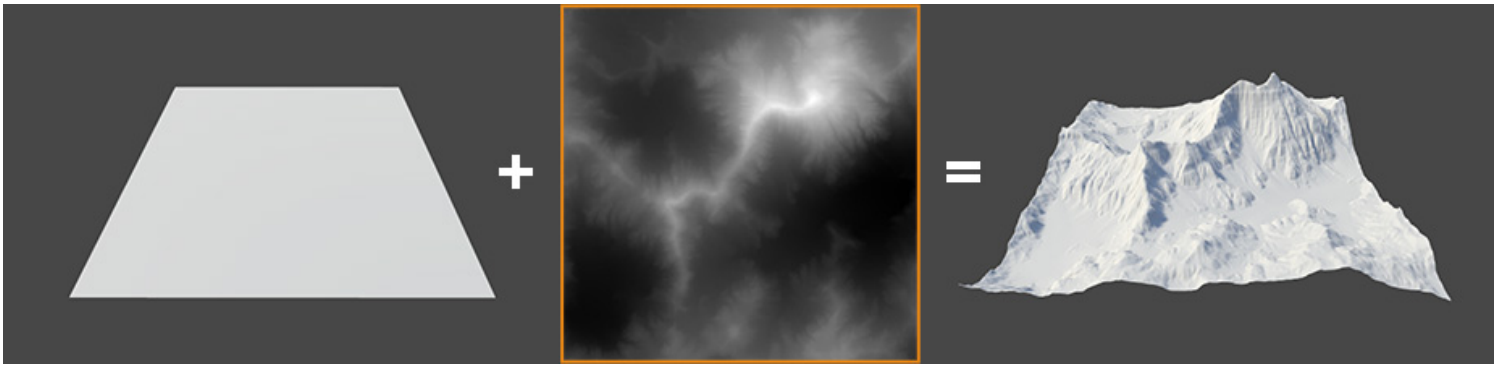
Curved World

Contents

What is Curved World and how it works

Curved World is a vertex transformation shader for creating various mesh bending effects.
It is like vertex displace shader, where texture is used to adjust vertex position.



Plane mesh | Displace texture | Vertex displace effect

Curved World does not use any textures and external resources, instead it calculates vertex transformation parametrically based on its world space position.

Common shader | Mesh bending (vertex displace) effect

Being a shader effect **Curved World** does not really modify mesh it renders. It means that if mesh is 'flat' before using **Curved World** shader - it still will be 'flat' after, just rendered differently. Because of this **Curved World** does not affect physics, animations, path finding or any other game features. If object needs to be moved from position A to position B along path C, after using **Curved World** everything will be the same, just rendered differently.

Installation

After importing Curved World asset into a project 3 ***.unitypackage*** files will be available inside **Amazing Assets -> Curved World -> Editor -> Installer** folder. Each one designed for one of the Unity's render pipeline.

1. **Built-in** - package for built-in render pipeline.
2. **Universal** - package for Universal scriptable render pipeline (URP).
3. **High Definition** - package for High Definition scriptable render pipeline (HDRP).

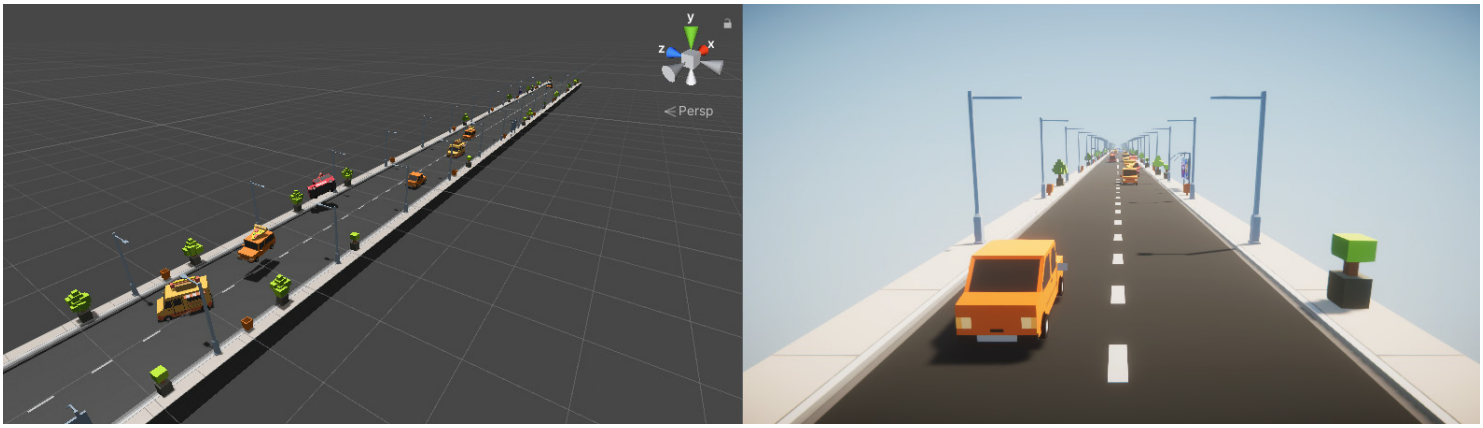
Based on project render pipeline import one of them.

Curved World minimum requirements are:

- Unity version 2019.4
- Universal RP 7.4.1
- High Definition RP 7.4.1

Quick start - Tutorial #1

Open Tutorial #1 - Introduction scene and run it.
It is a simple runner type scene. Objects are spawned at some distance from the player, moving along path (X axis in this case) and are destroyed behind the camera. Nothing related to the **Curved World**.

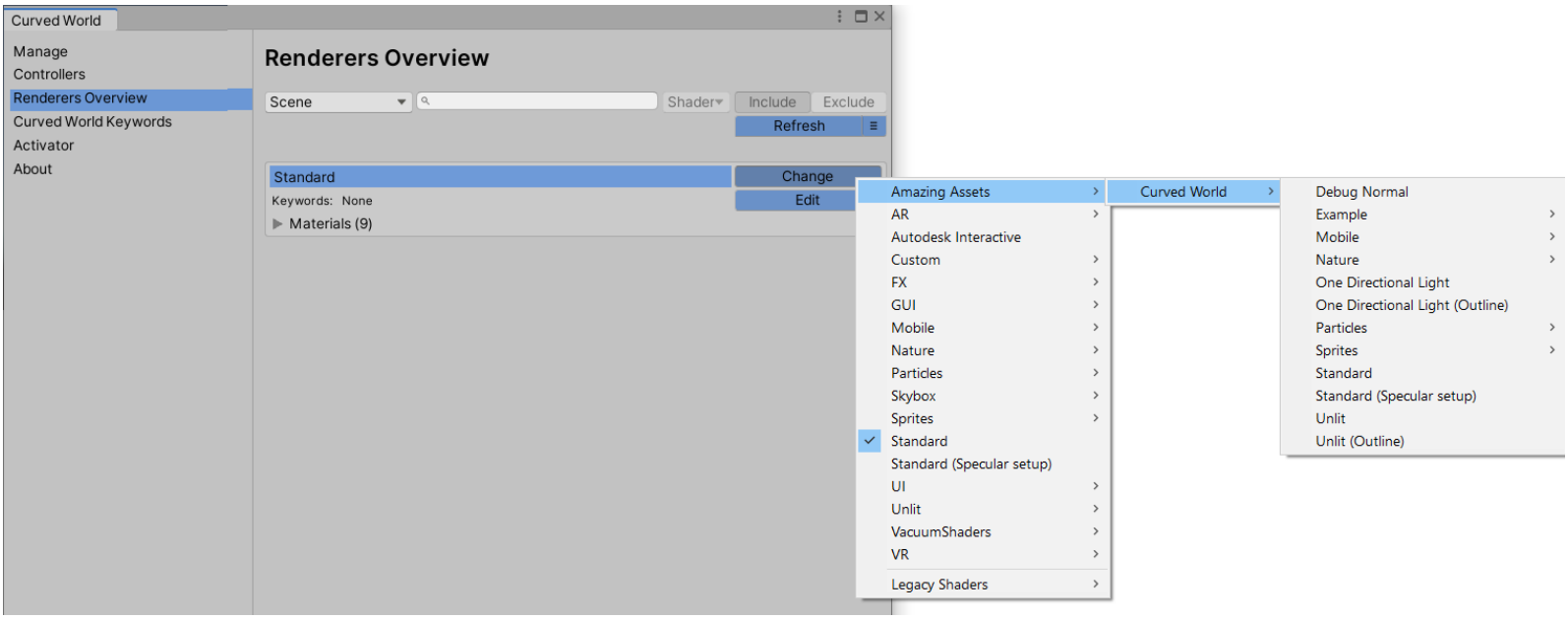


As bending effect created by Curved World is pure shader effect, meshes requiring such bending have to use shaders with integrated Curved World vertex transformation.
Currently all scene meshes are using Unity's default shaders. We have to change them.

Note, package includes all basic shaders with integrated Curved World. If those shaders are not enough, Curved World vertex transformation can be manually integrated into any shader. All required steps are described in chapter **Custom Shaders**.

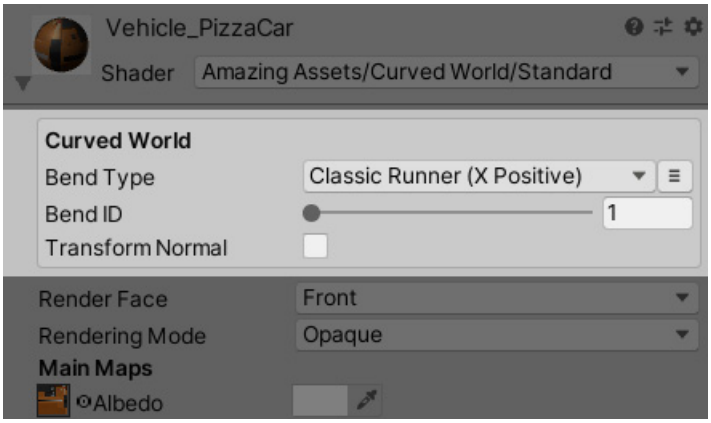
All mesh objects instantiated in this scene are in **Car Spawner** and **Chunk Spawner** game objects (find it in Hierarchy window). Go through each mesh there and change all materials shaders from default to the **Amazing Assets -> Curved World** shaders.

Or use editor tool from **Menu -> Window -> Amazing Assets -> Curved World**, go to the **Renderers Overview** tab. Here are displayed all scene materials and their shaders. Click on **Change** button and change shader from Unity to the **Amazing Assets -> Curved World** shader.

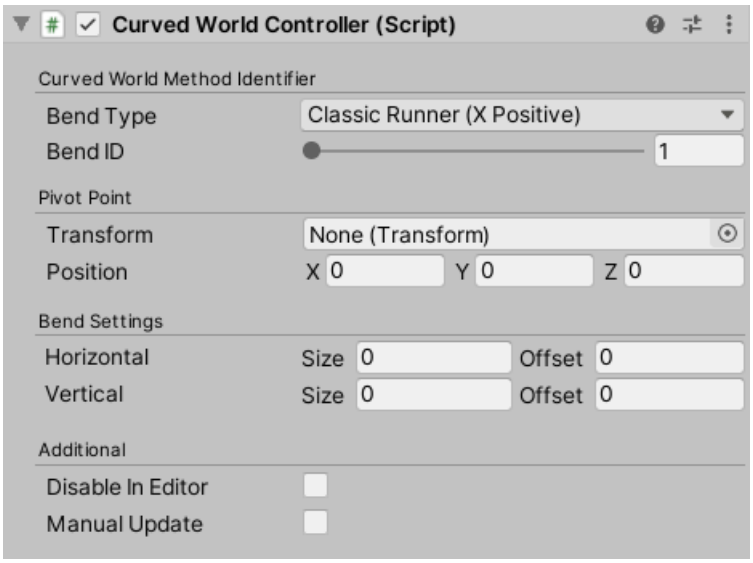


Note, Curved World editor window is explained in chapter **Editor Window**.

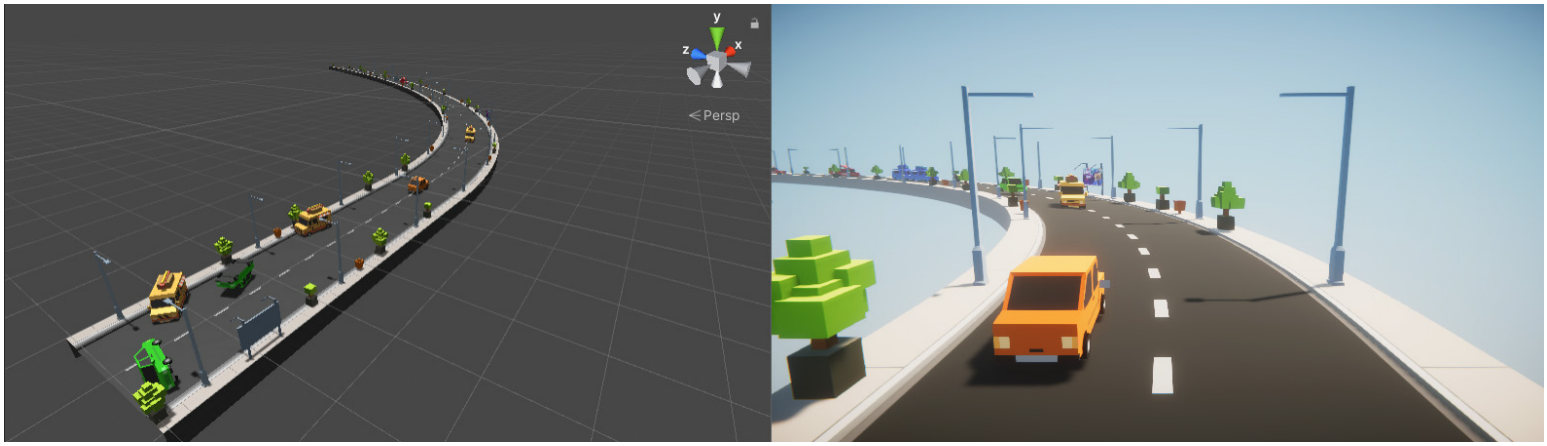
After changing shaders, make sure all materials are using **Classic Runner (X Positive)** bend type with **ID 1**.



Inside material editor Curved World shaders have options only to choose **Bend Type** with its **ID** and have no any properties for affecting bend settings itself. All bend settings are updated from script.
Select game object with name **Curved World Controller** and from the Components menu add **Amazing Assets -> Curved World -> Controller** script. Make sure it uses **Classic Runner (X Positive)** bend type with **ID 1**.



That's all. Everything is ready.
Enter game mode and change **Horizontal** and **Vertical** properties from controller script.



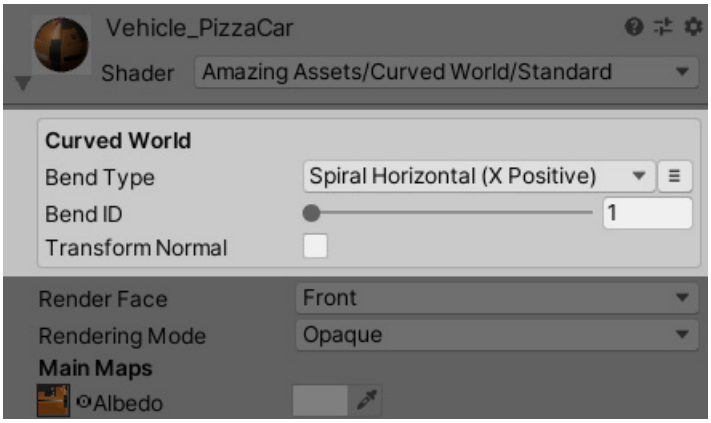
Note, bend settings can be update in editor and run-time using **Curved World Controller** script or any other custom script.
Curved World Controller script and its run-time API is explained in chapter [Package Scripts](#).

To summarize this simple tutorial. For enabling Curved World mesh bending effect all we did is:

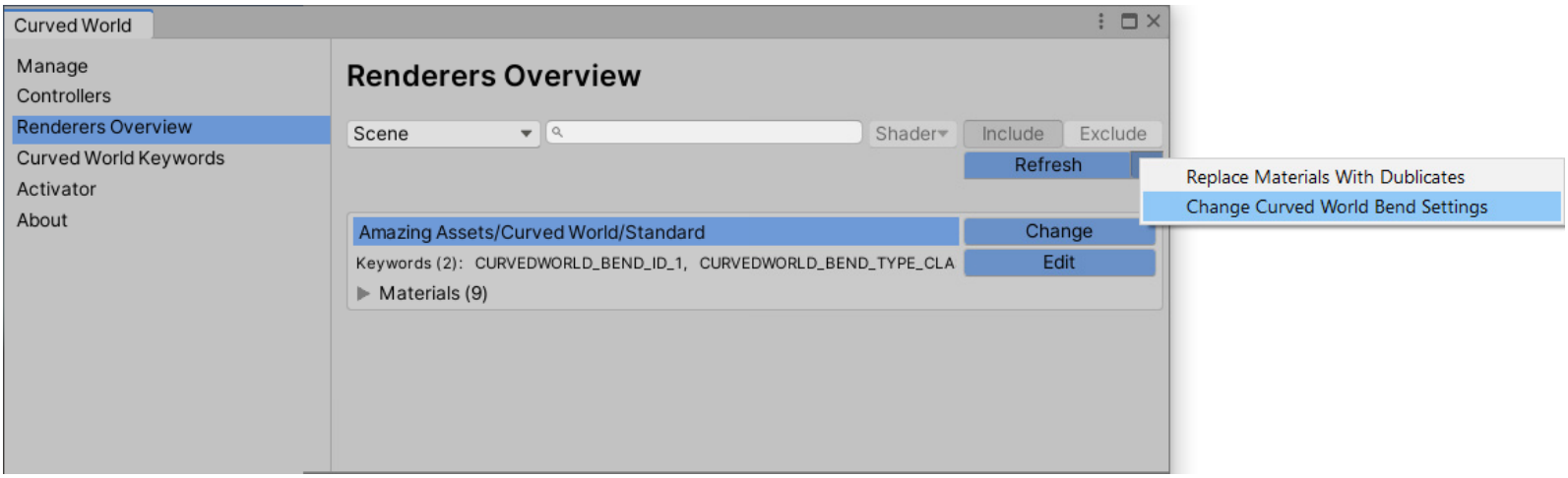
1. Changed mesh materials shaders from default to the Curved World shaders.
2. For materials chose desired **Bend Type** (**Classic Runner** in this case) and **Bend ID**.
3. Used script for updating bend settings.

That's how simple Curved World is. Nothing in the scene has changed: scripts, physics, animations, lighting and game design elements. Everything remains the same.

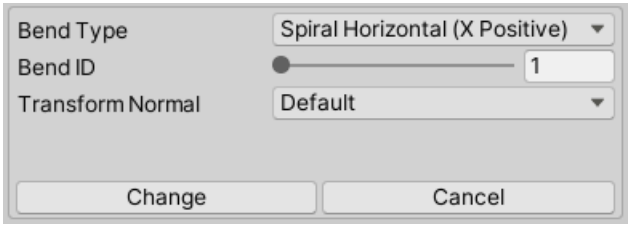
Let's try one more bend type and then explain Curved World key features in more details.
While Tutorial #1 - Introduction scene is still open, change all scene materials bend type from **Classic Runner** to **Spiral Horizontal (X Positive)**. Keep bend ID to 1.



Bend type can be changed directly from material editor or by using Curved World's editor window.
Inside **Renderers Overview** tab select menu ≡ button and choose **Change Curved World Bend Settings**.



Inside opened window choose **Spiral Horizontal (X Positive)** for the bend type and **1** for the bend ID.

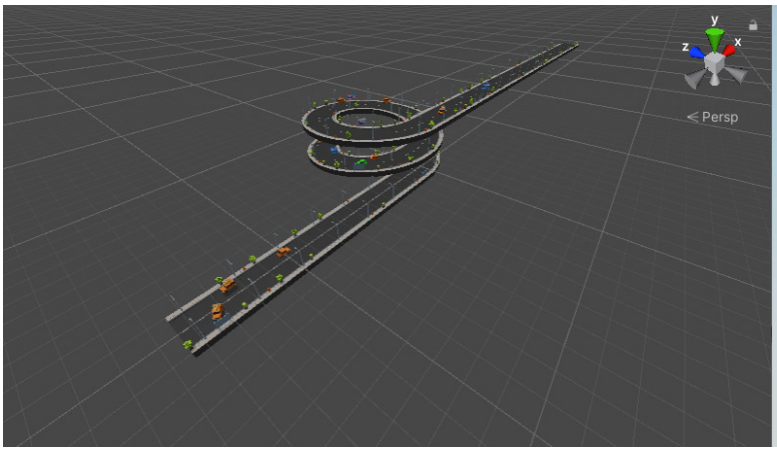


Click on **Change** button. Now all materials and shaders listed in the **Renderers Overview** tab (in our case all material used in this tutorial scene) will use new bend type and ID.

Note, changing bend settings from the **Renderers Overview** window affects only Curved World shaders that are included in the package or written by hand. Custom Curved World shaders created by graph tools or any other shaders are not affected and must be modified manually.
Creating custom Curved World shaders are described in chapter **Custom Shaders**.

After changing bend types for the materials it is time to update bend settings from script.
As scene already contains **Curved World Controller** script, select it and change its bend type to the **Spiral Horizontal (X Positive)** with ID 1 accordingly.
For **Rotation Center** select game object with the same name in the scene. Set **Angle** to 660 and **Minimal Radius** to 10.

That's all. Everything is ready.
Enter game mode and change **Angle** and **Minimal Radius** properties from controller script, move **Rotation Center** game object.



Before continue reading this file try **Spiral Vertical (X Positive)** bend type manually.

