Help and support

For help or questions:

- email: tmulgrew@slingshot.co.nz
- or post to this Unity discussion forum thread: https://forum.unity.com/threads/released-racetrack-builder.644332/

Also see the Unity asset store page for tutorial videos:
https://assetstore.unity.com/packages/tools/modeling/racetrack-builder-138353

Overview

This racetrack building system allows you to build racetracks by warping meshes around a series of curves. It is particularly suited to building elevated, rollercoaster-like tracks, but can also integrate with Unity terrains to create ground based tracks. It features the ability to create jumps, banked corners, loops and branching tracks.

The resulting mesh is fully drivable - for example using a 3rd party Car Controller asset - and contains runtime information for tracking a vehicle's progress.

Importing

Updating source files

When importing Racetrack Builder into recent Unity versions you will be prompted to automatically update the source files due to API changes.

Select one of the "Yes" options, e.g. "Yes, just for these files".

Racetrack Build should then function correctly.

Note: the changes are not drastic. Unity simply changes `.velocity` to `.linearVelocity` in a few places.

Universal Render Pipeline (URP)

Racetrack Builder supports URP, but the Built-in pipeline materials for the default track types must be converted to URP.

The materials are stored in `Assets/Racetrack Builder/Materials` in the project structure.

Select all the materials and click `Edit > Rendering > Materials > Convert Selected Built-In Materials to URP` from the main menu. Then repeat for the materials in the "Arcade" subfolder.

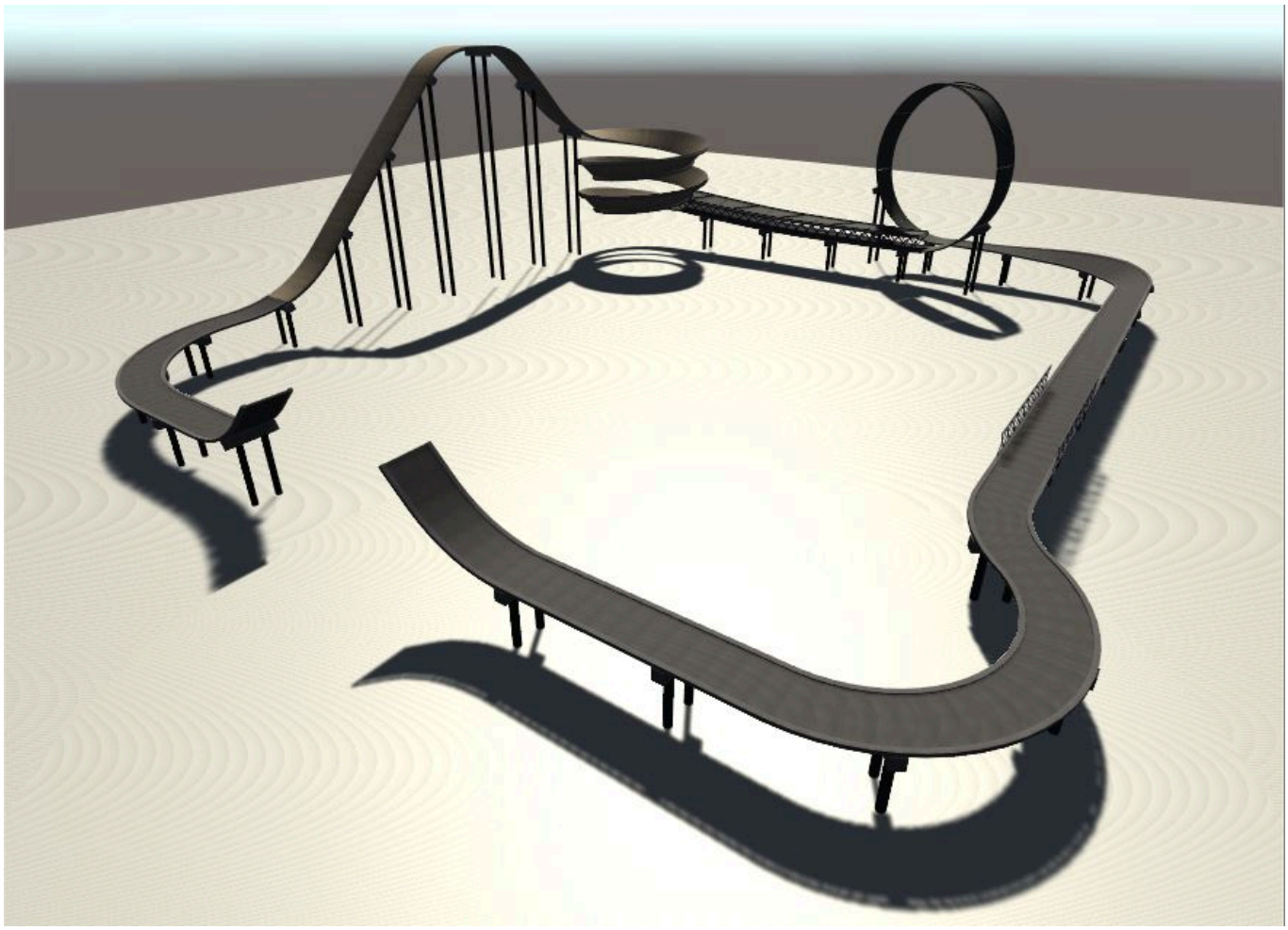
High Definition Render Pipeline (HDRP)

Racetrack Builder supports HDRP, but the Built-in pipeline materials for the default track types must be converted to HDRP.

Click `Edit > Rendering > Materials > Convert All Built-in Materials to HDRP` from the main menu.

Example scene

See Example1, -2, -3, -4 scenes in the Assets/Racetrack Builder folder for examples of various techniques.

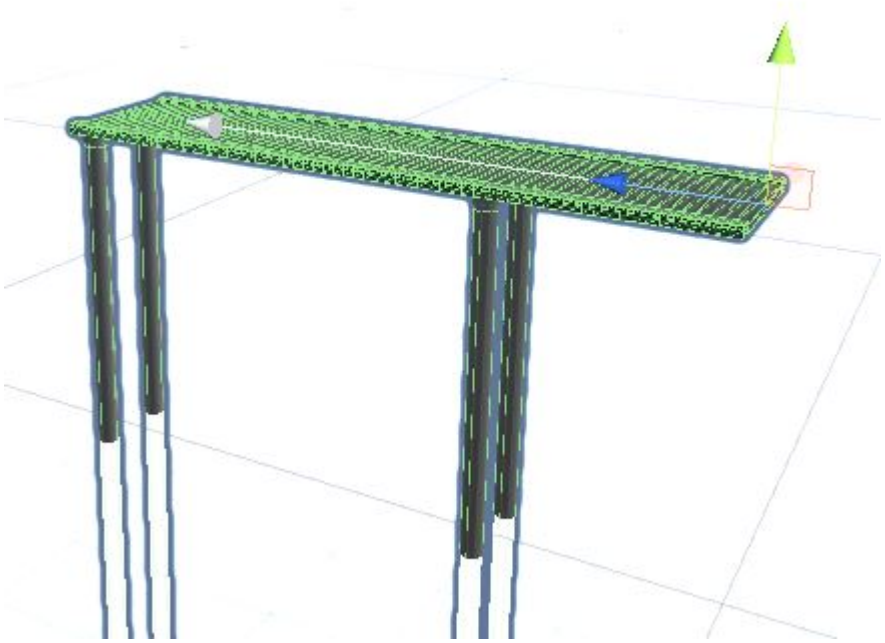


Quick start - Creating a track

It only takes a few minutes to get started building a race track:

First create or open a scene (File > New Scene), and make sure it has some ground (e.g. a plane).

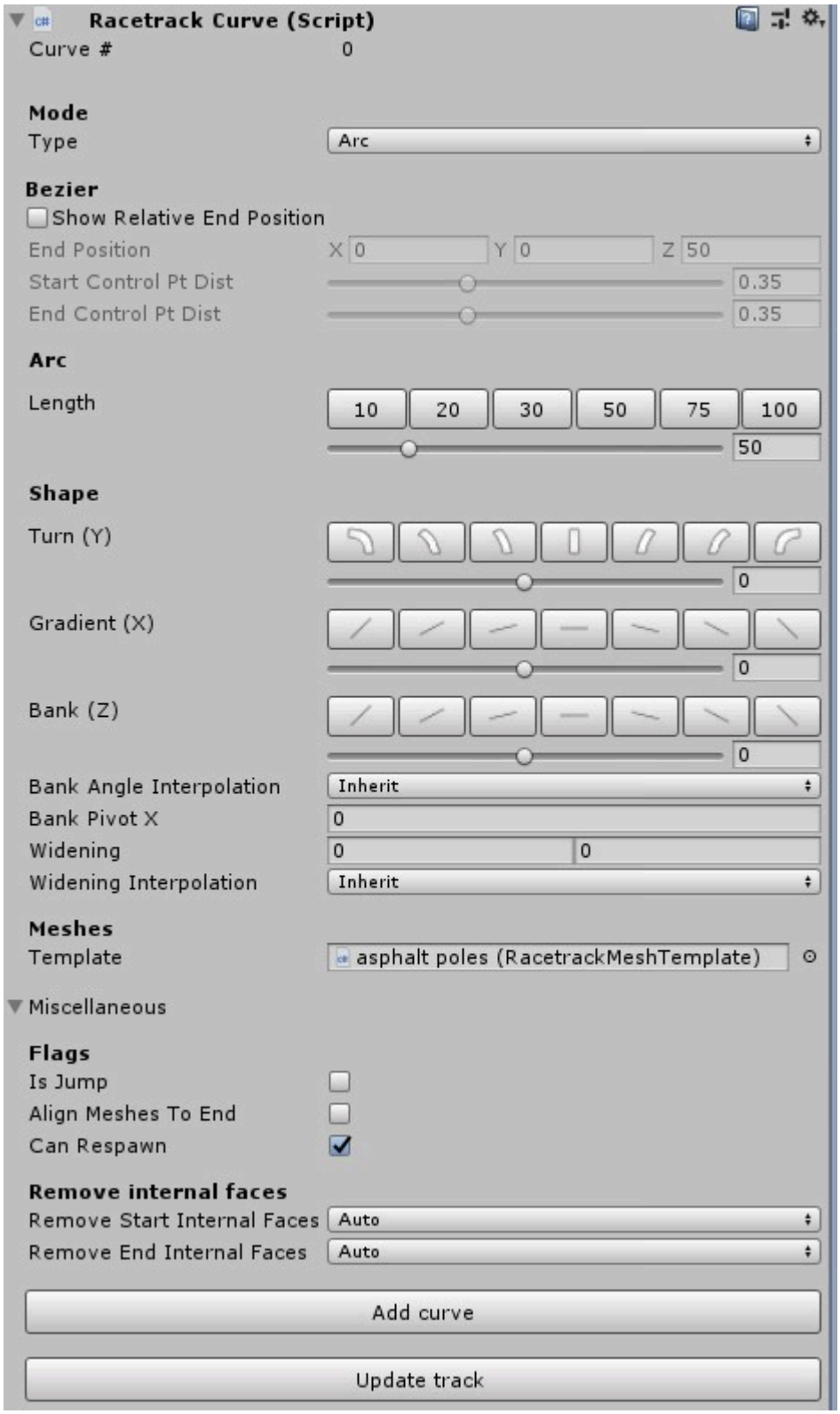
Now create the racetrack object by creating "GameObject > 3D Object > Racetrack" from the menu.



Note: If you need to reposition the racetrack, be sure to update the transform on the main ***Racetrack object*** *not the curve. Racetrack Builder automatically manages the curves' positions for you, and will overwrite any changes you make.*

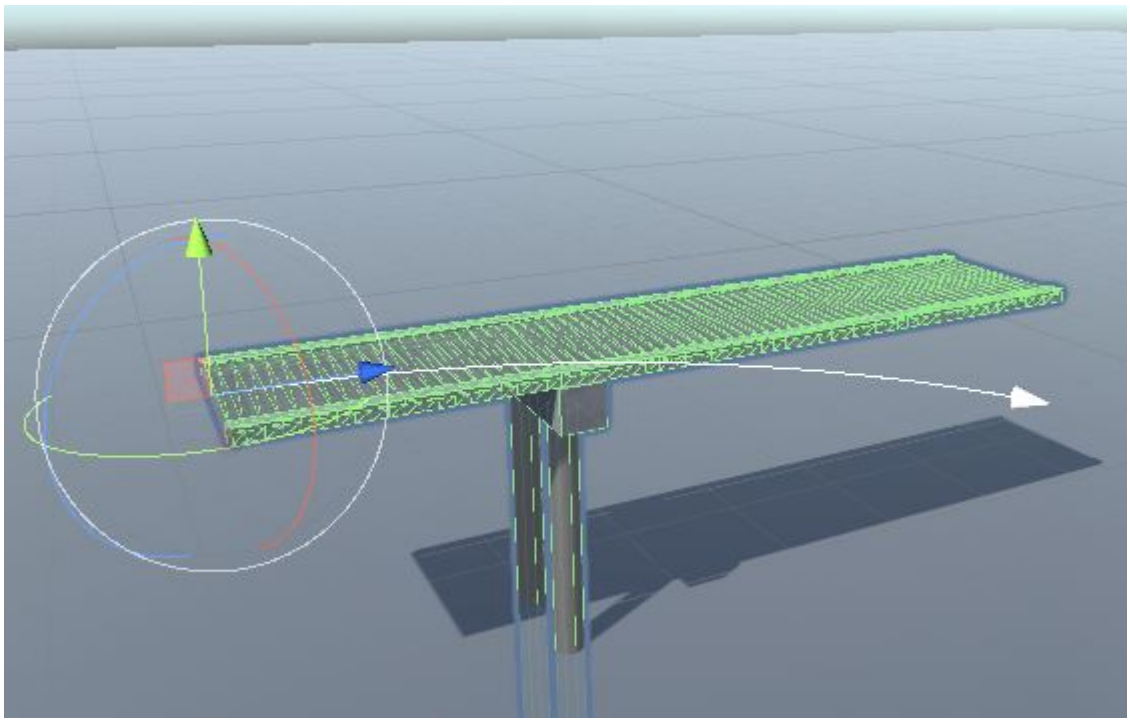
Racetrack Builder has created a racetrack with a single curve object. The white arrow shows the path of the curve, and the mesh has been fitted around it. The curve is initially a straight line, so the road mesh is accordingly straight.

The curve is automatically selected, and the Inspector window shows the UI for editing it:



Try clicking the buttons in the Shape section to see how the track adapts to the updated curve.

For finer control you can set the values via the slider or key them in. The curve line will update to reflect your change but the meshes will not update immediately (so that the slight delay does not interfere with the slider/typing).



Click on the "Update track" button to regenerate the meshes and bring the model up to date.

Once you're happy with the initial curve, click the "Add curve" button to create a new curve and add it onto the end of the racetrack.

This is the basic workflow for Racetrack Builder.

More about track building

Selecting corners

If you need to go back and change a previous corner, there are a couple of ways to do this.

You can expand the "Racetrack" object in the scene tree and click on the corresponding "Curve" child object.

Alternatively, you can click on the track in the editor view. This selects the curve that generated the part of the of track you clicked.

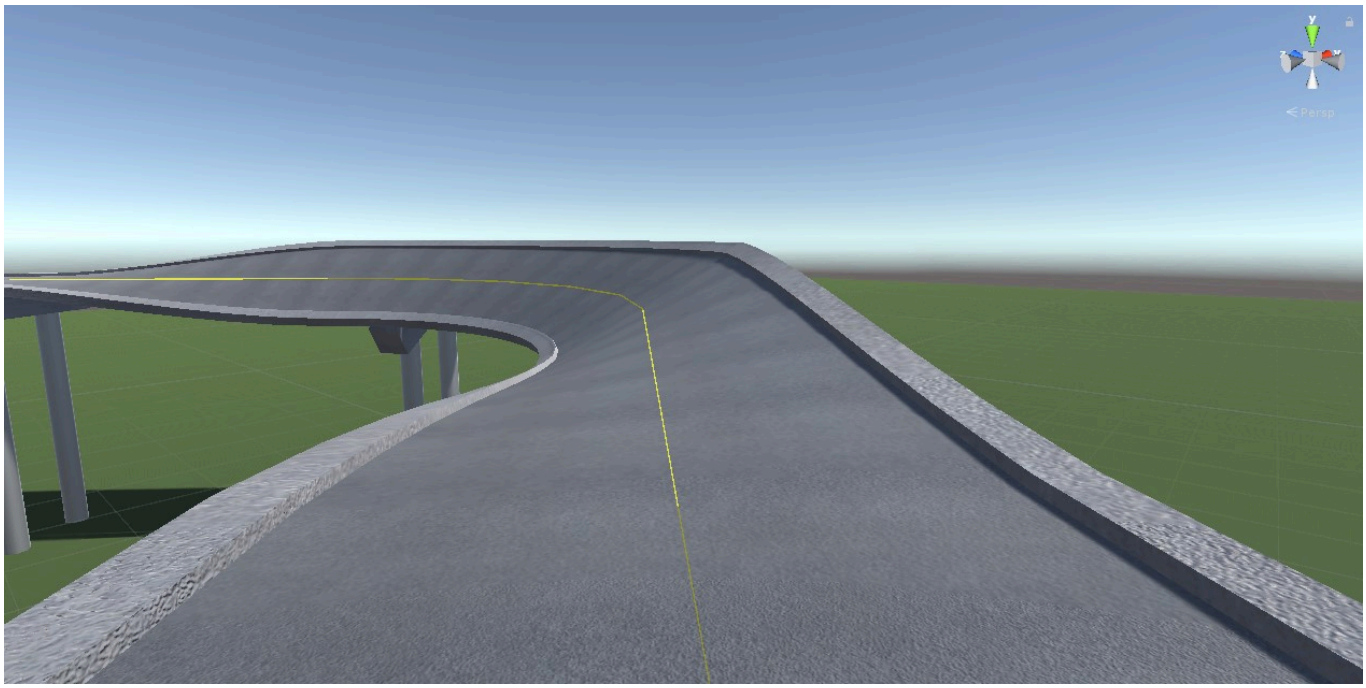
Undo and redo

When the racetrack model is updated, Racetrack Builder does not register the changes for Undo/Redo tracking. This is mainly for speed and to preserve memory, as Unity has to store a full copy of each version of each continuous mesh, and all its vertices. (And it's partly for performance too.)

However this means that undo will not automatically update the racetrack model, it will only undo the changes to the curve configuration. So after performing an undo, click the "Update track" to bring the racetrack model back in line with the curves.

Bank Pivot

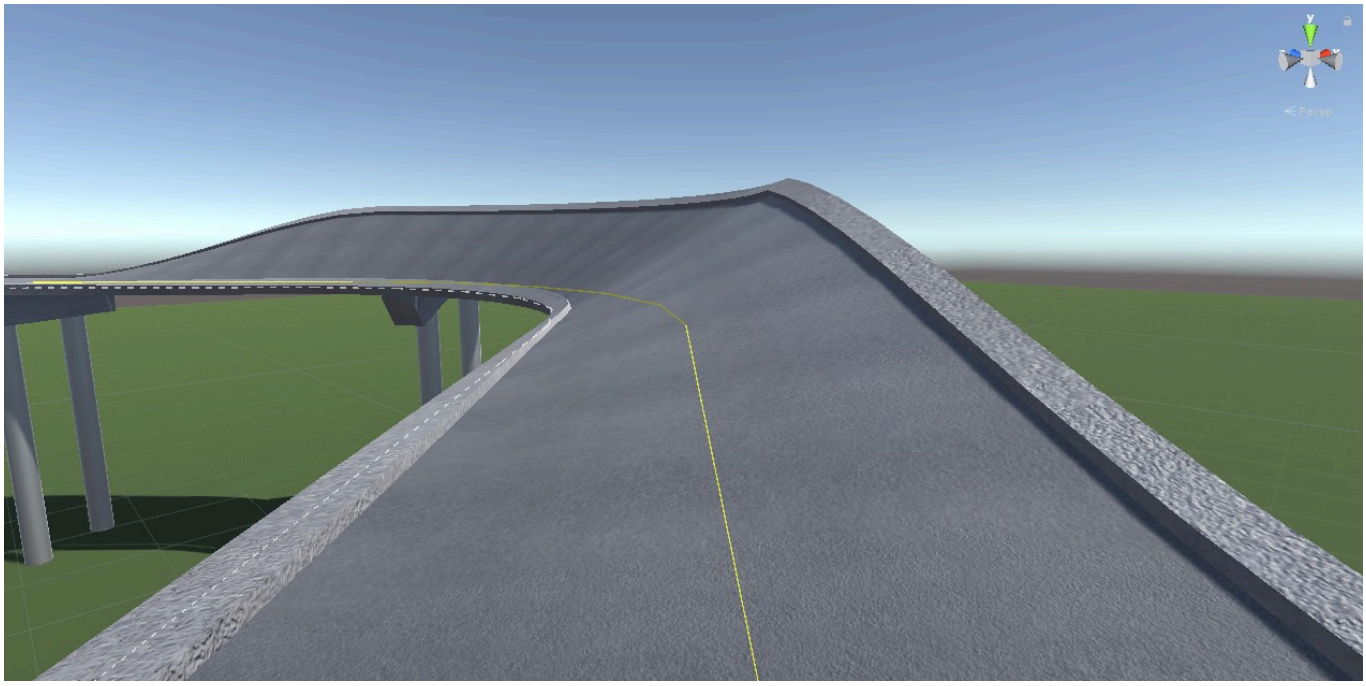
When you apply a "Bank (Z)" angle to the track, by default it will raise one side and lower the other by equal amounts. Essentially the track pivots around the center.



Sometimes you may require a different pivot point. For example, oval racetracks often bank the track by keeping the inside of the turn at the same level and raising the outside track.

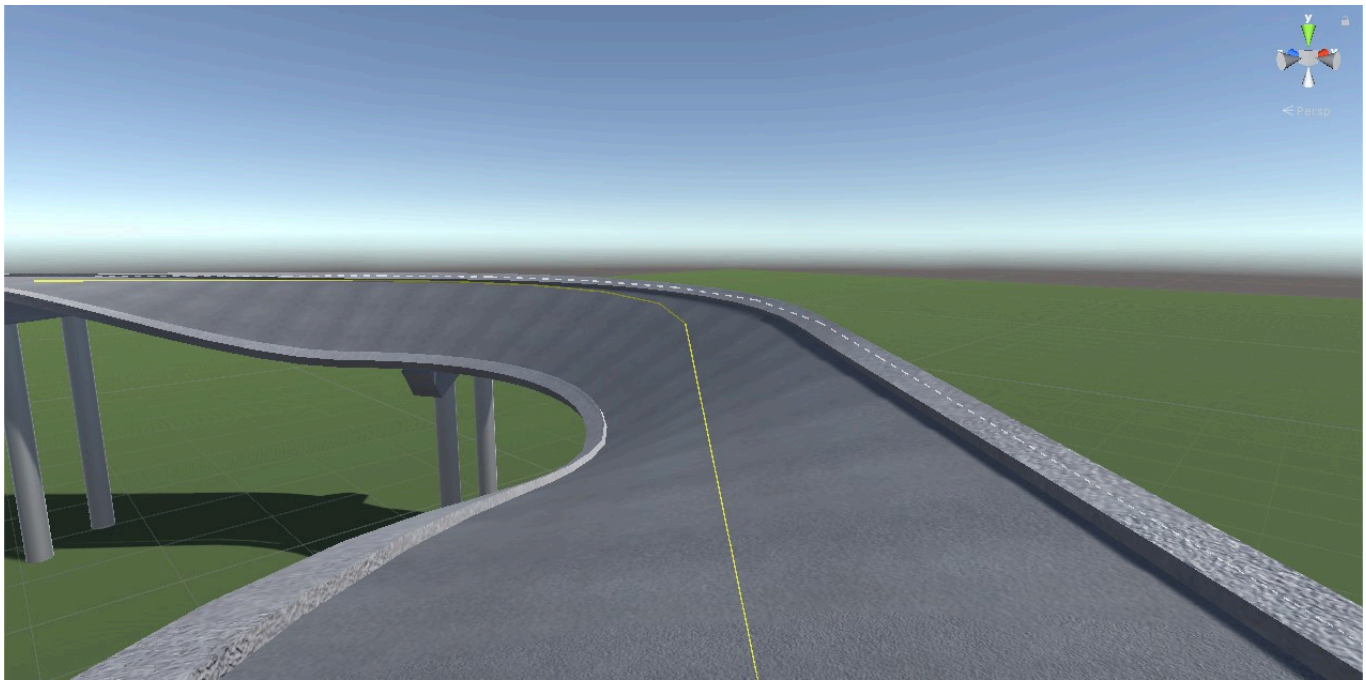
To achieve this, set the curve's "Bank Pivot X" property to the relative X position in world units.

For example, the default Racetrack Builder track types are 14 units wide, so setting the "Bank Pivot X" to -7 will pivot around the left hand edge, and 7 will pivot around the right.



Racetrack Builder displays the pivot as a dotted line when a curve or racetrack object is selected.

Note that "Bank Pivot X" specifies the value for the end of the curve. Racetrack Builder will linearly interpolate between the previous value along the length of the curve.



Changing the track type

You can change the type of track by dragging a "track templates" prefab and dropping it on the "Template" field in the properties of the corresponding curve.

