

# In-depth Unity 5.0 User Guide

---

## AI

- Navmesh contour may look different due to changed partitioning - in cases with narrow corridor/doorways or similar - this can lead to a difference in connectivity. Solve the issue by tweaking the voxel size for navmesh building.
- Setting the destination for a NavMeshAgent doesn't resume the agent after calling 'Stop' - call 'Resume' explicitly to resume the agent.
- *NavMeshAgent.updatePosition*: When updatePosition is false and the agent transform is moved the agent position doesn't change. Previously the agent position would be reset to the transform position - constrained to the nearby navmesh.
- *NavMeshObstacle* component: The default shape for newly created NavMeshObstacle components is a box. The selected shape (box or capsule) now applies to both carving and avoidance.
- Navmesh built with earlier versions of Unity is not supported. You must rebuild with Unity 5. You can use the following script as an example how to rebuild NavMesh data for all of you scenes.

### Rebake script example

```
#if UNITY_EDITOR
using System.Collections.Generic;
using System.Collections;
using System.IO;
using UnityEditor;
using UnityEngine;
public class RebakeAllScenesEditorScript
{
    [MenuItem ("Upgrade helper/Bake All Scenes")]
    public static void Bake()
    {
        List<string> sceneNames = SearchFiles (Application.dataPath, "*.unity");
        foreach (string f in sceneNames)
        {
            EditorApplication.OpenScene(f);

            // Rebake navmesh data
            NavMeshBuilder.BuildNavMesh ();

            EditorApplication.SaveScene ();
        }
    }
    static List<string> SearchFiles(string dir, string pattern)
    {
        List <string> sceneNames = new List <string>();
        foreach (string f in Directory.GetFiles(dir, pattern,
        SearchOption.AllDirectories))
        {
            sceneNames.Add (f);
        }
        return sceneNames;
    }
}
#endif
```

---

## Animation

## Asset Creation API

In 5.0 we introduced an API that allows to build and edit Mecanim assets in the editor. For users that have used the unsupported API - in `UnityEditorInternal` namespace – you will need to manually update your scripts to use the new API.

Here is a short list of the most encountered type changes :

`UnityEditorInternal.BlendTree` -> `UnityEditor.Animations.BlendTree`

`UnityEditorInternal.AnimatorController` -> `UnityEditor.Animations.AnimatorController`

`UnityEditorInternal.StateMachine` -> `UnityEditor.Animations.AnimatorStateMachine`

`UnityEditorInternal.State` -> `UnityEditor.Animations.AnimatorState`

`UnityEditorInternal.AnimatorControllerLayer` -> `UnityEditor.Animations.AnimatorControllerLayer`

`UnityEditorInternal.AnimatorControllerParameter` -> `UnityEditor.Animations.AnimatorControllerParameter`

Also note that most accessor functions have been changed to arrays:

`UnityEditorInternal.AnimatorControllerLayer layer = animatorController.GetLayer(index);`

becomes :

`UnityEditor.Animations.AnimatorControllerLayer layer = animatorController.layers[index];`

A basic example of API usage is given at the end of this blog post : <http://blogs.unity3d.com/2014/06/26/shiny-new-animation-features-in-unity-5-0/>

For more details refer to the Scripting API documentation

---

# Audio

## AudioClips

### 2D/3D sounds

A number of things has changed in the AudioClip. First, there is no longer a 3D flag on the asset. This flag has been moved into the AudioSource in form of the Spatial Blend slider allowing you to, at runtime, morphing sounds from 2D to 3D. Old projects will get imported in such a way that AudioSource's on GameObjects in the scene that have a clip assigned will get their Spatial Blend parameter set up according to the old 3D flag of the AudioClip. For obvious reasons this is not possible for scripts that dynamically assign clips to sources, so this requires manual fixing.

Also, while the default setting for the old 3D property was true, by default in the new system, the default of the Spatial Blend parameter is set to 2D.

### Format

The naming of the Format property has changed so that it reflects the method by which the data is stored rather than particular file format which deviate from platform to platform. So from now on Uncompressed refers to raw sample data, Compressed refers to a lossy compression method best suited to the platform and ADPCM refers to a lightweight (in terms of CPU) compression method that is best suited to natural audio signals which contain a fair amount of noise (footsteps, impacts, weapons etc) and are to be played in large amounts.