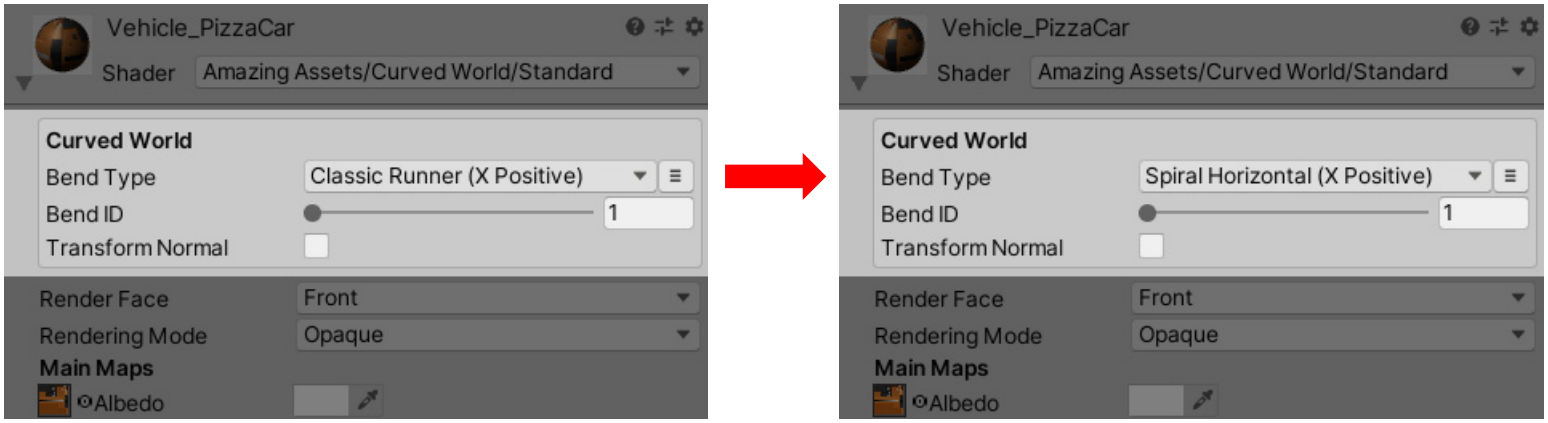
In the next tutorial we will create **Little Planet** bending effect and scene with multiple bend types, but before that let's check some key features of the Curved World.

How it really works

Curved World is a collection of vertex transformation methods (mathematical functions).
Each method is unique (by its name, ID and used parameters) and is described in its own **cginc** file.

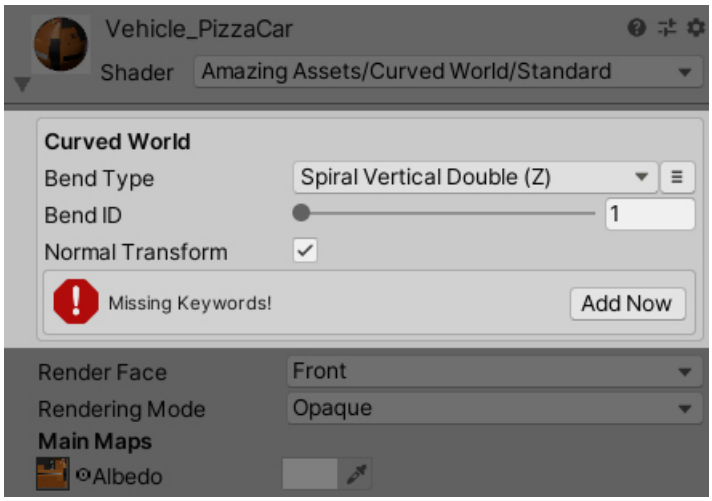
Those methods are used in shaders that in the result render mesh bending effects.

Shader can contain multiple vertex transformation methods. They can be mixed or used separately and enabled/disabled using keywords at any time from material editors or run-time scripts. For example in the previous chapter [Quick Start - Tutorial #1] we have initially used material with **Classic Runner** bend type and later changed it to the **Spiral Horizontal**.



In this case material editor just disabled keyword for **Classic Runner** bend type and enabled it for **Spiral Horizontal**. After that shader begins using different function for calculating bending effect.

By default Curved World package does not come with all bend types installed and shaders do not include all of them. They are added by user request. For example, if in the above material for the bend type choose **Spiral Vertical Double (Z)** material editor will display warning message, as the bend type with such name and ID is not installed.



By clicking on the **Add Now** button, all required **cginc** files will be generated and new vertex transformation method will be added to this shader.

Note, generating Curved World **cginc** files and automatically adding their methods to the shader directly from material editor is possible only for shaders that are included in the package or written by hand. Custom Curved World shaders created by graph tools or any other shaders must manage this process manually.

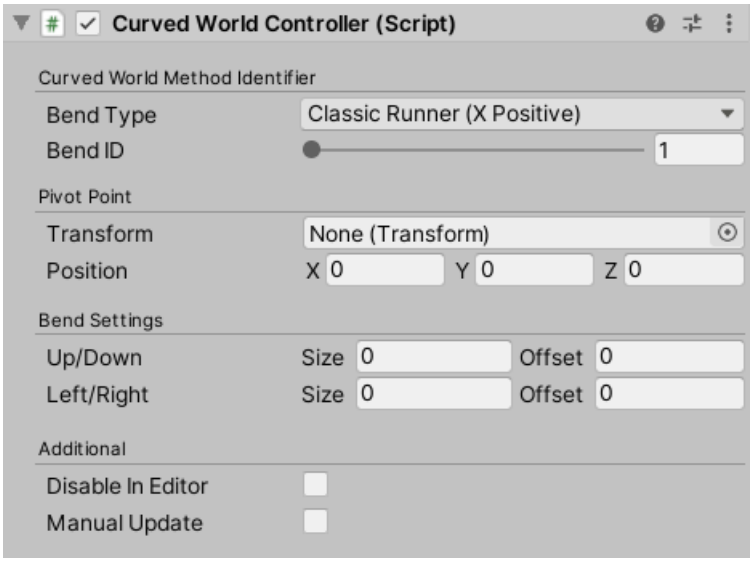
Note, Curved World **cginc** files can be generated using Curved World editor window too, described in chapter **Editor Window**.

Amount of Curved World transformation methods included in one shader is limited, as each one needs its own keyword and Unity has limit of 64 for local keywords.
In one shader can be used any combination of 32 bend methods. Other 32 keywords are left for Unity use.

Note, custom Curved World shaders created by graph tools or written by hand, may be keywords free.

Curved World vertex transformation method properties (bend settings) are always updated from script, not from the material editor. Bending properties are described in **cginc** files as global parameters and in Unity updating global shader properties is possible only from script.
Advantage of using global shader properties is that they are updated once and all shaders and materials automatically receive them, without the need to modify all of them manually.

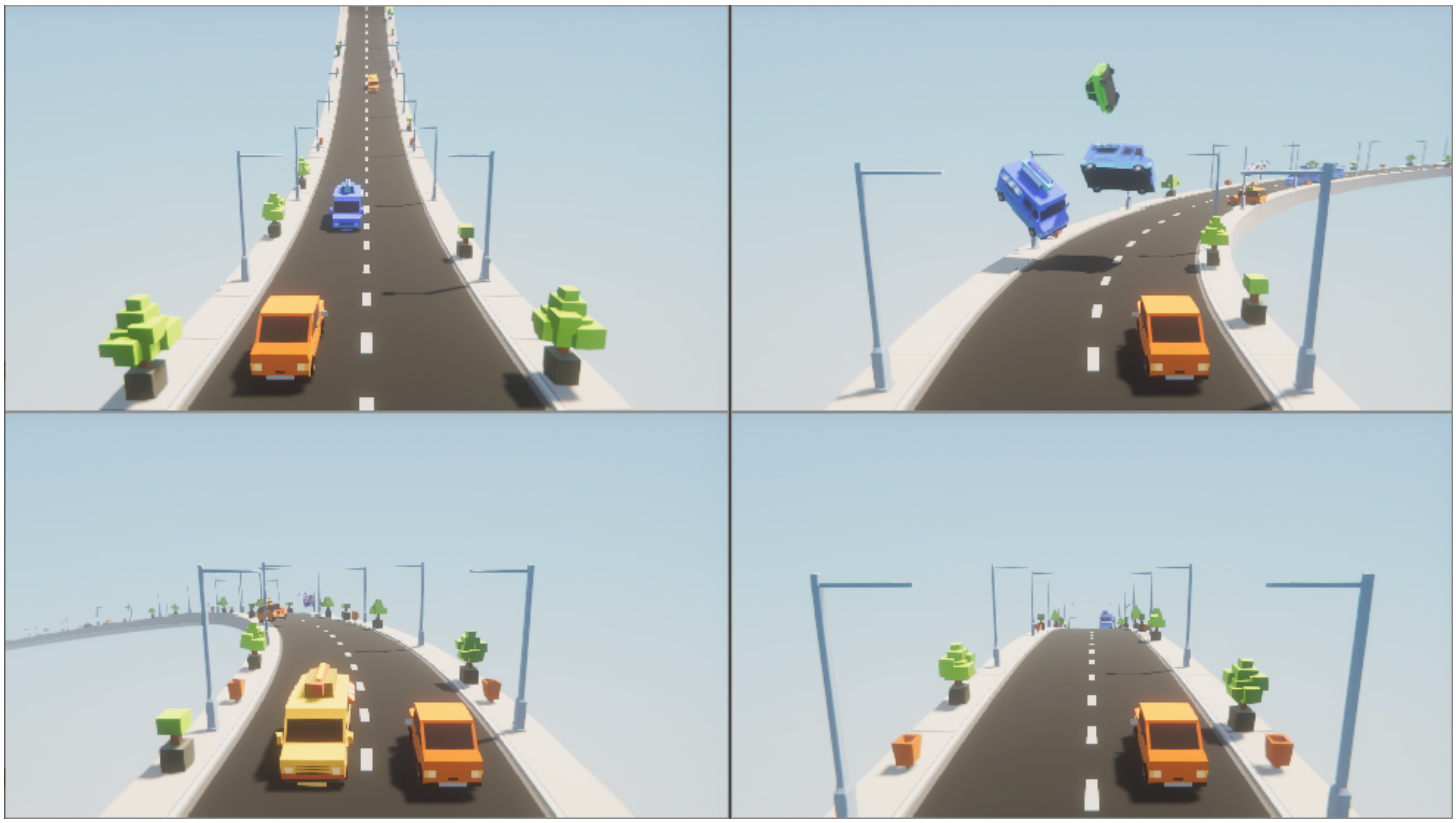
Package includes **Curved World Controller** script for updating bend settings based on **Bend Type** and **Bend ID**. Those two properties are the only available properties inside material editor too.



For the **Curved World Controller** script to correctly update bend settings, **Bend Type** and **Bend ID** inside material and scrip must be the same.

Note, **Curved World Controller** script is described in chapter **Package Scripts**.

Bend Type and **Bend ID** are unique identifiers for the Curved World transformation method.
Bend Type is the real name of the bend algorithm.
Bend ID is used for creating **Bend Type** variation. For example if scene needs several bending effects of the same type, but each one with separate bend settings, like on the image bellow:



This is one scene with 4 **Classic Runner** bend types using simultaneously on different set of meshes, and each bending effect is updated and controlled separately from each other. All materials in this scene are using just one shader containing 4 transformation methods:

- CurvedWorld_ClassicRunner_X_Positive_**ID1**
- CurvedWorld_ClassicRunner_X_Positive_**ID2**
- CurvedWorld_ClassicRunner_X_Positive_**ID3**
- CurvedWorld_ClassicRunner_X_Positive_**ID4**

Each of those methods are unique and while they all create the same **Classic Runner** bend types, their parameters are different, updated separately from each other and as a result create different bending effects.

We will create similar scene in the chapter **Tutorial #3 - Multiple Bends**.

Bend Types

Curved World offers 11 bending effects divided into two groups by used bending equation:

1. Classic Runner
2. Little Planet
3. Cylindrical Tower
4. Cylindrical Rolloff

Use parabola equation

5. Spiral Horizontal
6. Spiral Horizontal Double
7. Spiral Horizontal Rolloff
8. Spiral Vertical
9. Spiral Vertical Double
10. Spiral Vertical Rolloff

Use spiral equation

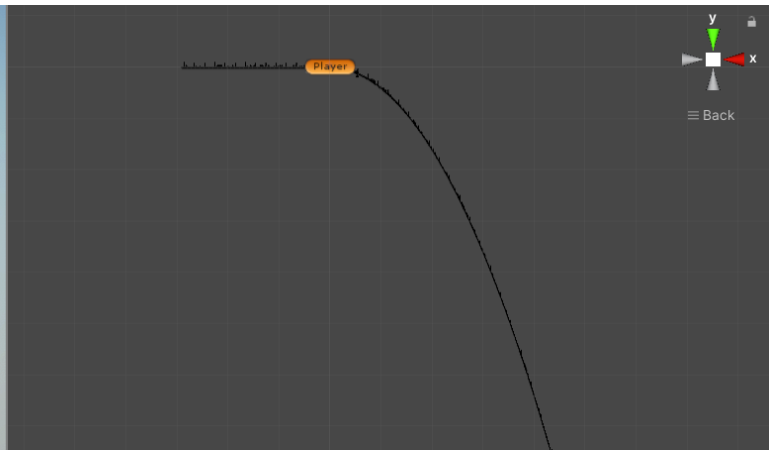
11. Twisted Spiral

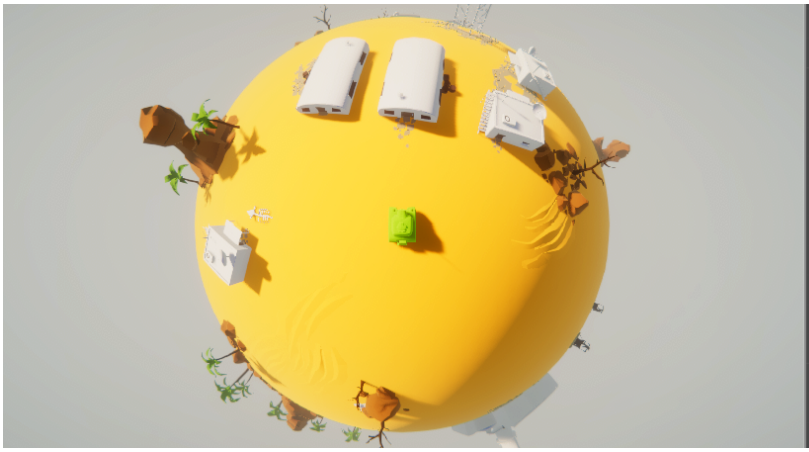
Uses both spiral and parabola equations

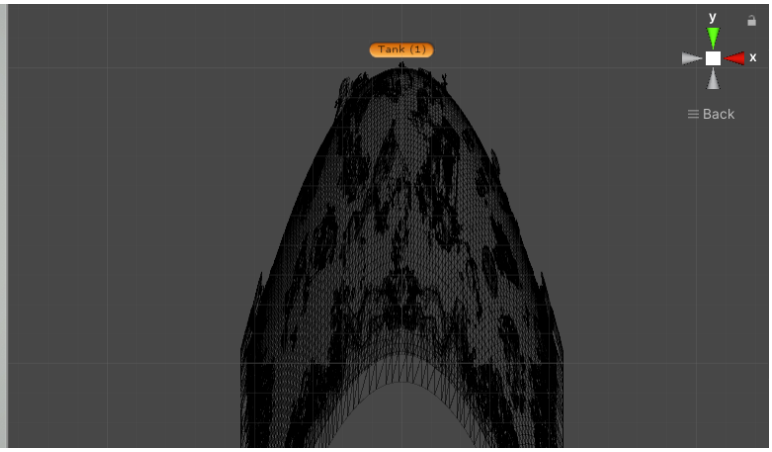
Main difference between those two groups is that **Parabola** equation is faster and uses less instructions count, but this algorithm '*pushes*' vertices in the bend direction and they gain more distortion further they are from a pivot point. This limits camera placement in a scene.

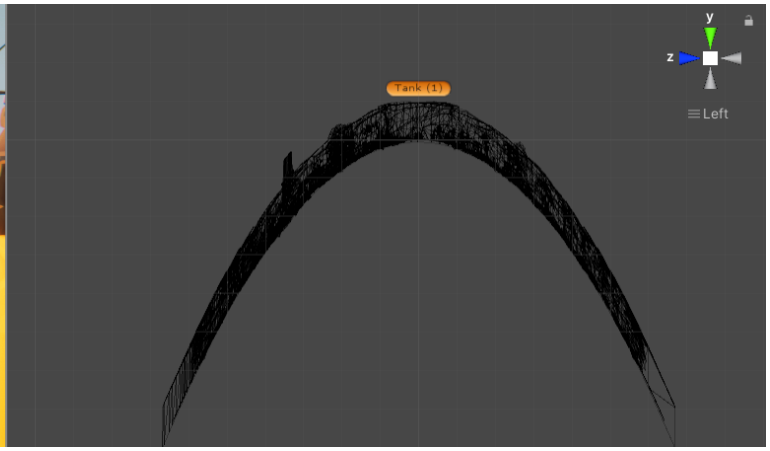
With those bending effects camera can be in a First Person, Third Person or Top Down positions. But observing scene from a big distance or making 360° orbital rotation is not a good idea.

Images below display **Classic Runner**, **Little Planet** and **Cylindrical Rolloff** bend types from '*correct*' position and side view.

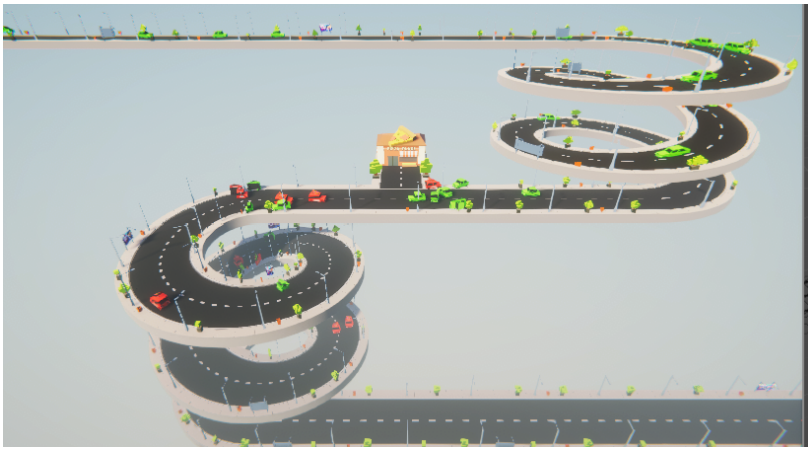


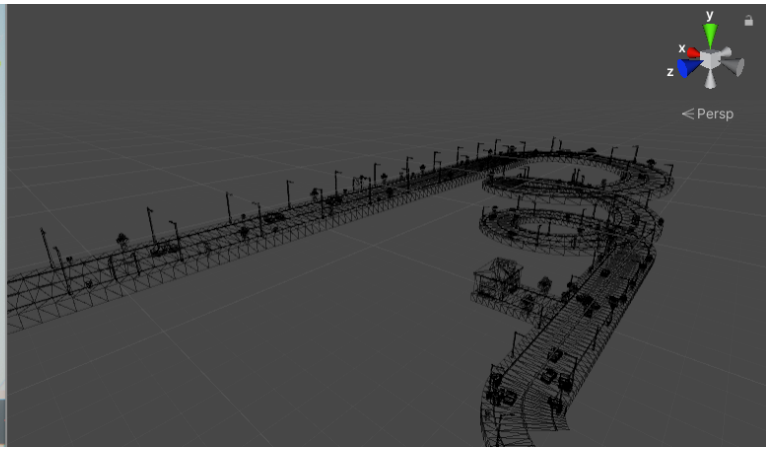


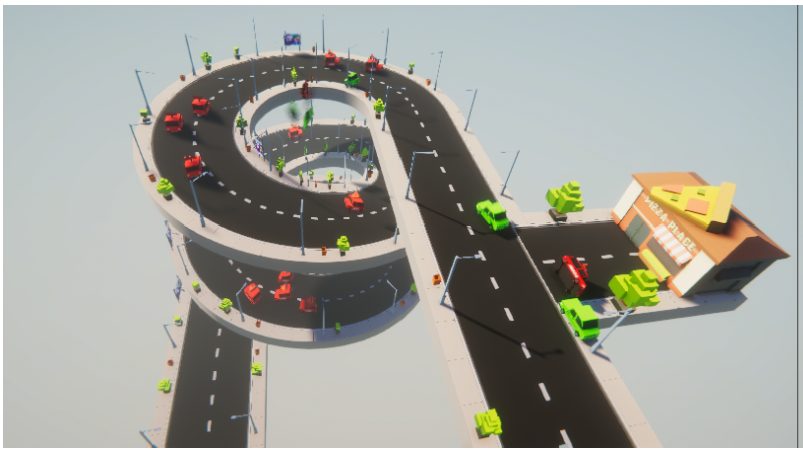


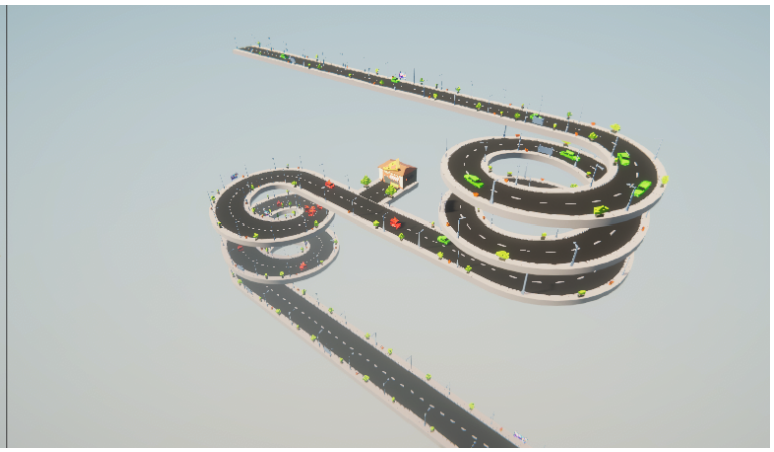


Meshes using **Spiral** bend equation can be observed from any point and camera can be placed anywhere in a scene.