Track templates are located in project folders:

- Assets/Racetrack builder/Prefabs/Track Templates
- Assets/Racetrack builder/Prefabs/Track Templates/Higher res

The "Higher res" versions have extra vertices and polygons that allow them to warp better, making them smoother in appearance and to drive on, particularly on banked corners. I recommend using them unless you have a very limited polygon budget.

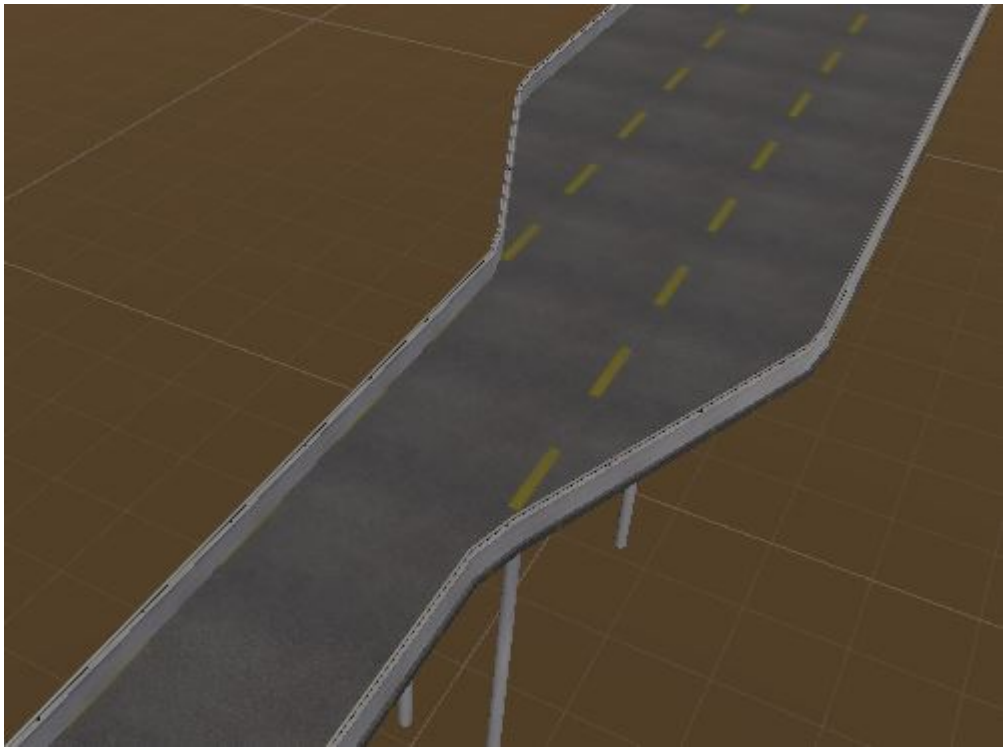
This will change the track type from that curve onwards, until it reaches another curve with a track type set (i.e. not set to "None").

Note: This action does not update the track automatically. Click "Update track" to apply the change.

Changing the road width

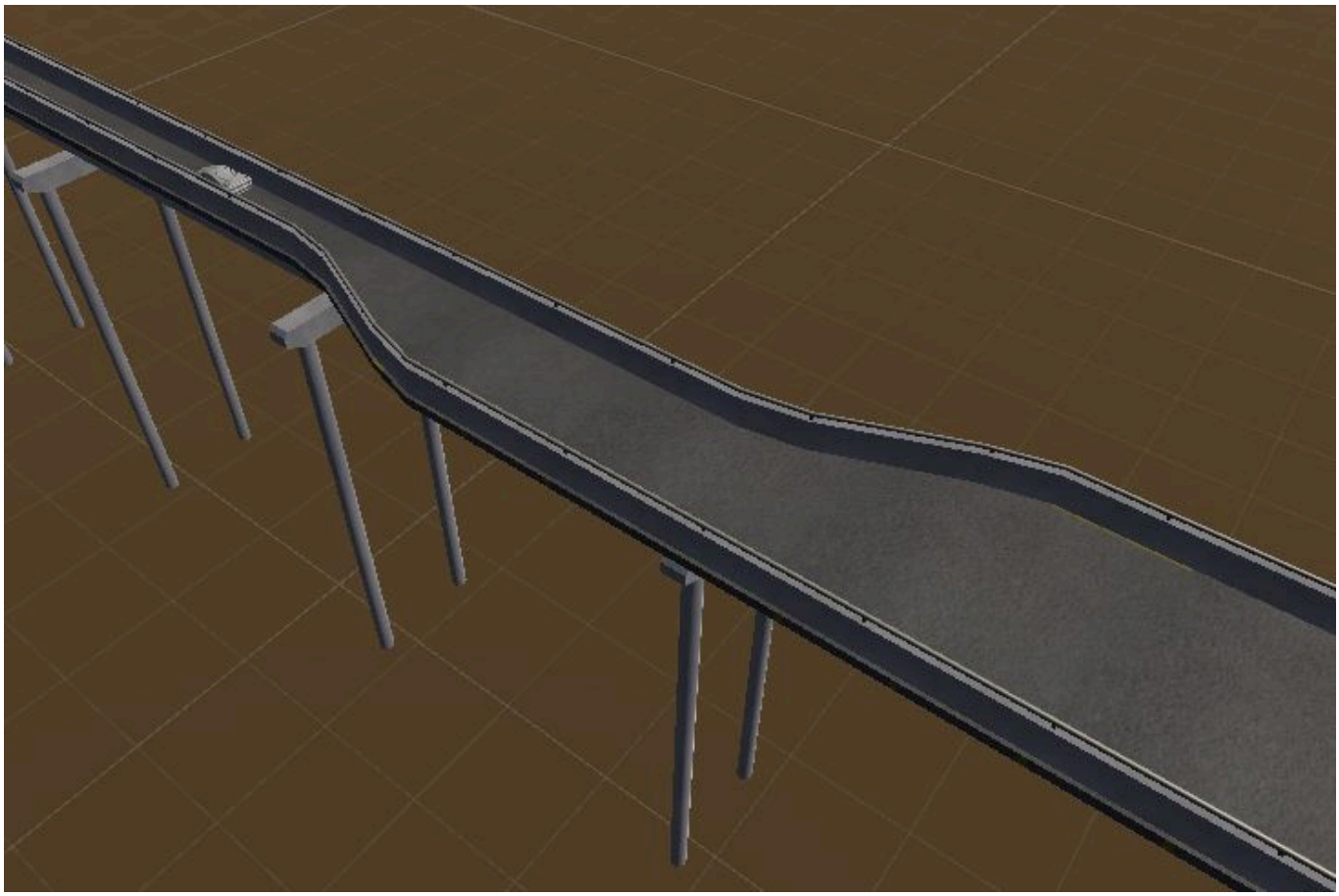
To make the racetrack wider (or narrower) adjust the "Widening" values.

There are two values - left and right - which extend the left and right hand side of the racetrack by the specified number of units respectively.



Note that these values are *added* to the original width.

If the track type supports it, you can also narrow the track, by specifying negative values. Be aware that track types typically will enforce a minimum width beyond which they will not go any narrower.

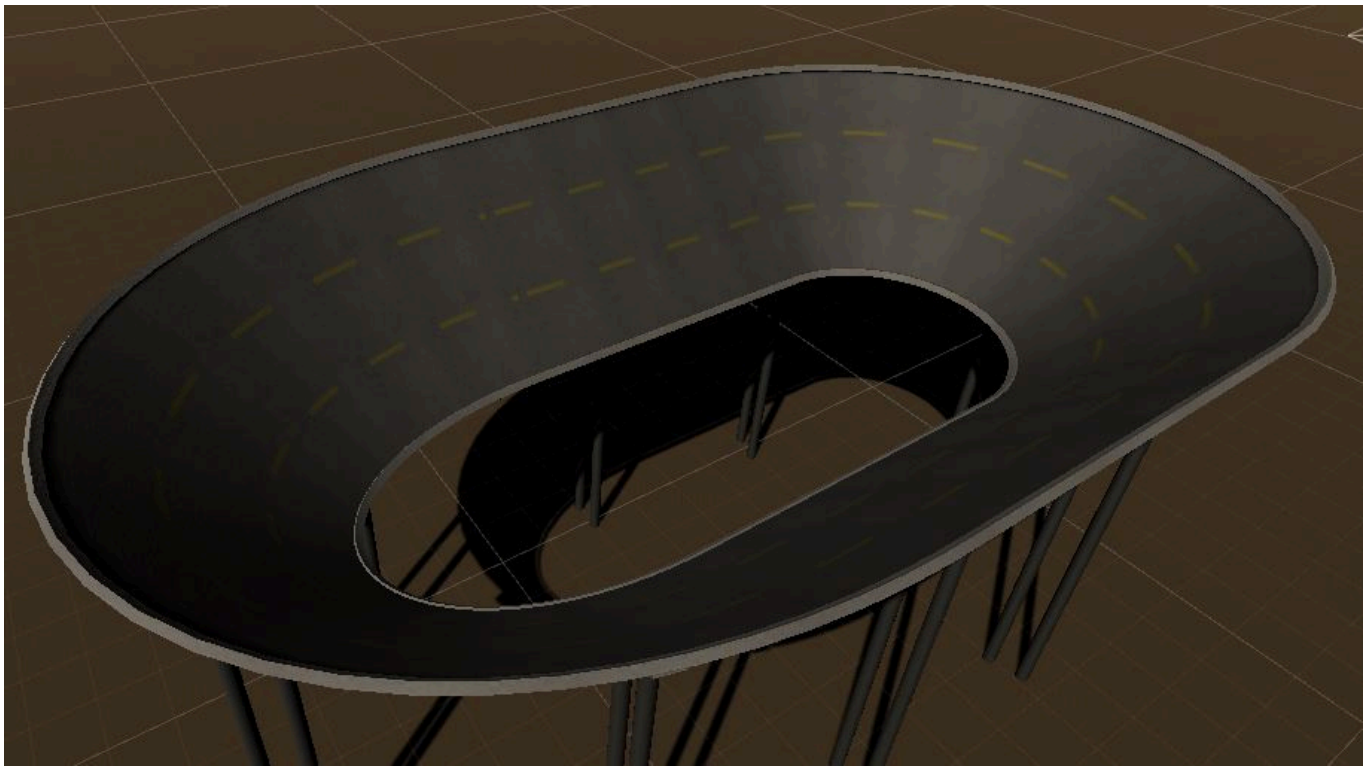


Note: This action does not update the track automatically. Click "Update track" to apply the change.

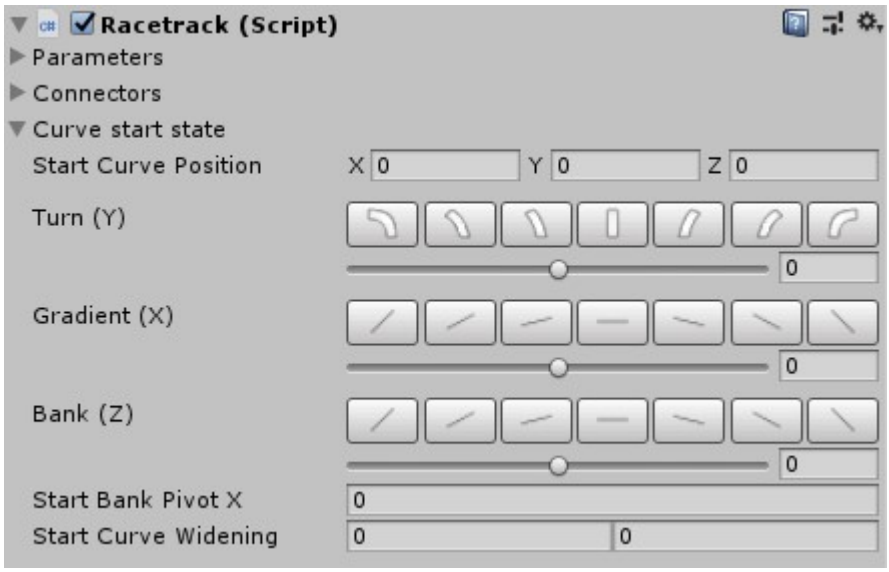
Setting the initial angles and widening

The curve angle and widening properties specify the values for the *end* of the curve, which then become the properties for the start of the *next* curve.

Sometimes you need to set the properties for the start of the *first* curve. This is useful for creating wide tracks, or banked circuits etc, where the initial curve does not start from a level and unwidened position.



To do this select the **Racetrack** object, and expand "Curve start state".



Set the gradient, bank, bank pivot and/or widening as required.

As with the curve controls, clicking the buttons will automatically update the meshes for the first curve. For other changes you will need to tell Racetrack Builder to update the meshes by selecting a curve and clicking the "Update track" button.

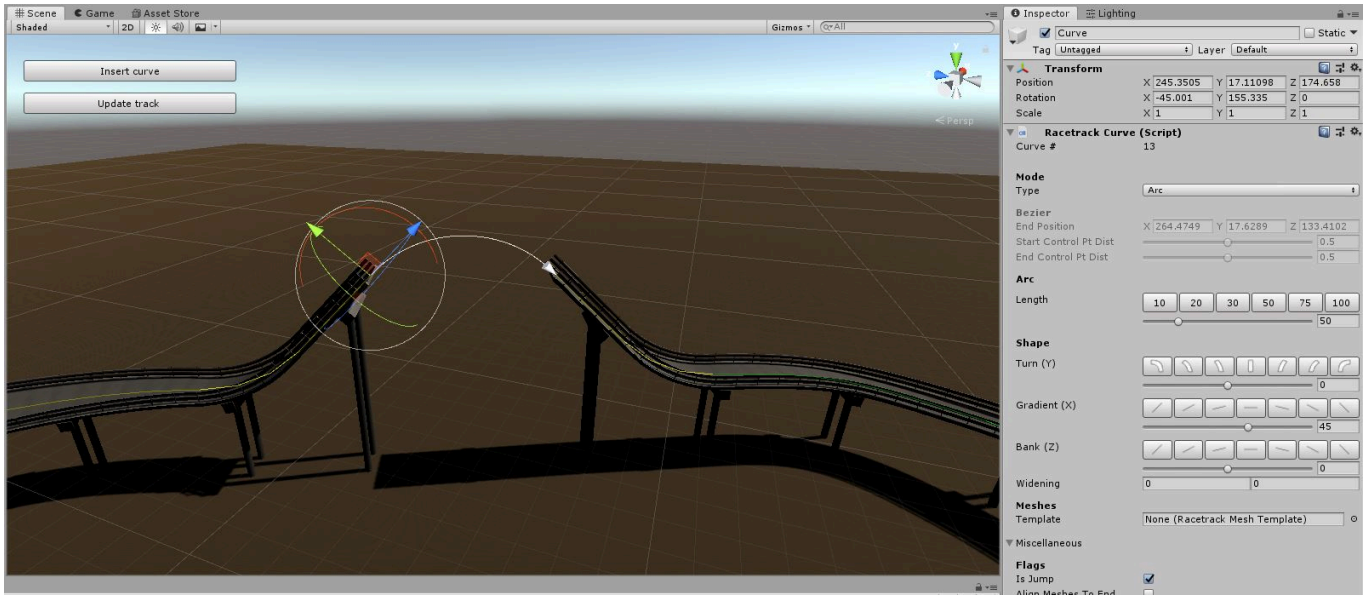
Note: If you have already created a closed circuit using the "Create closed circuit" command (described later on), you will notice the last curve does not automatically update to line up with the first curve. Simply click "Create closed circuit" again to fix this.

You can also set the start position and direction here. This has a similar effect to moving or rotating the racetrack object itself except that it doesn't move any bezier curve endpoints, and so can be used to move the start position without necessarily moving the entire racetrack. (The racetrack junction mechanism makes use of this when connecting the start of the **Racetrack** object to a junction.)

Creating jumps

To create a jump, select a curve, then in the inspector expand the "Miscellaneous" section and tick "Is Jump" property.

No meshes will be generated for the curve, resulting in a gap which the player must jump over.



Note: This action does not update the track automatically. Click "Update track" to apply the change.

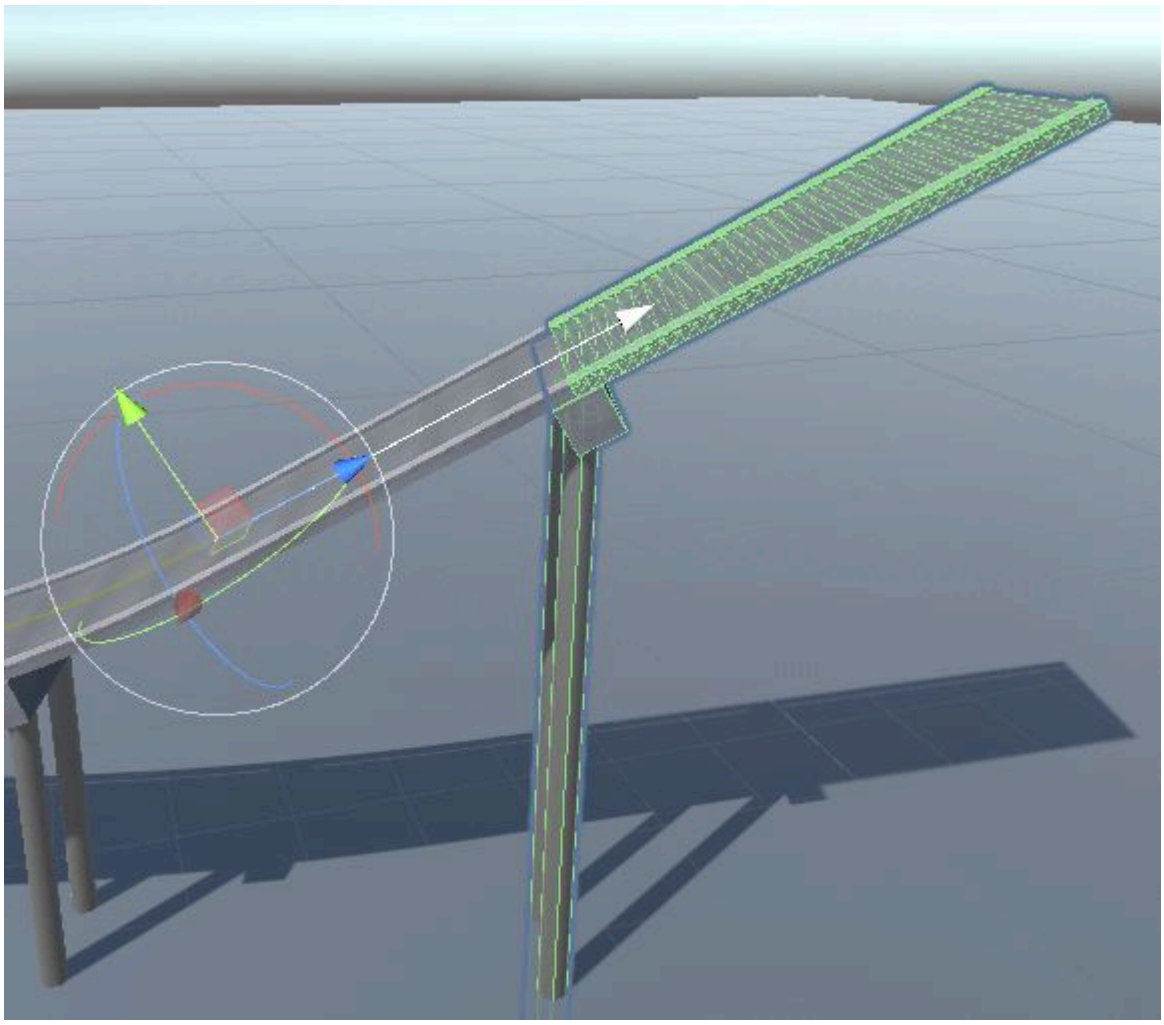
Aligning meshes to curves

Sometimes the mesh model will extend beyond the end of the curves.

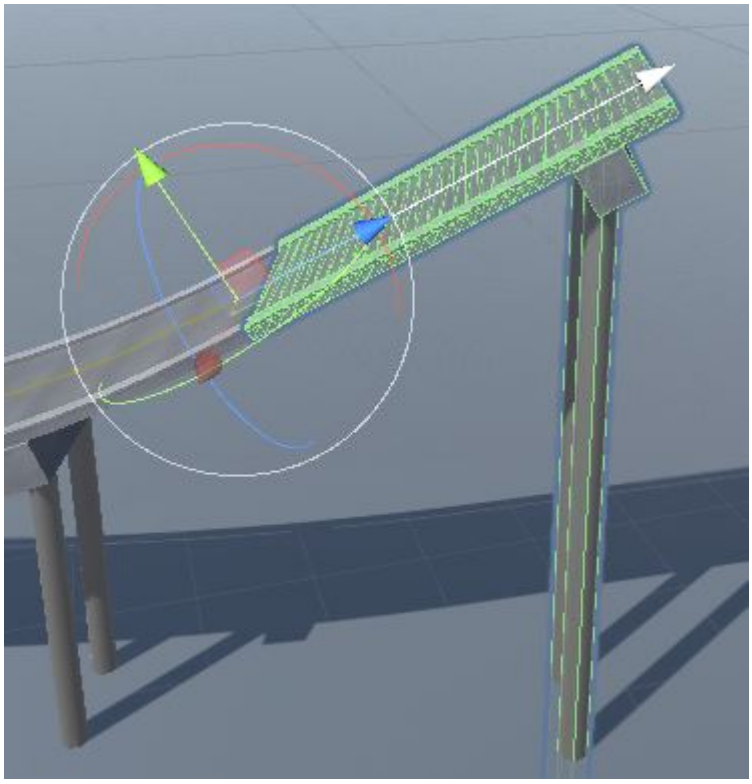
This can be evident:

- At the end of the racetrack
- At a jump
- When the track type changes

Racetrack Builder builds the mesh model by laying smaller mesh models end-to-end (the default track types have a 30 unit long mesh model). If the models don't fit exactly into the total length of the curves, there will be some overrun.



To avoid this, select the end curve, expand the "Miscellaneous" section in the property inspector and tick "Align Meshes To End".



This will adjust the meshes so that they stop exactly at the end of the curve.

Note: This action does not update the track automatically. Click "Update track" to apply the change.

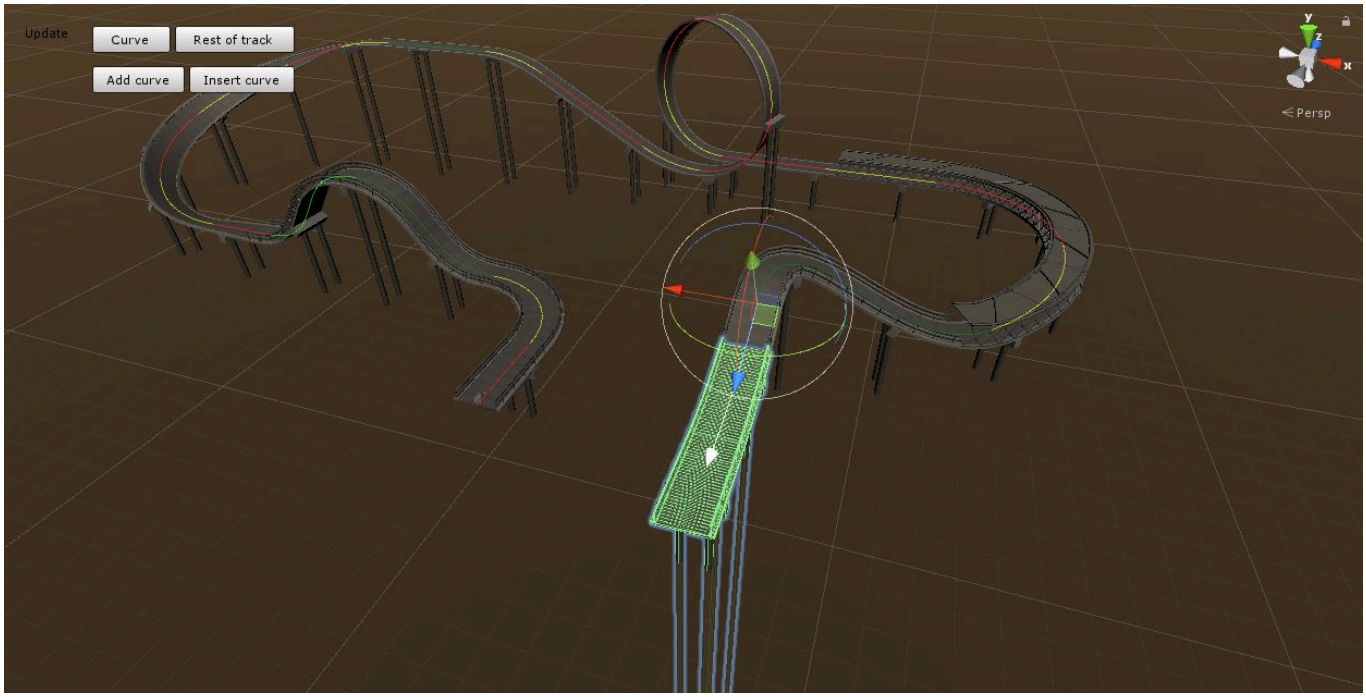
Racetrack Builder does this by scaling the meshes along the Z axis before they are warped along the curve path. The scaling is distributed back across the previous curves back to the start of the racetrack (or the last jump or align meshes point), to try to avoid scaling any single mesh by a large factor.