### Preloading and loading audio data in the background

A new feature of AudioClips is that they provide support for an option for determining whether to preload the audio data or not. Any property of the AudioClip is detached from the actual audio data loading state and can be queried at any time, so having the possibility to load on demand now helps keeping the memory usage of AudioClips low. Additionally to this AudioClips can load their audio data in the background without blocking the main game thread and causing frame drops. The load process can of course be controlled via the scripting API.

Finally, all AudioClips now support multi-editing, and the Force To Mono option now performs peak-normalization on the resulting down mix.

## Audio mixer

The AudioMixer is a new feature of Unity 5 allowing complex routing of the audio data from AudioSource's to mix groups where effects can

be applied. One key difference from the Audio Filter components is that audio filters get instantiated per AudioSource and therefore are more costly in terms of CPU if a game has a large number of AudioSources with filters or if a script simply creates many instances of a GameObject containing . With the mixer it is now possible to set up a group with the same effects and simply routing the audio from the AudioSource through the shared effects resulting in lower CPU usage.

The mixer does not currently support script-based effects, but it does have a new native audio plugin API allowing developers to write high-performance effects that integrate seamlessly with the other built-in effects.

---

## Data format changes

- Baked Occlusion Culling data format was changed. Rebaking is required for the occlusion culling data.
  - Baked Lighting data format was changed. Rebaking is required for the lighting data.
- 

## Plugins

Plugins are no longer required to be placed in Assets\Plugins\<Platform> folders, they now have settings for checking which platforms are compatible and platform specific settings (like setting compatible CPU, SDK's) etc. By default new plugins are marked as compatible with 'Any Platform'. To make transition easier from older Unity versions, Unity will check in what folder plugin is located, and set initial settings accordingly, for ex., if plugin is located in Assets\Plugins\Editor, it will be marked as compatible with Editor only, if plugin is located in Assets\Plugins\iOS, it will be marked as compatible with iOS only, etc. Also please refer to 'PluginInspector documentation' for more info.

## Native Plugins in the Editor

32-bit native plugins will not work in the 64-bit editor. Attempting to load them will result in errors being logged to the console and exceptions being thrown when trying to call a function from the native plugin.

To continue to use 32-bit native plugins in the editor, use the 32-bit editor version (provided as a separate installer).

To have both 32-bit and 64-bit versions of the same plugin in your project, simply put them in two different folders and set the supported platforms appropriately in the importer settings.

Restrictions:

- We currently do not provide a 32-bit editor for OSX.
- 

## Shaders

### Increased interpolator counts for some surface shaders

Built-in lighting pipeline in Unity 5 can in some cases use more texture coordinate interpolators (to get things like non-uniform mesh scale, dynamic GI etc. working). Some of your existing surface shaders might be running into texture coordinate limits, especially if they were targeting shader model 2.0 (default). Adding "#pragma target 3.0" can work around this issue. See <http://docs.unity3d.com/Manual/SL-ShaderPrograms.html> for the reference.

### Non-uniform mesh scale has to be taken into account in shaders

In Unity 5.0, non-uniform meshes are not "prescaled" on the CPU anymore. This means that normal & tangent vectors can be non-normalized in the vertex shader. If you're doing manual lighting calculations there, you'd have to normalize them. If you're using Unity's surface shaders, then all necessary code will be generated for you.

### Sorting by material index has been removed.

Unity no longer sorts by material index in the forward renderloop. This improves performance because more objects can be rendered without state changes between them. This breaks compatibility for content that relies on material index as a way of sorting. In 4.x a mesh with two materials would always render the first material first, and the second material second. In Unity 5 this is not the case, the order depends on what reduces the most state changes to render the scene.

### Fixed function TexGen, texture matrices and some SetTexture combiner modes were removed

Unity 5.0 removed support for this fixed function shader functionality:

- UV coordinate generation (TexGen command).
- UV transformation matrices (Matrix command on a texture property or inside a SetTexture).
- Rarely used SetTexture combiner modes: signed add (a+-b), multiply signed add (a\*b+-c), multiply subtract (a\*b-c), dot product (dot3, dot3rgba).