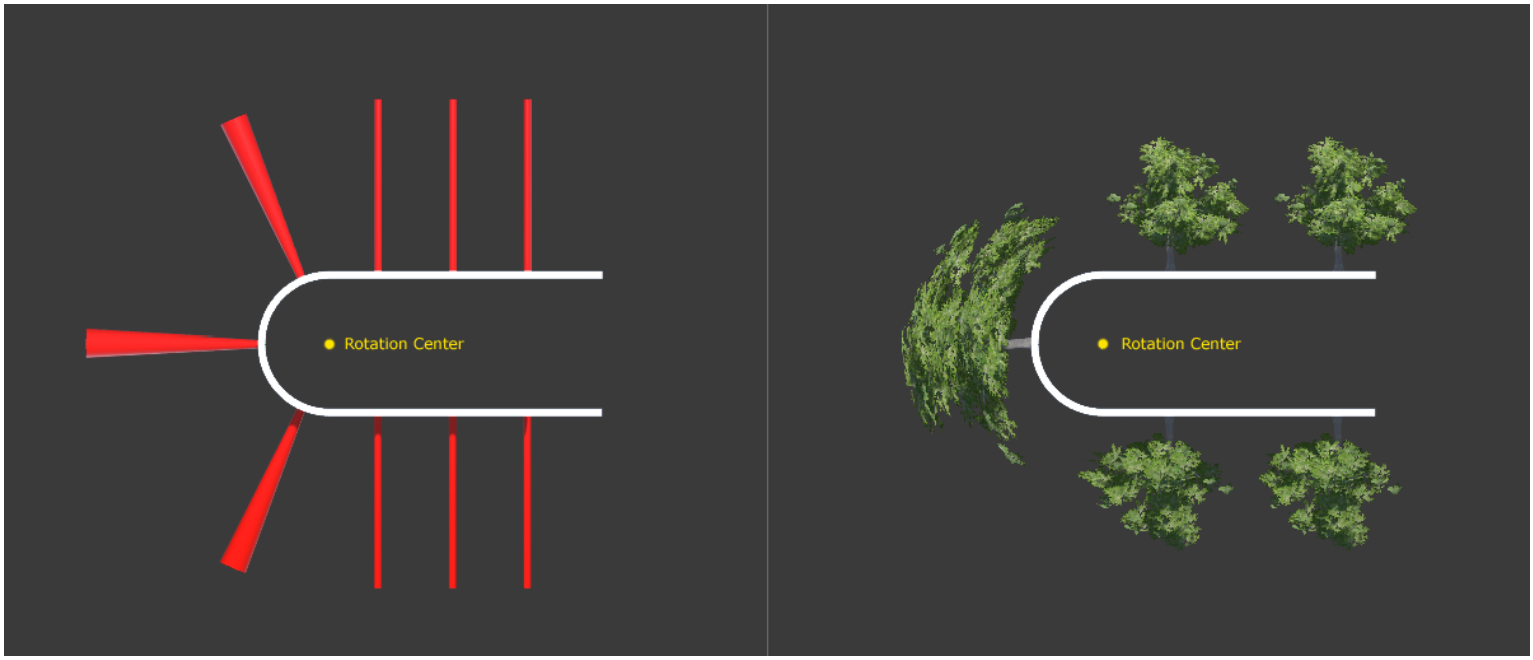




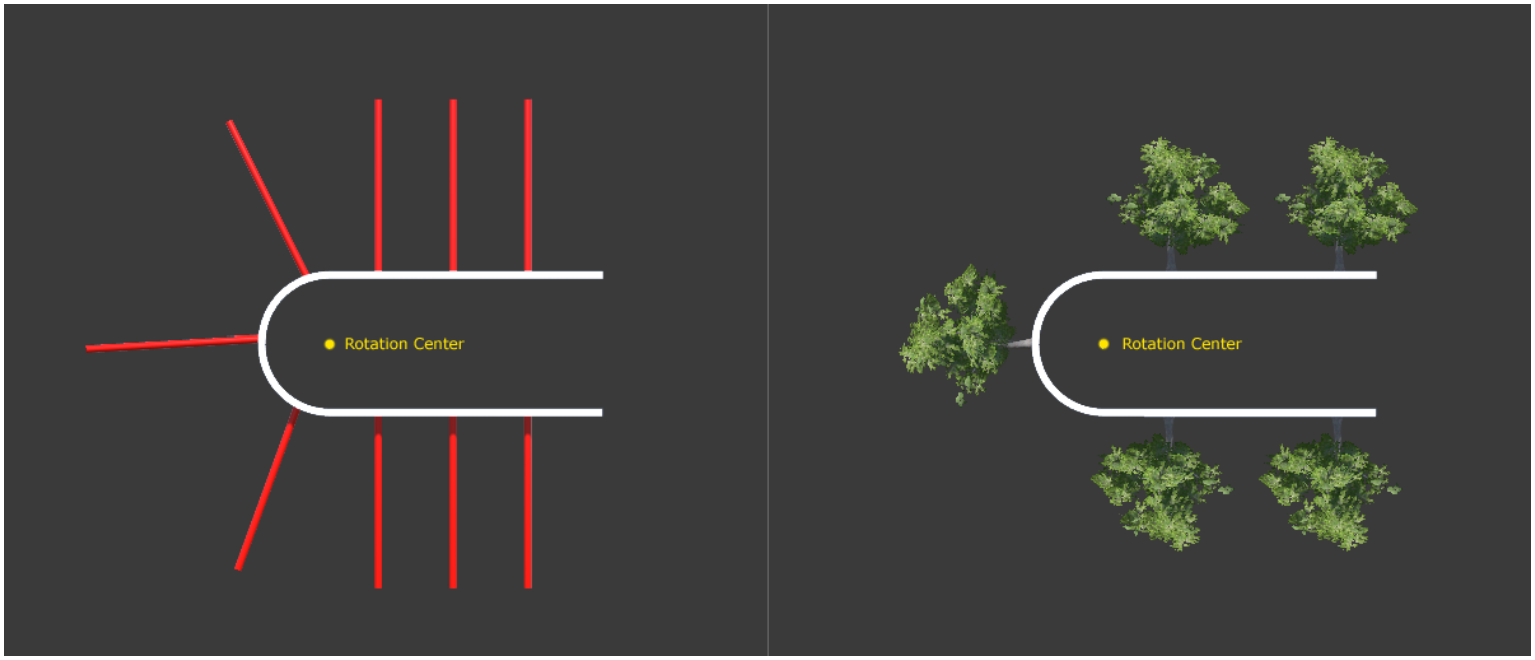


Negative part of the **Spiral** algorithm is that it uses more instruction than **Parabola** and it does not work well with tall objects, when curvature arc is small.

Image below demonstrates this problem. Vertices are stretched along rotation arc and distortion increase proportionally by the distance from rotation center to the vertex.



However this problem can be fixed by calculating object position using script (not shader), as described in Non Shader Bending chapter and demonstrated in Non - Shader Bending example scenes.



Curved World bend effect name consist of two parts: 1) Bend Type and 2) Bend Axis.
For example **Classic Runner (X Positive)** – means that **Classic Runner** type bending effect will affect only those vertices that are distributed along **X** (world **X**) axis and are in **Positive** side of the pivot point.

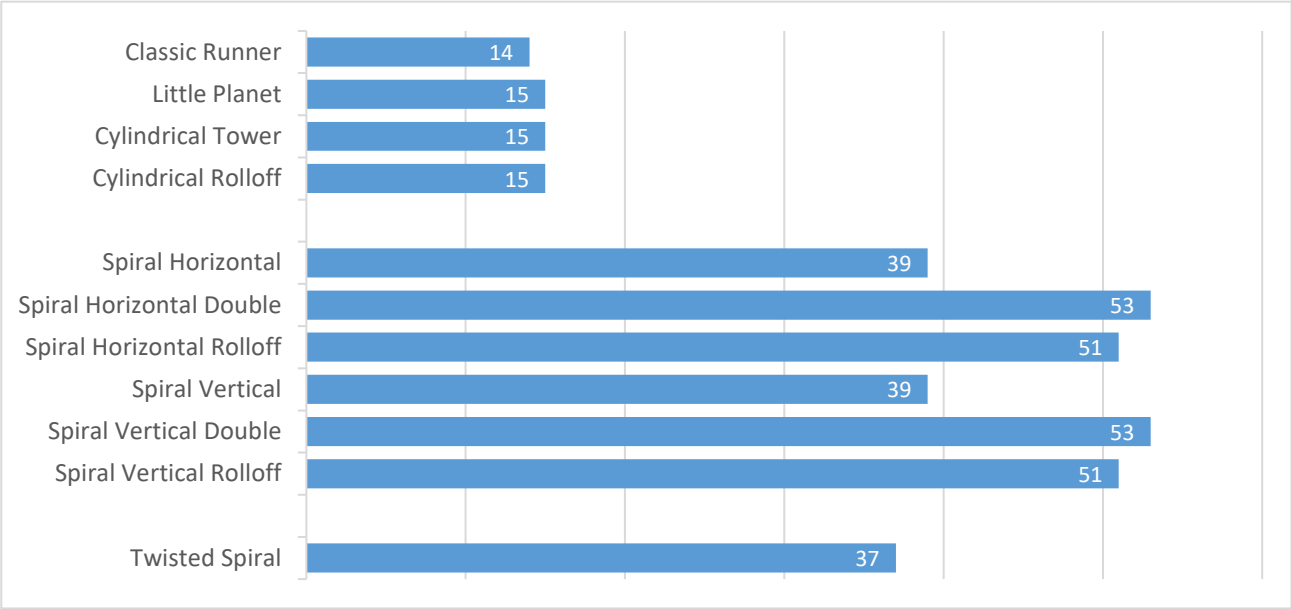
As we did in **Quick start – Tutorial #1** chapter. This scene initially was oriented along X axis and that is why we have used **Classic Runner** and **Spiral Horizontal** bend types working in **X Positive** axis direction. Using **Z** axis (Positive or Negative) will be incorrect and would not gave the desired bending effect.

Choosing bend axis depends only game design.

Some bend types have no option for choosing Positive/Negative sides of an axis, for example, **Cylindrical Rolloff (X)** or **Spiral Horizontal Double (Z)**, etc. This means that bending effect works along those axis on both sides of the pivot point.

Instruction count

Chart below displays shader instruction count for each bend type calculating vertex transformation only. Instruction count increases about three times if shader calculates normal transformation too.

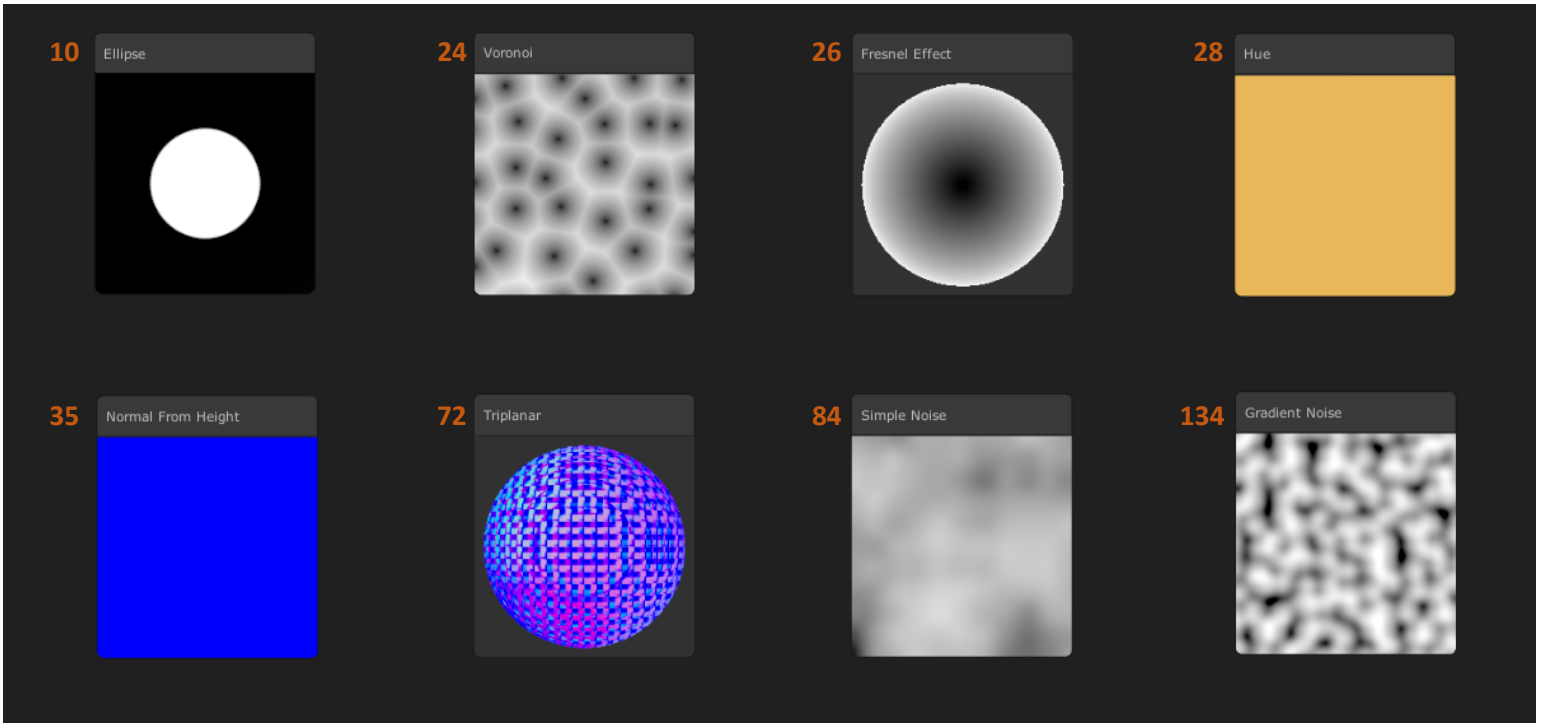


Instruction count is calculated using D3D compiler and may be different on other platforms.

Based on the provide data, is Curved World expensive and heavy effect? – No it is not.

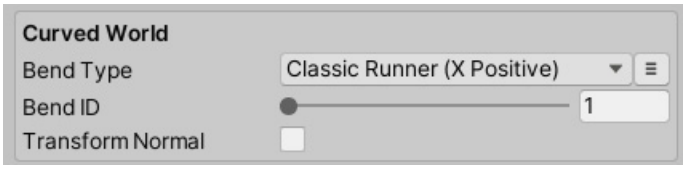
Note, Curved World vertex transformation is calculated per-vertex, not per-pixel.

For comparison below are some of the Unity Shader Graph nodes and there instructions count cost.

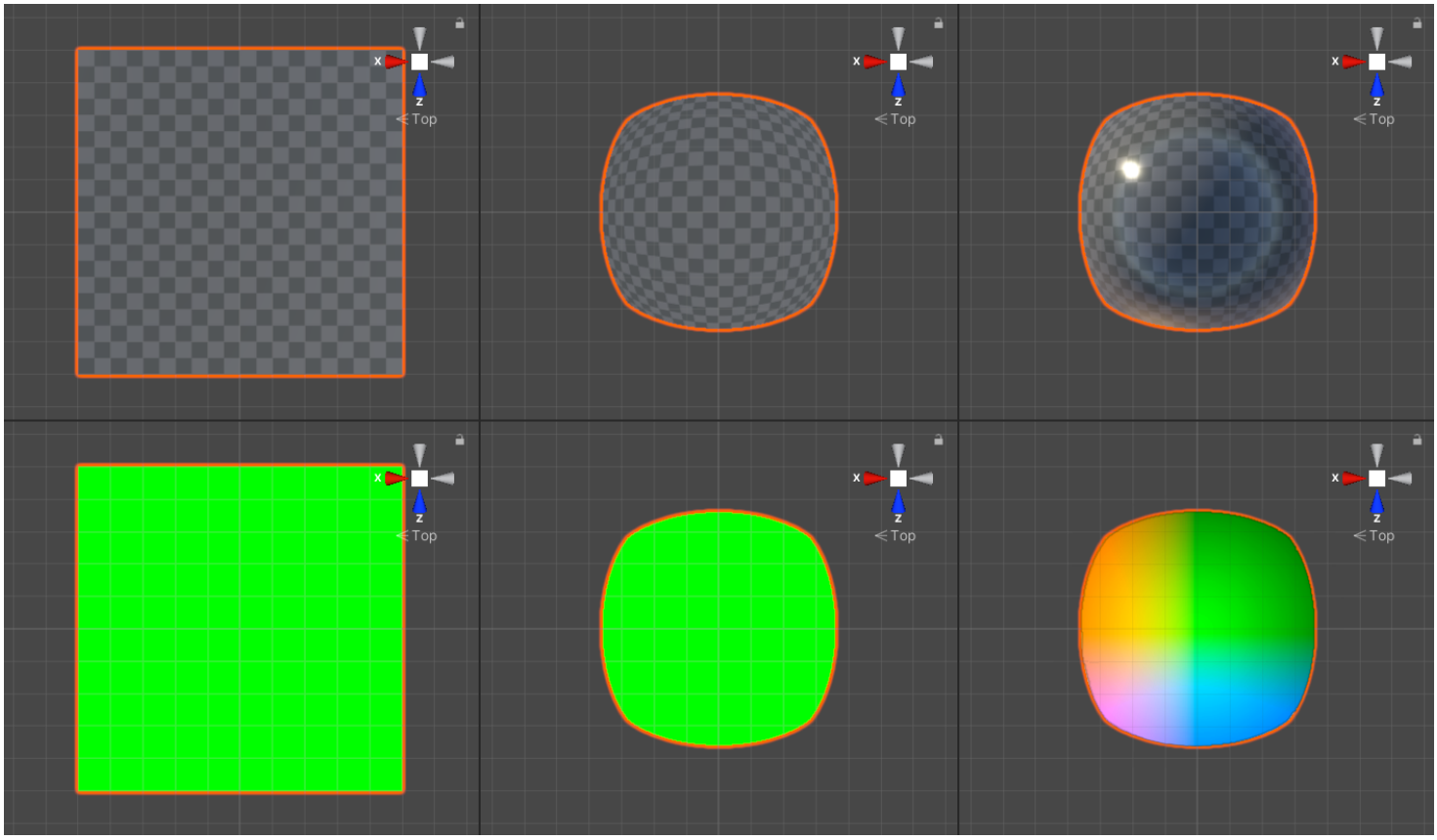


Normal transformation

For creating bending effect Curved World transforms mesh vertices only. By enabling **Transform Normal** check box inside material editor, shader will begin recalculating vertex ***normal*** vector too and it will follow bending curvature.



On the image below, left column displays plane mesh from the top view and its normal vector (0, 1, 0). Middle column displays the same mesh with Curved World bending effect. Normal vector is not affected. Right image displays the same Curved World bending effect with enabled normal recalculation.



Normal recalculation is not necessary, but it can add more realism and details to the bent meshes using lit or reflected shaders.
If enabled, normal vector will be recalculated in the same transformation method as the vertex.

Note, vertex and normal transformations are calculated per-vertex in vertex shader pass, not pixel or fragment passes.

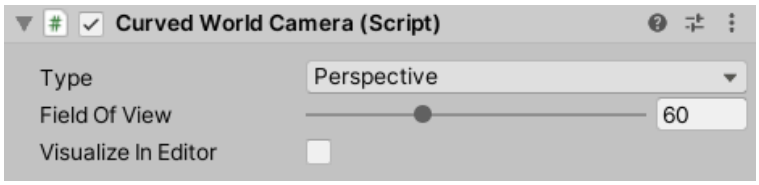
Note, normal recalculation requires vertex ***tangent*** too. If mesh has no tangents then new normal will not be correct.

Mesh disappearing and early culling

For vertex displace shaders it is a very common problem when at some camera view angles mesh 'suddenly' disappears. This happens when original mesh goes beyond camera's field of view and is excluded from rendering pipeline. In this case shader has nothing to render.

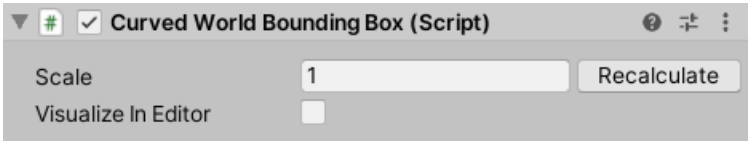
Curved World offers two solutions to fix this problem:

1. **Curved World Camera** script – Overrides camera's culling matrix before it begins frame rendering. Allowing to capture objects outside its field view.



   Script calculates culling matrix using **Perspective** or **Orthographic** algorithms. Near and far clipping planes are taken from the source Camera component.
   Script must be attached to the Camera object.

2. **Curved World Bounding Box** script – Scales individual renderer's (MeshRenderer and SkinnedMeshRenderer) bounding box component and makes it visible to the Camera, even if a mesh is outside of field of view.



   Script is also necessary if mesh is not visible to the dynamic light source (it has its own field of view), to avoid excluding mesh from shadow receiving/casting pass.
   Script must be attached to the object with MeshRenderer or SkinnedMeshRenderer component.

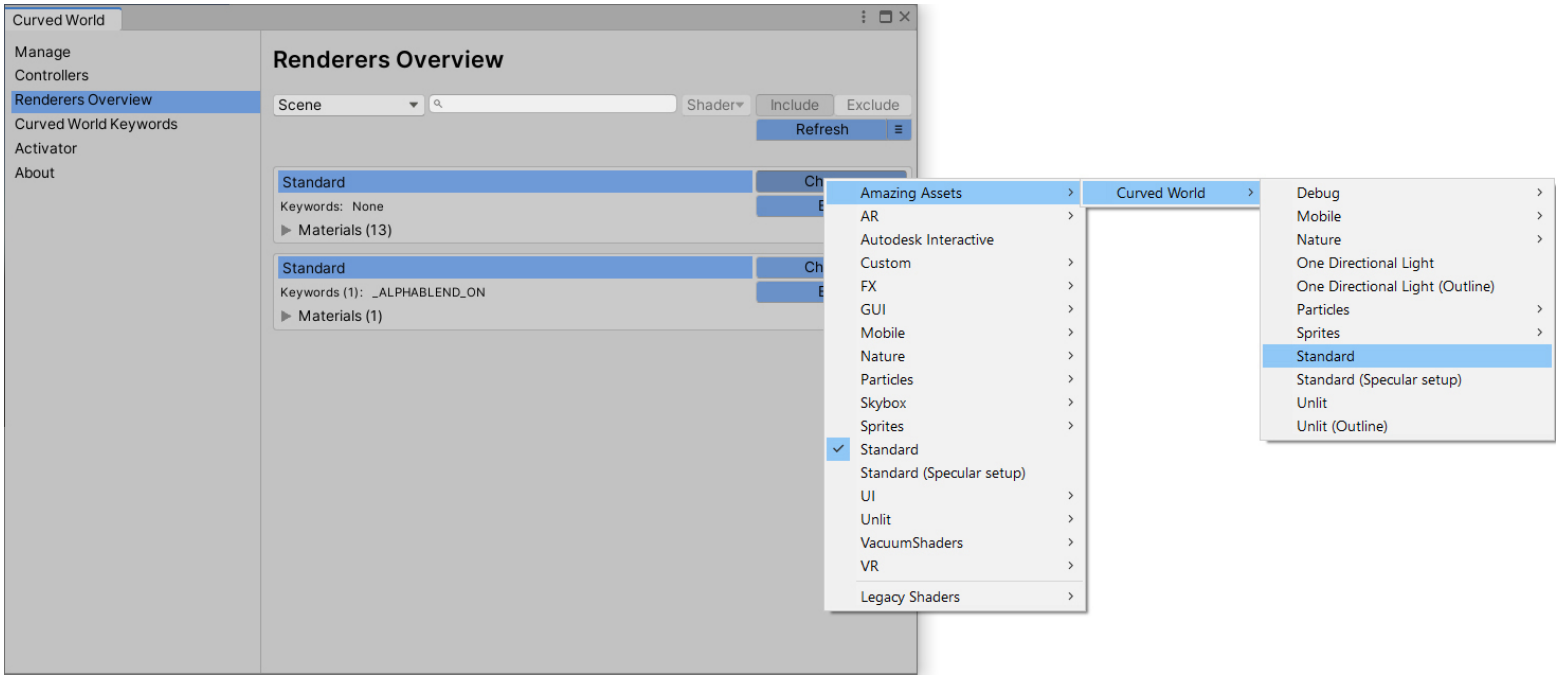
Note, it is very easy to find out why mesh with Curved World shader is not rendered or not receiving/casting shadows – just disable bending effect. Is mesh rendered now? If no, then it means that is outside of a camera view and will not be rendered regardless of a used shader and one of the above solutions must be used to bring it back to the render pipeline.