The scaling is only applied to the "continuous" meshes. Spaced content like poles will retain their original spacing. Likewise texture coordinates generated by the **Racetrack UV Generator** component will not be scaled.

Creating a closed circuit

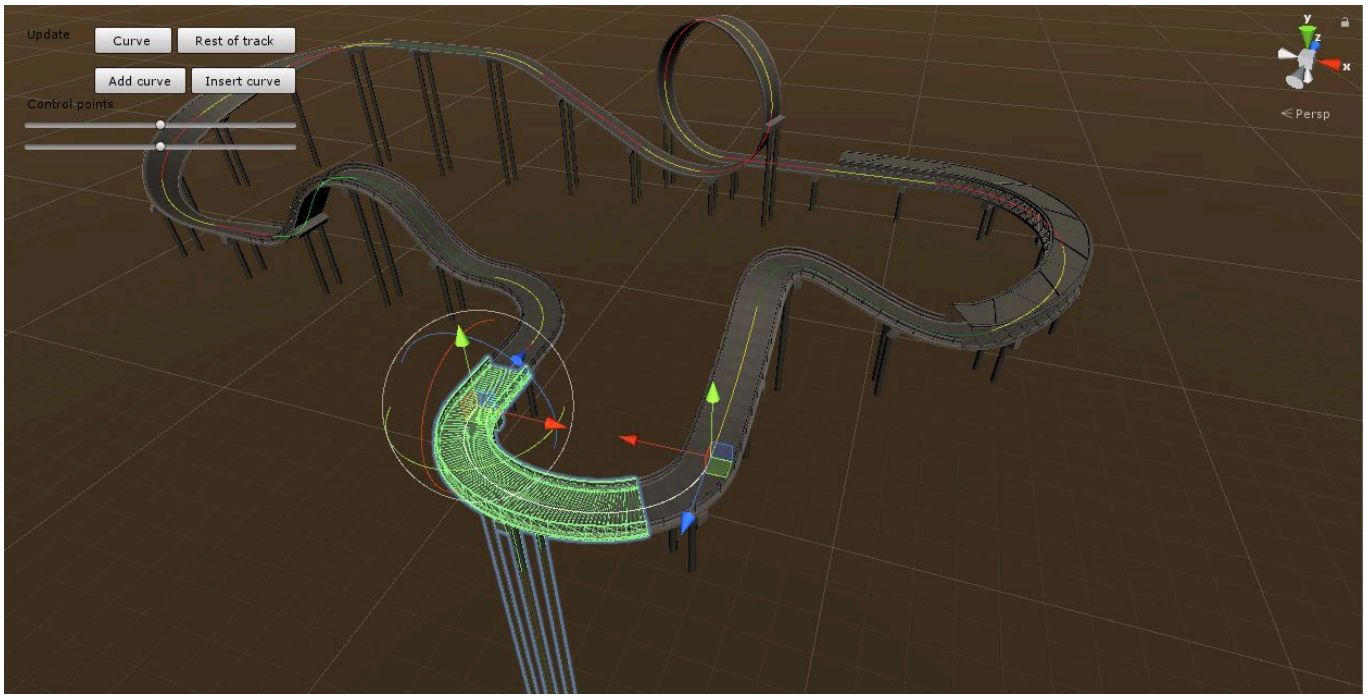
Racetrack Builder can create a seamless looped racetrack by adding a closing curve to the end of your racetrack.

For best results, start with an almost complete track, where a single curve would complete the loop. Because there is usually some overlap between the start and end meshes it can also be a good idea to set the last curve to the same track type.



Select the **Racetrack** object in the scene hierarchy then click "Create closed circuit" in the inspector.

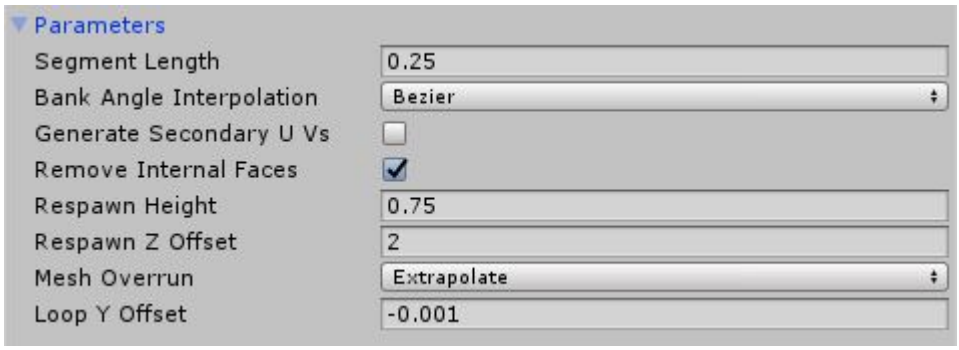
Racetrack Builder will create a single Bezier curve to close the loop.



(Bezier curves are described later in this document.)

You will likely notice that the last mesh of the track overlaps the first one. This happens when the meshes when laid end-to-end don't exactly match the length of the racetrack curves leaving a bit of overrun. Racetrack Builder attempts to hide the overrun by lining it up *almost* exactly with the start curve.

To avoid Z fighting Racetrack Builder offsets the looped meshes Y coordinates very slightly. This is controlled by the "Loop Y Offset" property on the **Racetrack** object "Parameters" section.



Alternatively you can select the last curve and tick "Align Meshes To End", as described earlier, to eliminate the overrun and have the end curve line up exactly to the start curve.

The looping behaviour is controlled by the "Mesh Overrun" property. The "Create closed circuit" function automatically sets this to "Loop" which enables the above behaviour. The other option is "Extrapolate" where the overrun part of the mesh simply carries on in a straight line in the direction the curve is facing.

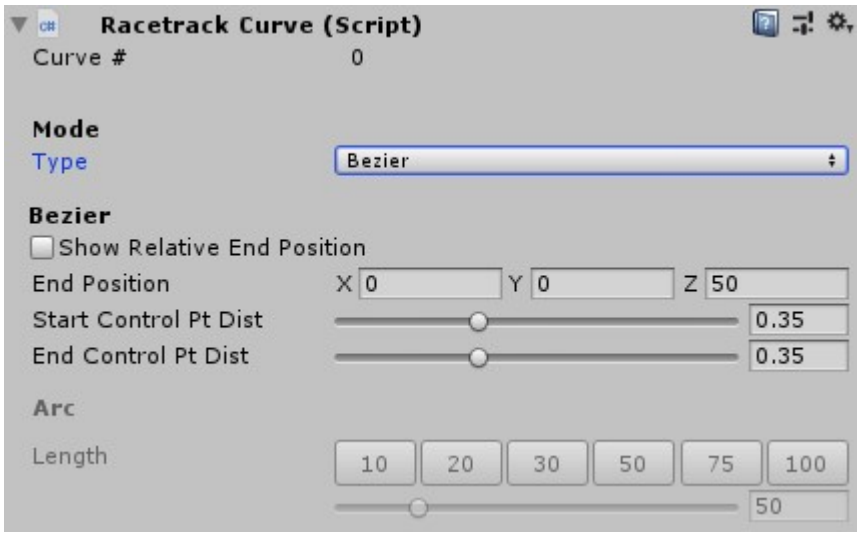
When "Mesh Overrun" is set to "Loop" the **Racetrack** and **Racetrack Curve** inspectors behave slightly differently:

- The "Add Curve" button does not display in the **Racetrack Curve** inspector. Only the "Insert Curve" button to insert a curve immediately after the current one (when applicable).
- Clicking "Create closed circuit" again in the **Racetrack** property inspector will realign the last curve of the racetrack, rather than adding a new curve.

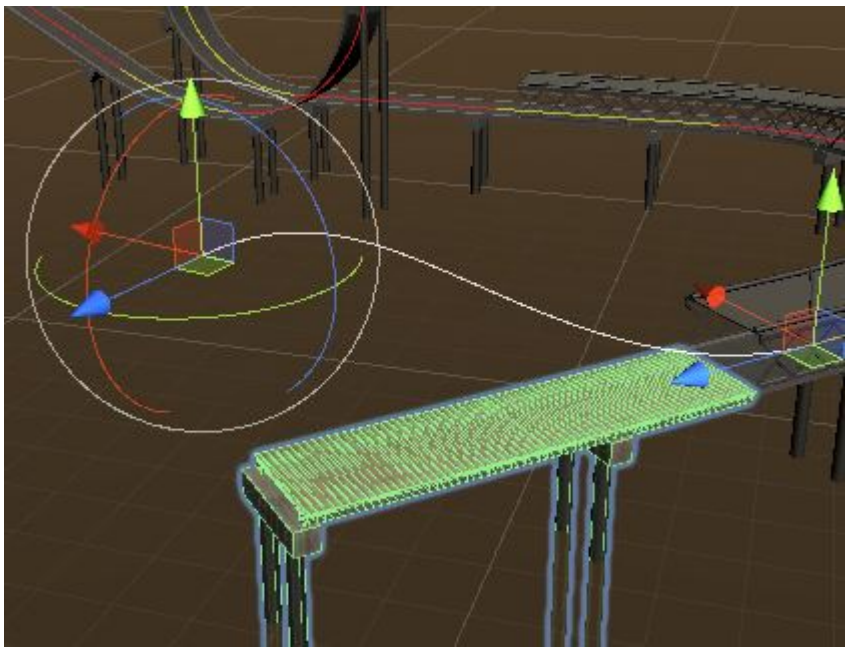
If you need to revert to the previous behaviour, set "Mesh Overrun" back to "Extrapolate".

Bezier curves

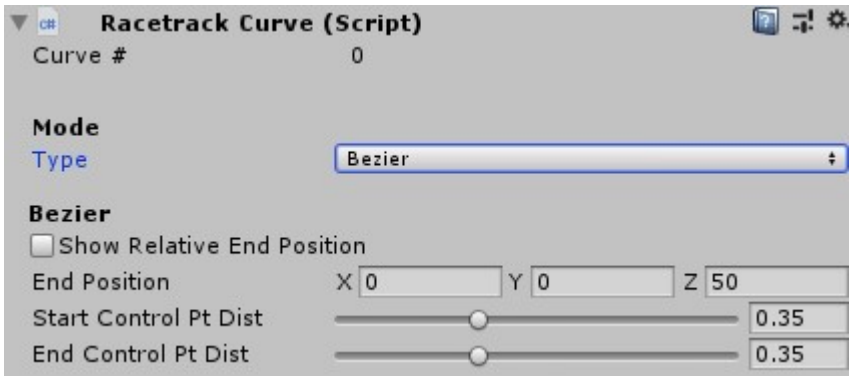
As well as the standard circle "arc" curves, you can also use Bezier curves. Select a curve and change its "Type" to "Bezier" in the drop down:



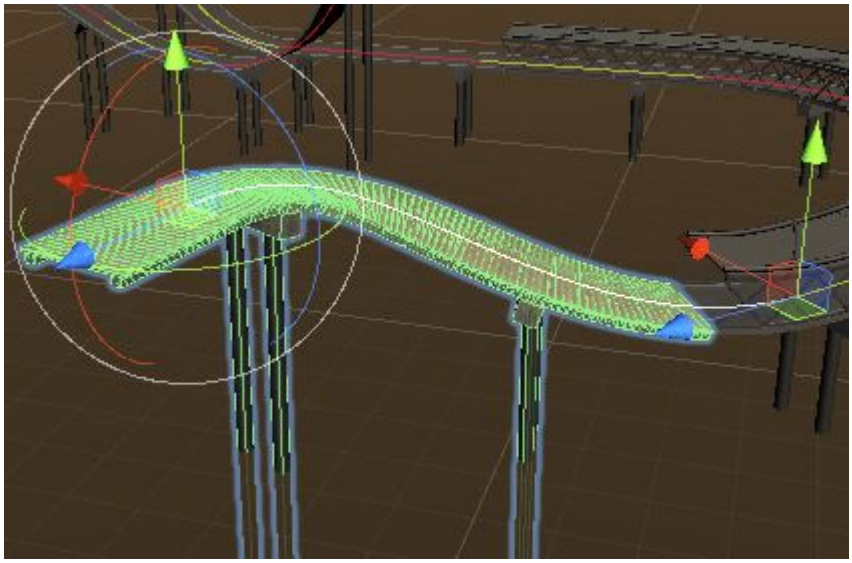
Drag and rotate handles will appear in the editor view to allow you to specify the position and direction of the end of the curve in world space. The curve will automatically update in response to your changes.



You can also control the curve shape by changing the "Start Control Pt Dist" and "End Control Pt Dist" in the curve properties. (Control point distances are proportional to straight-line distance between the curve start and end points.)



Click the "Update Track" button to recreate the road meshes around the updated curve.



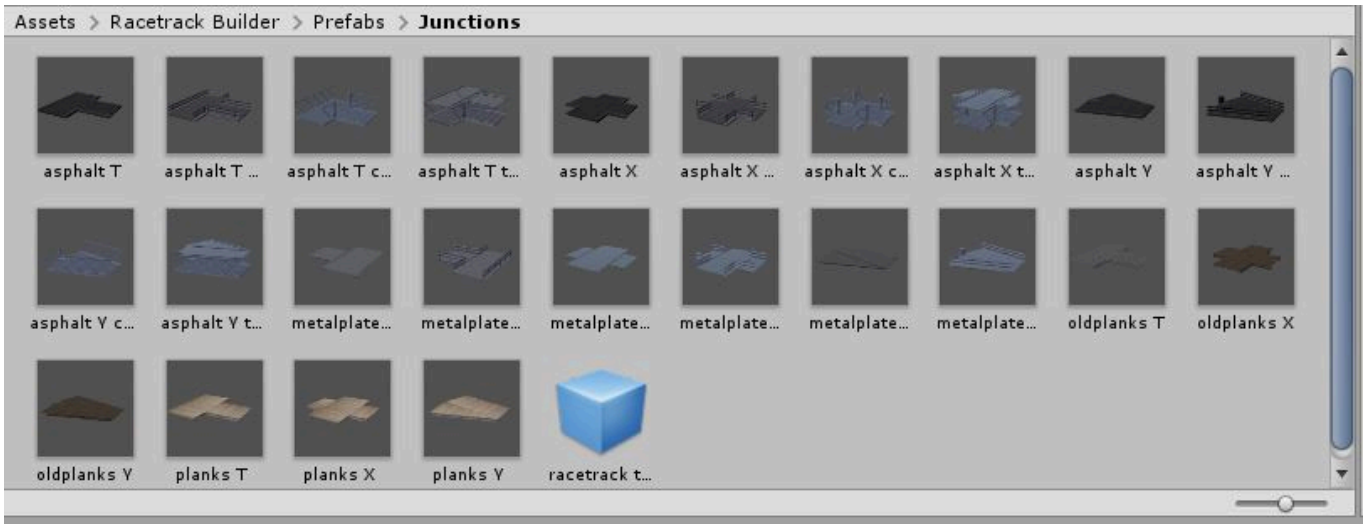
Bezier curves are useful when you need to control the exact position of the racetrack, e.g. when fitting around terrain or other game objects. They also have the useful property that the end point is fixed in world space regardless of where the start point is. You can use this to "lock down" parts of your racetrack by switching the curve immediately infront to a Bezier. They will then remain in the same place regardless of any changes to earlier curves in the racetrack.

Note: Bezier curves are based on the "cubic Bezier" equation, common in drawing and animation programs. The curve is defined by 4 points: 2 define the start and end of the curve - which in Racetrack Builder means the racetrack curve position and the end position that you specify. The other 2 points control the shape of the curve - in Racetrack Builder these lie on the tangent vectors of the start and end points, at the distance you specify.

Junctions

Junctions allow you to create branching racetracks with multiple paths. They are located in the folder:

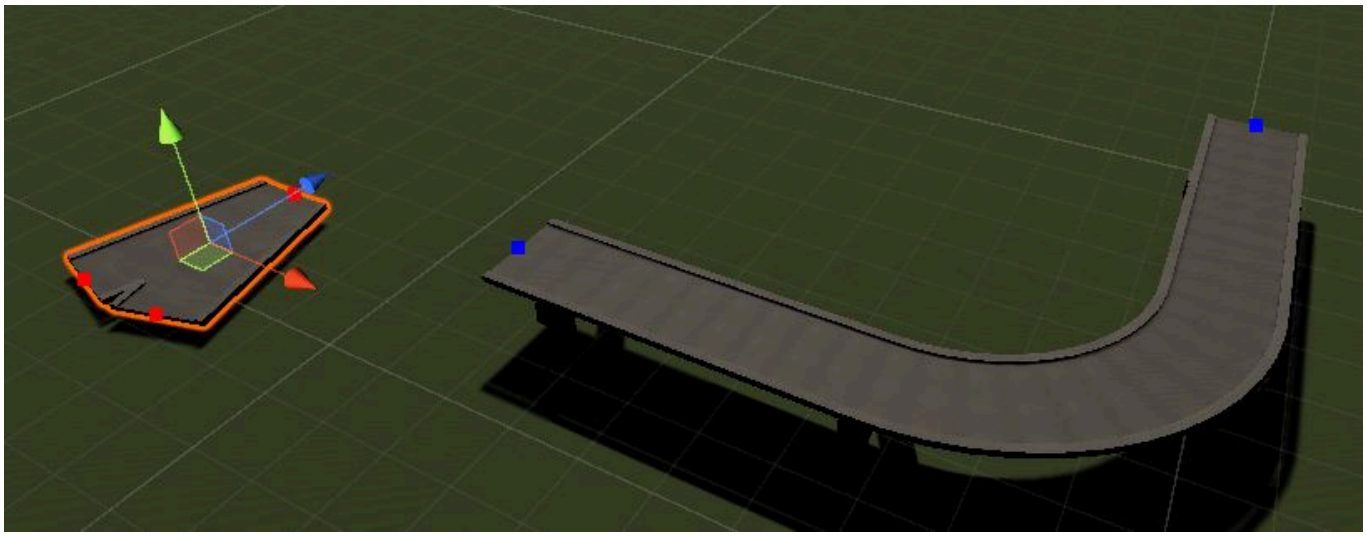
Racetrack Builder/Prefabs/Junctions



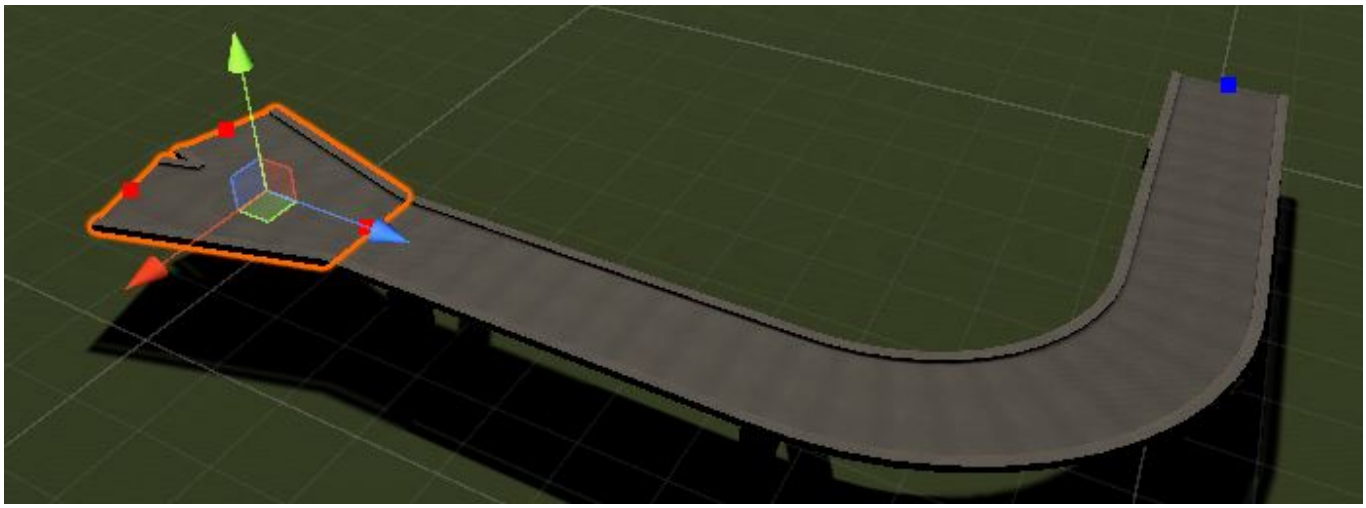
There are various permutations for the different track types and junction shapes. Select the one you wish to use and drag it into the scene.

Attaching junctions to racetracks

When the junction object is selected, its connection points will display as red squares. **Racetrack** objects will display blue squares at the start and end indicating where the junction can be attached to.

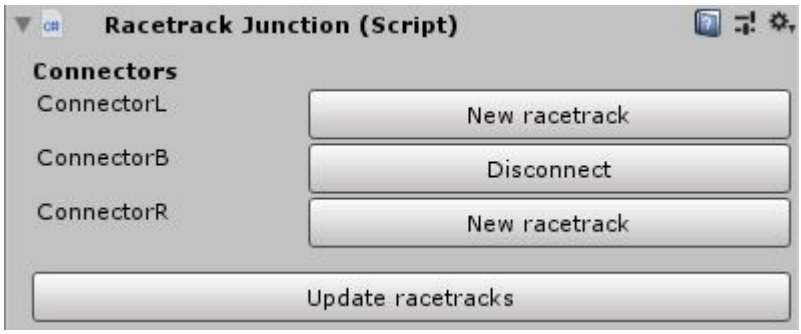


Attach the junction to the racetrack by dragging a red square to a blue square.



Junction properties

The junction is controlled by a **Racetrack Junction** component. In the inspector it displays a list of connection points, each with a button to either create and attach a "New racetrack" or "Disconnect" the existing racetrack.



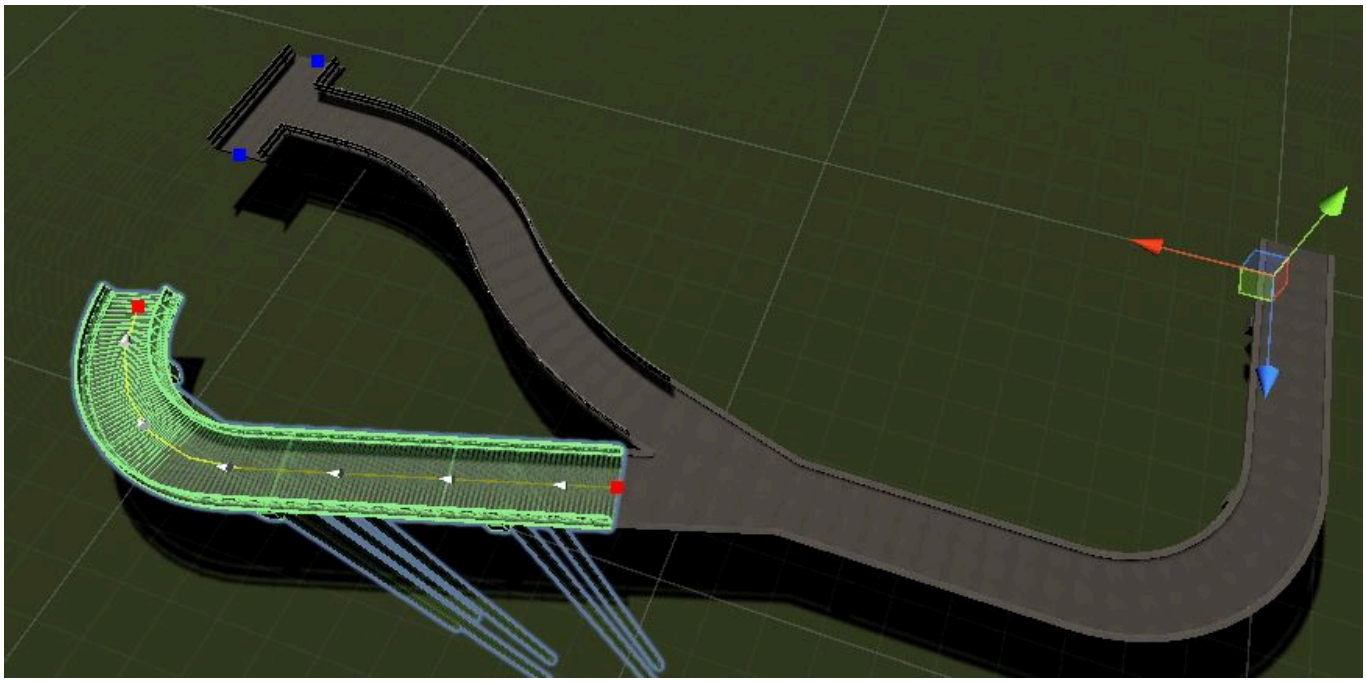
Clicking "New racetrack" creates a new racetrack and attaches it to the connection point. You can then add curves to build the racetrack in that direction.

Racetrack Builder remembers that the **Racetrack** objects are connected to the junction. After moving and/or rotating the junction object, click the "Update racetracks" button, and Racetrack Builder will update the **Racetrack** objects to connect seamlessly to the junction in its new orientation.

Connecting racetracks to junctions

Often it's necessary to connect a **Racetrack** object to a junction.

To do this, select the **Racetrack** object.