Any of the above will do nothing now, and shader inspector will show warnings about their usage. You should rewrite affected shaders using programmable vertex+fragment shaders instead. All platforms support them nowadays, and there are no advantages whatsoever to use fixed function shaders.

If you have fairly old versions of Projector or Water shader packages in your project, the shaders there might be using this functionality. Upgrade the packages to 5.0 version.

## Mixing programmable & fixed function shader parts is not allowed anymore

Mixing partially fixed function & partially programmable shaders (e.g. fixed function vertex lighting & pixel shader; or a vertex shader and texture combiners) is not supported anymore. It was never working on mobile, consoles or DirectX 11 anyway. This required changing behavior of Legacy/Reflective/VertexLit shader to not do that - it lost per-vertex specular support; on the plus side it now behaves consistently between platforms.

## D3D9 shader compiler was switched from Cg 2.2 to HLSL

Mostly this should be transparent (just result in less codegen bugs and slightly faster shaders). However HLSL compiler can be slightly more picky about syntax. Some examples:

- You need to fully initialize output variables. Use UNITY\_INITIALIZE\_OUTPUT helper macro, just like you would on D3D11.
- "float3(a\_4\_component\_value)" constructors do not work. Use "a\_4\_component\_value.xyz" instead.

## "unity\_Scale" shader variable has been removed

The "unity\_Scale" shader property has been removed. In 4.x unity\_Scale.w was the 1 / uniform Scale of the transform, Unity 4.x only rendered non-scaled or uniformly scaled models. Other scales were performed on the CPU, which was very expensive & had an unexpected memory overhead.

In Unity 5.0 all this is done on the GPU by simply passing matrices with non-uniform scale to the shaders. Thus unity\_Scale has been removed because it can not represent the full scale. In most cases where "unity\_Scale" was used we recommend instead transforming to world space first. In the case of transforming normals, you always have to use normalize on the transformed normal now. In some cases this leads to slightly more expensive code in the vertex shader.

```
// Unity 4.x
float3 norm = mul ((float3x3)UNITY_MATRIX_IT_MV, v.normal * unity_Scale.w);
// Becomes this in Unity 5.0
float3 norm = normalize(mul ((float3x3)UNITY_MATRIX_IT_MV, v.normal));
```

```
// Unity 4.x
temp.xyzw = v.vertex.xzxx * unity_Scale.xzxx * _WaveScale4 + _WaveOffset;

// Becomes this in Unity 5.0
float4 wpos = mul (_Object2World, v.vertex);
temp.xyzw = wpos.xzxx * _WaveScale4 + _WaveOffset;
```

## Shadows, Depth Textures and ShadowCollector passes

Forward rendered directional light shadows do not do separate "shadow collector" pass anymore. Now they calculate screenspace shadows from a camera's depth texture (just like in deferred lighting).

This means that LightMode=ShadowCollector passes in shaders aren't used for anything; you can just remove them from your shaders.

Depth texture itself is not generated using shader replacement anymore; it is rendered with ShadowCaster shader passes. This means that as long as your objects can cast proper shadows, then they will also appear in camera's depth texture properly (was very hard to do before, if you wanted custom vertex animation or funky alpha testing). It also means that Camera-DepthTexture.shader is not used for anything now. And also, all built-in shadow shaders used no backface culling; that was changed to match culling mode of regular rendering.

---

## Physics

Unity 5.0 features an upgrade to PhysX3.3 SDK. Please give [this blogpost](#) a quick look before taking any action on your 4.x projects. It should give you a taste of what to expect from the new codebase.

Please be warned that PhysX3 is not 100% compatible with PhysX2 and requires some actions from the user when upgrading.

## General overview

Unity 5.0 physics could be expected to work up to 2x faster than in previous versions. Most of the components you were familiar with are still there, and you will find them working as before. Of course, some behaviours were impossible to get the same and some were just weird behaviours caused by limitations of the pre-existed codebase, so we had to take changes. The two areas that got the most significant change are Cloth component and WheelCollider component. We're including a section about each of them below. Then, there are smaller changes all over the physics code that cause incompatibility.