Tutorial #2 – Little Planet

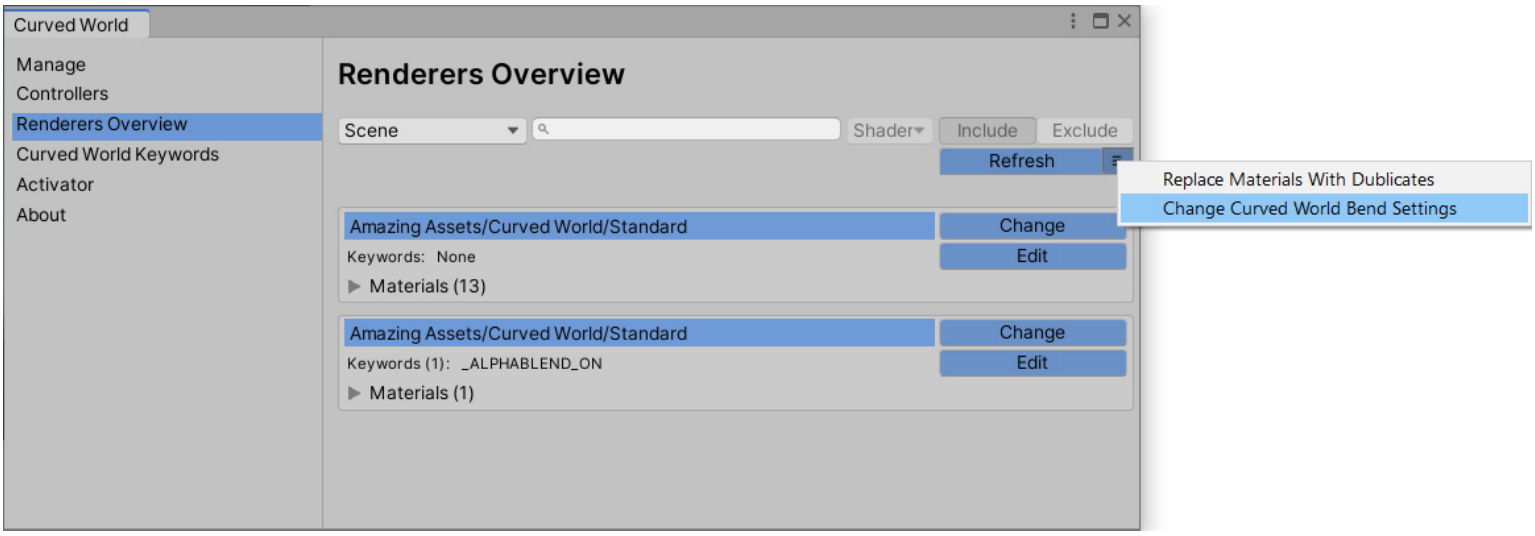
Open Tutorial #2 - Little Planet (part 1) scene. This is Unity's Tanks! Tutorial asset from the Asset Store. Enter Play mode. Use **WASD** keys for tank navigation and **Spacebar** for shooting. Currently this scene does not use any Curved World transformations. We will fix it.

As in the previous tutorials, to enable mesh bending effect all we need to do is - use shaders with Curved World vertex transformation and update bend settings from script.

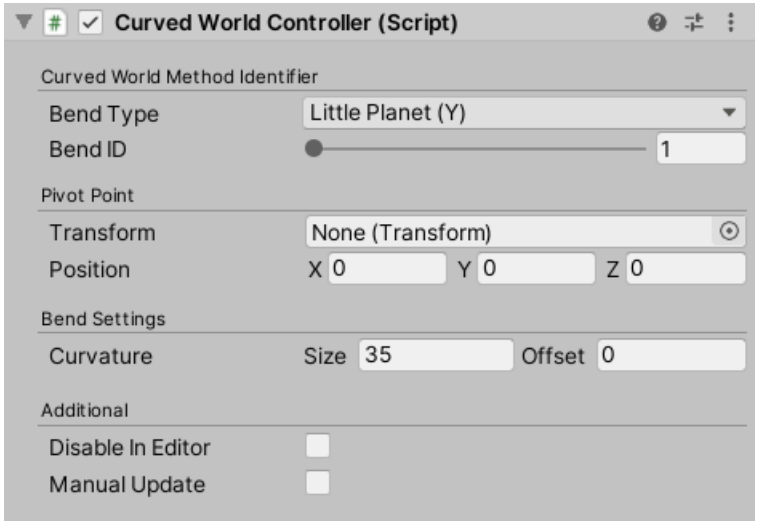
Open Curved World editor window change scene materials shaders from default to **Amazing Assets -> Curved World** shaders.



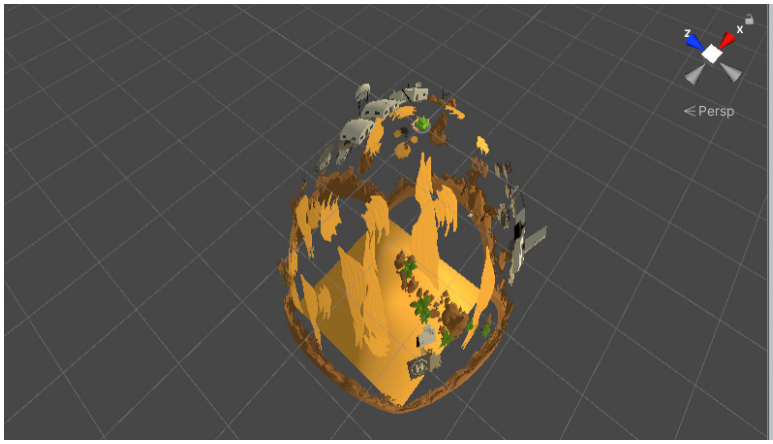
After that, change bending settings for all scene materials with **Little Plane (Y)** for bend type and **1** for **Bend ID**.



When all scene materials are set up to use Curved World shaders with **Little Planet (Y)** bend type, create **Curved World Controller** script with the same bend settings.

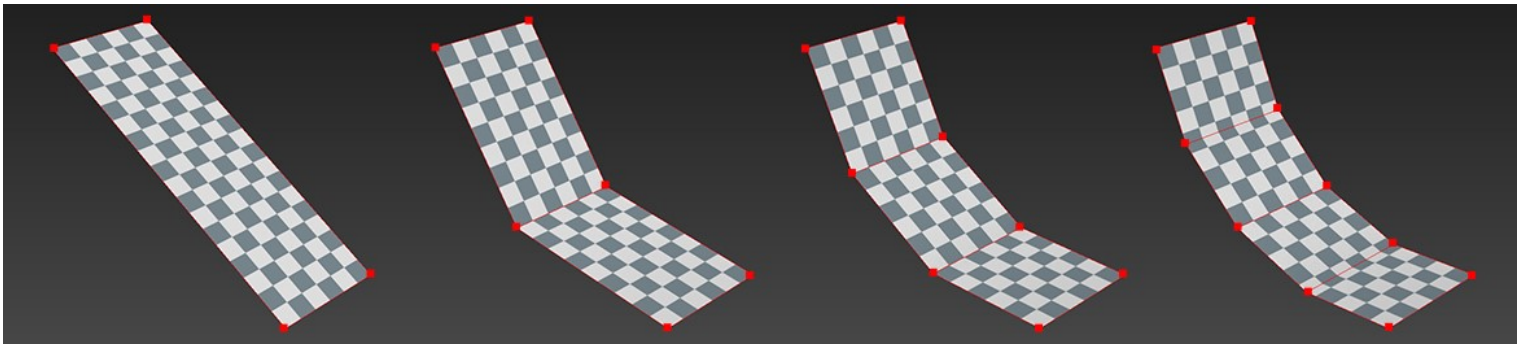


Now if try changing **Curvature** parameter inside controller script, scene will begin transforming but ground plane mesh will not be correctly affected.
And it is because, ground plane mesh has not enough vertices for creating smooth curvature.

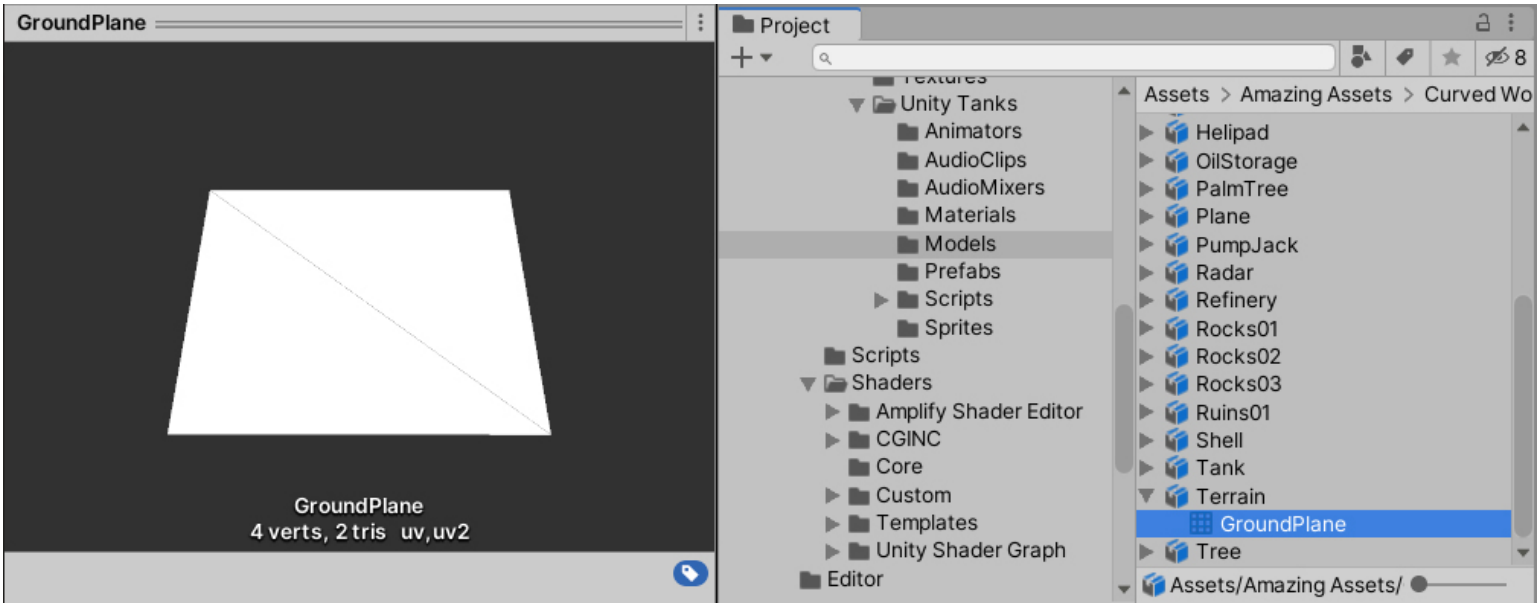


As Curved World bending effect is calculated per-vertex, mesh needs to have enough vertices for the bending effect to be smooth.

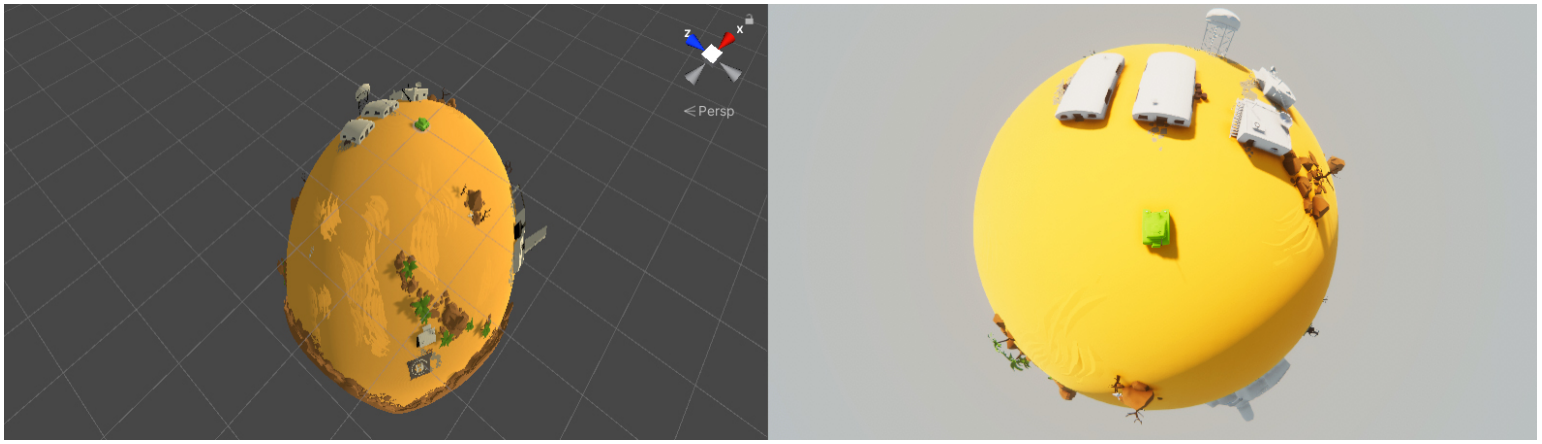
Image below demonstrates how vertex count affects curvature smoothness.



Current mesh used by the ground plane has just 4 vertices. Such meshes will never be bent.



Inside Hierarchy window select **GroundPlane** game object and change its mesh to the **Plane 20x20** mesh. Now scene looks much better.



Enter game mode and try to move the player tank.
Now, far tank is moved from scene's center, more distorted meshes are becoming and an illusion of a **Little Planet** is destroying. It is easily fixable.

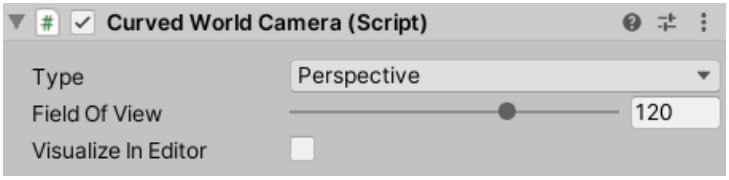
Each Curved World bending effect has a **Pivot Point**. It is the origin for the bending algorithm. It is a point from where curvature begins.

Pivot Point has to be update from **Curved World Controller** script. It can be any dynamic or static object.

Inside **Curved World Controller** script assign tank game object as the **Pivot Point** and enter game mode. Now everything is smooth, scene is bent around tank correctly and player is always in the center of a spherical world.

But if increase **Curvature** parameter inside controller script some meshes will begin disappearing and popping in and out. As described in chapter **Mesh disappearing and early culling**, that's because meshes are excluded from rendering pipeline and become invisible for the Curved World. It is easily fixable too.

Select **Main Camera** and from the Components menu add **Curved World Camera** script.
Set size of **Field of View** to 120 with **Perspective** type.



Now everything should be rendered correctly without any issues.

To summarize key features explained in this tutorial (beside enabling bending effect using Curved World shaders and updating them from script):

- For correct bending effect mesh needs to have enough vertices along curvature path.
- Providing correct **Pivot Point**, as it is a starting point for the curvature.
- When using Curved World shaders some meshes can disappear unexpectedly, creating mesh popping in and out effect. It is because mesh is excluded from rendering pipeline long before Curved world shader begins its rending. However this can be easily fixed by bringing those meshes back into rendering pipeline by using **Curved World Camera** script.

In the next tutorial we will add one more player into this Little Planet scene.

Note, we have missed to change shader for the tank bullet mesh (Shell). As it is instantiated only in run time and was not visible for **Renderers Overview** window in editor mode.
Pause game when shooting a bullet, go to the **Renderers Overview** window and change its material shader to Curved World and set proper bend type and ID for it.

Tutorial #3 – Little Planet (part 2)

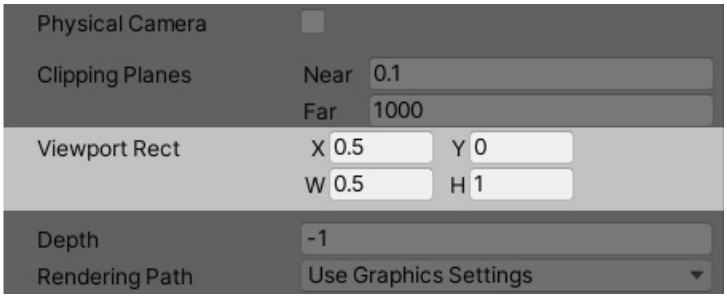
Note, this example scene does not work with **High Definition** render pipeline (**HDRP**), as there is a bug in Unity Engine related to the script used in this example.

In this tutorial we will continue working on the same Little Planet scene created in the previous chapter and add one more player.
For now disable **Curved World Controller** script to see scene without bending effect.

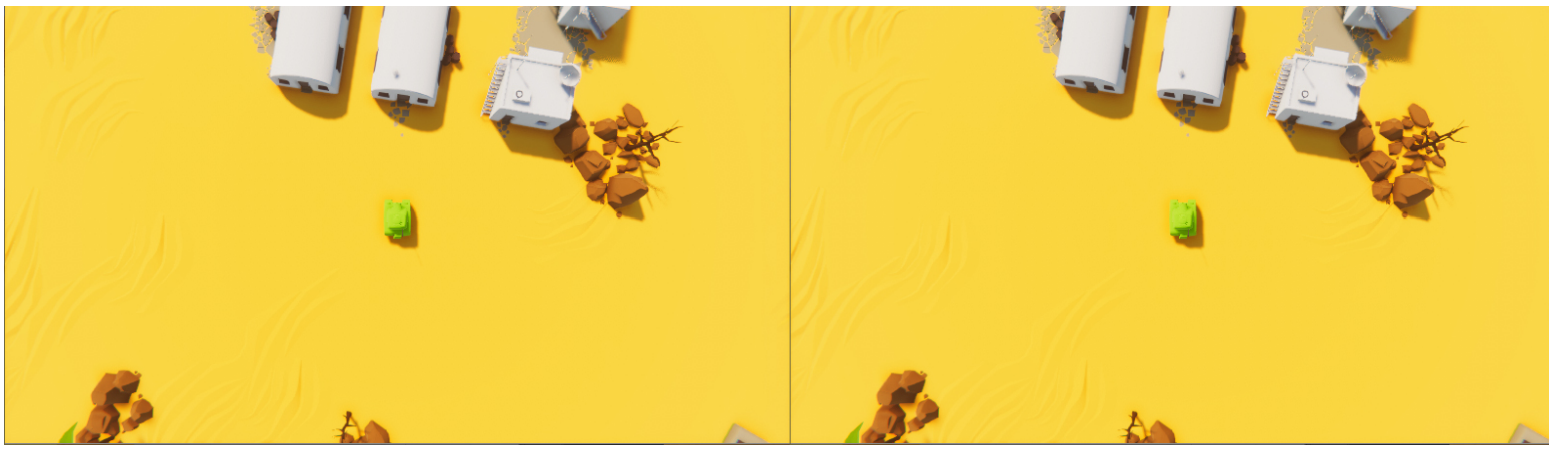
Duplicate Tank and Camera game objects.

For the new camera remove **Audio Listener** script (Unity will through warnings in Console window if there are more than one such scripts in a scene) and in attached **Camera Follow** script change target to be the new duplicated tank. For the new tank in **Tank Movement** script change Player ID to 2.

For each camera change **Viewport Rect** properties as on the image bellow.

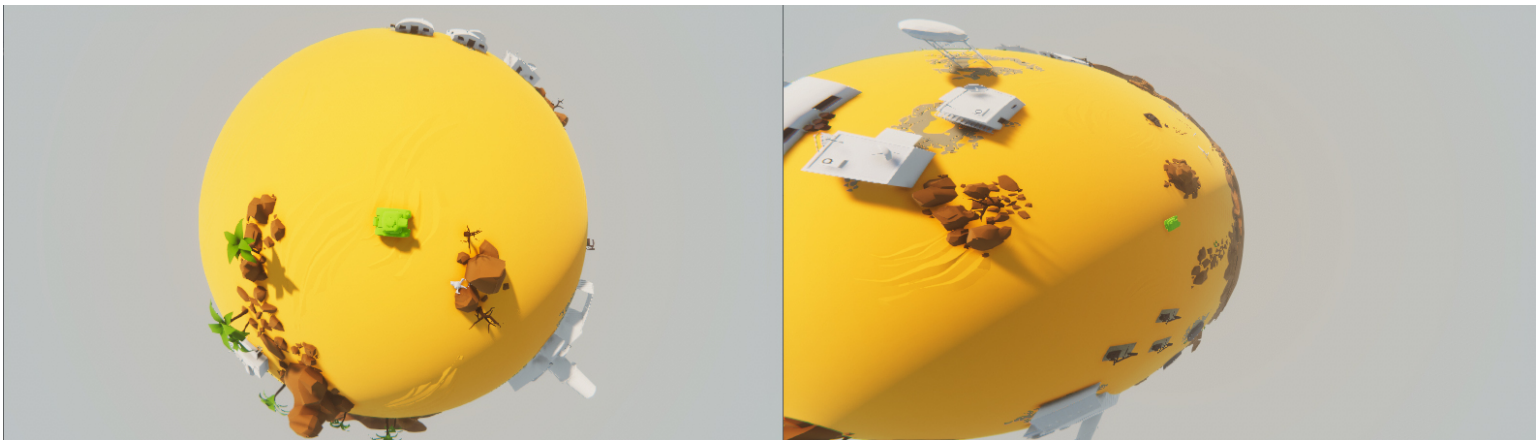


Now we have two tank players (Tank (1) and Tank(2)) and two cameras (Camera(1) and Camera(2)) following each players.
While **Curved World Controller** script still is disabled, enter game mode to test this scene (for the second tank, moving keys are Left/Right/Up/Down arrows, and right control key for shooting).



