has no Collider, disable the script with a warning
                if (gameObject.GetComponent<Collider>() == null && gameObject.GetComponentsInChildren<Collider>().Length == 0)
                {
                        Debug.LogWarning("No Collider found in the GameObject("+gameObject.name+") or its children, gestures will not
work!");
                        enabled = false;
                        return;
                }

                // Find an existing tracker plane for the corresponding tracker
                mTrackerPlane = null;
                foreach (Transform child in gameObject.transform.parent)
                {
                        if (child.gameObject.name == "TrackerPlane")
                        {
                                mTrackerPlane = child.gameObject;
                                break;
                        }
                }

                // Create a new tracker plane if not found
                if (mTrackerPlane == null)
                {
                        mTrackerPlane = GameObject.CreatePrimitive(PrimitiveType.Plane);
                        Component.Destroy(mTrackerPlane.GetComponent(typeof(Renderer)));
                        mTrackerPlane.name = "TrackerPlane";
                        mTrackerPlane.transform.parent = gameObject.transform.parent;
                        mTrackerPlane.transform.localScale = Vector3.one * 10000f;
```

```
                mTrackerPlane.layer = 8;
            }

            mSelected = false;
            mTranslationOffset = Vector3.zero;
            mLastTouchDistance = 0f;
            mLastTouchAngle = 0f;

            Debug.Log("Multitouch enabled: "+Input.multiTouchEnabled);
        }

        // Update is called once per frame
        void Update()
        {
            // Handle touch or click events only if the GameObject is currently
            // being rendered
            if (gameObject.activeSelf)
            {
#if UNITY_STANDALONE_WIN || UNITY_STANDALONE_OSX || UNITY_EDITOR
                handleClicks();
#elif UNITY_ANDROID || UNITY_IPHONE
                handleTouches();
#endif
            }
        }

        /// <summary>
        /// Handle touch events
        /// </summary>
        private void handleTouches()
        {
            if (Input.touchCount == 0)
            {
                return;
            }

            // if there's touch points and the phase is began-->try to select the GameObject
            if(Input.GetTouch(0).phase == TouchPhase.Began)
            {
                mSelected = selectGameObject(Input.GetTouch(0).position);

                if (mSelected)
                {
                    mLastSelectedGameObject = gameObject;
```

```
                // Disable colliders to increase the performance during transformations
                enableColliders(false);
        }

}

if (Input.touchCount > 1)
{
        float xd = Input.GetTouch(0).position.x - Input.GetTouch(1).position.x;
        float yd = Input.GetTouch(0).position.y - Input.GetTouch(1).position.y;

        // Save initial distance and angle    between two touch points
        if (Input.GetTouch(1).phase == TouchPhase.Began)
        {
                mLastTouchDistance = Vector2.Distance(Input.GetTouch(1).position, Input.GetTouch(0).position);
                mLastTouchAngle = Mathf.Atan2(yd, xd);
        }
        else if (Input.GetTouch(0).phase == TouchPhase.Moved || Input.GetTouch(1).phase == TouchPhase.Moved)
        {
                if (Input.GetTouch(0).deltaPosition.magnitude > 4f || Input.GetTouch(1).deltaPosition.magnitude > 4f)
                {
                        // Pinch
                        float currentDistance = Vector2.Distance(Input.GetTouch(1).position, Input.GetTouch(0).position);
                        pinchGameObject(currentDistance/mLastTouchDistance);
                        mLastTouchDistance = currentDistance;

                        // Rotate
                        float currentAngle = Mathf.Atan2(yd, xd);
                        rotateGameObject((mLastTouchAngle-currentAngle)*Mathf.Rad2Deg);
                        mLastTouchAngle = currentAngle;
                }
        }
}
// Drag only when single touch is moved
else if(Input.GetTouch(0).phase == TouchPhase.Moved)
{
        // Drag
        dragGameObject(Input.GetTouch(0).position);
}


if(mSelected && Input.GetTouch(0).phase == TouchPhase.Ended)
{
```

```
                enableColliders(true);
        }
}

/// <summary>
/// Handle mouse clicks
/// </summary>
public void handleClicks()
{
        // Try to select the geometry when left or middle mouse button is clicked
        if (Input.GetMouseButtonDown(0) || Input.GetMouseButtonDown(2))
        {
                mSelected = selectGameObject(Input.mousePosition);
        }

        // Drag
        if (Input.GetMouseButton(0))
        {
                dragGameObject(Input.mousePosition);
        }

        // Pinch
        float scrollWheel = Input.GetAxis("Mouse ScrollWheel");
        pinchGameObject(1f+scrollWheel);
}