## Changes that are likely to affect projects.

Adaptive force is now switched off by default (but you can switch it on in the editor physics properties: Edit -> Project settings -> Physics -> Enable adaptive force). Adaptive force was introduced to help with the simulation of large stacks, but it turned out to be great only for demos. In real games it happened to cause wrong behaviour.

Smooth sphere collisions are removed both from terrain and meshes. PhysX3 has the feature that addresses the same issue and it's no longer switchable as it's considered to be a solution without major drawbacks.

Springs expose larger amplitudes with PhysX3, you may want to tune spring parameters after upgrading.

Use `TerrainCollider.sharedMaterial` and `TerrainCollider.material` to specify physics material for terrain. The older way of setting physics material through the `TerrainData` will no longer work. As a bonus, you can now specify terrain physics materials on a per collider basis.

Shape casting and sweeping has changed:

- shape sweeps report all shapes they hit (i.e. `CapsuleCast` and `SphereCast` would return all shapes they hit, even the ones that fully contain the primitive)
- raycast filter out shapes that contain the raycast origin

When using compound colliders, `OnCollisionEnter` is now called once per each contact pair

From now on, you can have triggers only on convex shapes (a PhysX restriction):

- `TerrainCollider` no longer supports `IsTrigger` flag
- `MeshCollider` can have `IsTrigger` only if it's convex

Dynamic bodies (i.e. those having `Rigidbody` attached with `Kinematic = false`) can no longer have concave mesh colliders (a PhysX limitation).

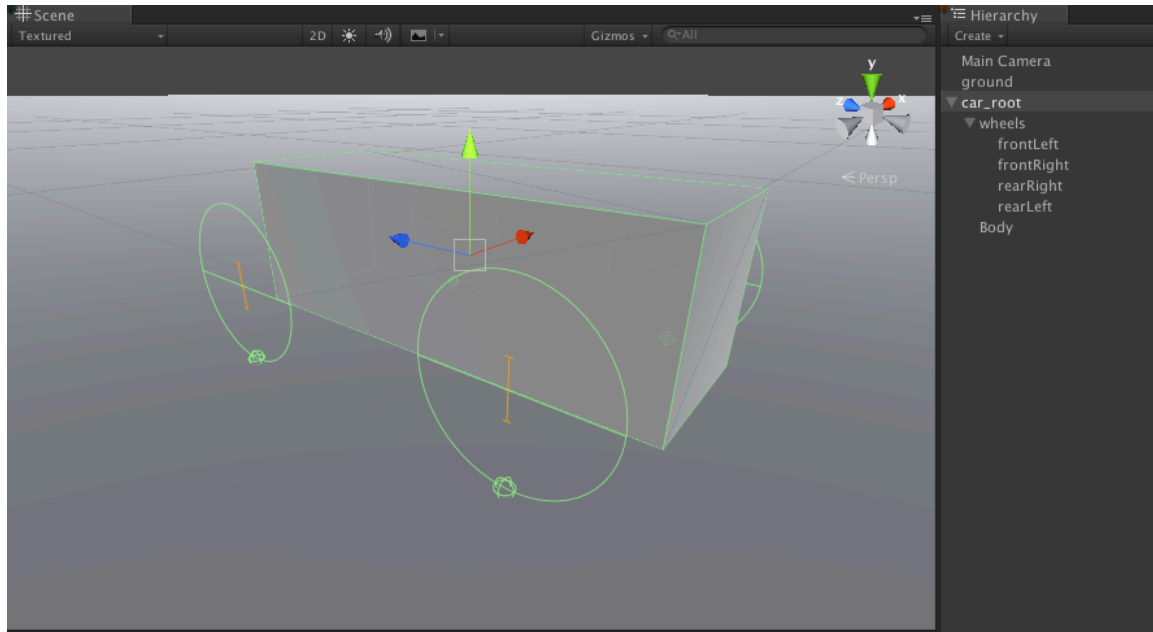
- if you want to have collisions with concave meshes, you can only have them on static colliders and kinematic bodies

## WheelCollider

The new WheelCollider is powered by the PhysX3 Vehicles SDK that is basically a completely new vehicle simulation library when compared to the code we had with PhysX2.