It is time to enable **Curved World Controller** script and enter Play mode.

If Tank(1) is assigned as a **Pivot Point** in the controller script, only for this player bend effect will be correct and completely distorted for the second player. If **Pivot Point** is not assigned at all, both camera renderings will be incorrect.
We need both players to be rendered with **Little Planet**!



It is easily achievable. We have to tell each camera to update bend settings independently from each other.

Note, scene materials still are using one shader, with one bend type and ID. We will change nothing here.

Duplicate game object with **Curved World Controller** script.
For **Pivot Point** for the first controller assign Tank(1) and for the second script – Tank(2).
For both scripts enable **Manual Update** check box.

Attach **Manual Controller Update** script to the cameras and assign **Curved World Controller** scripts - Controller (1) to Camera(1) and Controller (2) to Camera(2) accordingly.

Now enter Play mode.
Each player should be in the center of its **Little Planet** while moving and Cameras following accordingly.

If examine **Manual Controller Update** script, all it does is calling update function for the **Curved World Controller** before camera begins rending a frame.

By default **Curved World Controller** script automatically updates bend settings in its own MonoBehaviour.Update() function. But as we have two different cameras and need each one to be rendered with its own bend settings, we disabled controller scripts default Update (by enabling **Manual Update** check box) and call Update function manually for each camera from our custom script just before they begin rendering their frames. In this case even all scene materials are using one bend type with the same ID, in each frame bending effect is rendered with different settings.

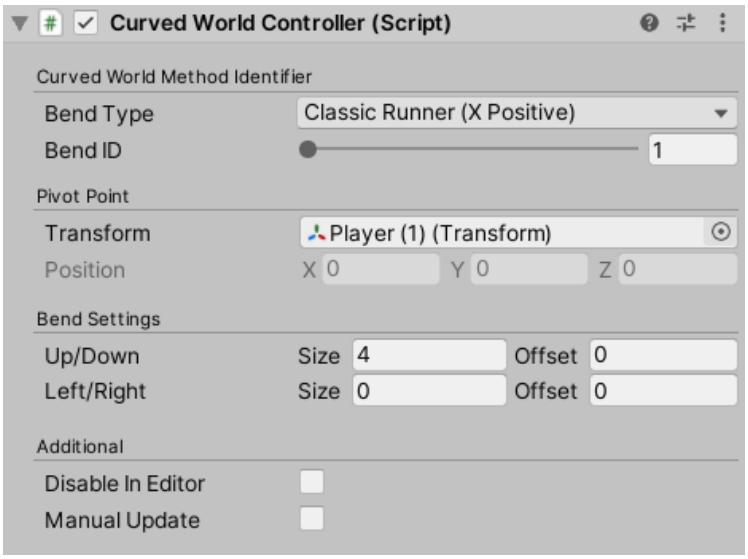
In the next tutorial we will create scene with multiple various bending effects.

Tutorial #4 – Multiple Bends

In the previous chapter we have created scene with one bend type **Little Planet**, and rendered it from different cameras with different settings. Goal of this tutorial is to create scene with multiple bend types using simultaneously.

Open Tutorial #4 – Multiple Bends scene and run it. It is exactly the same scene we have worked with in the **Quick start - Tutorial #1** chapter when creating runner type game.

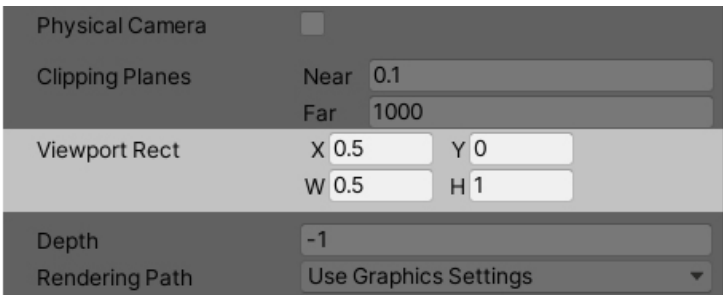
Make sure all scene materials are using Curved World shaders with **Classic Runner (X Positive)** bend type with **ID 1**, and **Curved World Controller** scrip uses Player (1) as Pivot Point.



Duplicate game objects:

- Camera (1)
- Player (1)
- Car Spawner (1)
- Chunk Spawner (1)

Move duplicates away on 1000 units along Z axis. Just to make sure that old and duplicated objects are not interacted with each other and not visible for other cameras.
Set **Viewport Rect** for each camera as on the image below.



Now we have two players visible in the Game view, each one render with its camera.

Currently all scene materials are using **Classic Runner (X Positive)** bend type with **ID 1** and changing bend settings from controller script will affect both players. But we need different bend types for each one.

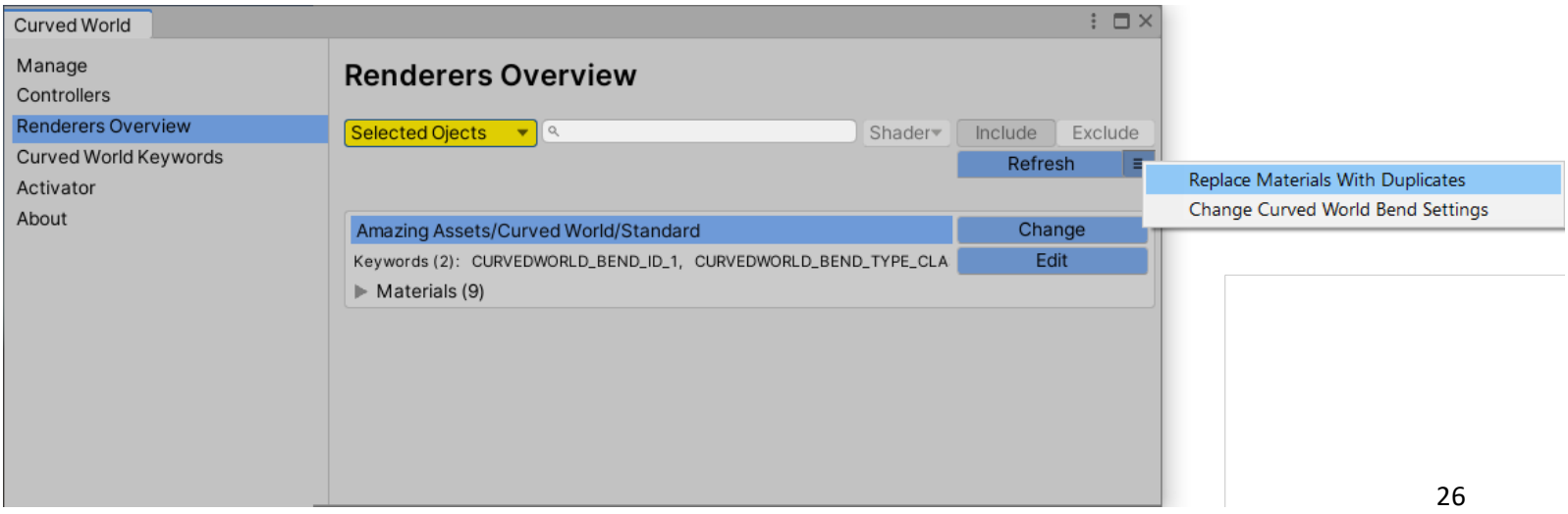
For the second player's ***world*** to have different bending effect – materials there have to use different bend method.
Here is a list of steps we need to do:

1. Go through all meshes instantiated for the second player and replace their materials with duplicates.
2. For all duplicated materials change **Bend Type** and **Bend ID** from material editor.
3. Create another **Curved World Controller** for updating bend settings for this new bend effect.

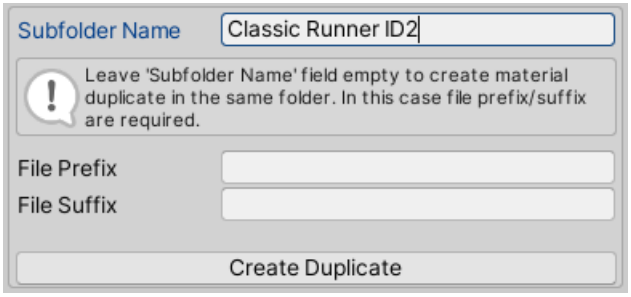
Note, instantiated meshes are inside **Object Pool** game object that is attached to the **Scene Manger** game object.

Steps 1) and 2) can be easily done by using Curved World editor window. Go to the **Renderers Overview** tab and choose **Selected Objects** from the drop down menu, then select **Player (2)** and **Scene Manager (2)** game objects inside Unity Hierarchy window.

Now all actions performed in **Renderers Overview** tab will affect only meshes that are currently selected and not all the meshes in this scene.
From the menu ≡ button choose **Replace Materials With Duplicates**.



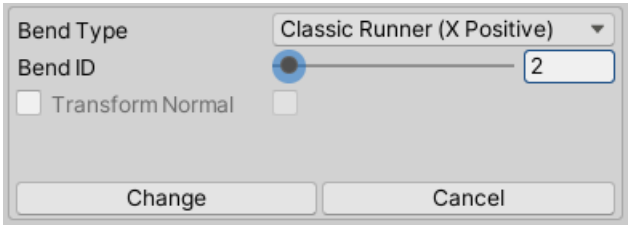
In the pop up window set **Classic Runner ID2** for the Subfolder name and click on **Create Duplicate** button.



This will create duplicate materials for all selected meshes and replace them in their renderers.

Note, this step can be Undo.

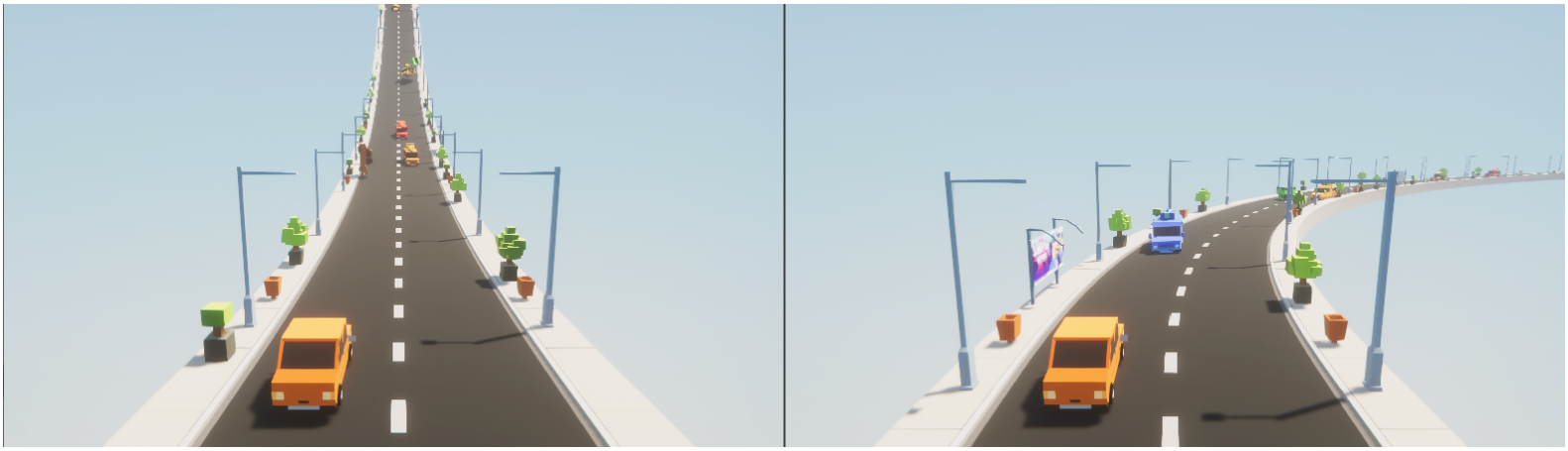
While still in **Renderers Overview** tab and **Player (2)** and **Scene Manager (2)** game objects are still selected, from the menu ≡ button choose **Change Curved World Bend Settings**.
Set **Classic Runner (X Positive)** for the **Bend Type** and 2 for **Bend ID**. Click on Change button.



Now all materials used for the second player's ***world*** rendering are using materials with different bending method.
Note, materials still are using the same one shader. But now this shader will have one more keyword describing additional bending method and for the second player's ***world*** materials is enabled this keyword.

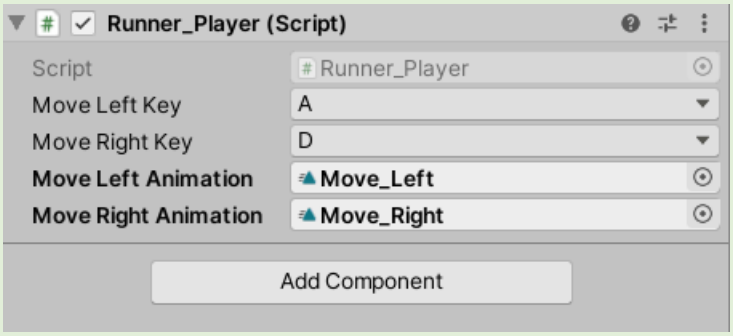
After materials are ready, it is time to update their bend settings. Select **Curved World Controllers** game object and from Components menu add one more **Curved World Controller** scrip.
Set **Bend Type** to **Classic Runner (X Positive)** and **Bend ID** to **2**, and set Pivot Point to be the **Player (2)**.

Now enter game mode.



If everything is done correctly ***world*** meshes for Player (1) and Player (2) should be bent differently according to their bend settings controlled from **Curved World Controller** scripts.

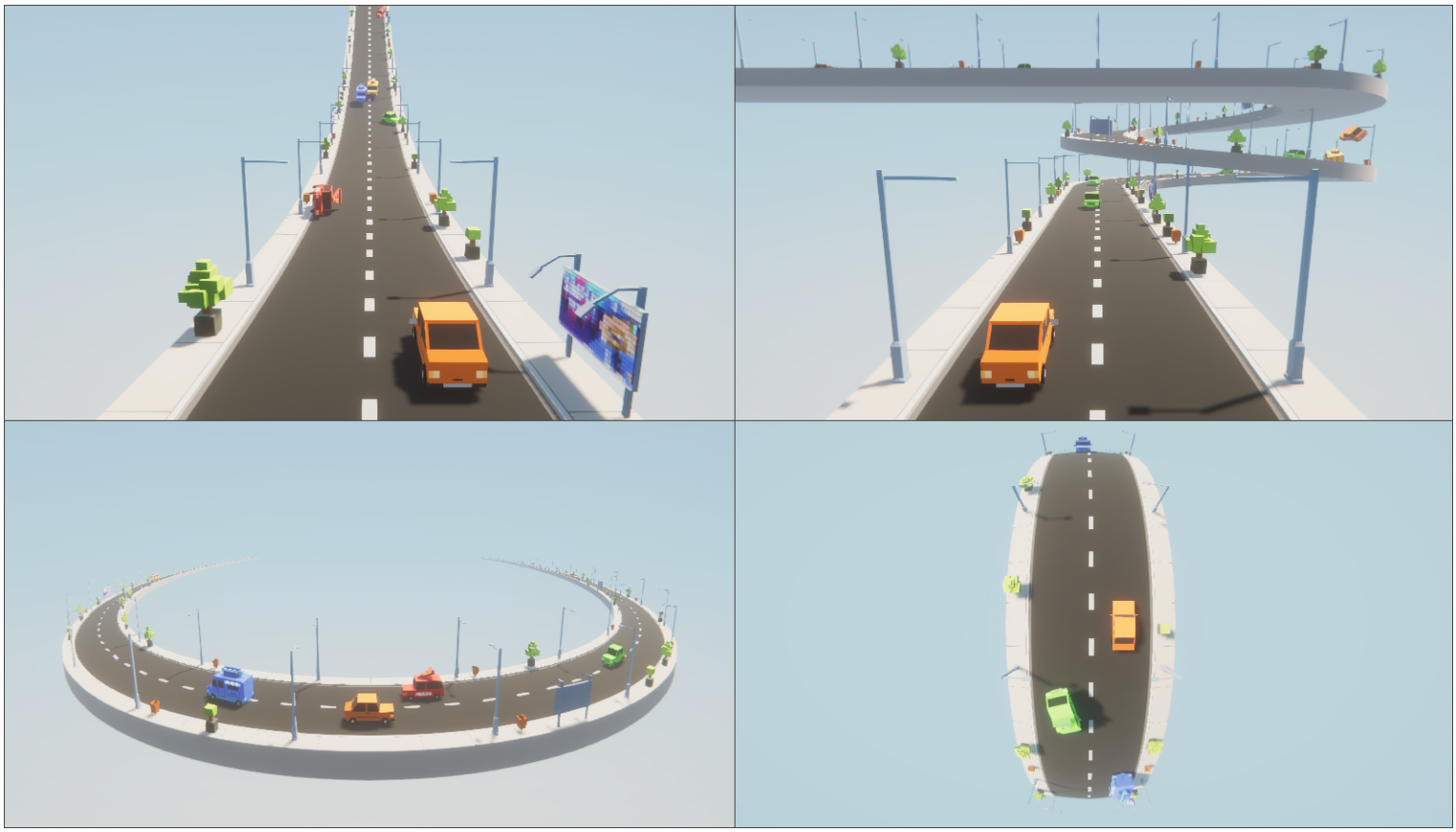
Note, player's navigation keys can be changed from the attached script.



To summarize this tutorial:

- Curved World bend methods are differentiate by **Bend Type** and **Bend ID**.
- Two bend effects with similar **Bend Types** but different **Bend IDs** – are two different bend effects.
- Shader can contain several bend effects.
- Materials switch bend effects using keywords (from material editors or run-time).
- Scene can contain any number of various materials with various bend types.
- Each bend effect must be updated separately, using **Curved World Controller** script or from any other custom script.

In this tutorial we have used two bending effects in one scene, but we can use even more.
Check Tutorial #4 - Multiple Bends Finished scene. Here are used 4 set of ***worlds***, each one with its unique bend effect updated and controlled independently from each other.

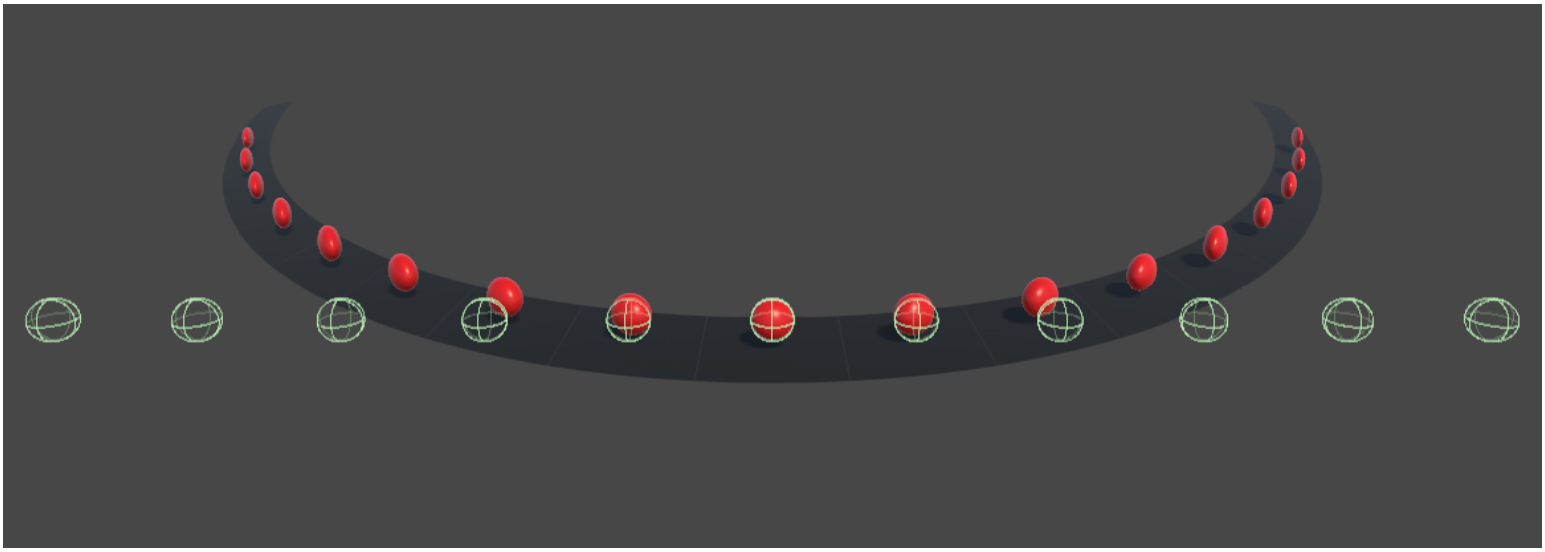


Non-shader bending

Bending effect created by Curved World is pure shader effect and presents only in rendered frame, real scene objects are not modified at all.

But what if we need to interact with bent meshes in run-time? For example picking them up by touching on screen, or selecting them by mouse click?

Image below represents a '*problem*' we may encounter – position of a bent mesh on screen and its real position in scene are different.



Curved World asset includes solution for this problem – calculating shader bending effect using script, helping transforming any '*world space*' position (variable in Vector3 format) into '*Curved World space*'.
This allows to transform almost any non-mesh object's position to follow the Curved World's curvature.

Check Non-Shader Bending example scenes to see how colliders, point lights and objects not using Curved World shaders are bent.

Note, script bending is explained in chapter **Package Scripts and run-time API**.