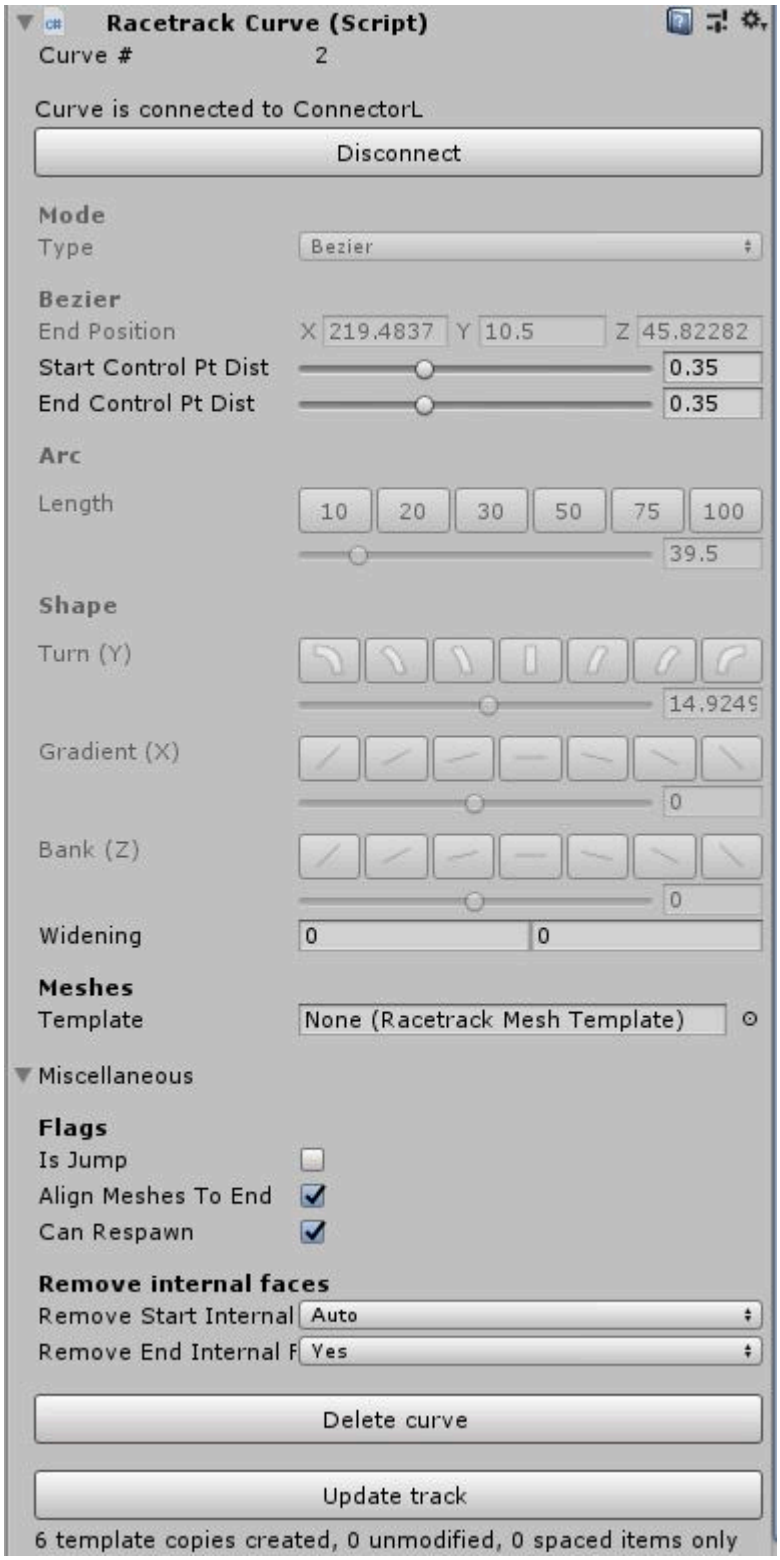


*Note: Clicking the racetrack in the scene view will select the clicked **Racetrack Curve** object, not the **Racetrack** object itself. Use the Hierarchy tree to find and select the main **Racetrack** object.*

When the **Racetrack** object is selected, its start and end connection points will display in red, and the available junction connection points will appear in blue. You can now connect the racetrack to the junction by dragging the red square to the appropriate blue square.

Connected curve properties

When you connect the end of a racetrack to a junction, Racetrack Builder will insert a new bezier curve at the end of the racetrack to join them up. If you select the curve, you can see in the inspector view that it is connected to a junction connector.



Most of the bezier property editors are disabled, indicating that Racetrack Builder is managing the curve to ensure it connects seamlessly to the junction.

Clicking "Disconnect" will disconnect the end of the racetrack from the junction, allowing you to manipulate it manually again.

Expanding the "Miscellaneous" section reveals that "Align Meshes To End" has been ticked, to ensure that the racetrack curve stops exactly at the connection point.

Connected racetrack properties

Similarly, when you connect the start of a racetrack to a junction, Racetrack Builder will manage the start position and orientation of the racetrack to ensure it connects seamlessly. You can see this in the inspector by selecting the **Racetrack** object and expanding the "Curve start state" section.

Once again most of the property editors are disabled.

Creating circuits

Branching "racetracks" are made from multiple "Racetrack" objects. So you cannot create a closed circuit the usual way by joining the end of a **Racetrack** object to its start - instead you need to join the end of the last racetrack to the start of the first one. But the junction system only supports joining junctions to racetracks or vice versa, so you cannot join two racetracks directly.

The solution is a special junction prefab called "racetrack to racetrack". This odd junction object has no meshes, and just contains two connectors on top of each other.

You use it by dragging it into the scene, then dragging one of the connectors to the start of the first racetrack so that it snaps to the racetrack's start position. Now you can select the last racetrack and drag its red endpoint to the blue junction connector to connect it up and complete the circuit.

Splitting racetracks

You can split a racetrack object in two by selecting a curve, expanding the "Miscellaneous" section in the inspector and clicking "Split track after curve". This splits the racetrack into two separate **Racetrack** objects.

The split operation is particularly useful for inserting junctions midway through existing racetracks. The process typically looks like:

1. Split the racetrack immediately before the desired junction position
2. Adjust/delete the last curve of the first racetrack to make room for a junction
3. Create a junction and attach to the start of the second racetrack
4. Join the first racetrack to the junction

Respawning

Note: Previously the car tracking component was called 'Racetrack Progress Tracker'. This has been deprecated in favor of 'Racetrack Car Tracker', due to subtle bugs. If you have an existing project, consider switching to 'Racetrack Car Tracker', which should be a drop-in replacement.

The **Racetrack Car Tracker** component can be added to the player car to automatically respawn them back on the track after they fall off. By default the player is respawned on the last curve that they drove on.

Sometimes a curve is not suitable as a respawn point, e.g. if the player cannot get enough speed to complete the next jump, and therefore cannot progress.

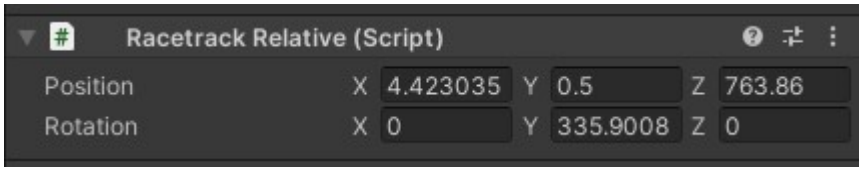
To avoid this, untick the "Can Respawn" property of the curve. The respawn logic will search backwards from the last curve the player drove on until it finds a curve where "Can Respawn" is ticked.

Placing objects on the track

The **Racetrack Relative** component can be attached to an object in order to position it relative to the racetrack. Position the object underneath a **Racetrack** or **Racetrack Curve** object *in the Unity scene hierarchy*. This tells Racetrack Builder that the object is to be positioned relative to that **Racetrack** or **Racetrack Curve**, and to automatically update the object's position whenever the racetrack changes to maintain the same relative position.

*Important: Don't place the object underneath the **Racetrack Template Copy** objects that each curve generates, as these are regularly deleted and recreated, meaning your object will be deleted too!*

Set the Position and Rotation properties to position the object relative to the racetrack or curve.



The Z position is the distance along the racetrack/curve. The X and Y position is relative to the racetrack cross section at that distance.

The rotation is relative to the racetrack direction at that point. So the local Z axis points in the racetrack direction, the Y axis points away from the surface of the racetrack.

Tip: Placing an object relative to a curve ensures it stays on that curve, even if you change the length of the curves before it.

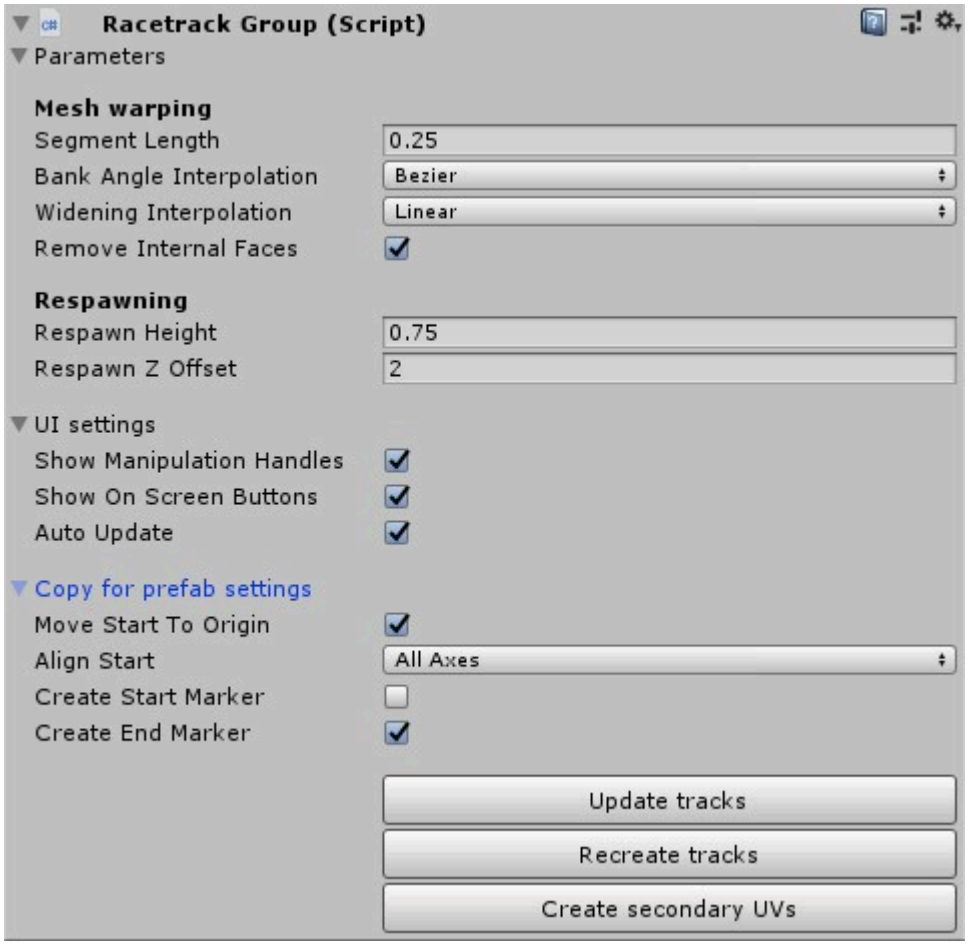
Racetrack Groups

The **Racetrack Group** component is an optional component that can be attached to a parent object for **Racetrack** and **Racetrack Junction** objects in the Unity scene hierarchy.

It provides a single location to store common settings that apply to all **Racetrack** objects, as well as the ability to update or recreate meshes for all **Racetrack** objects underneath it in the hierarchy. (This can be particularly useful when using junctions.)

You can create a **Racetrack Group** object from the main menu using: GameObject > 3D Object > Racetrack Group Or you can simply create an empty unity object, add the **Racetrack Group** component to it and move the **Racetrack** objects underneath it manually.

Settings



The "Mesh warping", "Respawning" and "UI settings" sections contain the same settings as exist on **Racetrack** objects, and override the settings on any descendent **Racetrack** objects in the hierarchy.

These settings are also *hidden* in the **Racetrack** component's property inspector if a parent **Racetrack Group** is present to avoid ambiguity.

"Copy for prefab settings" affect how prefabs of racetrack sections are created. See the Racetrack Prefabs section for more information.

Actions

The 3 buttons trigger actions that apply to all descendent **Racetrack** objects in the scene hierarchy.

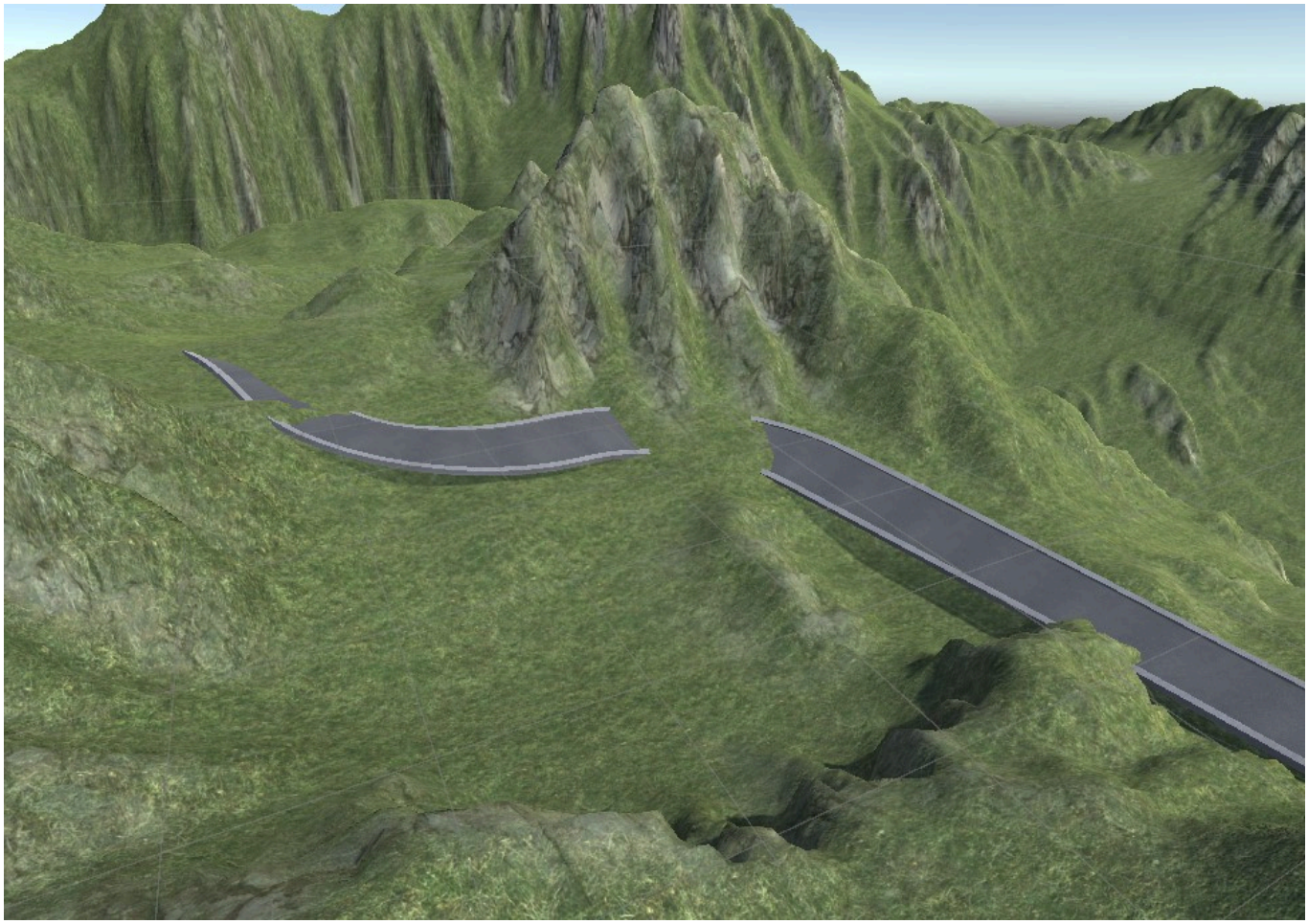
- "Update tracks" is equivalent to clicking "Update track" on each individual **Racetrack** object.

- "Recreate tracks" is equivalent to clicking "Recreate track" on each individual **Racetrack** object.
- "Create secondary UVs" is equivalent to clicking "Create secondary UVs" on each individula **Racetrack** object.

Working with terrains

As well as elevated racetracks, Racetrack Builder can be used to create regular "on the ground" racetracks, by combining it with a Unity **Terrain** object.

Attempting to fit a terrain and a racetrack together without gaps or overlaps is not an easy task however.



Therefore Racetrack Builder provides a component to help with this, the **Racetrack Terrain Modifier**.

Racetrack Terrain Modifier

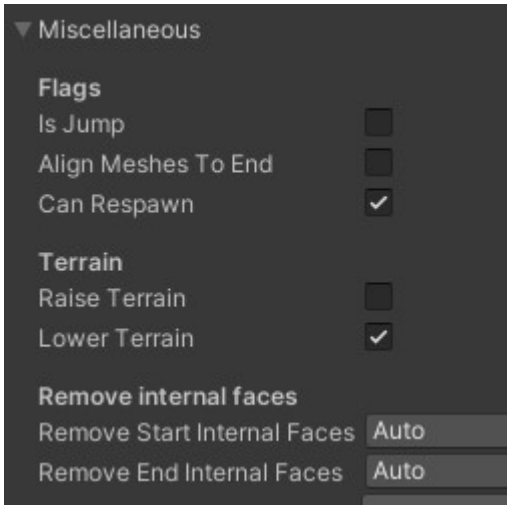
The **Racetrack Terrain Modifier** component is added to a Unity Terrain object.

It's main function is a large "Update Terrain" button. Clicking this button modifies the terrain heightmap to fit around the racetrack(s) in the scene.

The default behaviour is to move the terrain down - when necessary - so that the racetrack remains above it.

Racetrack Curve terrain properties

You can change the behaviour per **Racetrack Curve** by expanding the "Miscellaneous" section and editing the "Terrain" section.

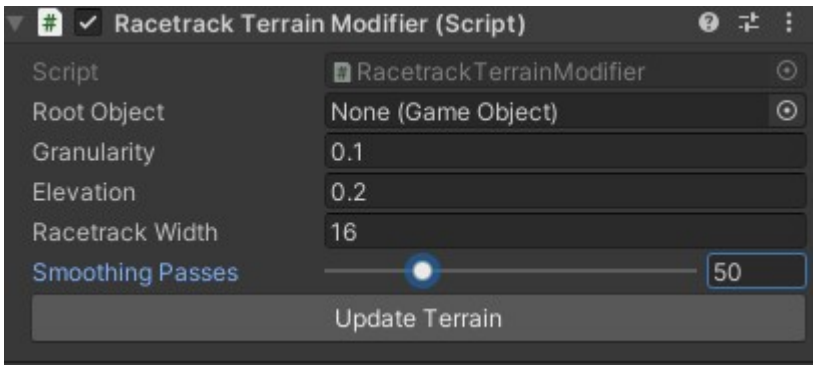


"Lower Terrain" lowers the terrain - when necessary - to ensure the racetrack will not be underneath it. Typically this will always be ticked unless you really do want the racetrack to penetrate underneath the terrain.

"Raise Terrain" will raise the terrain up - when necessary - to eliminate any gap between the terrain and the racetrack. This is useful when you want the racetrack to lie flat on the ground.

Racetrack Terrain Modifier properties

The **Racetrack Terrain Modifier** is controlled by the following properties:



"Root Object" can be set to a Unity object to control which Racetracks are considered when adapting the terrain. Racetracks that are not underneath the root object in the scene hierarchy will be ignored when modifying the terrain.

If the Root Object is unset, all Racetracks in the scene are considered.

"Granularity" controls the distance between height samples taken along the racetrack.

The terrain modifier works by sampling points along the racetrack and determining their height in the terrain, so that the raise and lower constraints can be applied.

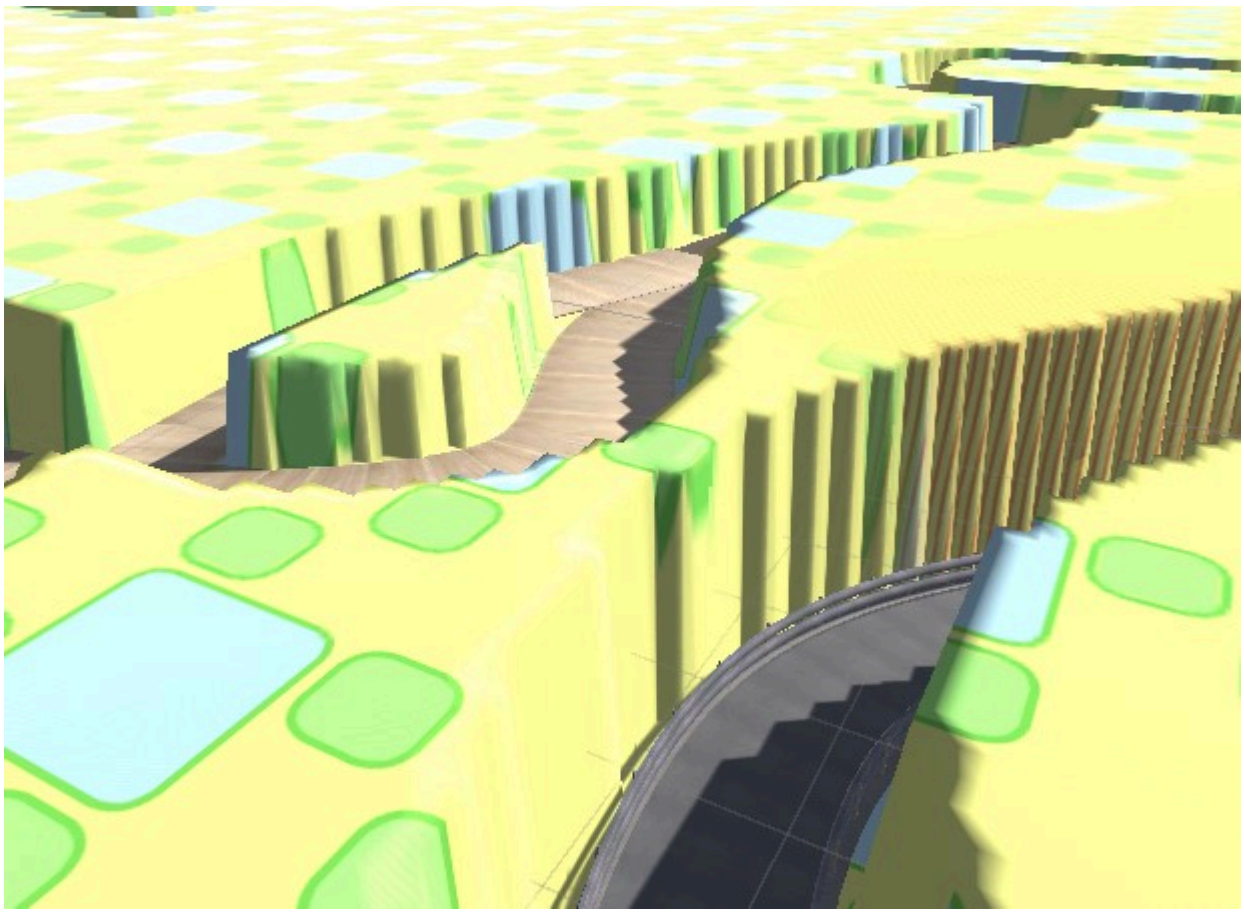
If you have a high resolution terrain and some heightmap vertices do not appear to be modified when they should, you may need to lower the granularity. However this results in more samples during the terrain modification process which can make it take longer, so be careful.

"Elevation" sets the desired elevation of the racetrack driving surface above the terrain, in world space units. Setting it to 0 would result in the racetrack surface lying flat on the terrain (generally not recommended due to Z-fighting).