Let's go through the process of creating a basic functioning car in Unity 5.0.

1. Start by Game Object -> Create General -> Plane. This is the ground the car is going to drive on. Make sure the ground has zero transform (Transform -> Reset) for simplicity. Scale it by putting something like 100 in Transform scale components.
2. Create a basic car skeleton:
  - a. First, add the car root object: Game Object -> Create Empty. Change the name to `car_root`.
  - b. Add `Rigidbody` component to `car_root`. The default mass of 1 kg is way too light for the default suspension settings. Change to 1500 kg.
  - c. Now create the car body collider: Game Object -> Create General -> Box. Parent the box under `car_root`. Reset the transform to make it aligned perfectly in local space. Since our car is going to be oriented along the Z axis, scale the box along the Z axis, put 3 in z scaling.
  - d. Add the wheels root. Select `car_root` and Game Object -> Create Empty Child. Change the name to `wheels`. Reset the transform on it. This node is not mandatory, but is for tuning convenience later.
  - e. Create the first wheel: select `wheels` object, Game Object -> Create Empty Child, name it `frontLeft`. Reset the transform on it. Input (-1, 0, 1) as position.
  - f. Duplicate `frontLeft` object (Cmd-D or Control-D). Change the x position to 1. Change the name to `frontRight`.
  - g. Select both the `frontLeft` and `frontRight` objects. Duplicate them. Change the z position of both objects to -1. Change the names to `rearLeft` and `rearRight` respectively.
  - h. Finally, select the `car_root` object and using the transform manipulators, raise it slightly above the ground.
3. Now in play mode, you should be able to see something like this:



4. To make this car actually drivable we need to write a controller for it. Let's dive into some scripting:

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

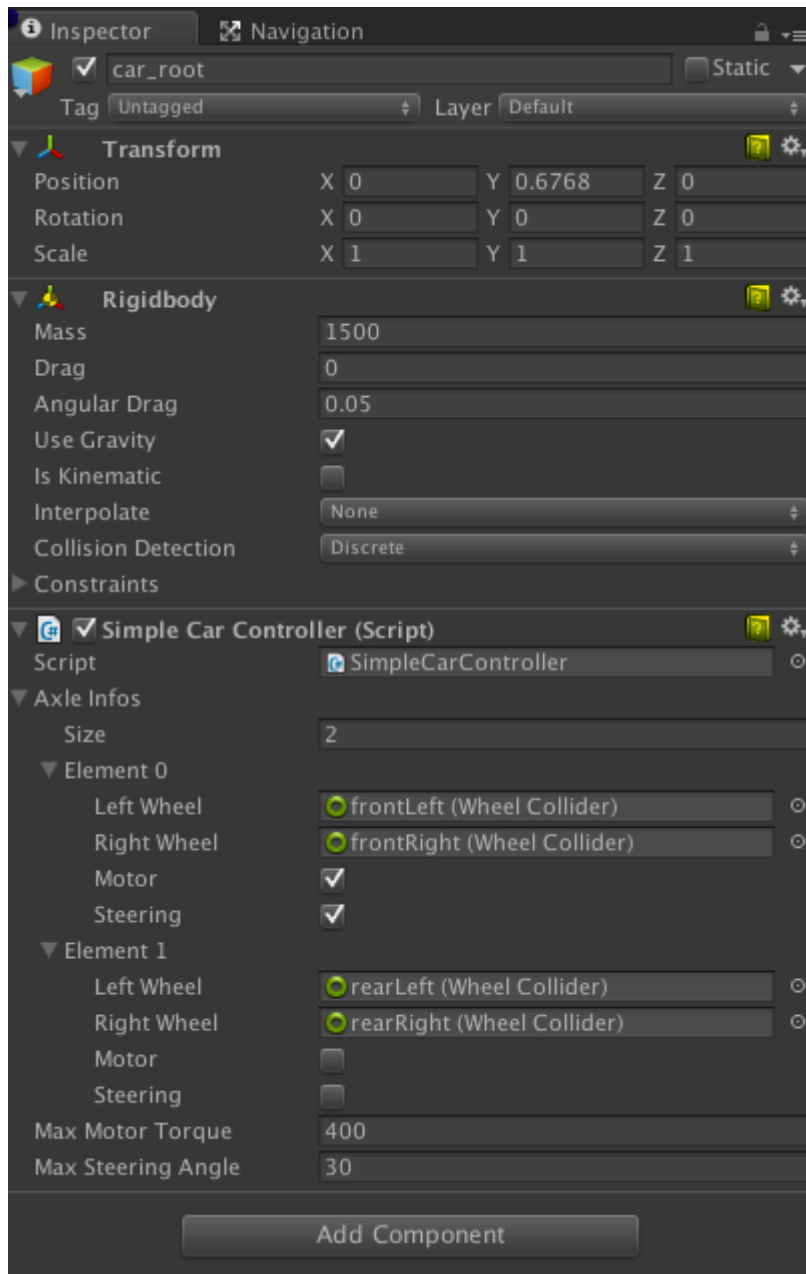
[System.Serializable]
public class AxleInfo {
    public WheelCollider leftWheel;
    public WheelCollider rightWheel;
    public bool motor; // is this wheel attached to motor?
    public bool steering; // does this wheel apply steer angle?
}