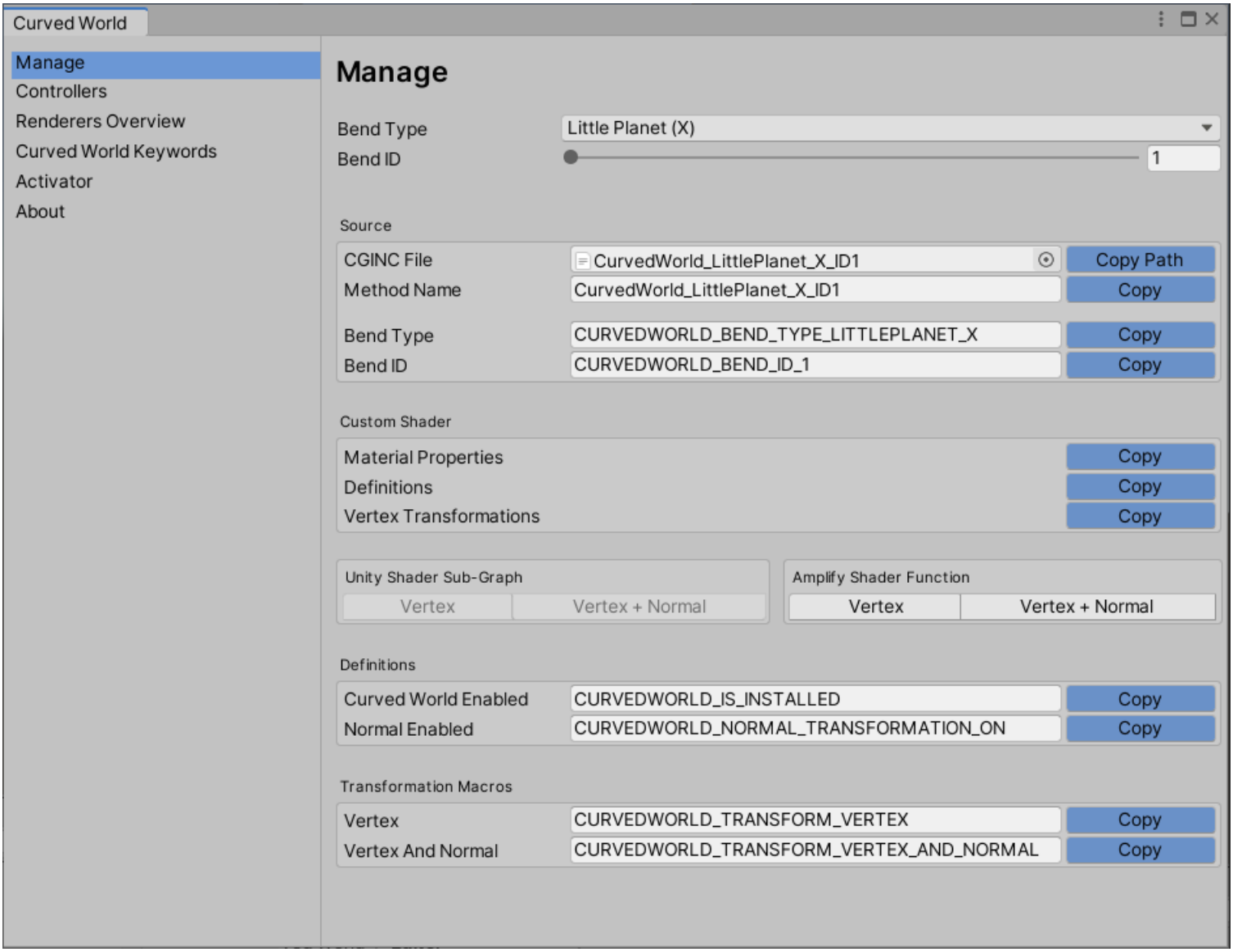
Editor window

Curved World editor window can be open from **Main Menu -> Window -> Amazing Assets -> Curved World.**

1. Manage

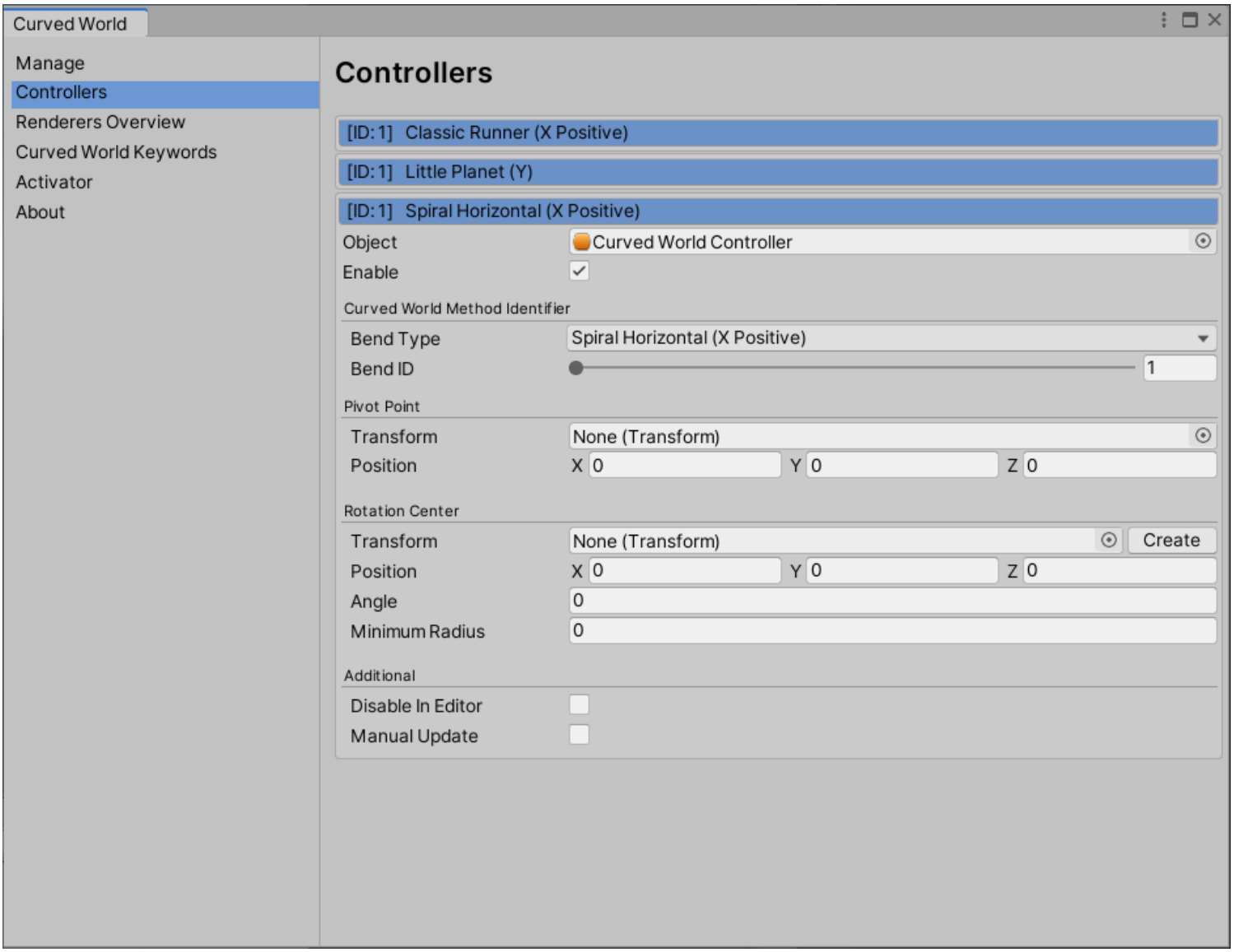
Allows generating Curved World **cginc** files based on **Bend Type** and **Bend ID** and displays all resources related to this bend.
If shades are using bend types not installed in project, they will not be compiled and display errors about missing **cginc** file. This tab should be used to generate all bend types and **cginc** files associated with them.

All resources displayed here are useful when creating custom Curved World shaders and are explained in chapter **Custom Shaders**.



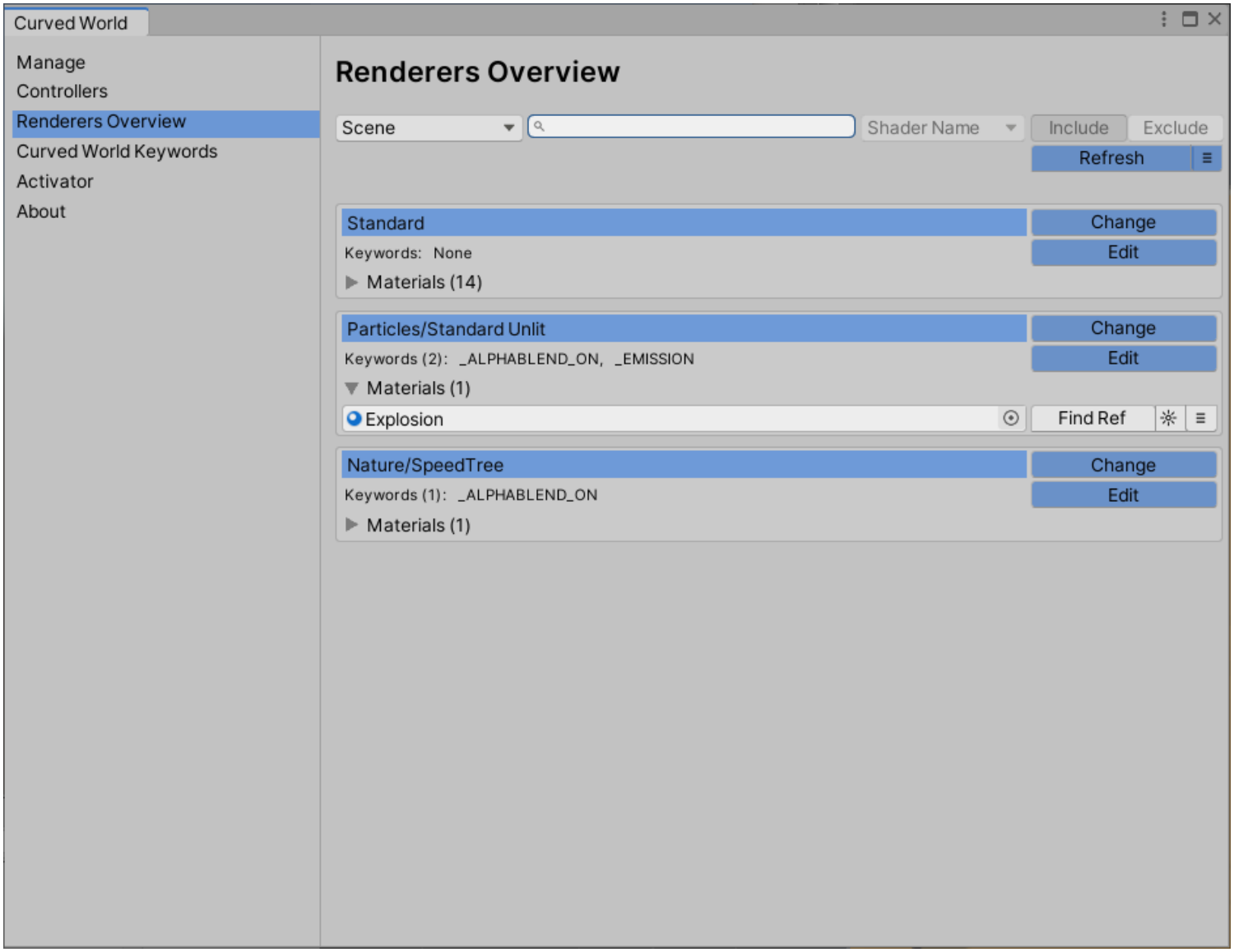
2. Controllers

From this tab can be created and controlled all scene **Curved World Controller** scripts.

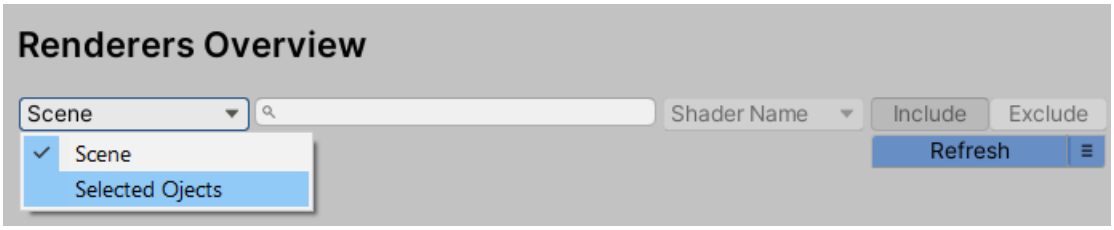


3. Renderers Overview

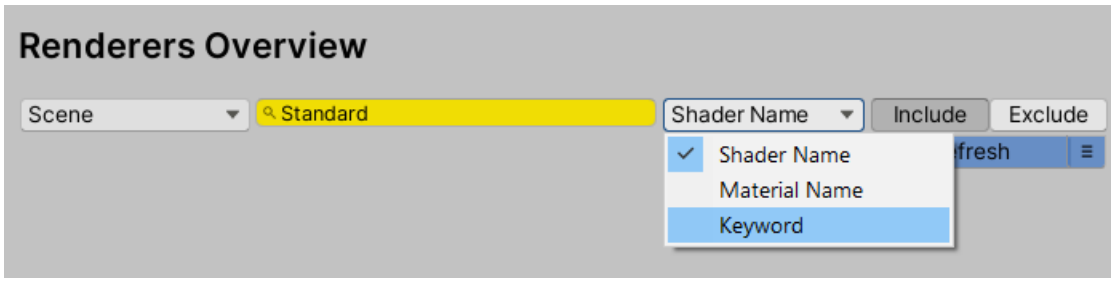
Here are displayed information about materials and shaders for current scene or selected objects (renderers). Allows to quickly change shaders for a group of materials or modify their keywords.



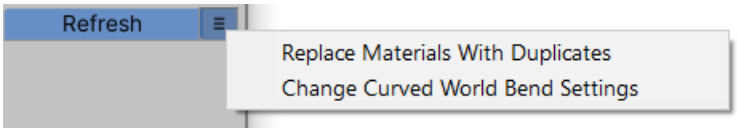
By default window displays information about all scene materials and shaders, but it can be changed to analyze only **Selected Objects** currently selected in Hierarchy window.



Filter bar can be used to filter displayed data by shader or material names, or by used keyword.



Refresh button updates displayed materials and shaders data.

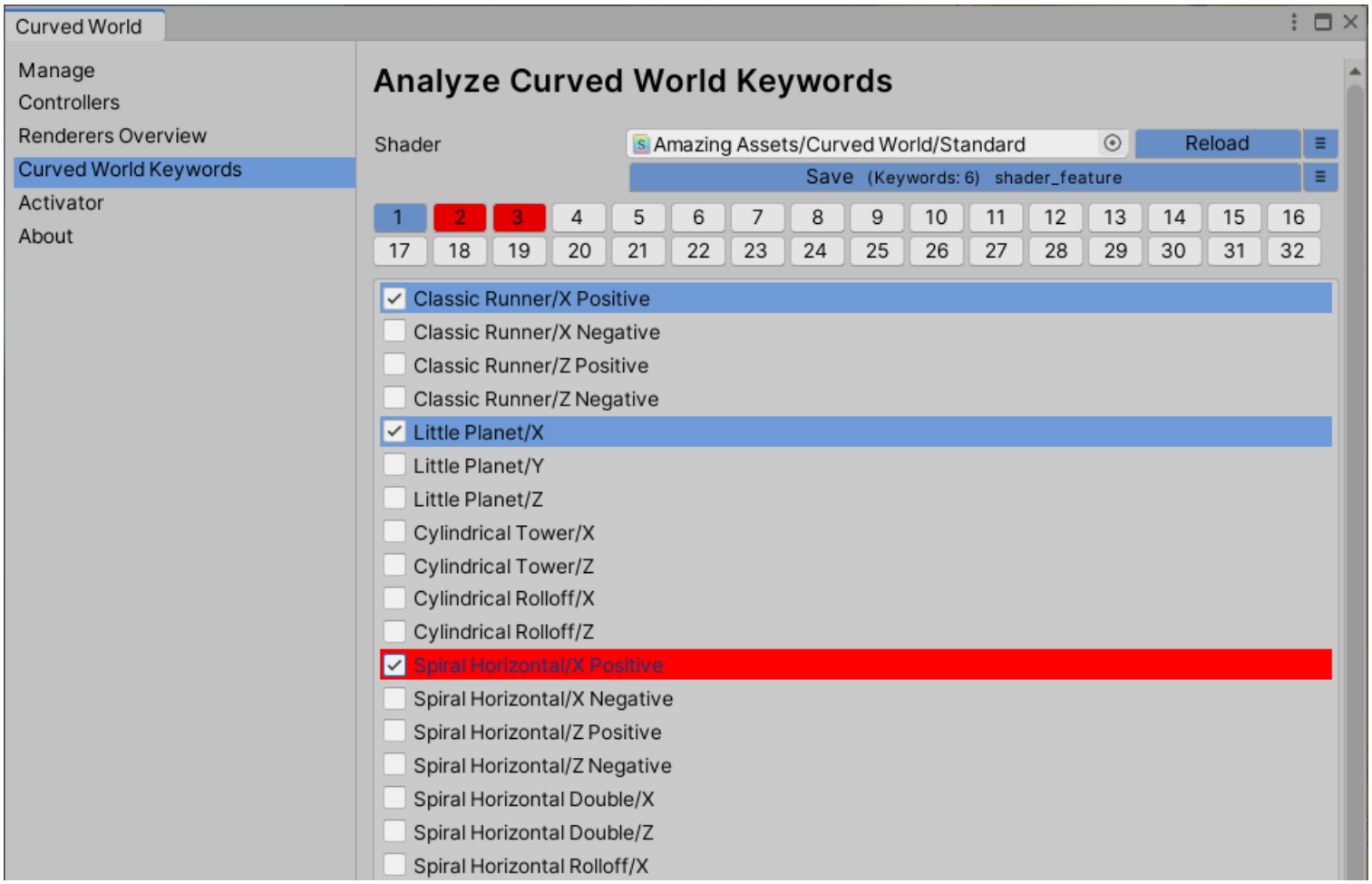


Menu ≡ button offers:

- Replace Materials With Duplicates – Replaces all materials currently displayed in tab with duplicates.
- Change Curved World Bend Settings – Modifies materials (displayed in tab) **Bend Type** and **Bend ID** properties. This action affects only those materials that are using Curved World shaders included in the asset or created by hand. Custom Curved World shaders created by shader graph tools or any other shaders are not affected. For such shaders **Bend Type** and **Bend ID** properties must be modified manually.

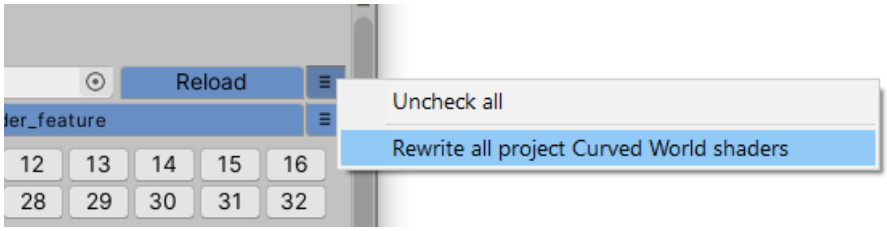
4. Curved World Keywords

Displays shaders Curved World keywords. Each **Bend Type** and **Bend ID** added to a shaders has its own keyword. Those keywords are enabled and disabled from materials editor or run-time scripts when switching bend types. This tab helps to visualize all such keywords.



Upper grid displace **Bend IDs**, bellow are **Bend Types**. In blue color are displayed keywords already available in this shader and **cginc** files associated for those bend effects are installed in project.
Save button rewrites shaders to have only those keywords (**Bend Types** and **Bend IDs**) that are currently selected in this window.
Keywords inside shader can be added as **shader_feature** or **multi_compile**.

From menu ≡ is possible to rewrite all project Curved World shaders to have selected **Bend Types** and **Bend IDs** only. This affects only shaders that are included in the asset package or written by hand. Other shaders are not affected.

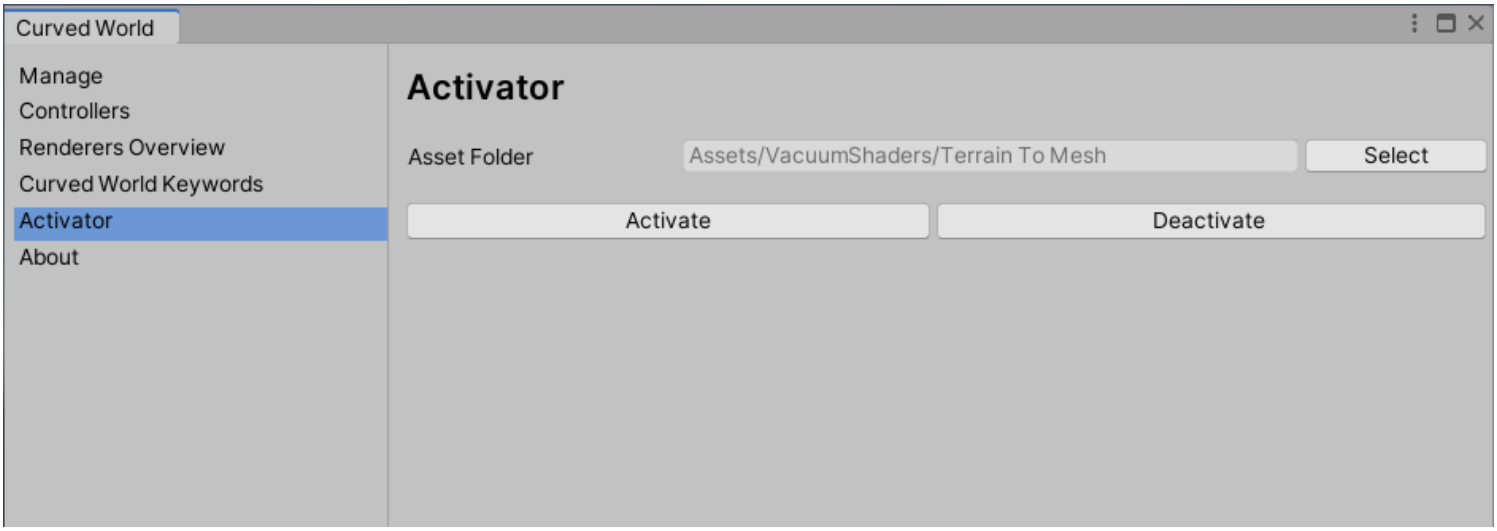


5. Shader Packages

By default when importing Curved World asset not all shaders are installed. From this window can be imported additional shaders.

6. Activator

Used for activating Curved World transformation integrated into assets purchased from the Asset Store.



If Asset Store publisher made asset Curved World compatible, after importing that asset package into project, Curved World vertex transformation in shaders will be disabled by default. To activate it, select asset folder location and click on **Activate** button.