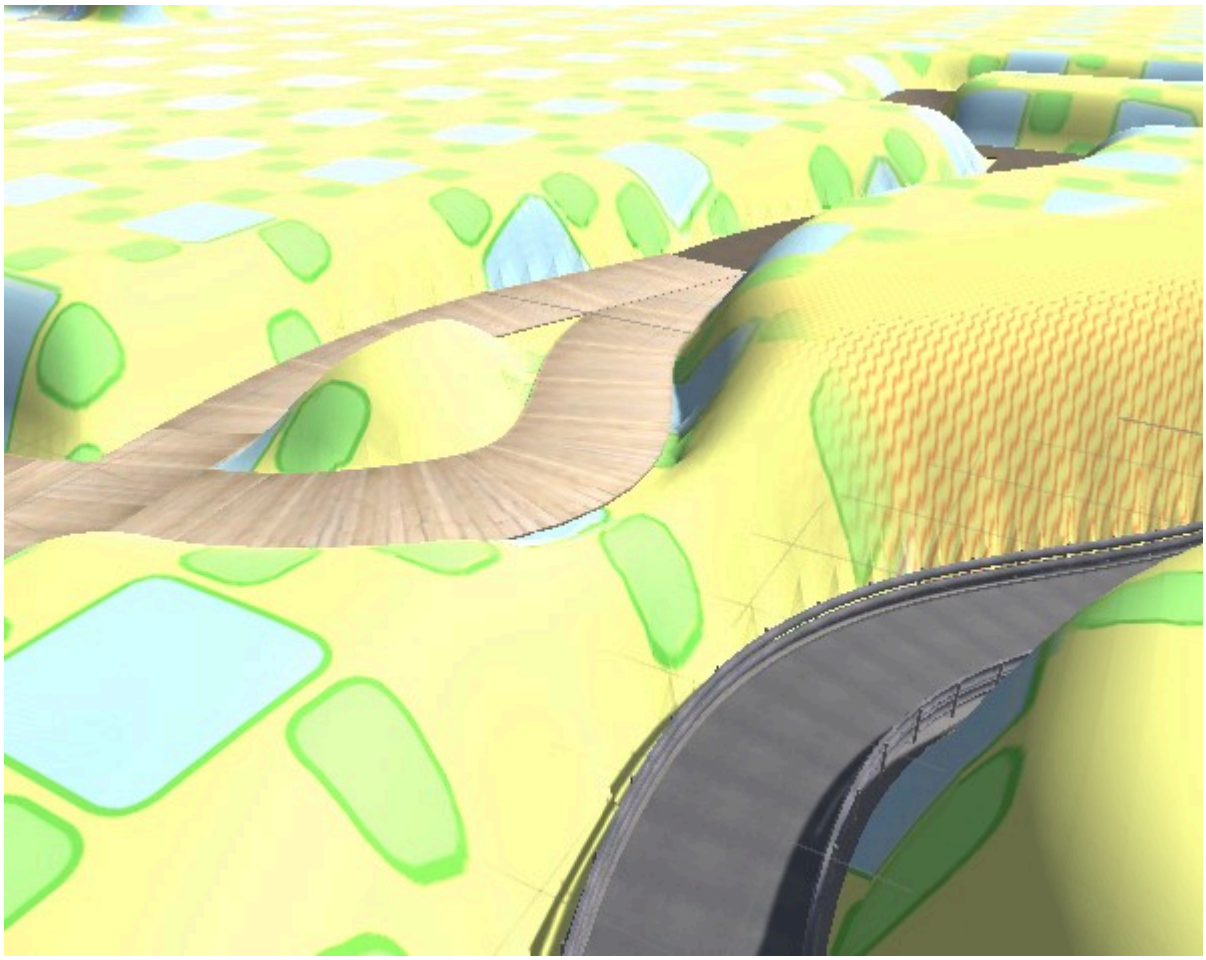
"Racetrack Width" tells the terrain modifier how wide to treat the racetrack. This value is combined with the curve "widening" settings to calculate the final width.

"Smoothing Passes" is number of smoothing passes to apply to the height map deltas, after the initial adjustments have been made.

Without any smoothing harsh jagged edges are likely to be introduced, particularly if large height adjustements are made to the terrain.



More smoothing passes create softer edges (at the expense of longer processing time).



Note: The smoothing is applied to the *height adjustments*, and only in areas where adjustments need to be made. This preserves map details, unlike smoothing the map using the Unity Terrain tools, which is a different operation.

Junctions

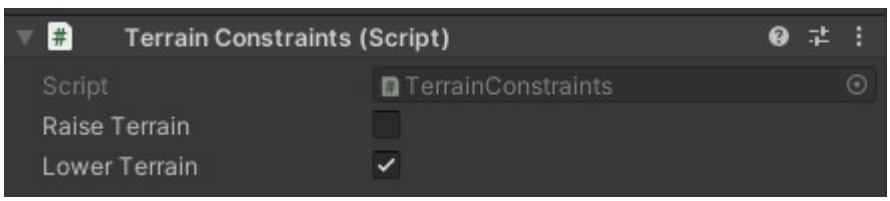
Racetrack Builder has two more components that are used to fit the terrain around junction objects.

*These components are mostly only relevant if when building/modifying **Racetrack Junction** objects. Otherwise the main point is to look for the **Terrain Constraints** component on a junction to find its "Raise Terrain" and "Lower Terrain" settings.*

Terrain Constraints

The **Terrain Constraints** component is attached to the junction object. It flags it to the **Racetrack Terrain Modifier** as an object to fit the terrain around.

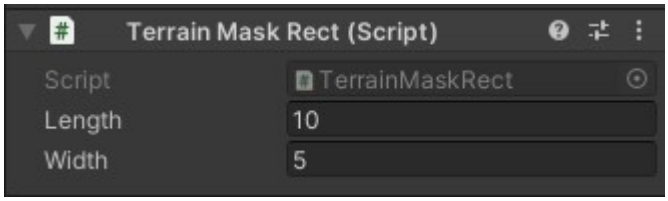
It also has "Raise Terrain" and "Lower Terrain" properties, to configure how the terrain fits around the junction.



These behave the same as for **Racetrack Curve**s.

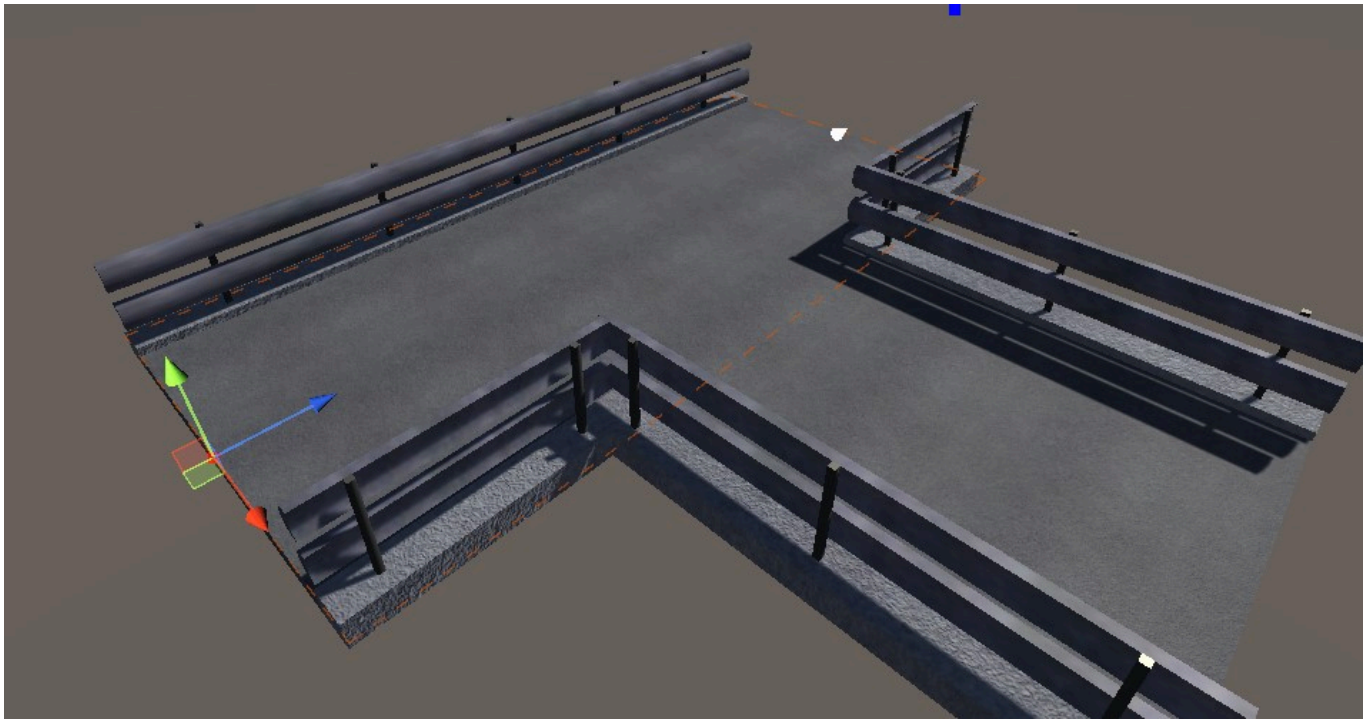
Terrain Mask Rect

The **Terrain Mask Rect** component is used to describe the shape of the junction to the **Racetrack Terrain Modifier**. Each one defines a rectangular region, and is typically attached to a **Racetrack Connector** object.



The base of the rectangle is centered at the local origin, and the length extends along the local Z axis.

The rectangle is displayed in the scene view as a dotted orange line when the corresponding component is selected.



Two or more rectangles are often needed to correctly describe the shape of a junction.

Re-using meshes

By default Racetrack Builder does not try to re-use each mesh that it generates when building the racetrack mesh model - every mesh is a full clone of all the mesh vertices, stored in the scene itself.

Alternatively, Racetrack Builder can be configured to share duplicate meshes when it detects them, as well as saving them as project assets so they can be shared between scenes. This can be particularly useful for mobile platforms, as it can save a large amount of storage space and GPU RAM.

Be aware of the following compromises however:

- Reusing meshes interferes with the RacetrackUVGenerator texture-coordinate generation, resulting in texture seams.
- To gain any meaningful benefit from mesh re-use the racetrack must contain a decent number of identical meshes. In practice this means restricting the different types of curves used.

Racetrack Mesh Manager

To enable mesh re-use, add a **Racetrack Mesh Manager** component to an object in your scene.

You can add it to any object, but giving it its own object is recommended, as it means you can easily turn it into a prefab to share between scenes, once it has been configured.

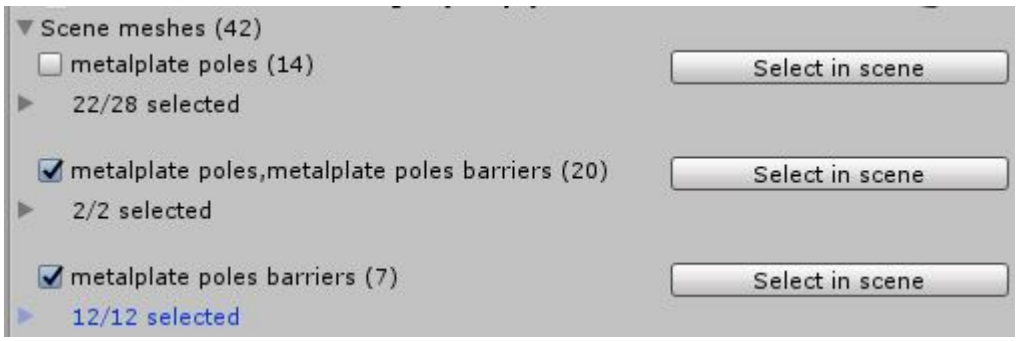
Racetrack builder will automatically detect the mesh manager and start sharing duplicate meshes as you build your racetrack. If you have already built a track, you can recreate the meshes (select the main **Racetrack** object and click "Recreate track" in the inspector) in order to update the meshes for the existing track.

Scene meshes

Select the **Racetrack Mesh Manager** to review the meshes generated and stored in the current scene.

These are shown under "Scene meshes" (the count is the total number of meshes).

Expanding "Scene meshes" displays the meshes grouped (loosely) by the track type that generated them. In this case the number in brackets is the maximum mesh re-use count.



You can also expand a group to view the meshes themselves, and their usage counts.

Click "Select in scene" to see the corresponding meshes in the scene.

Maximising mesh reuse

Racetrack builder will automatically re-use meshes if it detects they are identical, but it is up to you to generate identical meshes in the first place.

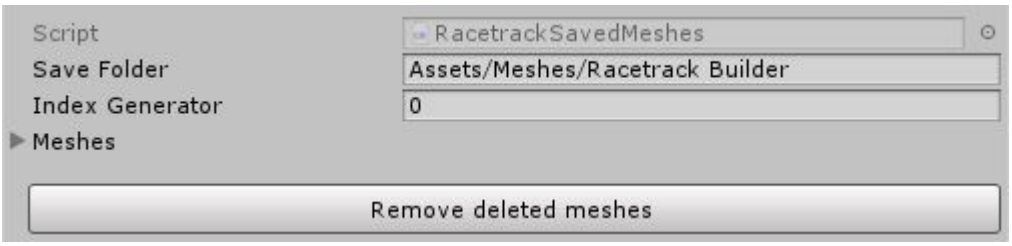
Here are some tips:

- Use "Arc" curves. Avoid bezier curves unless absolutely necessary (e.g. creating a closed circuit).
- Set the curve length to a multiple of 30. This is the length of the racetrack mesh templates that are distributed with Racetrack builder. (Unless you have created your own mesh templates, in which case use a multiple of their length).
- Decide on a set of curve shapes to use, e.g. straight, 90 degree turns, 45 degree slopes etc, and try to stick to them.
- Avoid using a lot of different track types.
- If you use a bezier curve, set "Align Meshes To End" on the bezier curve, *and the previous* curve, to ensure the arc curves before and after the bezier are not stretched/squashed.

Racetrack Saved Meshes

In order to save generated meshes as assets in your project, you must create a **Racetrack Saved Meshes** object. This is a Scriptable Object that will reside in a project folder rather than in the scene.

To create it, right click on a folder in your project and select "Create > Scriptable Objects > Racetrack Saved Meshes".



This object stores information about saved mesh assets, so that Racetrack builder knows how they were generated and when they can be re-used.

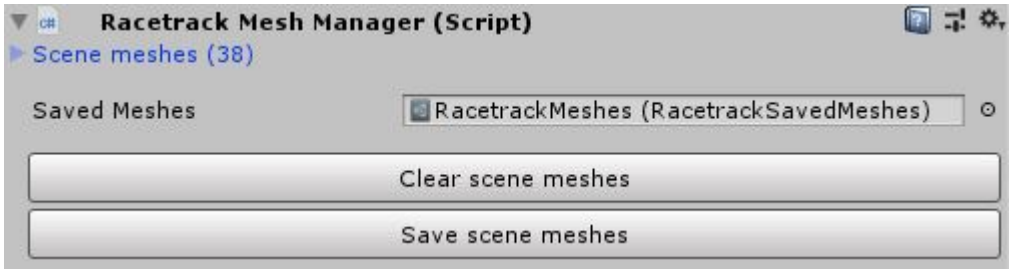
The "Save Folder" is the path to the folder in which the meshes will be saved. You can change it to another folder if you wish, but be aware that the top most folder must always be "Assets". If the folder does not exist Racetrack Builder will automatically create it (and any missing parent folders) before saving any generated meshes.

Racetrack Builder automatically manages the "Index Generator" and "Meshes" properties, and I don't recommend changing them manually. The "Remove deleted meshes" button will be discussed below.

Saving meshes

Once you have created a **Racetrack Saved Meshes** object, select the **Racetrack Mesh Manager** object in your scene, and drag the new **Racetrack Saved Meshes** object to its "Saved Meshes" property in the inspector.

You should see a "Save scene meshes" button appear.



Tick the meshes you wish to convert into project assets in the "Scene meshes" list. Racetrack builder automatically ticks any "Arc" curves that have not been stretched/squeezed as a basic rule of thumb, but you can override it if you need to.

Then click "Save scene meshes" to save them as mesh assets in your project.

The meshes will be removed from the "Scene meshes" list in the **Racetrack Saved Meshes** object. You should be able to see them in your project folder.

The mesh references in the scene will automatically be updated to reference the saved meshes. The saved meshes will automatically be used for any new racetrack curves you build if Racetrack Builder detects that they match.

Sharing meshes across scenes

To use the saved meshes in a different scene you will need to create a **Racetrack Mesh Manager** in that scene and set the "Saved Meshes" property to the **Racetrack Saved Meshes** object in your project.

Racetrack Builder should automatically detect and use the saved meshes where appropriate.

*To save time you may wish to create a prefab of the **Racetrack Mesh Manager** with the **Racetrack Saved Meshes** object already populated.*

Deleting saved meshes

If for any reason you decide you don't want a saved mesh asset, you can delete it from your project folder.

You will need to find any scenes containing racetracks that referenced the mesh and recreate their meshes (select the **Racetrack** object and click "Recreate track" in the inspector).

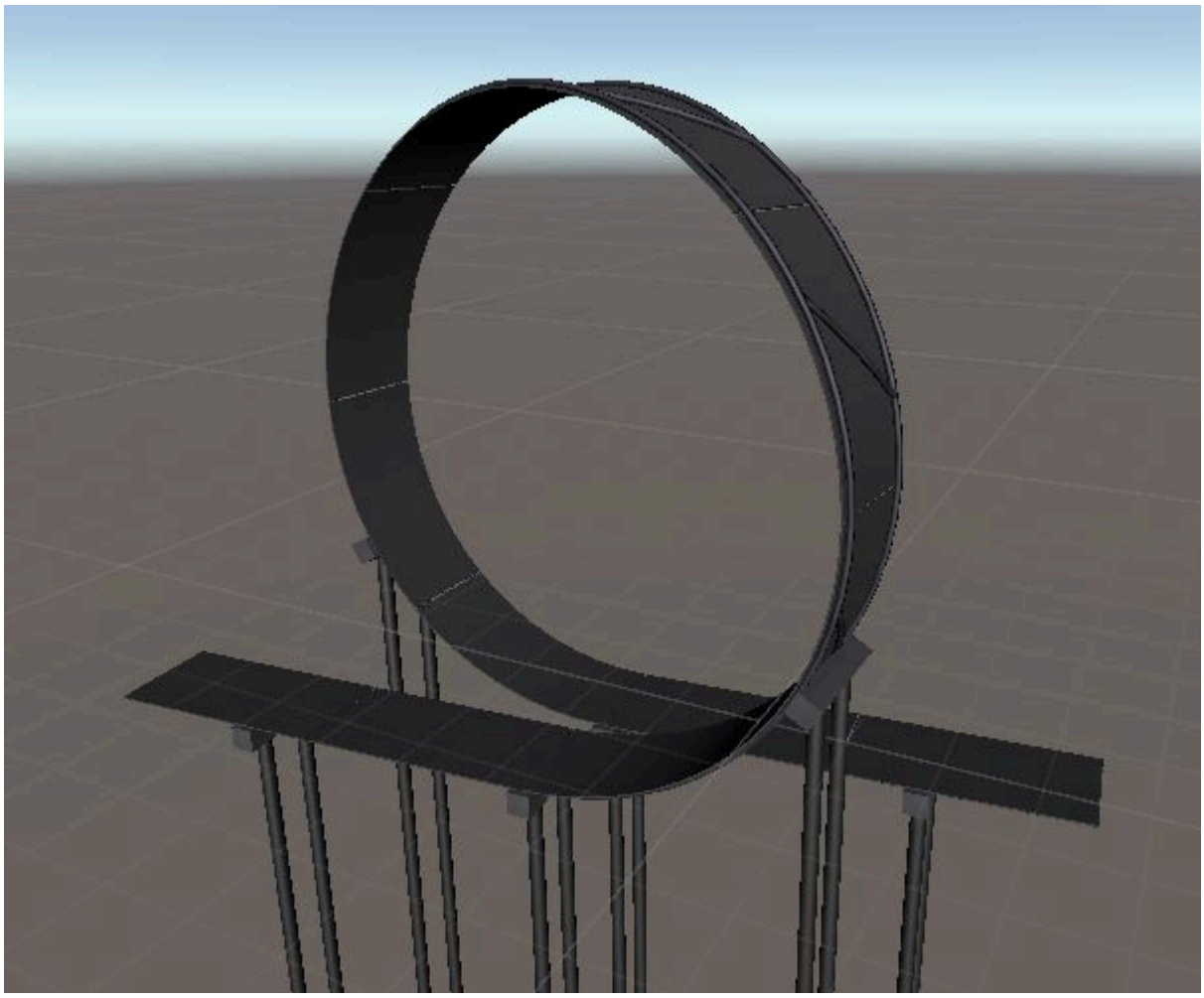
Racetrack Builder will detect that the mesh is missing and use a scene mesh instead.

You can also remove the mesh reference from the **Racetrack Saved Meshes** object by selecting it and clicking "Remove deleted meshes" in the inspector. This is not strictly necessary but keeps things tidy.

Racetrack prefabs

You can use Racetrack Builder to create prefabs for curves or sections of racetrack etc that can be exported and imported into other projects (including projects that don't use Racetrack Builder).

You must ensure *all meshes have been saved as project assets* for any part of a racetrack you wish to convert into a prefab.



The easiest way to turn a section of racetrack into a prefab is:

1. Select the curve or curves to convert.
2. In the inspector, find the **Racetrack Curve** component, click the drop down menu icon and select "Create copy for prefab".

Racetrack Builder will create a new object named "Racetrack section" containing a copy of the selected curves. You can now drag the object into a project folder to create a prefab.

If Racetrack Builder does not create a new object, check the console for an error like "Mesh 'xxx' has not been saved as a project asset". Racetrack Builder will automatically select the offending object in the object hierarchy. You will need to rectify this and try again.

Racetrack Builder strips all Racetrack Builder specific components from the prefab copy.

Copy for prefab settings

In addition it performs the following modifications. These can be configured by creating a parent **Racetrack Group** object in the "Copy for prefab settings" section:

"Move Start to Origin" determines whether the copy is moved so that the start point aligns with the origin:

- Ticked (default) - Start point will be aligned to the world space origin (0,0,0).
- Unticked - Start will remain at the same point in world space.

"Align Start" determines how the copy will be oriented in world space:

- All Axes - The copy will be oriented so that the start of the track aligns with the world space Z axis, and the track surface normal aligns with the Y axis.
- Y Axis (default) - The copy will be rotated around the Y axis only, so that the start of the track aligns with the world space Z axis *when viewed from above*. *Note: This is usually the best option if you are using support poles, as it ensures they remain vertical in the copy.*
- No - The copy will not be rotated at all. It will retain the same orientation as the source meshes.

Start and end marker objects can be added to help script code join prefabs together seamlessly.

"Create Start Marker" specifies whether to create a "start marker" object:

- Ticked - An empty Unity object named "Start marker" will be created and oriented to the section start position and direction.
- Unticked (default) - No start marker object will be created.

"Create End Marker" specifies whether to create an "end marker" object:

- Ticked (default) - An empty Unity object named "End marker" will be created and oriented to the section end position and direction.
- Unticked - No end marker object will be created.

Creating mesh templates

A "mesh template" provides the meshes that are generated and fitted to the path of the curves, in order to create the racetrack.

They are a little bit like Unity prefabs, and are often stored as one. But they are instantiated a little differently, mainly due to the need to warp the meshes along the curves and clone the repeating parts (like support poles).

This package contains a set of mesh templates to get you started, but you can easily build your own to create your own road types.

Supplied templates

The sample mesh templates are in: Assets/Racetrack builder/Prefabs/Track Templates

There are also prefabs that can be composed together to create mesh templates in: Assets/Racetrack builder/Prefabs/Template parts

These are grouped into folders:

- Driving surface - The main part the player drives on
- Sides - Barriers etc that attach to the side of the track
- Supports - Support poles that hold up the track

Example walkthrough

As an example, we can can create a mesh template consisting of wooden planks, support poles and side barriers.

First navigate to the "Assets/Racetrack builder/Prefabs/Track Templates" project folder and drag the "template base" prefab into your scene. This creates the basic skeleton structure for a mesh template. You can examine the object in the scene tree:

- The main object ("template base") contains a **Racetrack Mesh Template** script component. This marks it as a mesh template and is mandatory.
- Underneath we have a "continuous" object with a **Racetrack Continuous** script. This denotes the meshes underneath it as "continuous", meaning they will be warped along the racetrack curves. In this prefab the driving surface and barrier meshes will be "continuous", so that they follow the racetrack curves.
- We also have a "spaced" object, which is an empty placeholder for now. This will be used for objects that will be placed along the track at regular intervals, like the supporting poles.

Create the road surface as follows:

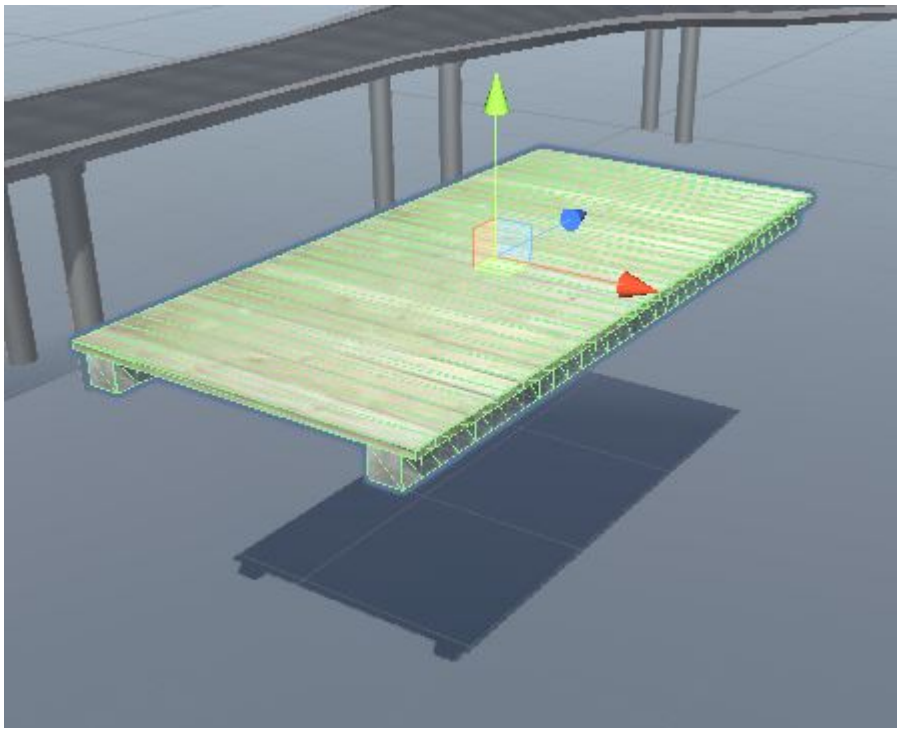
1. Make sure the "template base" object is expanded in the scene tree, and the "continuous" child object is visible.
2. Open the "Assets/Racetrack builder/Prefabs/Template parts/Driving surface" project folder.