public class SimpleCarController : MonoBehaviour {
    public List<AxleInfo> axleInfos; // the information about each individual axle
    public float maxMotorTorque; // maximum torque the motor can apply to wheel
    public float maxSteeringAngle; // maximum steer angle the wheel can have

    public void FixedUpdate()
    {
        float motor = maxMotorTorque * Input.GetAxis("Vertical");
        float steering = maxSteeringAngle * Input.GetAxis("Horizontal");

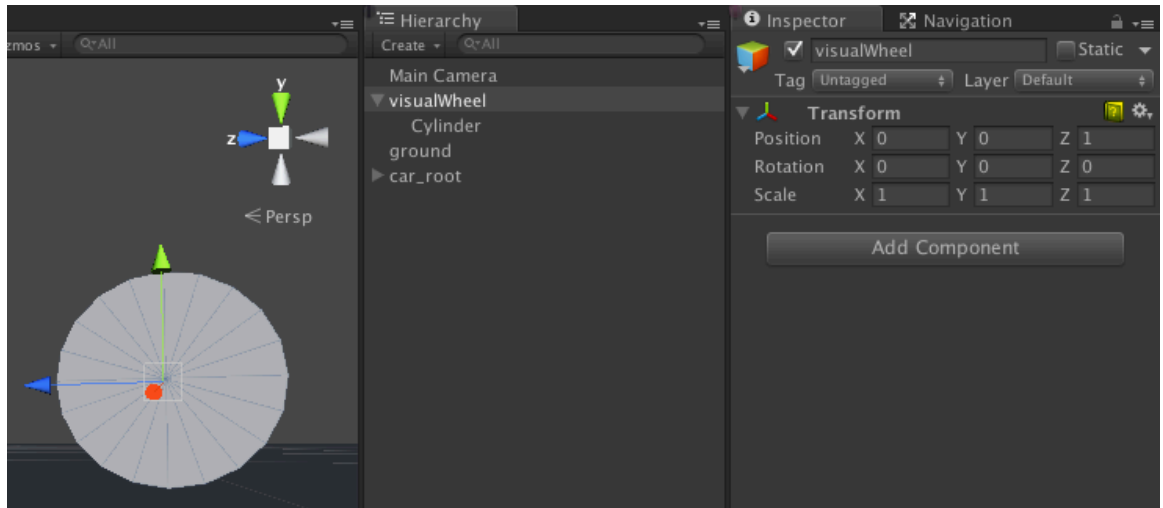
        foreach (AxleInfo axleInfo in axleInfos) {
            if (axleInfo.steering) {
                axleInfo.leftWheel.steerAngle = steering;
                axleInfo.rightWheel.steerAngle = steering;
            }
            if (axleInfo.motor) {
                axleInfo.leftWheel.motorTorque = motor;
                axleInfo.rightWheel.motorTorque = motor;
            }
        }
    }
}
```

Just drop this snippet on the car\_root object, tune the script parameters and kick off to play mode. Here you can have some fun with settings, I'm just attaching my parameters that seemed to work reasonably well:

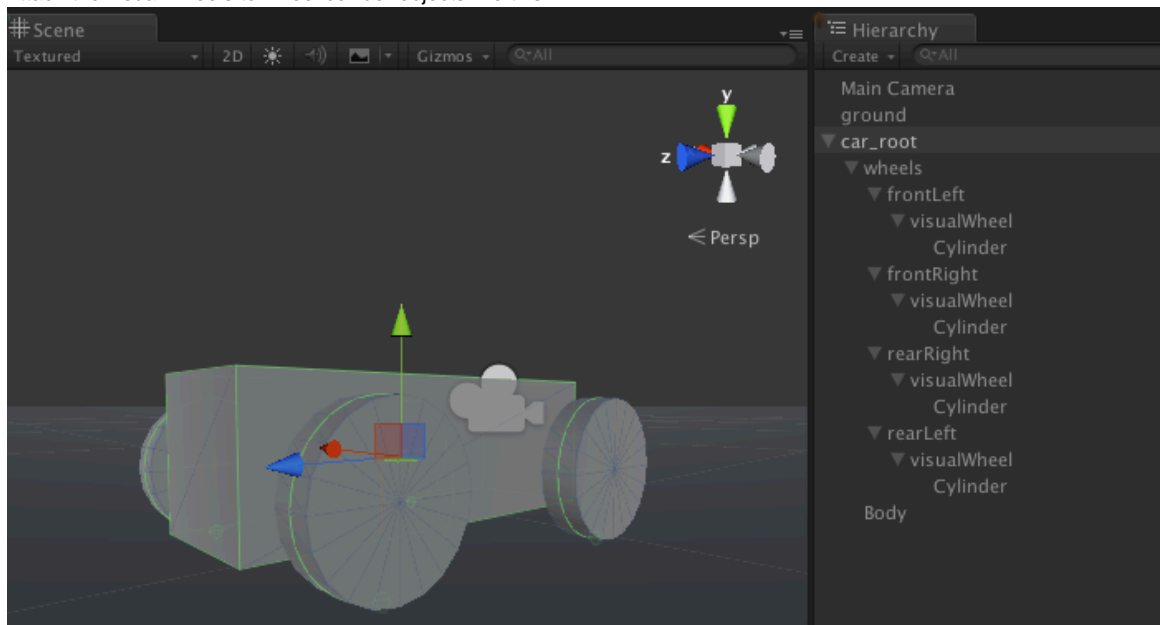


You can have up to 20 wheels on a single vehicle instance with each of them applying steering, motor or braking torque.