Deactivate button removes Curved World compatibility from shaders.

Note, activator woks only with hand written shaders. If asset shaders are created using graph tool, Curved World bend nodes can be added manually.

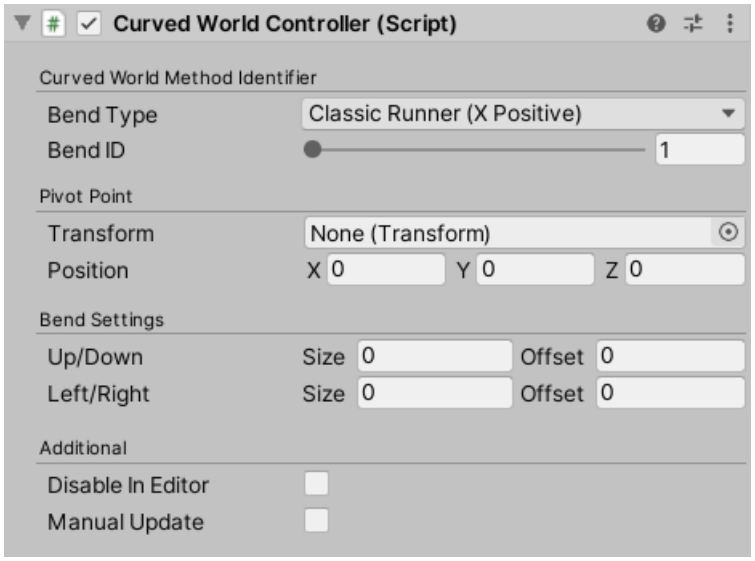
Note, make sure asset is Curved World compatible before using Activator tool.

Note, using activator resets all bend types and IDs added to a shader.

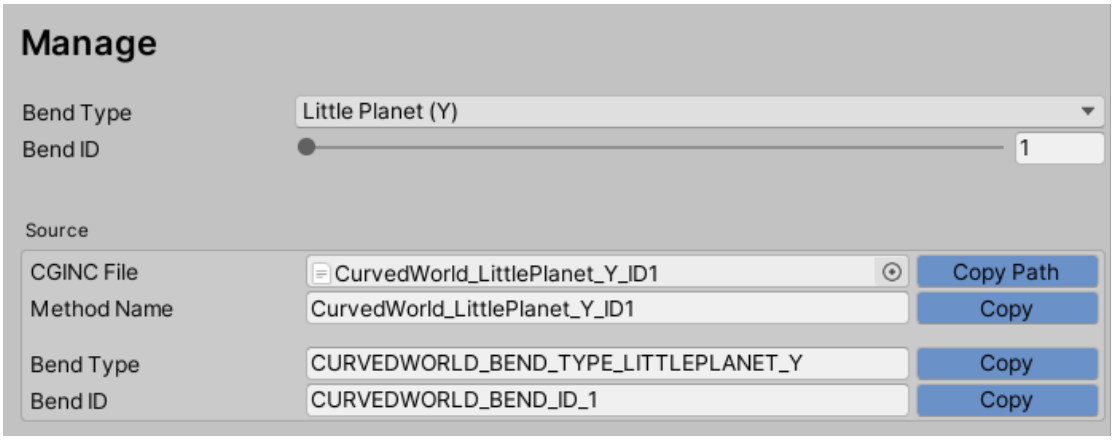
Package scripts and run-time API

1. Curved World Controller

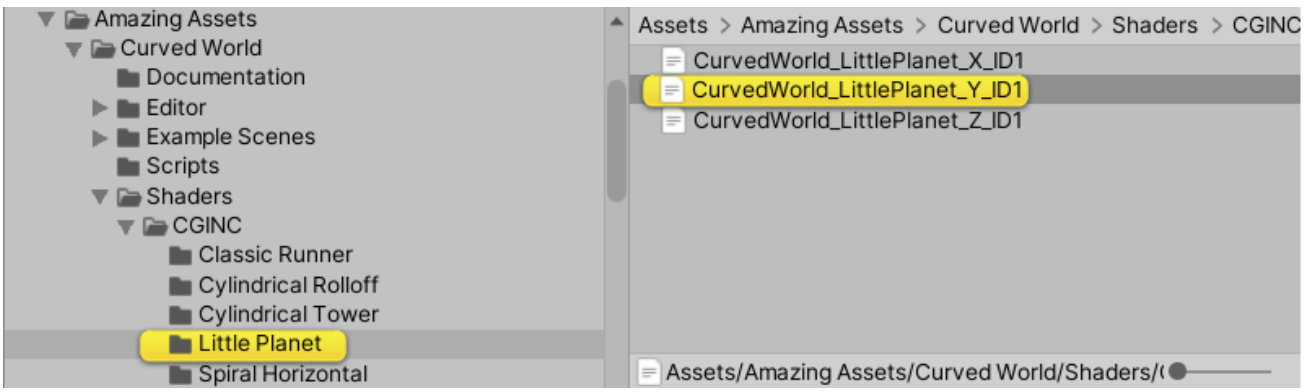
Updates bend variables described in its **cginc** file and affects all materials using **Bend Type** and **Bend ID** selected here.



Each bend method and its control variables are described in its own unique **cginc** file. To see all those variables in the Curved World editor window in Manage tab choose desired **Bend Type** and **Bend ID** and below in the **Source** section will be provided **CGINC File** associated with this bend effect.



Clicking on this filed will select and highlight file in the Project window. All variables and method of this bend effect are described in this file.



Curved World Controller script can be brought into scope with this using directive:

C#
```
using AmazingAssets.CurvedWorld;
```

Public variables
```
BEND_TYPE bendType;
int bendID;

Transform bendPivotPoint;
Vector3   bendPivotPointPosition;

Transform bendRotationCenter;
Vector3   bendRotationCenterPosition;

Transform bendRotationCenter2;
Vector3   bendRotationCenter2Position;

float bendVerticalSize, bendVerticalOffset;
float bendHorizontalSize, bendHorizontalOffset;
float bendCurvatureSize, bendCurvatureOffset;

float bendAngle;
float bendAngle2;
float bendMinimumRadius;
float bendMinimumRadius2;
float bendRolloff;


bool disableInEditor;
bool manualUpdate;
```

void DisableBend()

Disables current bend effect. Materials affecting by this script will be rendered without Curved World transformation.

void EnableBend()

Enables current bend effect. Script will continue updating Curved World bend settings.

void ManualUpdate()

Forces controller script to update bend settings.

```
Vector3 TransformPosition(Vector3 vertex)
```

Transforms **vertex** World space position into Curved World space.

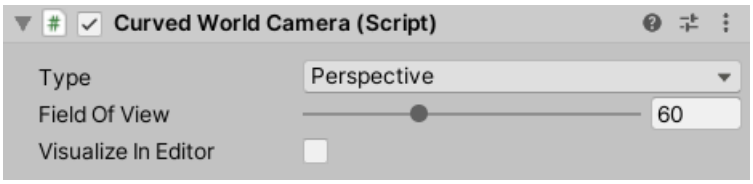
```
Quaternion TransformRotation(Vector3 vertex, Vector3 forwardVector, Vector3 rightVector)
```

Calculates **vertex** rotation (Normal vector direction) that follows Curved World curvature.

TransformVertex and **TransformRotation** methods are used in Non - Shader Bending example scenes.

2. Curved World Camera

Overrides camera's culling matrix before it begins frame rendering. Allowing to capture objects outside camera's original field view.



Script calculates culling matrix using **Perspective** or **Orthographic** algorithms. Near and far clipping planes are taken from the source Camera component.
Script must be attached to the Camera object.

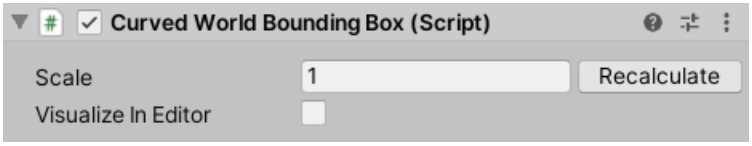
Curved World Camera script can be brought into scope with this using directive:

C#
```
using AmazingAssets.CurvedWorld;
```

Public variables
```
MATRIX_TYPE matrixType;

float fieldOfView;
float size;
bool nearClipPlaneSameAsCamera;
float nearClipPlane;
```

3. Curved World Bounding Box

Scales individual renderer's (MeshRenderer or SkinnedMeshRenderer) bounding box component and makes it visible to the Camera, even if mesh is outside of a Camera field of view.



Script is also necessary if mesh is not visible to a dynamic light source (it has its own field of view), to avoid excluding mesh from shadow receiving/casting pass.
Script must be attached to the object with MeshRenderer or SkinnedMeshRenderer component.

Curved World Bounding Box script can be brought into scope with this using directive:

C#
```
using AmazingAssets.CurvedWorld;
```

Public variables
```
float scale;
```

```
void RecalculateBounds ()
```

Recalculates renderer's original bounds.

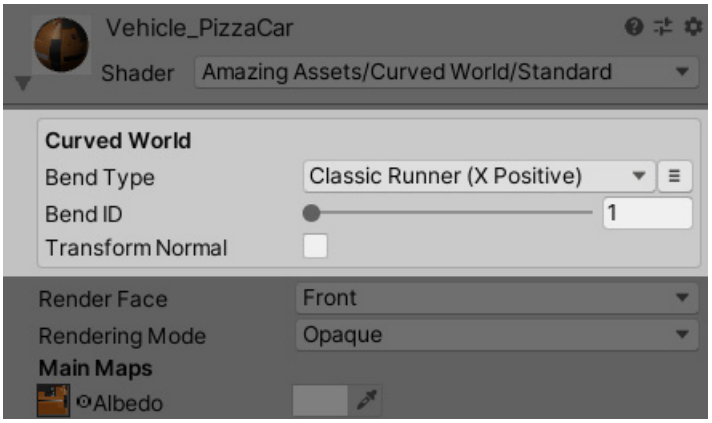
Custom shaders

Curved World vertex transformation can be integrated into hand written shader or added as a custom single node to the Unity Shader Graph and Amplify Shader Editor.

1. Hand written

Integrating Curved World into hand written shader requires:

- Adding Curved World material property to the shader that will be visualized using material editor, allowing switching **Bend Types**, **Bend IDs** and **Normal Transformation**. Like on the image below.

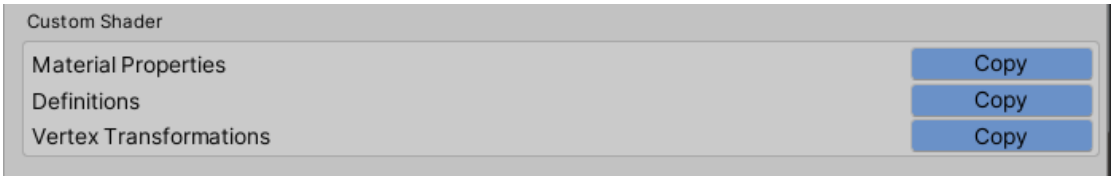


- Adding paths to the transformation **cginc** files and keyword definitions.
- Adding vertex transformations to the shader's vertex stage.

To help with steps described above Curved World's editor window already includes pre-defined fields with string data to just copy them and paste into a shader.

Open Curved World's editor window and in the Manage tab choose desired **Bend Type** and **Bend ID** (generate transformation if it is not installed), for this example **Classic Runner (X Positive)** bend type with 1 ID.

Editor window now will display all resources related to this transformation, with pre-defined fields for hand written shaders in **Custom Shader** group.

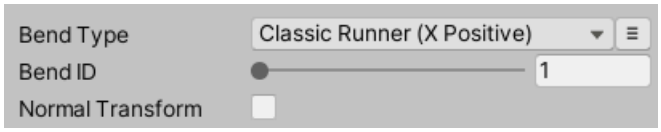


Note, to follow steps described here open and modify Tutorial Unlit Shader (from Curved World/Shaders/Custom/Tutorial folder). This is simple Unlit shader.

Click on Copy button for the **Material Properties** field. This will copy string into system keyboard memory. Paste it into the shader file inside Properties block.

```
Shader "Amazing Assets/Curved World/Tutorial/Unlit Shader"
{
    Properties
    {
        [CurvedWorldBendSettings] _CurvedWorldBendSettings("0|1|1", Vector) = (0, 0, 0, 0)

        _MainTex ("Texture", 2D) = "white" {}
    }

    SubShader
    {
        Tags { "RenderType"="Opaque" }
        LOD 100
```

If shader does not use custom editor, after saving shader file material editor will display Curved World properties:



Changing those properties from material editor currently will not have any effect. We have to add Curved World transformation definitions and keywords to this shader first.

Note, do not modify `_CurvedWorldBendSettings` string and its data manually.

For the second integration step click on Copy button for the **Definitions** field.

Paste copied string into the shader below Unity's default `#includes`, but before the vertex stage:

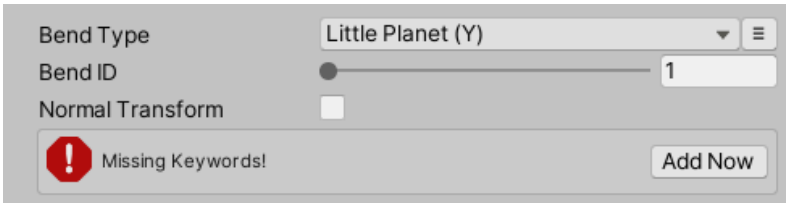
```
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #pragma multi_compile_fog

            #include "UnityCG.cginc"

#define CURVEDWORLD_BEND_TYPE_CLASSICRUNNER_X_POSITIVE
#define CURVEDWORLD_BEND_ID_1
#pragma shader_feature_local CURVEDWORLD_DISABLED_ON
#pragma shader_feature_local CURVEDWORLD_NORMAL_TRANSFORMATION_ON
#include "Assets/Amazing Assets/Curved World/Shaders/Core/CurvedWorldTransform.cginc"
```

5 lines of text will be added to the shader:

- First line contains keywords for **Bend Types**. Everytime **Bend Type** property is changed from material editor, shader file will be modified to include keyword for that bend type. Currently shader has keyword definition only for **Classic Runner (X Positive)** bend type:

```
#define CURVEDWORLD_BEND_TYPE_CLASSICRUNNER_X_POSITIVE
```

If try changing **Bend Type** property from material editor, for example adding **Little Planet (Y)**, material editor will display Missing Keywords warning.



Clicking on **Add Now** button will add required keyword to the shader and enable it for the material.

```
#pragma shader_feature_local CURVEDWORLD_BEND_TYPE_CLASSICRUNNER_X_POSITIVE  CURVEDWORLD_BEND_TYPE_LITTLEPLANET_Y
```

- Same way works the second line defining **Bend ID** keywords.
- Third line contains keyword for disabling Curved World vertex transformation. This keyword is not visible inside material editor and can be enabled only from script.
- Fourth line contains keyword for **Normal Transformation**.
- Fifth line contains path to the `CurvedWorldTransformation.cginc` file.

The final step is using transformation method in vertex stage. Click on Copy button for the

Vertex Transformations field.

Paste copied string into a vertex stage before calculating pixels screen space position.

```
v2f vert (appdata_full v)
{
    v2f o;

    #if defined(CURVEDWORLD_IS_INSTALLED) && !defined(CURVEDWORLD_DISABLED_ON)
      #ifdef CURVEDWORLD_NORMAL_TRANSFORMATION_ON
        CURVEDWORLD_TRANSFORM_VERTEX_AND_NORMAL(v.vertex, v.normal, v.tangent)
      #else
        CURVEDWORLD_TRANSFORM_VERTEX(v.vertex)
      #endif
    #endif

    o.vertex = UnityObjectToClipPos(v.vertex);
    o.uv = TRANSFORM_TEX(v.texcoord, _MainTex);
    UNITY_TRANSFER_FOG(o,o.vertex);

    return o;
}
```

`CURVEDWORLD_IS_INSTALLED` is Curved World assets global definition for shaders described in `CurvedWorldTransformation.cginc` file, indicating validation of the Curved World vertex transformations.

Below are two Curved World vertex transformations (provided as macros):

- `CURVEDWORLD_TRANSFORM_VERTEX_AND_NORMAL` – Transforms vertex and normal to follow Curved World curvature. Executing only if **Transform Normal** keyword is enabled.
- `CURVEDWORLD_TRANSFORM_VERTEX` – Transforms only vertex.

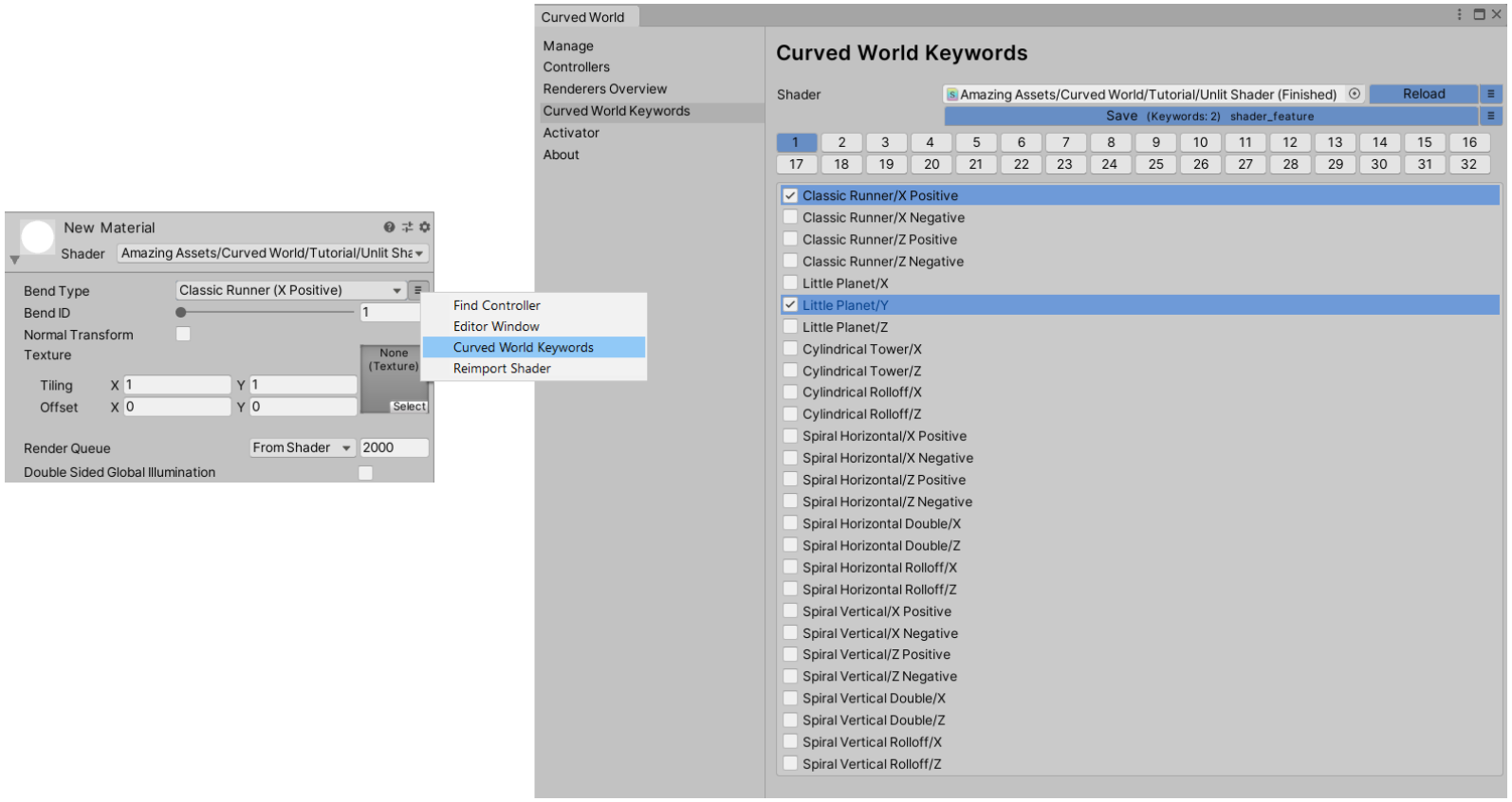
That is all. Save file.
Shader now is fully compatible with Curved World.

Note, pay attention to the vertex data semantic provided in the transformation methods. In the example above `vertex`, `normal` and `tangent` variables are described in `appdata_full` class with provided names. In other custom shaders those names may be different.

Note, if shader uses other per-vertex transformations, like texture displace, wind, etc. Use Curved World transformation after them.

Note, actions described in steps 2 (adding Curved World transformation definitions and keywords) and 3 (adding transformation methods to the vertex stage) must be used in every pass of a shader.

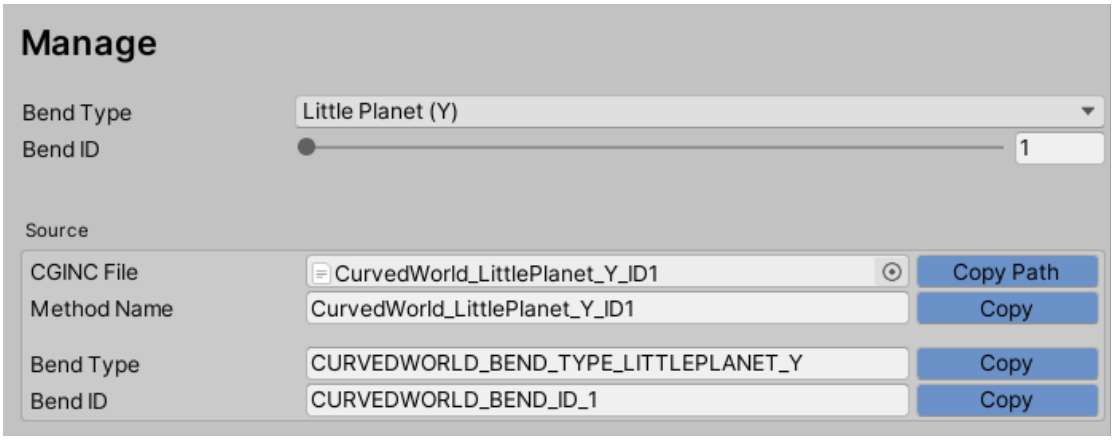
Note, check/add/remove Curved World keywords from shader using Curved World editor window.



Steps described above adds Curved World transformation to a shader in the ***Universal*** way. It means that such shader does not need any further hand modifications and it will automatically support all bend types with multiple bend IDs. But the only one backside of such shader is that it can reach keyword count limit, as each bend type and ID here has its own unique keyword.