3. Drag the "woodplanks" prefab into the *scene tree* and drop it onto the "continuous" child object.
4. Reset the "Position" of the new object to (0,0,0) in the Inspector window.

Placing the woodplanks mesh underneath the "continuous" object means it will be warped along the racetrack curves. The first continuous mesh is also special, as it will be treated as the main driving surface. This has certain implications:

- It defines the length of the mesh template.
- It means a **Racetrack Surface** script component will be attached to the mesh after it has been copied and warped. This is used by the **Racetrack Car Tracker** component to detect when the player is above the road. It also links back to the curve that generated it, so that the player's progress along the track can be calculated.

If you view the template object in the scene, you'll see it now has a woodplanks surface. This is now a fully functional mesh template, which you can drag from the scene tree and drop onto the "Template" property of a racetrack curve. But we are not finished yet.



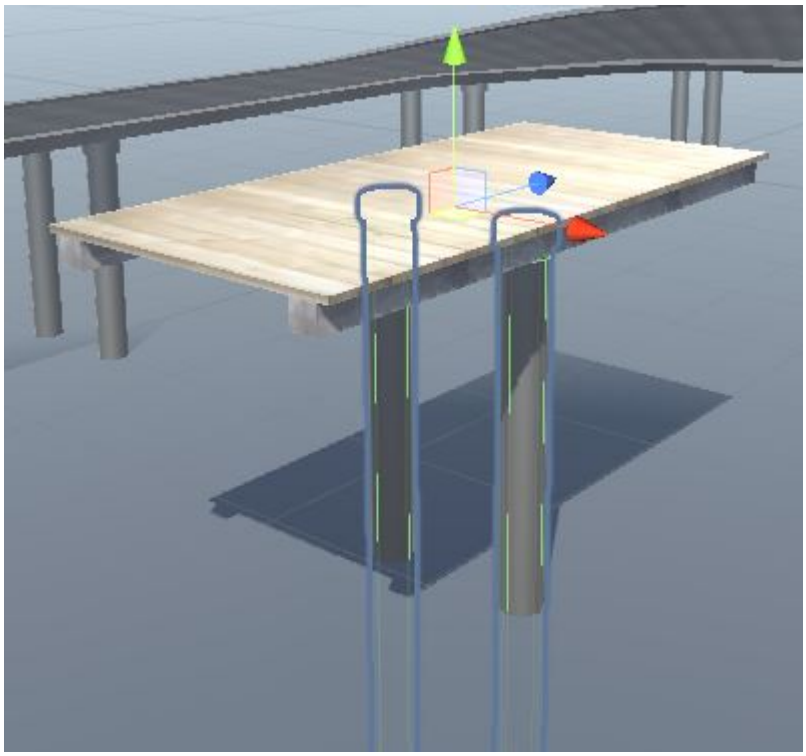
You may notice the template looks a little small. This is because the default mesh templates are all scaled by a factor of 3. Correct this as follows:

1. Click on the "template base" object in the scene tree.
2. Set the X, Y and Z components of the "Scale" to 3 in the Inspector window.

Now add some poles to hold up the track:

1. Make sure "template base" is still expanded in the scene tree, and that "spaced" is visible.
2. Open the "Assets/Racetrack builder/Prefabs/Template parts/Supports" project folder.
3. Drag the "Metal poles" prefab into the *scene tree* and drop it on the "spaced" child object.
4. Set the Y position to -0.33 in the Inspector window.

The mesh template now has two poles underneath the track. Once again it is fully functional. If you assign it to a curve and regenerate the track, the poles will be spaced along it at regular intervals.



Expand the new "Metal poles" object in the scene tree and examine it.

The "Metal poles" object has a **Racetrack Spacing Group** script component. This indicates that the content underneath will be spaced evenly along the track, and specifies the spacing.

The "Index" property is important to get correct spacing. Objects spaced at *different* intervals should be assigned to spacing groups with *different* indices, in order to get correct results. You should organise your spaced objects into distinct groups, each with a unique index between 0 and 15. The supplied mesh templates use index 0 for road support poles and index 1 for poles that hold up the side barriers.

The other properties are:

- Spacing Before - Amount of space to add before an object
- Spacing After - Amount of space to add after an object

In this case we have 10 units before and after each, meaning the poles will be spaced 20 units apart.

Underneath the "Metal poles" object we have two child objects, "left" and "right". Each has a **Racetrack Spaced** script component, indicating it has content to be repeated, with properties describing how:

- Is Vertical - This forces the object to be aligned vertically in world space. If unticked the object will be aligned with surface of the curves.
- Max Z angle - Maximum bank angle (positive or negative). If the curve exceeds this angle, the spaced content will not be created.
- Max X angle - Maximum pitch angle (positive or negative). Same as above.
- Apply Widening - This causes the object to be moved horizontally when the road is widened (or narrowed). If unticked, the object position is not affected by widening.

Important: "Max Z angle" and "Max X angle" only apply when "Is Vertical" is ticked.

In this case "Is Vertical" is ticked, so that the poles remain vertical regardless of how the road surface pitches or banks. The max X and Z angles ensure poles are not created if the track is upside down. (See the loop in

the example scene for an example of upside-down track).

Underneath the "left" and "right" objects is the content to be generated. In this case we have a standard Unity mesh to represent the pole visually, and a capsule collider.

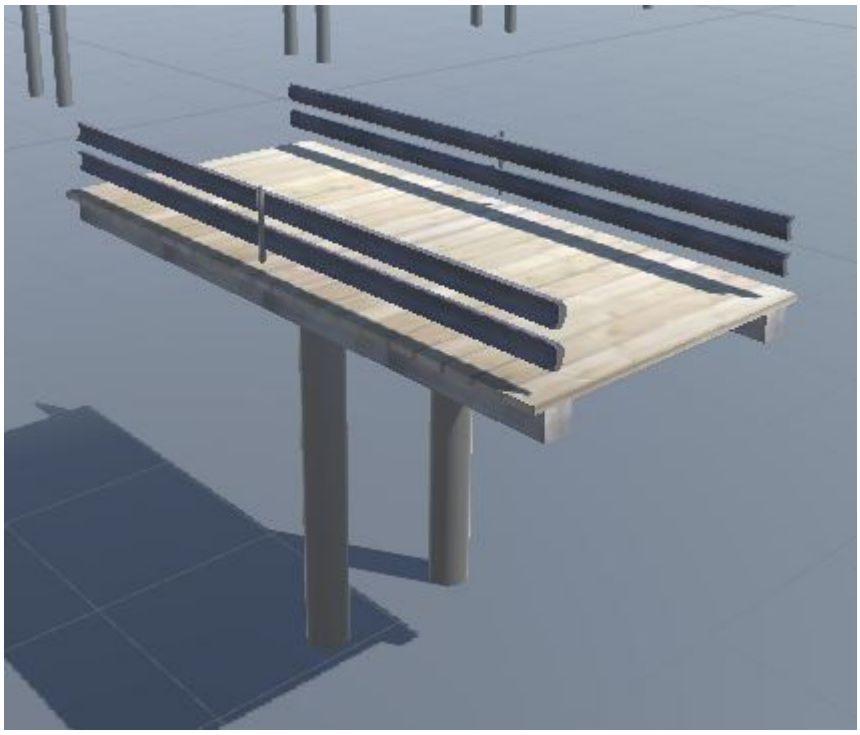
To complete the mesh template, add the side barriers as follows:

1. Open the "Assets/Racetrack builder/Prefabs/Template parts/Sides" project folder.
2. Drag the "rail barrier" prefab and drop it on the "template base" object in the scene tree. *Do not* drop it on the "continuous" or "spaced" child objects.

The prefab should not be a child of the "spaced" or "continuous" objects, because it contains both spaced and continuous content. If you expand the "rail barrier" object, you'll see it has its own "continuous" and "spaced" sub objects with the appropriate components attached to each. The "continuous" child object contains two railing meshes, which will be warped along the curve path. The "spaced" child object contains a small pole configured to be spaced every 6 units along the track.

The barrier runs along one side of the mesh template. To create the barrier on the other side:

1. Right click the "rail barrier" object in the scene tree and select "Duplicate"
2. In the Inspector window, set the rotation Y component to 180.

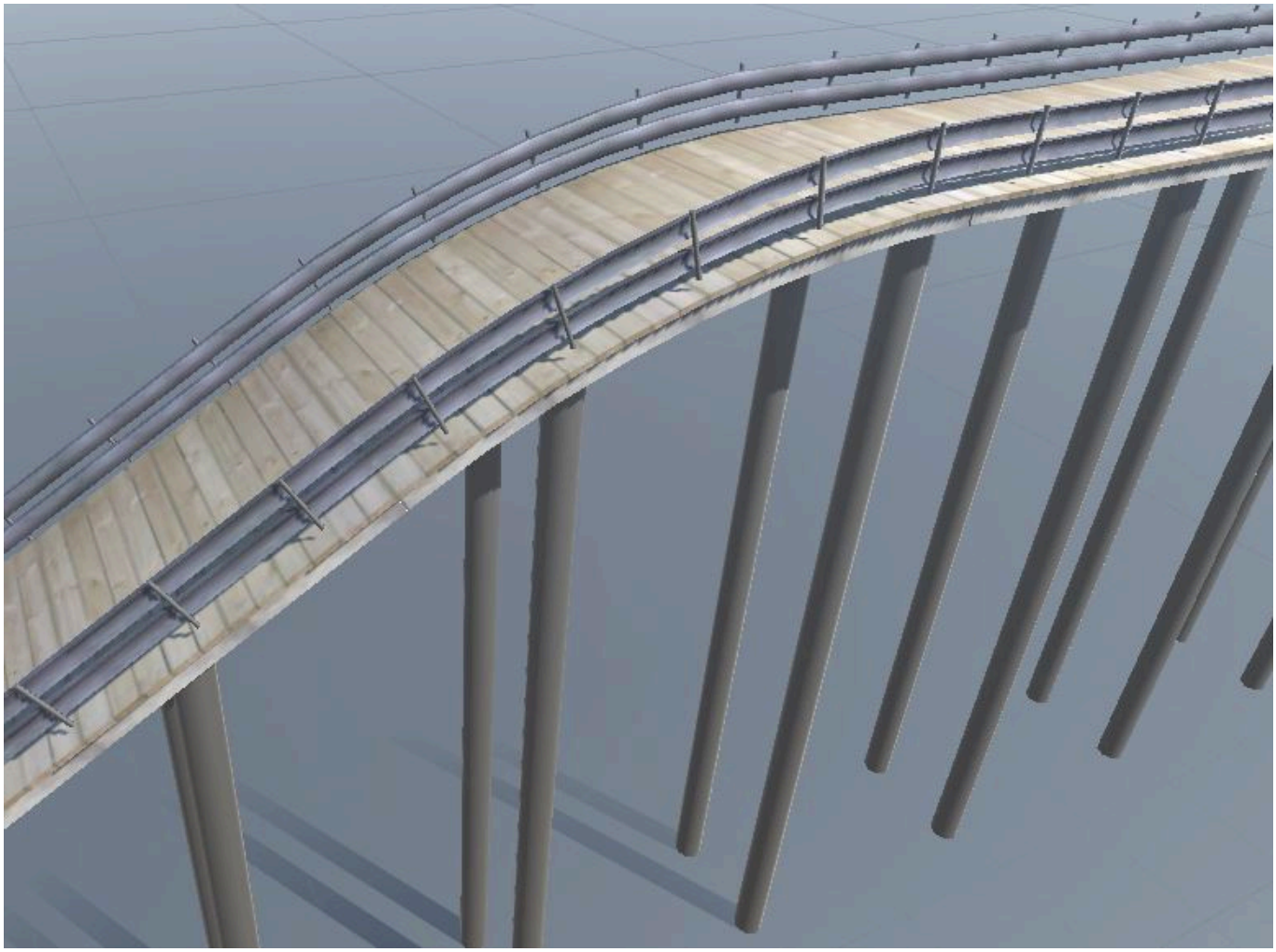


The mesh template is ready to use. The last step is to convert it into a prefab, so that it can be reused easily:

1. Click on the "template base" object in the scene tree.
2. Key in a new name in the Inspector window. E.g. "woodplank barriers"
3. Navigate to the "Assets/Racetrack builder/Prefabs/Track Templates" project folder.
4. Drag the new object from the scene tree into the project window to create a new template.
5. When prompted click "Prefab Variant" (although it doesn't make much difference in this case).

Once the prefab has been created, you can delete the object from your scene.

The template is now complete. Drag it from the project folder onto the "Template" field of a racetrack curve and click "Rest of track" to apply it to your track.



Creating meshes

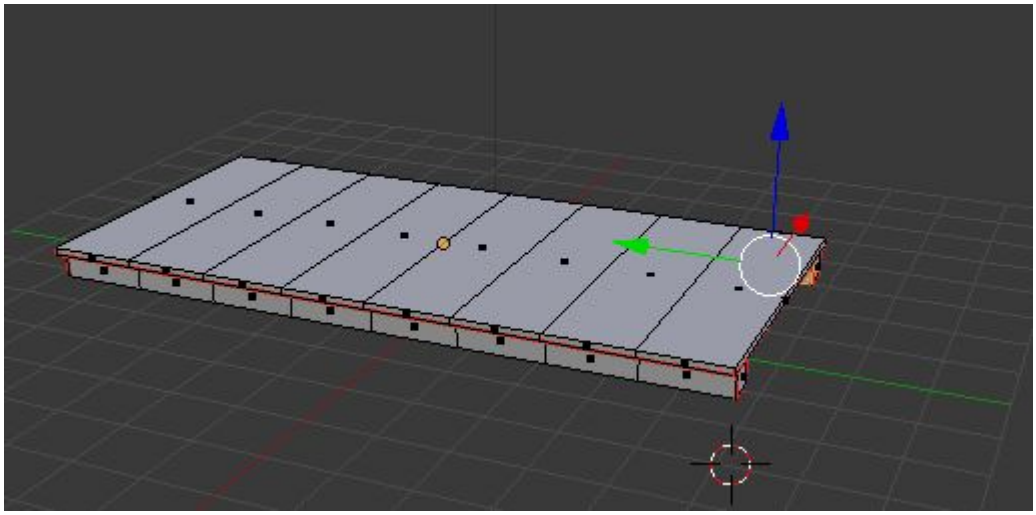
You can of course use your own meshes to create track templates and build your own custom tracks.

The standard Unity workflow applies. I.e. create them in a 3rd party modeller, import them, and use them to build your mesh template prefabs.

For continuous meshes, like the road surface, there are some guidelines to ensure they will warp around the racetrack curves correctly, and provide a smooth driving experience.

Split along the Z axis

Meshes are warped to fit a curve by transforming their existing vertices. The process does not split existing polygons or introduce new vertices, so you must provide sufficient vertices to make the result look smooth.



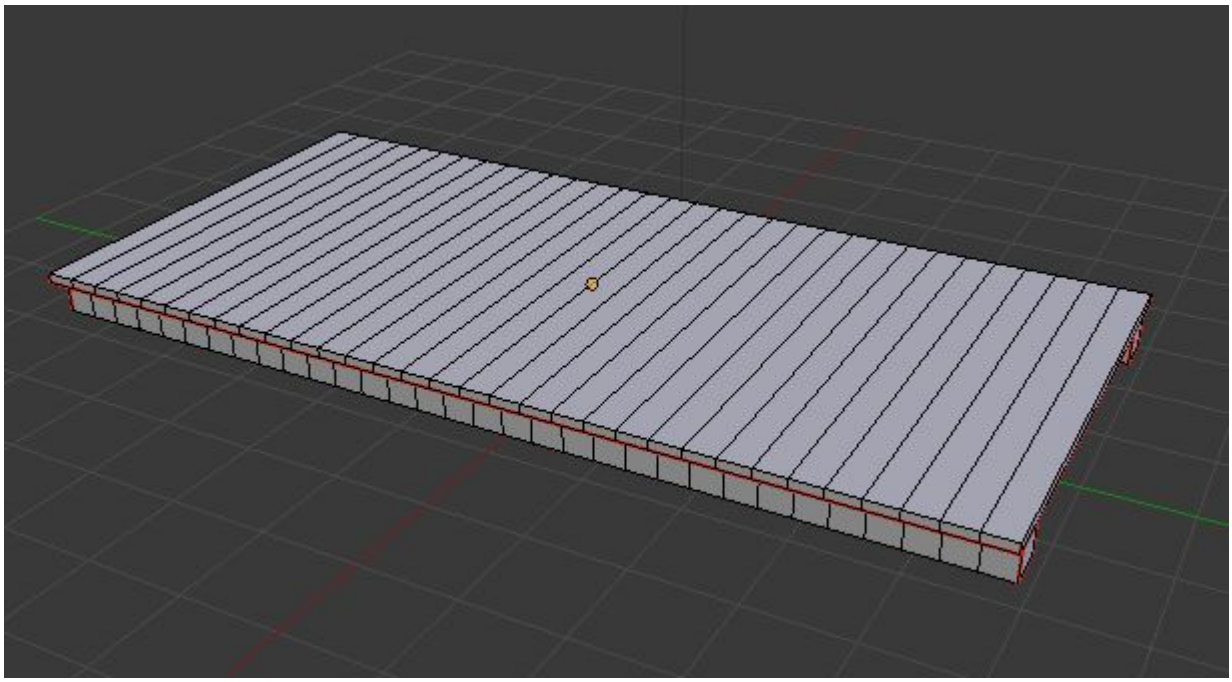
Here the road surface mesh has been sliced into 8 pieces, at regular intervals along the Z axis. (In this case the Blender3D "Loop cut" tool was used. Other 3D modelers should have a similar function.)

This is the visual mesh used for the "woodplank" road surface.

Separate high poly mesh for collisions

The visual mesh *looks* decent when warped around the racetrack curves, but does not have adequate detail to be used as the collision model. It will feel juddery to drive over when the road pitches or banks.

To fix this, create a second mesh, identical to the first, but with more polygons and vertices, so that it can be warped into a smoother surface.



Assign this mesh explicitly to the "Mesh Collider" component of your mesh object or prefab.

Unlike the rendered mesh, having a high resolution collision mesh has little performance penalty. Although it is worth keeping in mind the storage requirements of the mesh, especially as it will be duplicated many times along the different curves of the racetrack.

In this example the mesh has been sliced into 32 pieces, which gives a decent driving feel without being too excessive.

Note: For continuous meshes that the player will not be driving on, like barriers or walls, it is not necessary to have a separate high poly mesh

Use a mesh collider

For continuous meshes the collision mesh must be warped along the racetrack curves along with the visible mesh. Although other colliders, like the box collider, may fit neatly around the mesh template, they cannot be warped, which will result in an incorrect collision model on your racetrack.

Therefore always use a mesh collider with continuous meshes.

Align the top of drivable surfaces with the Y=0 plane

This is more of a suggestion than a rule, however it makes life a lot easier.

It means a continuous surface when the track switches from one mesh template to another. Also, applying a different scale factor to a mesh template will not alter the height of the surface, in case you want to vary the size of the track.

Spaced objects

Spaced objects are much simpler than continuous meshes. There is no mesh warping. They are simply instantiated (using standard Unity object instantiation) and positioned at regular intervals along the track.

This means:

- You do not need to add extra vertices.
- You do not restricted to a mesh collider. For example, you can use a box or capsule collider if suitable.

Appendix - Component reference

Track building components

Location: *Assets/Racetrack Builder/Scripts/Track*

Use these components to define the racetrack curves.

Racetrack

Represents the racetrack as a whole. Acts as the parent object for the **Racetrack Curve** objects that define the curves of the track.

Use "GameObject > 3D Object > Racetrack" to create a **Racetrack** object.

Properties - Parameters

Segment Length

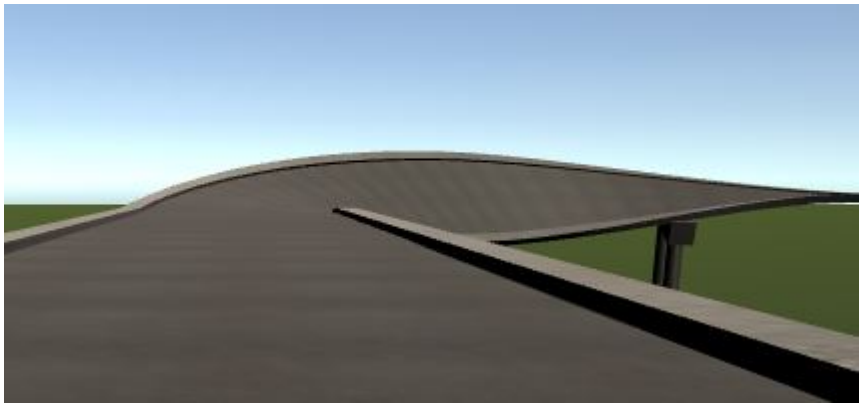
Length of line segment generated from curves. Each curve is converted into a sequence of small line segments. Generally you should not need to change this setting.

Bank Angle Interpolation

Controls how bank (Z) angles are interpolated along a curve.

This sets the default for the entire racetrack, but can be overridden on individual curves.

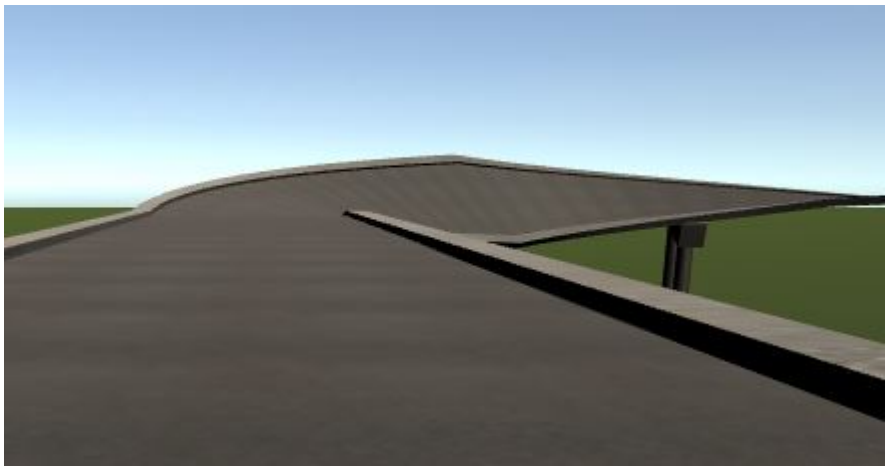
The default value is "Bezier" which applies a smooth Bezier curve interpolation. This avoids sharp angles at the end of curves, and generally gives a smoother driving surface.



Note that there are two Bezier options, "Bezier" and "Bezier Unclamped". The regular "Bezier" clamps the bank angles so that they always fall between the original values, whereas "Bezier Unclamped" does not.

For example, if the bank angle is defined as 0 on the previous curve and 45 on the current curve, "Bezier" will ensure the interpolated bank angle remains between 0 and 45, whereas with "Bezier Unclamped" it may bank over 45 degrees or less than 0 degrees (i.e. bank the other way) in order to create the smoothest Bezier curve.

"Linear" applies a simple linear interpolation.



Note: The remaining value "Inherit" should not be used when defining interpolation at the racetrack level. It is intended for ***Racetrack Curves****.*

Widening Interpolation

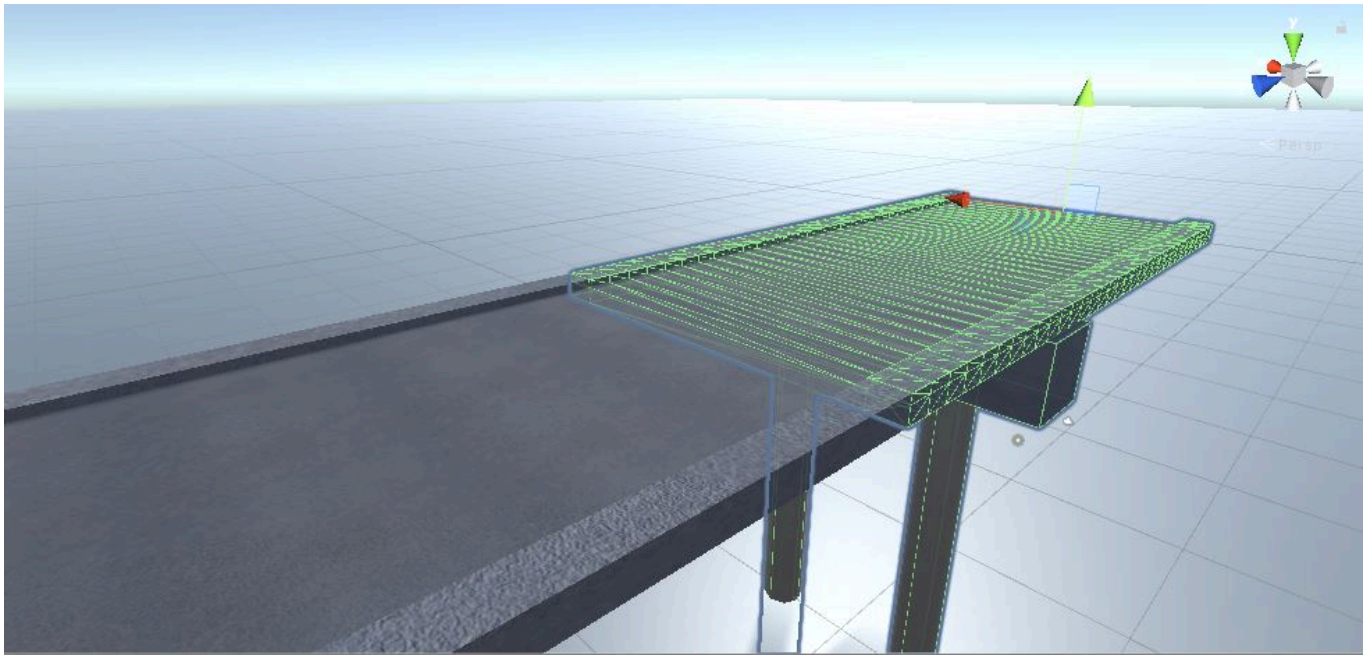
Controls how widening values are interpolated along a curve.