5. Moving on to visual wheels. As you could notice, WheelCollider don't apply the simulated wheel position and rotation back to WheelCollider's transform. So adding visual wheel requires some tricks.
  - a. We need some wheel geometry here. I've made a simple wheel shape out of a cylinder:



- b. Now there could be several approaches to adding visual wheels: making it so that we have to assign visual wheels manually in script properties or writing some logic to find the corresponding visual wheel automatically. We'll follow the second approach.
- c. Attach the visual wheels to wheel collider objects like this:



- d. Now change the controller script:



```

using UnityEngine;
using System.Collections;
using System.Collections.Generic;

[System.Serializable]
public class AxleInfo {
    public WheelCollider leftWheel;
    public WheelCollider rightWheel;
    public bool motor;
    public bool steering;
}

public class SimpleCarController : MonoBehaviour {
    public List<AxleInfo> axleInfos;
    public float maxMotorTorque;
    public float maxSteeringAngle;

    // finds the corresponding visual wheel
    // correctly applied transform
    public void ApplyLocalPositionToVisuals(WheelCollider collider)
    {
        if (collider.transform.childCount == 0) {
            return;
        }

        Transform visualWheel = collider.transform.GetChild(0);

        Vector3 position;
        Quaternion rotation;
        collider.GetLocalPose(out position, out rotation);