/// <summary>
/// Select the tracker plane from given ray
/// </summary>
/// <returns>
/// true if the tracker plane is selected
/// </returns>
/// <param name='ray'>
/// Ray that is casted from touched or clicked position in viewport
/// </param>
private bool selectTrackerPlane(Ray ray)
{
        return (Physics.Raycast(ray, out mHit, Camera.main.farClipPlane, 1<<mTrackerPlane.layer)
                && mHit.collider.gameObject == mTrackerPlane);
}

/// <summary>
/// Select this GameObject from the given viewport coordinates
/// </summary>
```

```
/// <returns>
/// true if this GameObject is selected
/// </returns>
/// <param name='position'>
/// Touched or clicked position in viewport
/// </param>
private bool selectGameObject(Vector2 position)
{
	Debug.Log("selectGameObject: "+position);

	// get a ray from the touch point
	Ray ray = Camera.main.ScreenPointToRay(position);

	// Select the GameObject by casting a ray
	if (Physics.Raycast(ray, out mHit, Camera.main.farClipPlane, ~(1<<mTrackerPlane.layer)))
	{

		Debug.Log("GameObject found: "+mHit.collider.gameObject.name);

		if (mHit.collider.gameObject == gameObject ||
			isChildOf(gameObject.transform, mHit.collider.gameObject.transform))
		{
			// Find the initial translation offset on the tracked plane
			if (selectTrackerPlane(ray))
			{
				mTranslationOffset = mHit.point - gameObject.transform.position;
				return true;
			}
		}
	}

	mTranslationOffset = Vector3.zero;
	mLastTouchDistance = 0f;
	mLastTouchAngle = 0f;
	return false;
}

/// <summary>
/// Is this GameObject selected
/// </summary>
/// <returns><c>true</c>, if selected, <c>false</c> otherwise.</returns>
private bool isSelected()
{
	return mSelected || (mLastSelectedGameObject != null && mLastSelectedGameObject == gameObject);
```

```
	}

	/// <summary>
	/// Drag this GameObject after selected
	/// </summary>
	/// <param name='position'>
	/// Touched or clicked position in viewport
	/// </param>
	private void dragGameObject(Vector2 position)
	{
		if (!(enableDrag && mSelected))
		{
			return;
		}

//		Debug.Log("dragGameObject: "+position);

		// cast a ray on layer 8 (the plane) to calculate the hit position of the plane
		Ray ray = Camera.main.ScreenPointToRay(position);
		if(selectTrackerPlane(ray))
		{
			// move the GameObject to the intersect of the ray and the plane, offset should be accounted for
			gameObject.transform.position = mHit.point - mTranslationOffset;
		}
	}

	/// <summary>
	/// Pinch the GameObject with the given scale factor
	/// </summary>
	/// <param name='factor'>
	/// Scale factor, must be greater than 0
	/// </param>
	private void pinchGameObject(float factor)
	{
		if (!(enablePinch && isSelected() && (factor <= 0.99f || factor >= 1.01f)))
		{
			return;
		}

//		Debug.Log("pinchGameObject: "+factor);
		gameObject.transform.localScale *= factor;
	}

	/// <summary>
```

```
        /// Rotate the GameObject by the given angle around y-axis
        /// </summary>
        /// <param name='angle'>
        /// Angle in degrees
        /// </param>
        private void rotateGameObject(float angle)
        {
                if (!(enableRotate && isSelected()))
                {
                        return;
                }

//              Debug.Log("rotateGameObject: "+angle);
                gameObject.transform.Rotate(Vector3.up * angle);
        }

        /// <summary>
        /// Recursively Determine if the given child transform is the child of the
        /// given parent transform
        /// </summary>
        /// <returns>
        /// true if child transform is the child of the parent transform in hierarchy
        /// </returns>
        /// <param name='parentTransform'>
        /// Parent transform
        /// </param>
        /// <param name='childTransform'>
        /// Child transform to check
        /// </param>
        private bool isChildOf(Transform parentTransform, Transform childTransform)
        {
                if (parentTransform.childCount > 0)
                {
                        if (childTransform.parent == parentTransform)
                        {
                                return true;
                        }
                        else
                        {
                                foreach (Transform child in parentTransform)
                                {
                                        if (isChildOf(child, childTransform))
                                                return true;
                                }
```

```
                }
            }

            return false;

        }

        /// <summary>
        /// Enable or disable the colliders of this and all GameObjects in the hierarchy
        /// </summary>
        /// <param name='enable'>
        /// true to enable, false to disable
        /// </param>
        private void enableColliders(bool enable)
        {
            MeshCollider[] colliders = gameObject.GetComponentsInChildren<MeshCollider>();
            foreach(MeshCollider collider in colliders)
            {
                collider.enabled = enable;
            }
        }
}
```