This has the same options and behaves the same as "Bank Angle Interpolation" (see above).

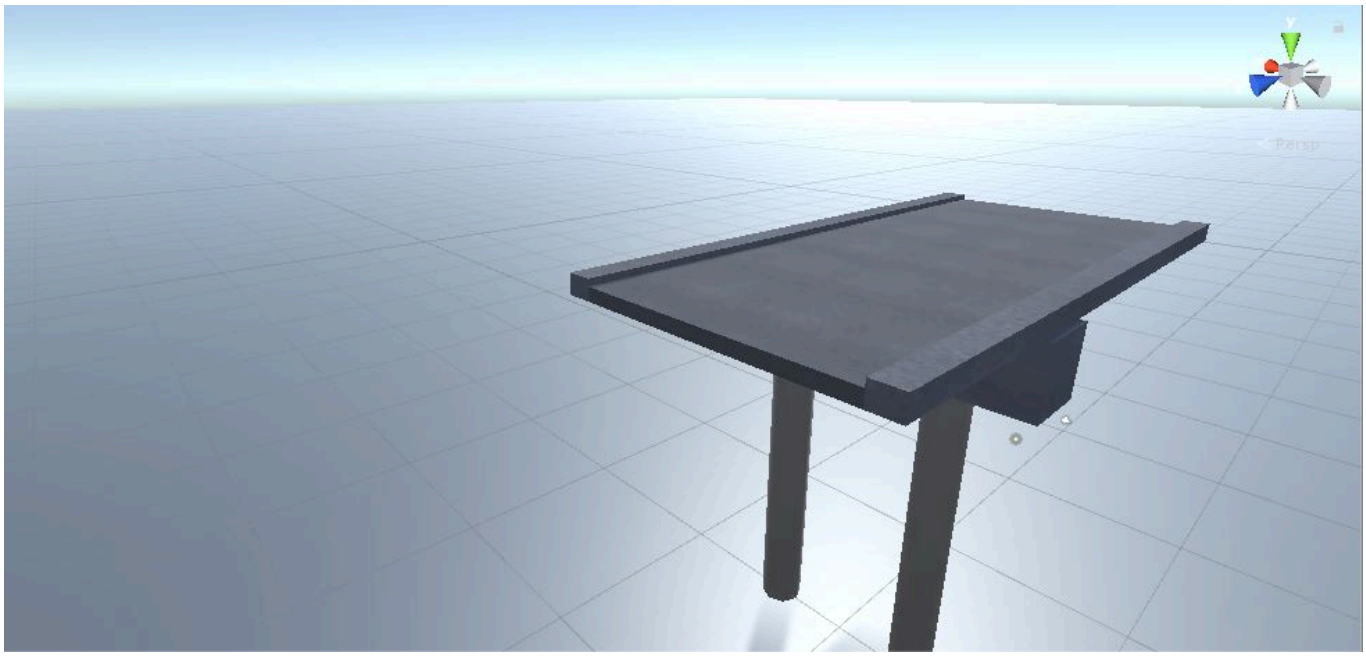
Remove Internal Faces

Whether to remove internal faces that are hidden due to consecutive meshes being adjacent to each other.

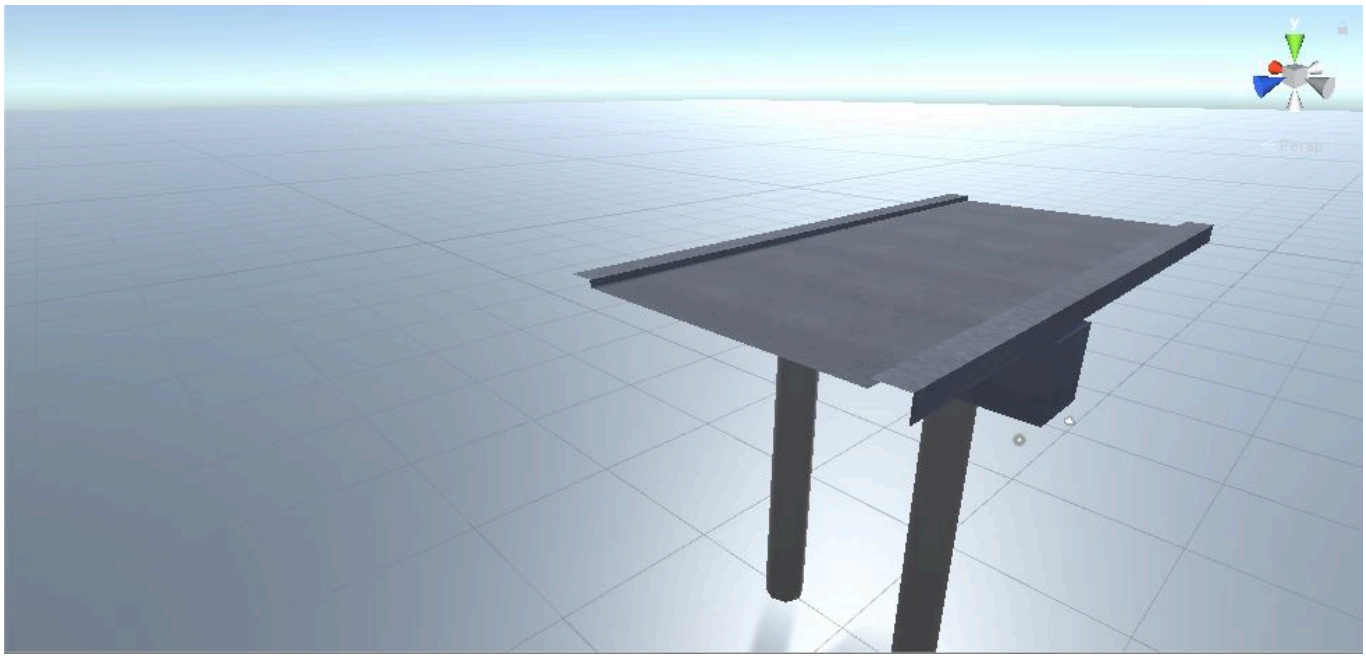
For example, consider these two consecutive asphalt meshes:



If we hide the second mesh instance and examine the first mesh with "Remove Internal Faces" *un-ticked*, we can see that it has polygons at the cross section between the two instances. These will never be seen in practice, but can sometimes poke through due to Z fighting causing visual artifacts.



Here's the same mesh instance generated with "Remove Internal Faces" *ticked*. The cross sectional faces are automatically removed.



By default Racetrack Builder will only remove faces between two continuous meshes in consecutive mesh template instances generated from the same RacetrackMeshTemplate. It will not remove faces that might be visible, such as at the start or end of the track, or around jumps.

You can override this for a specific **Racetrack Curve** by setting the "Remove Start Internal Faces" and/or "Remove End Internal Faces" properties of the curve.

Properties - Respawning

Respawn Height

Height of the respawn points above the racetrack surface.

Respawn Z Offset

How far along each curve to place its respawn point. *Note: Racetrack Builder will never place a respawn point further than halfway down a curve, regardless of this value.*

Properties - Looping

Mesh Overrun

What to do with the part of the last mesh that overruns the end of the curves. For more information see the Creating a closed circuit section earlier in this document.

Loop Y offset

Offset to apply to mesh Y position when Mesh Overrun mode is "Loop". Also see the Creating a closed circuit section.

Properties - Connectors

Connectors store which **Racetrack Connector** objects the start and end of the racetrack is connected to. This is how Racetrack Builder tracks which racetracks are connected to which junctions.

Typically these properties are not set directly in the inspector, but as a side effect of dragging connection handles in the scene view.

Start Connector

The **Racetrack Connector** to which the start of the racetrack is connected. "None" if not connected.

End Connector

The **Racetrack Connector** to which the end of the racetrack is connected. "None" if not connected.

Properties - Curve start state

This section defines the start position of the racetrack, and the curve state at the start of the first curve.

Properties - UI settings

Show Manipulation Handles

Whether to show drag and rotate handles when a **Racetrack Curve** is selected (in addition to the regular Unity handles), allowing the curve to be manipulated directly on screen.

Show On Screen Buttons

Whether to display the curve manipulation buttons on screen when a **Racetrack Curve** is selected.

Auto Update

Whether to automatically update the racetrack model when changes are made to the curves.

Ticking this option causes the racetrack model to be updated:

- When a curve is added/inserted
- When a curve angle or length is changed *using the inspector buttons*
- When using the "Create closed circuit" function

Important: Regardless of this setting, no meshes will ever be automatically recreated in response to adjusting a slider, keying a value or using the on-screen handles.

Miscellaneous

This section contains buttons for less commonly used actions.

Delete meshes

Delete the entire racetrack model (but not the curves).

Note: You can recreate the model using the "Recreate track" (racetrack) or "Update track" (curve) buttons.

Clear templates

Clear the track templates from all curves (sets them to null).

Combine static batches

Calls `StaticBatchingUtility.Combine()` on the racetrack object.

This usually reduces the number of batch calls when rendering, which may increase performance in some cases. It may also decrease performance, so be sure to measure before and after.

See the Unity manual for more information.

Note: If you make any changes to the racetrack after combining static batches, the meshes will be replaced and the static batching effect will be lost. You will need to click "Combine static batches" again.

Buttons

Update track

Updates the track meshes as necessary to bring them up to date with any changes to the racetrack or curves. This button has the exact same effect as the "Update track" button for **Racetrack Curve** objects. It is made available here for convenience.

Recreate track

Delete and recreate all generated meshes for all curves. This can take several seconds for larger racetracks.

Usually this is unnecessary as "Update Track" is more efficient and should have the same net result. But if the racetrack appears to have broken somehow in a way that "Update Track" cannot correct, then "Recreate track" will often fix this.

Create secondary UVs

By default Racetrack builder does not generate secondary UVs for continuous surfaces, as they are relatively slow to generate.

Click this button to regenerate the meshes with secondary UVs for baked lighting. Depending on the size of your racetrack, this could take several seconds.

You can then bake your lighting as normal.

Note: If you make any changes to the racetrack after generating secondary UVs, the UV information will be lost. You will need to use the "Create secondary UVs" function again before baking lighting again.

Create closed circuit

Creates a closed circuit racetrack by adding a Bezier curve. Refer to the **Creating a closed circuit** section earlier in this document.

*Note: This button is not available if the **Racetrack** is connected to a junction.*

Racetrack Curve

Represents a single curve in the racetrack. Stores the shape of the curve.

Properties

Type

Selects the type of curve. "Arc" is the traditional curve type where the curve is defined by its angles and length. "Bezier" is a cubic Bezier curve defined by angles and an explicit end position.

End Position

The end position of the curve in world space. This is readonly for "Arc" curves.

Start Control Pt Dist

The distance of the start Bezier control point from the curve start point. As a fraction of the straight line distance between the curve start and end points. Applies to "Bezier" curves only.

End Control Pt Dist

The distance of the end Bezier control point from the curve end point. As a fraction of the straight line distance between the curve start and end points. Applies to "Bezier" curves only.

Length

The curve length. This is readonly for "Bezier" curves.

Turn (Y)

How far the curve turns around the Y axis. Positive values create right hand corners.

Gradient (X)

The X axis angle at the *end* of the curve. The curve curves from the previous gradient to the new one. Negative values create uphill slopes.

Bank (Z)

The Z axis angle at the *end* of the curve. The curve curves from the previous bank angle to the new one. Positive values bank the road right.

Bank Angle Interpolation

Allows the bank angle interpolation method to be overridden for the current curve.

The default value is "Inherit", meaning the curve will use the value set on the **Racetrack** or **Racetrack Group** object.

Refer to the "Bank Angle Interpolation" property of the "Racetrack" component for details.

Bank Pivot X

Affects how the track banks left and right. See the "Bank Pivot" section for details.

Widening

The number of units to widen the left and right hand side of the racetrack by. Negative values will narrow the racetrack, if and to the degree that the track type supports it. Applies to the *end* of the curve. The curve interpolates from the previous widening values to the new ones.

Widening Interpolation

Allows the widening interpolation method to be overridden of the current curve.

The default value is "Inherit", meaning the curve will use the value set on the **Racetrack** or **Racetrack Group** object.

Refer to the "Widening Interpolation" property of the "Racetrack" component for details.

Template

The **Racetrack Mesh Template** that defines the meshes that will be warped along the **Racetrack Curve** objects to create the final model. Can be left set to "None" to use the previous curve's template.

Properties - Miscellaneous

Is Jump

If true, no meshes will be created for the curve, leaving a gap in the racetrack.

Align Meshes To End

Scales the meshes so that the end of the last mesh aligns exactly with the end of the curve. See the Aligning meshes to curves section.

Can Respawn

If true, the **Racetrack Car Tracker** component will respawn the car above this curve. Otherwise it will skip it and search backwards along the racetrack for a curve with "Can Respawn" set to true. Useful to ensure there is enough run-up before jumps etc.

Raise Terrain

Instructs the **Racetrack Terrain Modifier** to raise the terrain if necessary to meet the curve. See the Working with terrains section.

Lower Terrain

Instructs the **Racetrack Terrain Modifier** to raise the terrain if necessary to ensure the curve does not pass underneath the terrain surface. See the Working with terrains section.

Remove Start Internal Faces

Whether to remove internal faces at the start of each mesh instance. One of the following:

- Auto - The behaviour is controlled by the **Racetrack Builder** "Remove Internal Faces" property (see above).
- Yes - Remove the internal faces from the start of all mesh instances generated for this curve.
- No - Do not remove internal faces from the start of any mesh instances generated for this curve.

Remove End Internal Faces

Whether to remove internal faces at the end of each mesh instance. Has the same options as "Remove Start Internal Faces".

Buttons

Add curve

Add a new curve to the end of the track and select it.

(This button is not available when "Mesh Overrun" is set to "Loop" in the Racetrack object's inspector window.)

Insert curve

Insert a new curve in the track immediately after this one.

(This button does not appear when the last curve is selected.)

Update track

Update the mesh model to bring it inline with changes made to the curves.

Use this button after making one or more changes that don't automatically update the mesh model (see the "Auto Update" property on the Racetrack component for more on this.)

Racetrack Builder attempts to find the minimum number of changes required to update the mesh model, reusing existing racetrack mesh template copy objects whenever possible. The number of (re)created and unmodified template copies will be displayed under the button. "Spaced items only" refers to existing template copies that were partially reused, reusing the continuous meshes, but regenerating the repeating content.

Racetrack Junction

The **Racetrack Junction** component is used in conjunction with the **Racetrack Connector** to create junctions. These are objects with defined connection points that Racetrack objects can connect to.

Racetrack Builder is released with a number of pre-built junction prefabs in the /Prefabs/Junctions subfolder.

The **Racetrack Junction** is placed on the object that represents the junction as a whole. This is the object that is moved when a the junction is connected to a racetrack by dragging a junction connector to a racetrack connection point.

The component also provides a convenient inspector UI for creating and connecting Racetrack objects, and updating connected racetracks after the junction has been moved or rotated.

No properties

Buttons

New racetrack

Displayed next to each **Racetrack Connector** that is not currently connected to a racetrack.

This button creates a new racetrack and automatically connects its start position to the connector. Specifically it sets the Racetrack object's "Start Connector" to the **Racetrack Connector** object and sets the Racetrack's "Curve start state" so that it starts at the connector position, facing outwards.

Disconnect

Displayed next to each **Racetrack Connector** that is connected to a racetrack.

This button disconnects the Racetrack object from the connector. Specifically it sets the Racetrack object's "Start Connector" or "End Connector" to "None" as appropriate.

Update racetracks

This updates all connected racetracks to ensure they connect cleanly with the junction.

Typically this button is used after the junction has been moved and/or rotated to bring the racetrack meshes up to date.

A Racetrack object is connected if its "Start Connector" or "End Connector" is set to a **Racetrack Connector** belonging to the junction. If connected via the "Start Connector", the racetrack's start position is adjusted by setting its "Curve start state". If connected via the "End Connector", the racetrack's end position is adjusted by setting the properties of its last curve, which is always a bezier curve in this case.

Racetrack Relative

Defines an object's position relative to the racetrack so that it is automatically repositioned when the racetrack is modified.

See the **Placing objects on the track** section.

Properties

Position

Specifies the position in relation to the parent racetrack or curve.

The Z coordinate is the distance along the racetrack/curve.

The X and Y coordinates are relative to the racetrack/curve cross section at that distance.

Rotation

Specifies the object rotation in relation to the track where it is positioned.

In this coordinate space the Z axis points along the racetrack path, and the Y axis points "up" away from the track surface.

Buttons

Update position

Updates the object's position based on the racetrack relative "Position" and "Rotation".

Note: Editing the "Position" or "Rotation" properties automatically updates the position as well. This button can occasionally be useful after changing the object's parent to a different curve, for example.

Racetrack Mesh Manager

Tracks meshes generated when **Racetrack Mesh Templates** are warped along **Racetrack Curves** and automatically re-uses duplicate meshes when detected.

Adding a single **Racetrack Mesh Manager** to any object in the scene enables this functionalty.

The **Racetrack Mesh Manager** can be combined with a **Racetrack Saved Meshes** scriptable object to save warped meshes as assets in a project folder and automatically use them where applicable.

See the **Re-using meshes** section in this document for more information.

Properties

Scene meshes

Shows the meshes that have been generated and stored in the current scene.

Meshes are grouped by the **Racetrack Mesh Template**(s) that generated them.

Each mesh has a tickbox allowing you to select which meshes you wish to save as project assets.

Saved Meshes

Optional link to a **Racetrack Saved Meshes** scriptable object to enable meshes to be saved as project assets for re-use across multiple scenes.

Buttons

Select in scene

Displayed next to meshes and mesh groups when "Scene meshes" is expanded.

Selects all matching meshes in the scene.

Clear scene meshes

Clears all scene mesh references from the **Racetrack Mesh Manager**.

Meshes will not be deleted immediately as they will typically still be referenced by Mesh Filter and/or Mesh Collider components in the racetrack mesh model.

Usually you don't need to use this function.

Save scene meshes

This button only appears when the "Saved Meshes" property has been set.

Clicking this button converts the selected scene meshes into assets in your project.

The project folder is specified in the "Save Folder" property of the referenced **Racetrack Saved Meshes** scriptable object.

Racetrack Saved Meshes

Note: This is a Scriptable Object, not a component.

The **Racetrack Saved Meshes** object is a scriptable object that can be created inside a project folder by:

1. Right clicking the project folder
2. Selecting: Create > Scriptable Objects > Racetrack Saved Meshes

This object tracks which meshes have been saved as assets to a project folder, so that they can automatically be re-used across multiple scenes.

To use it, create a single instance in your project and reference it from a **Racetrack Mesh Manager** component in each scene.

See the Re-using meshes section in this document for more information.

Properties

Save Folder

The path to the project folder to save the mesh assets into.

Must be a subfolder of the main "Assets" folder.

Racetrack Builder will automatically create this folder when required, if it doesn't already exist.

Index Generator

Used to generate unique asset names.

You shouldn't need to change this property. (Doing so could cause mesh assets to be tracked incorrectly.)

Hash Method

The method used to generate hashes for tracking meshes.

You shouldn't need to change this property. (Doing so would prevent the object from finding and reusing existing saved meshes.)

This property exists to ensure backwards compatibility with projects created with older Racetrack Builder versions that used a simpler hashing method.

Meshes

An array of tracked mesh assets.

This array contains the information Racetrack Builder needs to detect when a mesh asset can be re-used in a racetrack.

You shouldn't need to change this property. (Doing so could cause mesh assets to be tracked incorrectly.)

Buttons

Remove deleted meshes

If tracked meshes have been deleted from the project, use this button to remove the corresponding entries from the "Meshes" array.

This is not strictly necessary, as Racetrack Builder will ignore "Meshes" entries for deleted mesh assets anyway, but it helps keep things tidy.

Mesh building components

Location: *Assets/Racetrack Builder/Scripts/Template*

Use these components to build the mesh templates that are used to create the 3D model of the racetrack.

Racetrack Mesh Template

Marks an object as a mesh template. This component is required in order to be assigned to a **Racetrack Curve** object's "Template" field.

Properties

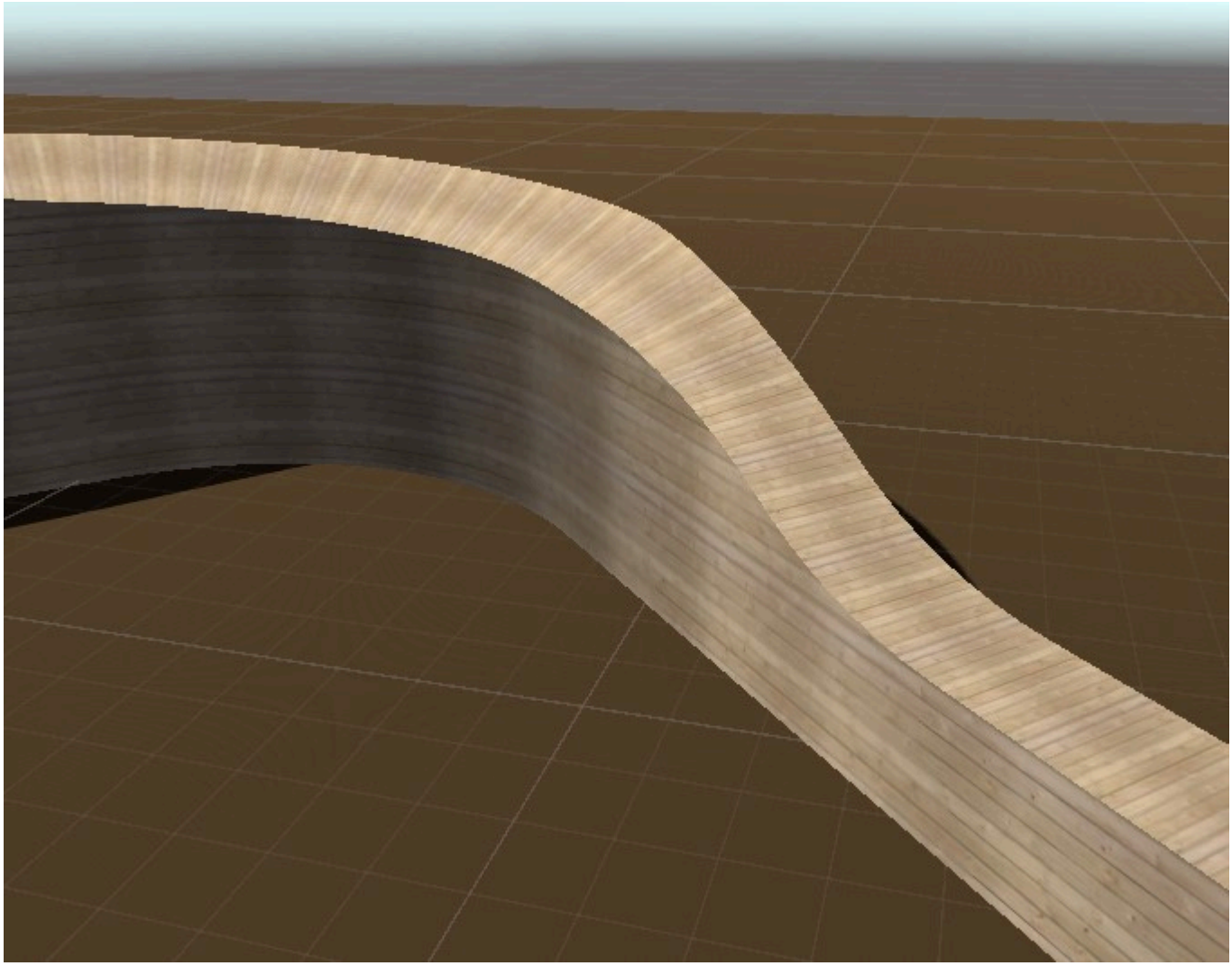
XZ Axis Transform

This specifies the matrix transform used to transform the mesh around the X and Z axes.