        // visualWheel is parented to wheel collider, so we need to do a
        little space transformation here
        visualWheel.transform.position =
        collider.transform.parent.TransformPoint(position);
        visualWheel.transform.rotation =
        collider.transform.parent.rotation * rotation;
    }

    public void FixedUpdate()
    {
        float motor = maxMotorTorque * Input.GetAxis("Vertical");
        float steering = maxSteeringAngle * Input.GetAxis("Horizontal");

        foreach (AxleInfo axleInfo in axleInfos) {
            if (axleInfo.steering) {
                axleInfo.leftWheel.steerAngle = steering;
                axleInfo.rightWheel.steerAngle = steering;
            }
            if (axleInfo.motor) {
                axleInfo.leftWheel.motorTorque = motor;
                axleInfo.rightWheel.motorTorque = motor;
            }
            ApplyLocalPositionToVisuals(axleInfo.leftWheel);
            ApplyLocalPositionToVisuals(axleInfo.rightWheel);
        }
    }
}

```

Unity 5 uses the completely rewritten cloth solver provided by the new PhysX SDK. This cloth solver has been designed with character clothing in mind, and is a great improvement compared to the old version in terms of performance and stability. Unity 5 replaces the SkinnedCloth and InteractiveCloth components in Unity 4 with a single Cloth component, which works in conjunction with a SkinnedMeshRenderer. The functionality is similar to the previous SkinnedCloth component, but it is now possible to assign arbitrary, non-skinned meshes to the SkinnedMeshRenderer, so you can still handle cloth simulation on any random mesh. However, some functionality which was available on the old InteractiveCloth is now no longer supported by the new version of PhysX as it is difficult to implement these with good performance. Specifically, you can no longer use cloth to collide with arbitrary world geometry, tearing is no longer supported, you can no longer apply pressure on cloth, and you can no longer attach cloth to colliders or have cloth apply forces to rigidbodies in the scene.

---

## Standalone Player

Cursor lock and cursor visibility are now independent of each other.

```
// Unity 4.x
Screen.lockCursor = true;

// Becomes this in Unity 5.0
Cursor.visible = false;
Cursor.lockState = CursorLockMode.Locked;
```

## Windows Phone 8

When building over existing Visual Studio project you might get errors stating that UnityPlayer.UnityApp does not contain definitions for IsLocationEnabled, EnableLocationService and SetupGeolocator. Location services are now automatically initialized by Unity so you can safely remove locations code from OnNavigatedTo method and entire SetupGeolocator method in MainPage.xaml.cs file.

## Windows Store Apps

'Metro' keyword was replaced to 'WSA' in most APIs, for ex., BuildTarget.MetroPlayer became BuildTarget.WSAPlayer, PlayerSettings.Metro became PlayerSettings.WSA. Defines in scripts like UNITY\_METRO, UNITY\_METRO\_8\_0, UNITY\_METRO\_8\_1 are still there, but along them there will be defines UNITY\_WSA, UNITY\_WSA\_8\_0, UNITY\_WSA\_8\_1.

---

## Other script API changes that cannot be upgraded automatically

- UnityEngine.AnimationEvent is now a struct. Comparisons to 'null' will result in compile errors.
- GetComponent(string) / AddComponent(string), when called with a variable cannot be automatically updated to the generic versions - GetComponent<T>() / AddComponent<T>(). In such cases the API Updater will replace the call with a call to APIUpdaterRuntimeServices.GetComponent() / APIUpdaterRuntimeServices.AddComponent(). These methods are meant to allow you to test your game in editor mode (they do a best effort to try to resolve the type at runtime) but they are not meant to be used in production, so it is an error to build a game with calls to such methods). On platforms that support Type.GetType(string) you can try to use GetComponent(Type.GetType(typeName)) as a workaround.
- AssetBundle.Load, AssetBundle.LoadAsync and AssetBundle.LoadAll have been deprecated. Use AssetBundle.LoadAsset, AssetBundle.LoadAssetAsync and AssetBundle.LoadAllAssets instead. Script updater cannot update them as the loading behaviors have changed a little. In 5.0 all the loading APIs will not load components any more, please use the new loading APIs to load the game object first, then look up the component on the object.