Its values are:

- Rotate (Default)
- Shear

"Shear" is useful for creating track types with vertical sides, as the shear transform preserves vertical surfaces. This means the side walls remain vertical regardless of how the track banks around the Z axis or pitches around the X axis.



When using a "shear" track type you should avoid pitch angles greater than 90 degrees or bank angles approaching 90 degrees, as these will distort the mesh model into incorrect shapes. This means you cannot create loops (where the player drives upside down) with these track types.

Auto Min Max Z

When set (ticked) the "Min Z" and "Max Z" properties are calculated automatically from the main surface mesh.

The main surface is the first **Racetrack Continuous** mesh in the template. The minimum and maximum Z properties are calculated as the minimum and maximum Z values of the vertices in that mesh.

Min Z

Sets the minimum Z value of the mesh template.

Max Z

Sets the maximum Z value of the mesh template.

When mesh template copies are generated along the racetrack path, they are positioned such that the Max Z of a mesh template aligns with the Min Z of the next template.

Thus subtracting Max Z from Min Z gives the effective length of the mesh template.

Racetrack Continuous

Indicates that all the meshes in the subtree are "continuous". The track building algorithm will copy the subtree, find all its meshes, and warp them around the racetrack curves.

Used for the road and other surfaces that follow the racetrack curvature, like barriers, tunnel walls etc.

IMPORTANT: The first "continuous" object containing a mesh defines the racetrack "surface". This is the mesh above which the car must be to be considered on the racetrack, and affects how the ***Racetrack Car Tracker*** *detects the car's progress along the racetrack and whether they have fallen off.* ("First" in this case is the first applicable object based on a depth first search of the mesh template.)

A mesh template typically will have at least one of these.

No properties.

Racetrack Spacing Group

Assigns all **Racetrack Spaced** objects in the subtree to a spacing group, and specifies their spacing.

Used to space out repeating objects like support poles at regular intervals. Spaced objects are positioned along the racetrack curves, but their meshes are not warped to the curve.

Properties

Index

The spacing group index. The mesh building algorithm maintains 16 "spacing groups", indices 0 to 15, which step down the racetrack spacing out objects. If a mesh template has different objects with different spacing, they should each be assigned to a spacing group with a different index.

Spacing Before

Space to apply before an object

Spacing After

Space to apply after an object

Racetrack Spaced

Indicates the subtree is a "spaced" object that will be copied and placed along the racetrack at regular intervals.

The actual spacing must be specified in a **Racetrack Spacing Group** object higher up in the object hierarchy.

Properties

Is Vertical

True to align the object's Y axis with the world Y axis. False to align it to the curve's local Y axis.

Max Z Angle

Applies only when Is Vertical=true. Specifies the maximum posive or negative Z axis angle (or "bank" angle) of the curves where an object will be placed. If the curve exceeds this angle, no spaced object will be created.

Max X Angle

Applies only when Is Vertical=true. Specifies the maximum posive or negative X axis angle (or "pitch" angle) of the curves where an object will be placed. If the curve exceeds this angle, no spaced object will be created.

Note: Max Z and X Angles are useful to prevent road support poles from being created for upside down portions of the racetrack, or when the track is very steep.

Racetrack Widen Ranges

Describes how to widen or narrow the the mesh template, or part of it.

The widen range affects the object it is attached to and its subtree, and controls how continuous meshes are warped, and spaced objects moved horizontally (if "Apply Widening" is ticked).

The component has a "Left" and "Right" ranges, each defining a horizontal region that is stretched or squeezed in order to widen or narrow the road.

*If no **Racetrack Widen Range** component is found the mesh template defaults to two 0 width ranges in the center of the mesh template, meaning that widening simply moves all content to the left or right depending on which side of the center line it is, and narrowing is not supported.*

Properties

Left

The left range

- X0 - X coordinate of the start (left) of the range
- X1 - X coordinate of the end (right) of the range
- Min Width - Minimum width when road is narrowed. The range will not squeeze narrower than this value.

Right

The right range. Has the same properties as the left range.

Y

Y coordinates of the guide lines when displayed in the editor. This has no effect at all on the racetrack generation.

Buttons

<< Mirror to left

Sets the left range to a mirror copy of the right range.

Mirror to right >>

Sets the right range to a mirror copy of the left range.

Racetrack UV generator

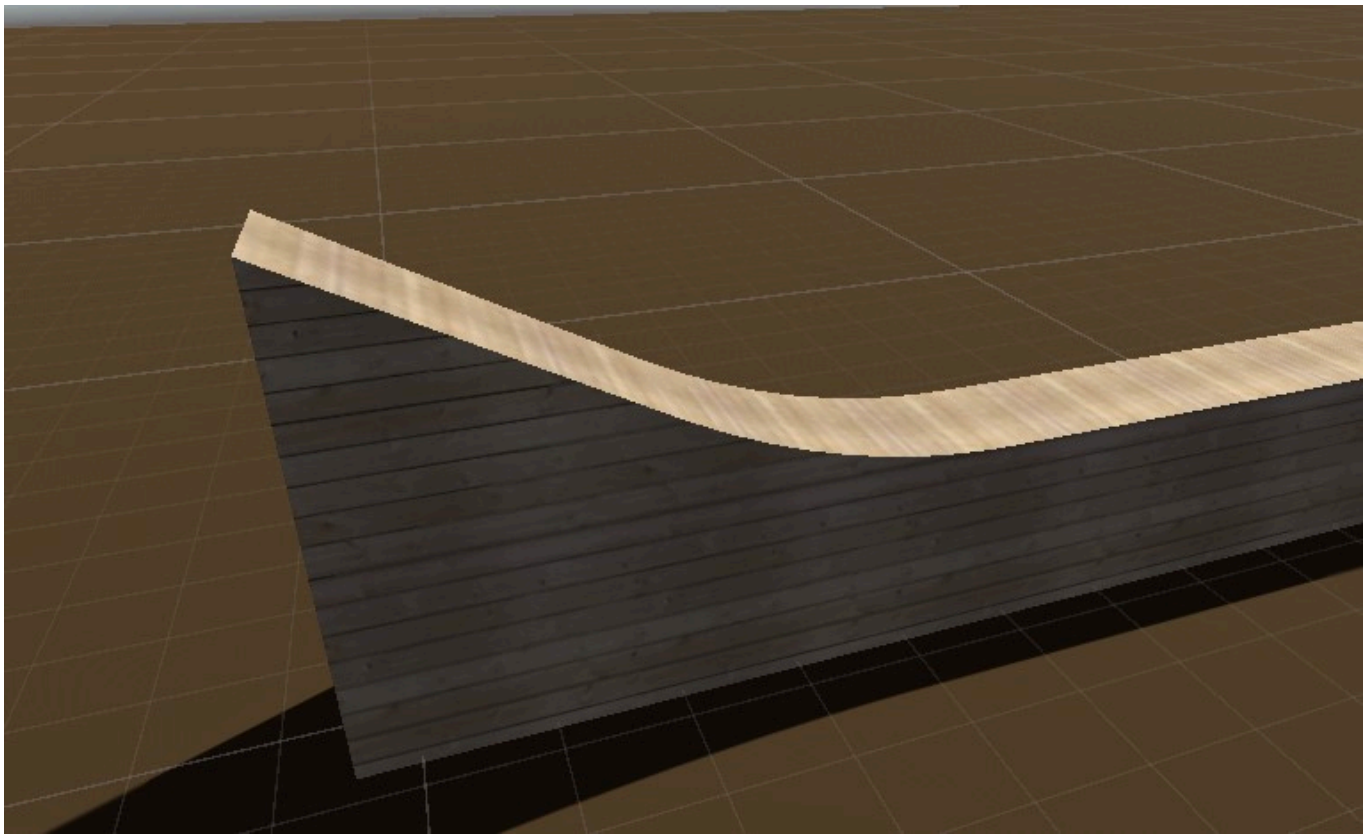
Can be used to generate UV coordinates racetrack surfaces.

The most common use is to texture the main driving surface, but it can also be used to texture the sides and the front or back surfaces when the racetrack breaks for a jump.

The UV generator is particularly useful when road widening is used, as otherwise the vertices will be moved while the texture coordinates reamin the same, resulting in stretched textures which often is not desired.

The UV generation takes the widening into account, generating UV coordinates based on the vertices X and Z coordinates *after* wideing, but *before* warping the mesh along the curve path.

Another use is texturing vertical sides to align the V texture coordinate with the world-space height.



And it can be used to seamlessly apply a repeating texture along the racetrack that doesn't exactly match the length of the mesh template.

The Racetrack UV Generator must be added to the mesh object, specifically the object that contains the MeshRenderer. UV coordinates will then be generated when the racetrack is generated from the **Racetrack**

Curve objects. Please note that this component does not change the UV coordinates on the mesh template itself. You must generate/update the racetrack to see its effects.

Properties

Materials

Specify the materials for which UV coordinates will be generated

Side

Which side(s) of the mesh to generate UV coordinates for. Values are self explanatory:

- Top
- Bottom
- Top And Bottom
- Left
- Right
- Left And Right
- Front
- Back
- Front And Back

Max Angle

The maximum angle between the polygon and the up vector, if generating UVs for "top" facing polygons, or down vector for "bottom" facing polygons. UVs will

Scale

X and Y scale factors. Note that the UV coordinates are actually *divided* by these factors, so larger numbers make the texture appear larger.

Offset

2D offset added to the UV.

Rotation

An optional rotation amount (in degrees) to apply to the UV coordinates.

U Source

Specifies where the initial value for the U (horizontal) texture coordinate comes from. One of:

- Track X - The X coordinate relative to the racetrack (default)
- Track Y - The Y coordinate relative to the surface of the racetrack
- Track Z - The Z coordinate relative to the start of the racetrack (i.e. the distance along the racetrack)
- World X - The world space X coordinate
- World Y - The world space Y coordinate

- World Z - The world space Z coordinate

V Source

Specifies where the initial value for the V (vertical) texture coordinate comes from.

Seel U Source (above) for possible values. The default is "Track Z".

Racetrack Terrain Modifier

Modifies a Unity Terrain heightmap to fit around the Racetrack and **Racetrack Junctions** in the scene.

This component must be added to the Unity terrain object that it is to modify.

See the Working with terrains section.

Properties

Root Object

Specifies which Racetracks and **Racetrack Junctions** should be considered when modifying the terrain. Only descendents of this object in the Unity scene hierarchy will considered.

If set to None, the terrain modifier will consider all Racetracks and **Racetrack Junctions** in the scene.

Granularity

The spacing between height samples taken along the racetrack or junctions, in world units.

For correct results this should be less than the effective spacing of heightmap samples in world space.

Note: Smaller values will cause the **Racetrack Terrain Modifier** to take longer, as more samples must be calculated.

Elevation

The target elevation of the racetrack surface above the terrain, in world space units.

Racetrack Width

The effective width of the racetrack to make space for.

Note: Racetrack widening is applied to this value to calculate the actual width to use.

Smoothing Passes

The number of smoothing passes to apply to the height map deltas, after the initial adjustments have been made.

Buttons

Update Terrain

Triggers the terrain modification process.

Terrain Constraints

Tells the **Racetrack Terrain Modifier** component how to adapt the terrain around the current object. The object must have one or more **Terrain Mask Rect** components (typically on child objects) to describe where the bottom of the object.

*Note: This component was designed to be used with **Racetrack Junction**s, but should also work with other objects in general.*

See the Working with terrains section.

Properties

Raise Terrain

Instructs the **Racetrack Terrain Modifier** to raise the terrain if necessary to meet the curve.

Lower Terrain

Instructs the **Racetrack Terrain Modifier** to raise the terrain if necessary to ensure the curve does not pass underneath the terrain surface. See the Working with terrains section.

Terrain Mask Rect

A rectangle describing the bottom of an object to the **Racetrack Terrain Modifier**. At least one of these should be attached to an object with a **Terrain Constraints** component.

The front edge of the rectangle is centered at the origin of the owning object. The back edge is along the object's local Z axis.

*Note: This component was designed to be used with **Racetrack Connector**s, but should also work with other objects in general.*

See the Working with terrains section.

Properties

Length

The length of the rectangle.

Width

The width of the rectangle.

Runtime components

Location: *Assets/Racetrack Builder/Scripts/Runtime*

Racetrack Car Tracker

This component tracks a vehicle's progress along the racetrack, detects when it has come off the track and places it back on.

It must be added to an object with a "Rigid Body" component (typically a "Car" game object).

Properties

Racetrack

The Racetrack object that the car is currently on. Combines with "Current Curve" to describe the car's position. If there is only one Racetrack object in the scene, this value will be set automatically at runtime. If you have multiple Racetrack objects, such as when creating a branched racetrack, you should set this to the racetrack the car will initially be on.

Current Curve

The index of the curve the car has progressed to.

Curve Search Ahead

How many curves to check to determine whether the car is on the track and whether it has progressed to the next curve. Must be at least 2 - one to check if the car is still on the current curve, and one to check if it has reached the next curve. May need to be higher if your racetrack has short curves, as they sometimes do not generate any meshes, so the tracker cannot detect the car driving over them. This value determines the number of ray casts per physics update.

Off Road Timeout

The number of seconds of the car being off the racetrack before it will be automatically reset onto the track. The car is considered "on" the racetrack if it is above the racetrack surface mesh. This includes when the car is airborne - it does not have to actually be touching the road surface. Also "above" means above in the racetrack curve's local space (which can even be *below* in world space, if the track is upside down).

IMPORTANT: If a car is jumping over an "Is Jump" curve, they will not register as being on the racetrack. If you have large jumps with a lot of airtime, make sure to set the Off Road Timeout long enough that the car can complete the jump!

Auto Reset

Enables the automatic reset logic.

Can Go Backwards

Whether the car can drive backwards around the track.

If set to false then the car will be treated as having gone off the racetrack, if it drives backwards onto an earlier curve. The off road timeout and automatic reset will behave accordingly.

Ray Offset

The **Racetrack Car Tracker** detects whether the car is above the track by casting a ray from the car object towards where the racetrack should be.

This property defines the offset of the start of that ray from origin of the car object.

*Note: If the **Racetrack Car Tracker** is not correctly detecting that the car is above the road, it may be because the ray is starting too low, so increasing the Y offset slightly may solve this.*

Finish Line Racetrack

Combines with "Finish Line Curve" to define the position of the finish line for lap counting. If not specified this defaults to the "Racetrack" setting.

Finish Line Curve

Combines with "Finish Line Racetrack" to define the position of the finish line for lap counting.

Lap Count

Increased after each lap for circuit racetracks.

Last Lap Time

The last lap time in seconds.

Best Lap Time

The best (i.e. quickest) lap time in seconds.

Current Lap Time

The current lap time in seconds.

Off Road Timer

The amount of time the car has been off the racetrack. Once this value reaches the *Off Road Timeout* the car will be reset on the track. *Valid if AutoReset=true*. Otherwise will always be 0.

Is above road

Whether the **Racetrack Car Tracker** considers the car to be above the road.

Buttons

Reset car

Place the car back on the racetrack, on the *Current Curve* (or before, if *Can Respawn*=false for that curve.)

Methods

PutCarOnRoad

Places the car back on the racetrack. (Has exactly the same behaviour as clicking "Reset car".)

Generated components

These components are automatically attached to the objects that are created when building the racetrack meshes.

Note: These objects should be considered to be automatically "generated". You should not modify them directly, as they are often destroyed and regenerated, meaning your changes will be lost.

Racetrack Template Copy

Indicates that the object was created by copying (and modifying) a **Racetrack Mesh Template**.

These objects are created when the racetrack is built, and placed underneath their corresponding **Racetrack Curve** objects in the scene hierarchy.

Properties

Template

Points back to the **Racetrack Mesh Template** object from which this object was created.

Racetrack Surface

Attached to the copy of the first continuous mesh in the **Racetrack Mesh Template** to indicate that this is the main driving surface.

The **Racetrack Car Tracker** looks for this component to determine if the car is above the racetrack and which curve.

Properties

Start Curve Index

The index of the first curve the surface belongs to.

End Curve Index

The index of the last curve the surface belongs to. Meshes can span two curves if their lengths don't line up with the curve lengths (or even more if there are curves that are shorter than the meshes).

Racetrack Builder scripting

Racetrack Builder exposes various data structures and routines that runtime scripts can use to:

- Position objects on the racetrack, or move them along the racetrack.
- Calculate an object's position in relation to the racetrack.
- Generate racetrack meshes at runtime.

Here are some examples of runtime scripts.

Position box colliders along the race-track

This was requested by a developer who was implementing their own progress tracking system (as an alternative to the supplied **Racetrack Car Tracker** component). It positions and aligns box colliders along the track, sizing them to fit the width of the road at each point.

This was implemented as a component that could be dropped onto the main Racetrack object.

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

[RequireComponent(typeof(Racetrack))]
public class CreateBoxColliders : MonoBehaviour
{
    void Start()
    {
        Racetrack racetrack = GetComponent<Racetrack>();
        for (float z = 0; z < racetrack.Path.TotalLength; z += 10.0f)
        {
            // Find segment for given z, and the Z offset from start of segment
            float segmentZOffset;
            RacetrackSegment segment = racetrack.Path.GetSegmentAndOffset(z, out
segmentZOffset);

            // Get transformation matrix for segment
            Matrix4x4 segmentToTrack = segment.GetSegmentToTrack(segmentZOffset);
            Matrix4x4 segmentToWorld = transform.localToWorldMatrix *
segmentToTrack;

            // Get road width at that position
            RacetrackWidening widening = segment.GetWidening(segmentZOffset);
            float leftWidth = 6.0f + widening.Left;
            float rightWidth = 6.0f + widening.Right;
            float width = leftWidth + rightWidth;

            // Find middle of the road
            float x = (rightWidth - leftWidth) / 2;

            // Create object for collider
            GameObject colliderObject = new GameObject();
```

```
            // Position collider on racetrack
            colliderObject.transform.position = segmentToWorld.MultiplyPoint(new
Vector3(x, 0.0f, segmentZOffset));
            colliderObject.transform.rotation =
Quaternion.LookRotation(segmentToWorld.GetColumn(2),
segmentToWorld.GetColumn(1));
            colliderObject.transform.localScale = new Vector3(width, 6.0f, 1.0f);

            // Add collider component
            BoxCollider collider = colliderObject.AddComponent<BoxCollider>();
        }
    }
}
```

The key is the "Path" property of the Racetrack object. This describes the racetrack path (essentially a spline) as defined by the **Racetrack Curve** objects, represented as an array of **Racetrack Segment** objects which represent small straight segments.

```
public class RacetrackSegment
{
    public Vector3 Position;
    public Vector3 PositionDelta;               // Added to position to get next
segment's position
    public Vector3 Direction;                   // Direction as Euler angles
    public Vector3 DirectionDelta;              // Added to Direction to get next
segment's direction (and used to lerp between them)
    public RacetrackWidening Widening;          // Left/right side widening
amount
    public RacetrackWidening WideningDelta;     // Added to Widening to get next
segment's widening (and used to lerp between them)
    public float Length;                        // Copy of
Racetrack.SegmentLength for convenience
    public RacetrackCurve Curve;                // Curve to which segment belongs
    ...
}
```

The path helper method "GetSegmentAndOffset" returns the segment at a specific distance along the racetrack. The segment's "GetSegmentToTrack" helper method returns an matrix that converts from "segment space" into "track space", which is the coordinate space of the main Racetrack object. Multiplying by the Racetrack object's localToWorldMatrix creates a matrix that converts from "segment space" into world space.

The segment also returns the "widening" values of the racetrack. I.e. how much wider (or narrower if negative) the racetrack is than the default width. In this case the sample assumes the default Racetrack Builder track templates are used, which are all 12 units wide by default.

Move an object along the track

Here's an example of moving an object along the racetrack.

This component is used by adding it to a Unity object and setting the Racetrack property to the corresponding racetrack.

```
using UnityEngine;

public class MoveAlongTrack : MonoBehaviour
{
    public Racetrack Racetrack;
    public float Speed = 10.0f;
    public float Distance = 0.0f;
    public bool Loop = true;

    private void FixedUpdate()
    {
        if (Racetrack == null) return;

        // Advance along racetrack
        Distance += Speed * Time.fixedDeltaTime;

        // Loop/clamp logic
        float length = Racetrack.Path.TotalLength;
        Distance = Loop
            ? Distance - Mathf.Floor(Distance / length) * length
            : Mathf.Clamp(Distance, 0, length);

        // Find segment for given z, and the Z offset from start of segment
        float segmentZOffset;
        RacetrackSegment segment = Racetrack.Path.GetSegmentAndOffset(Distance,
out segmentZOffset);

        // Get transformation matrix for segment
        Matrix4x4 segmentToTrack = segment.GetSegmentToTrack(segmentZOffset);
        Matrix4x4 segmentToWorld = Racetrack.transform.localToWorldMatrix *
segmentToTrack;

        // Position object on racetrack
        this.transform.position = segmentToWorld.MultiplyPoint(new Vector3(0.0f,
0.0f, segmentZOffset));
        this.transform.rotation =
Quaternion.LookRotation(segmentToWorld.GetColumn(2),
segmentToWorld.GetColumn(1));
    }
}
```

Essentially it uses the same logic as the CreateBoxColliders example.

Creating meshes at runtime

You can generate meshes at runtime by calling the Build method of the **Racetrack Builder** static class, and passing it the Racetrack.

For example:

```
using UnityEngine;
[RequireComponent(typeof(Racetrack))]
public class BuildTrackMeshesAtRuntime : MonoBehaviour
{
    void Start()
    {
        var racetrack = GetComponent<Racetrack>();
        RacetrackBuilder.Build(racetrack);
    }
}
```

This can be used to trim a few megabytes from your scenes, by not storing the actual meshes:

- Build your racetrack in the Unity editor
- Select the main Racetrack object, expand "Miscellaneous" and click "Delete meshes"
- Save your scene
- Call RacetrackBuilder.Build() at runtime.

Note: The Build method also has a "generateSecondaryUVs" parameter, which is ignored when the method is used at runtime.

The drawback is a slight pause when the scene loads at runtime. And obviously you cannot use baked lighting with this method.

Generating the track at runtime

You can also generate an entire track at runtime by constructing and configuring Racetrack and **Racetrack Curve** objects, then calling RacetrackBuilder.Build to generate the meshes.

This could be used for:

- Procedurally generated racetracks
- Implementing an in-game track editor

For example:

```
using UnityEngine;
public class GenerateTrackAtRuntime : MonoBehaviour
{
    // Drag template onto here
    public RacetrackMeshTemplate Template;

    void Start()
    {
        // Create the main racetrack object
        var trackObj = new GameObject();
        trackObj.name = "Track";
        var track = trackObj.AddComponent<Racetrack>();

        // Add curve objects
        AddCurve(track, 50.0f, 45.0f);
```

```
        AddCurve(track, 50.0f, -45.0f);
        AddCurve(track, 50.0f, 45.0f);
        AddCurve(track, 50.0f, -45.0f);
        AddCurve(track, 100.0f, 90.0f);
        AddCurve(track, 100.0f, -90.0f);
        track.CurvesModified();             // Track must be notified that the
curves have changed.

        // Build the meshes
        RacetrackBuilder.Build(track);
    }

    private void AddCurve(Racetrack track, float length, float yAngle)
    {
        var curveObj = new GameObject();
        curveObj.name = "Curve";
        curveObj.transform.parent = track.transform;
        var curve = curveObj.AddComponent<RacetrackCurve>();
        curve.Length = length;
        curve.Angles.y = yAngle;
        curve.Template = Template;
    }
}
```

It is vital to call Racetrack.CurvesModified after creating or modifying **Racetrack Curves** at runtime, so that the Racetrack object knows to invalidate its internal caches.

Generating the track from Racetrack section prefabs

An alternative method for generating a racetrack at runtime is to join together prefabs containing pre-made sections of racetrack.

This method is less flexible than using the Racetrack Builder track generation engine but much more lightweight, and should be able to handle on-the-fly track generation, e.g. for an "infinite runner" type of game.

It also does not require Racetrack Builder at all at runtime.

See the "Re-using meshes" section in this document for instructions on how to create racetrack section prefabs using the "Create copy for prefab" function. This function includes an "End marker" object for linking prefabs together seamlessly.

Here's an example component that takes an array of prefabs and strings them together in a random sequence at runtime:

```
using UnityEngine;

public class SectionTrackGenerator : MonoBehaviour
{
    public GameObject[] SectionPrefabs;
    public int Length = 10;
```

```
    // Start is called before the first frame update
    void Start()
    {
        GenerateTrack();
    }

    public void GenerateTrack()
    {
        if (SectionPrefabs.Length == 0)
            return;

        var currentPos = transform.localToWorldMatrix;
        for (int i = 0; i < Length; i++)
        {
            // Select a random prefab
            int index = Mathf.FloorToInt(Random.value * SectionPrefabs.Length);
            var prefab = SectionPrefabs[index];

            // Clone it
            var clone = GameObject.Instantiate(prefab);
            clone.transform.parent = transform;

            // Position it
            clone.transform.position = currentPos.GetColumn(3);
            clone.transform.rotation = currentPos.rotation;

            // Find end marker and set position for next section
            var endMarker = clone.transform.Find("End marker");
            currentPos = endMarker.localToWorldMatrix;
        }
    }
}
```