It is possible to integrate Curved World into a shader without any keywords. But, it will not have Curved World properties inside material editor and will not be detected by Curved World editor window.

Inside Manage tab choose desired bend type and ID.



Click on **Copy Path** button for the **CGING File** field and paste it into a shader below Unity's default `#includes`, but before the vertex stage.

```
Pass
{
    CGPROGRAM
    #pragma vertex vert
    #pragma fragment frag
    #pragma multi_compile_fog

    #include "UnityCG.cginc"


    #include "Assets/Amazing Assets/Curved World/Shaders/CGINC/Little Planet/CurvedWorld_LittlePlanet_Y_ID1.cginc"
```

In this case this is the direct path to the method's **cginc** file and not to the `CurvedWorldTransformation.cginc` file.
Click on **Copy** button for the **Method** name field and paste copied string in vertex stage before calculating pixels screen space position.
This will be Curved World transformation method name (not macros). For transforming vertex and normal, or only for vertex, it uses the same method but with different parameters.

```
v2f vert (appdata_full v)
{
    v2f o;


    //Vertex and Normal calcualtion
    CurvedWorld_LittlePlanet_Y_ID1(v.vertex, v.normal, v.tangent);

    //Vertex transformation only
    CurvedWorld_LittlePlanet_Y_ID1(v.vertex);


    o.vertex = UnityObjectToClipPos(v.vertex);
    o.uv = TRANSFORM_TEX(v.texcoord, _MainTex);
    UNITY_TRANSFER_FOG(o,o.vertex);

    return o;
}
```

Integration method described here is keyword free and is dedicated for shaders that will support only specific bend types without possibility to modify them externally (for example from material editor).

Note, Tutorial Shader (Keywords Free) uses integration method described here.

Note, integration method described here can be used to blend/mix multiple bend types that is not possible using ***Universal*** method described above.

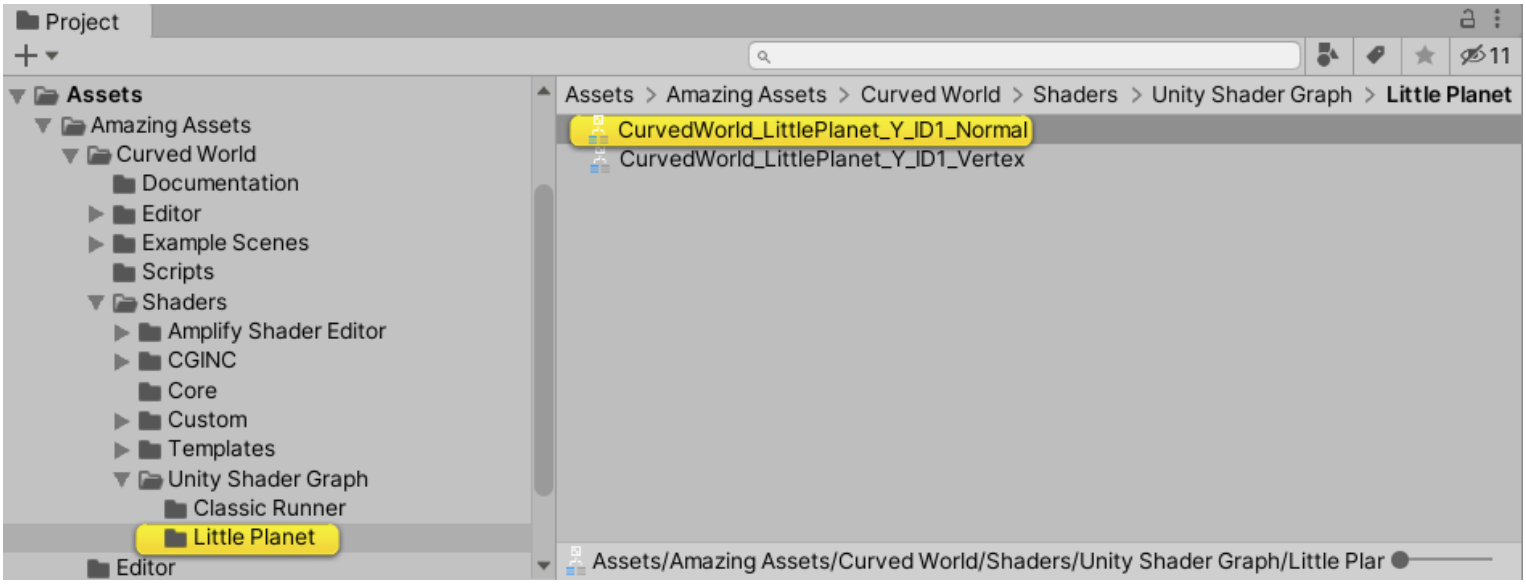
2. Shader Graphs

Curved World asset supports one node integration for Unity Shader Graph and Amplify Shader Editor.

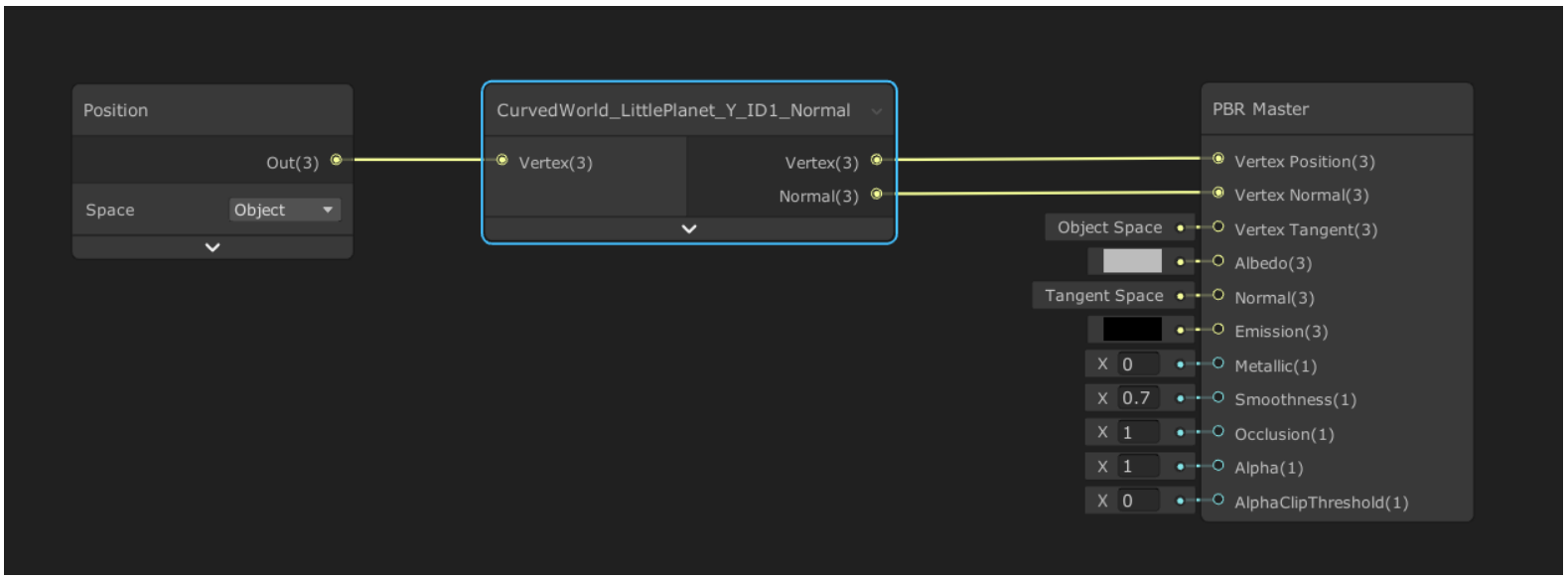
Each node represents only one Bend Type with its ID. Inside shader can be used any number of bend nodes in any combination and various IDs.

To easily navigate to the required node, inside Curved World editor window's Manage tab choose desired **Bend Type** and **ID**.
Nodes for Unity Shader Graph and Amplify Shader Editor will become available.

For each graph tool is available two separate nodes, for vertex transformation only and both vertex/normal transformation.
Click on required node button and it will be highlighted inside Project window.



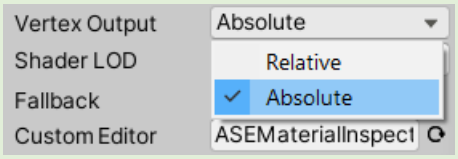
Drag and drop node file into shader editor and connect output nodes to the Master node.



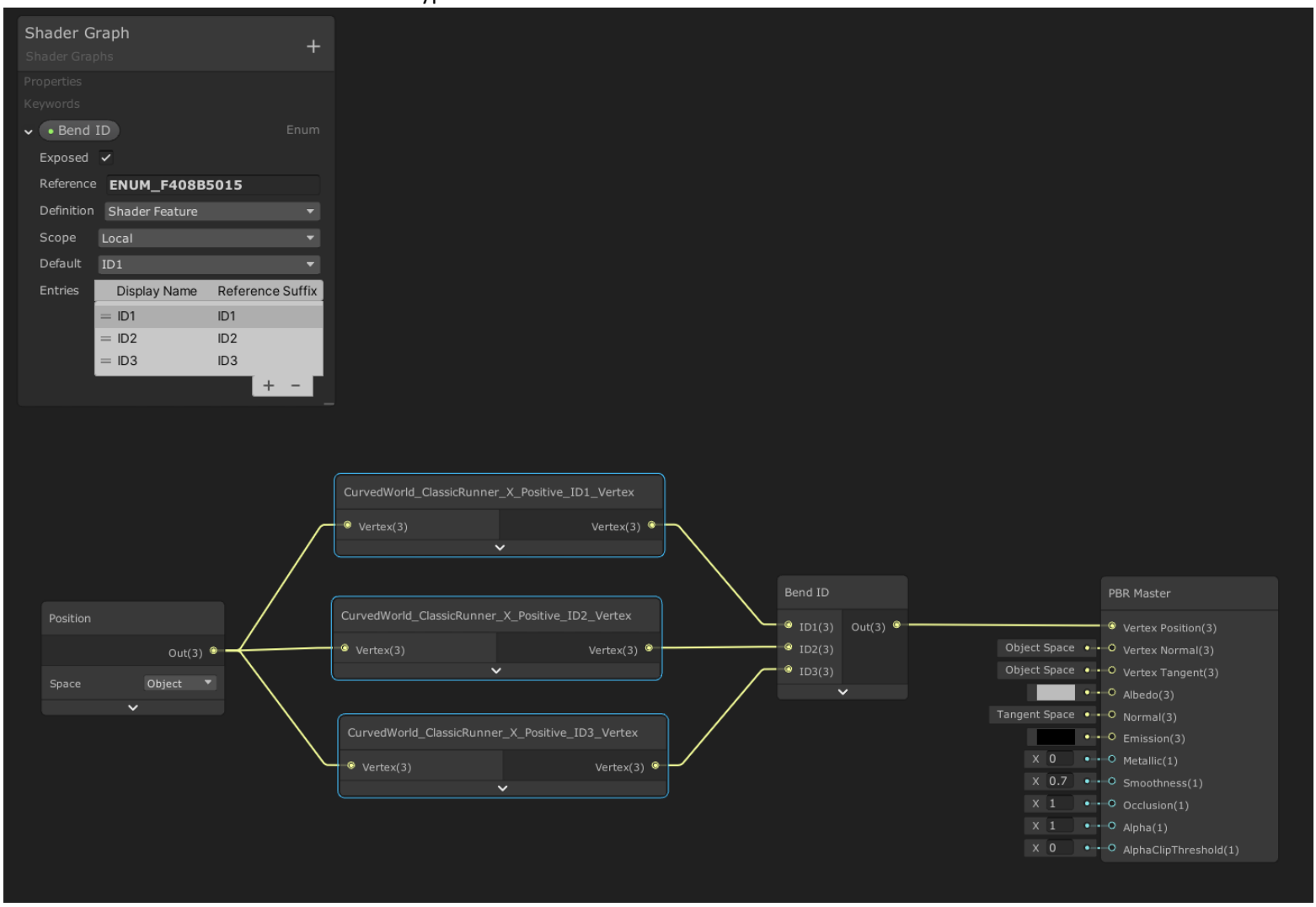
Curved World transformation node requires vertex **object space** position.
If shader includes other per-vertex transformations, like texture displace, wind, etc. Use Curved World transformation after them and make sure their resultant vertex position (used as input for Curved World node) is in **object space**.

Note, Curved World node is calculated in vertex pass and results of this node cannot be used in fragment/pixel pass.

Note, Amplify Shader Editor has two options for output vertex. Choose **Absolute** in General settings.



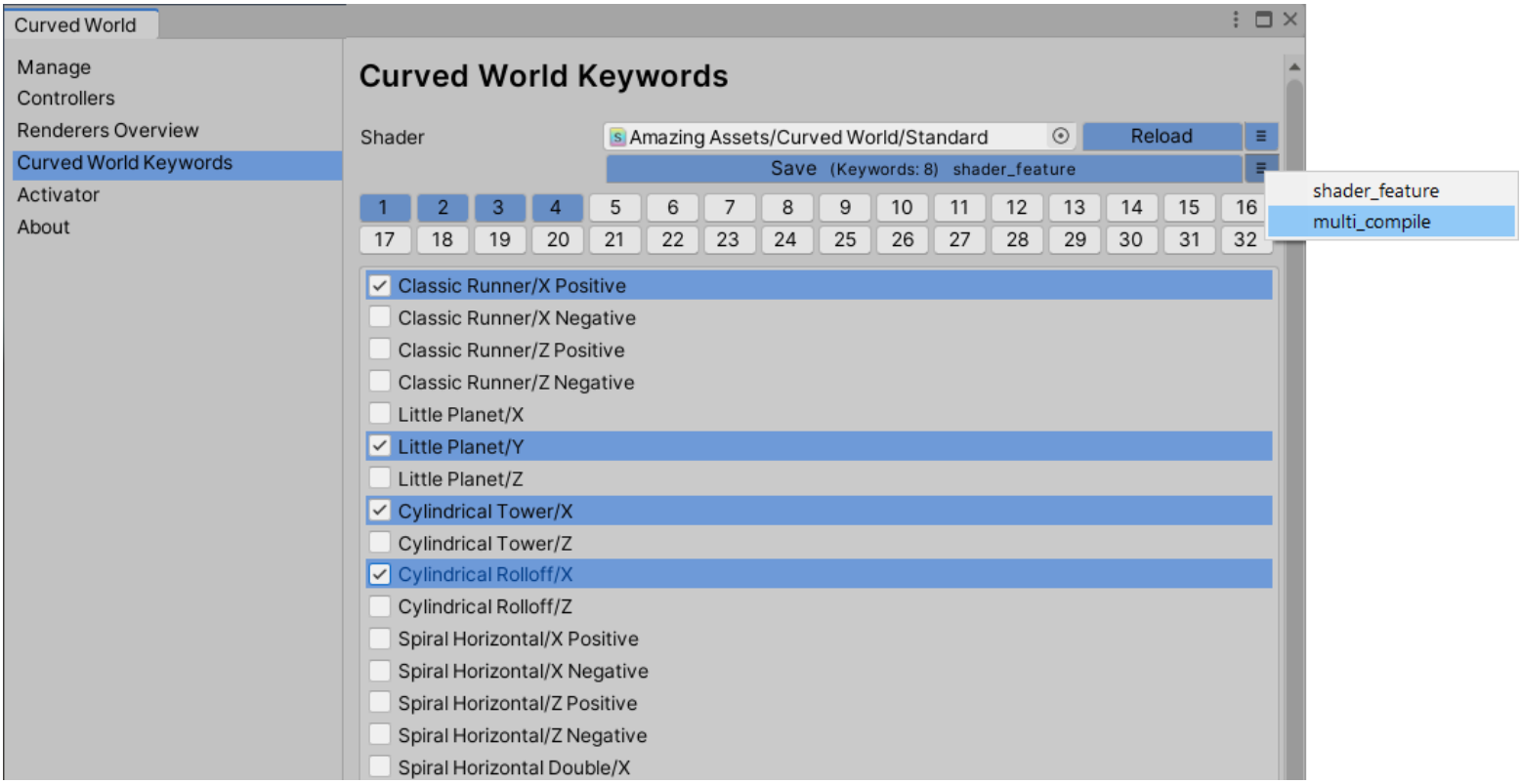
As shader graph tools support shader variations using keywords, it is possible to use several Curved World nodes for different bend types and IDs and switch them from material editor.



Building Project

When building project with Curved World shaders using multiple bend effects make sure:

- All Curved World keywords are included in build. This can be achieved by creating shader variants collection or defining all required Curved World bend types and ID keywords as **multi_compile**, this can be done from Curved World's editor window:



Note, Curved World Keywords window works only with shaders included within asset package or written by hand. Shaders created using graph tools or any other shader must manage **multi_compile** manually.

- Make sure all fallback shaders using Curved World transformation are also included in build and they include all bend types keywords with their IDs used in project.

- (Only for Standard (built-in) render pipeline) Curved World asset package includes CurvedWorld-DepthNormalsTexture.shader (Curved World/Shaders/Custom/Internal folder) containing normal calculations for post-processing and image effects.
  Use it in combination with Unity's DepthNormalsTexture shader and include it in build by adding it in **Always Included Shaders** array of the Graphics Settings.
  For the final build remove all unused Render Types from this shader and make sure it includes all required bend keywords with **multi_compile**.

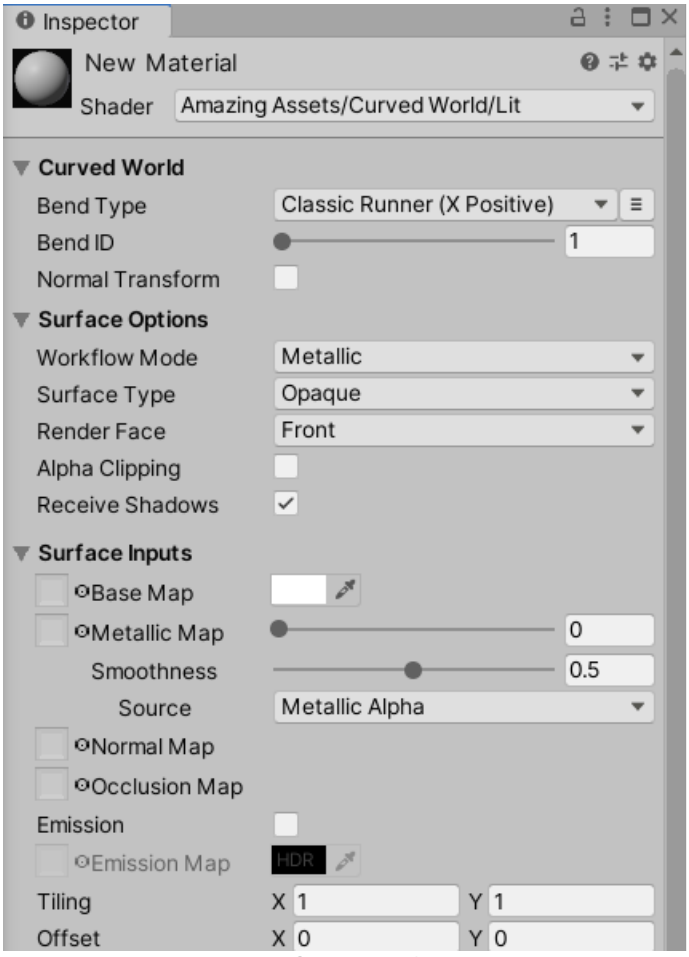
Known Issues

Two know issues when using Curved World asset are related to the Unity Engine itself.

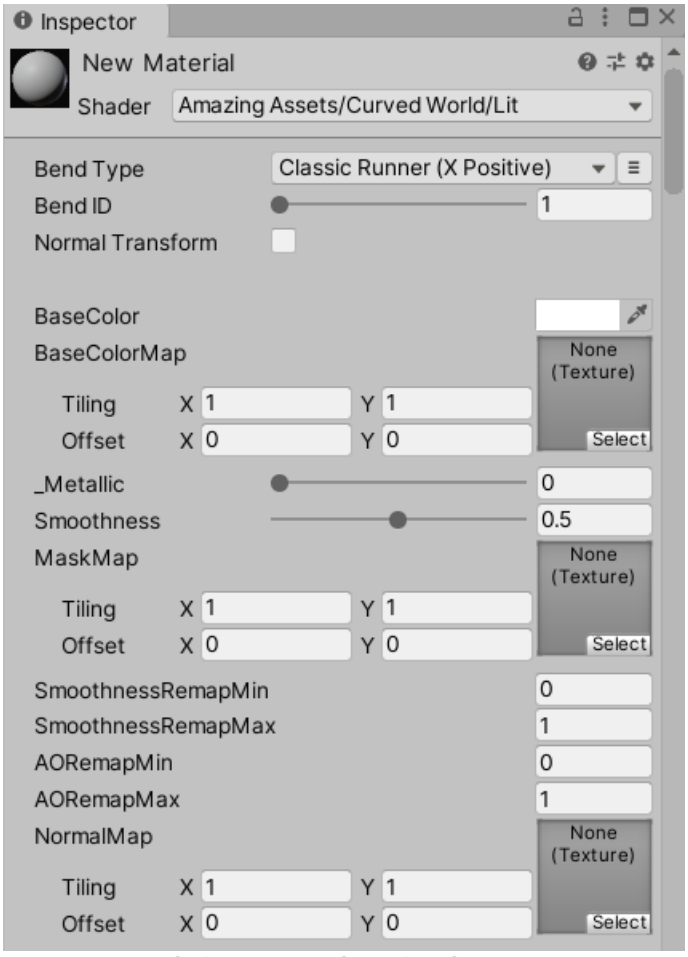
1. Terrain Shaders – Curved World generates fully functional terrain shaders with all required sub-shaders, passes and dependent files, but they have one limitation – Unity does not support overriding multiple Grass/Detail shaders. If scene uses multiple bend types with different terrains rendering grass and detail meshes will be possible only just for one terrain.

   Note, in some Unity versions there is problem (Engine bug) and overridden terrain Grass/Detail shaders are not recognized at all. If Curved World terrain is not rendered with bent grass and details – it may be only due to Unity bugs.

2. High Definition render Pipeline (HDRP) – Curved World shaders have no custom material editors. For Built-in and Universal render pipelines Unity material editor source files are ***public*** and can be overridden/extended for displaying Curved World shader properties. In HDRP such files are not available and Curved World shaders in material editor are displayed using default system GUI.



Custom GUI for Universal RP

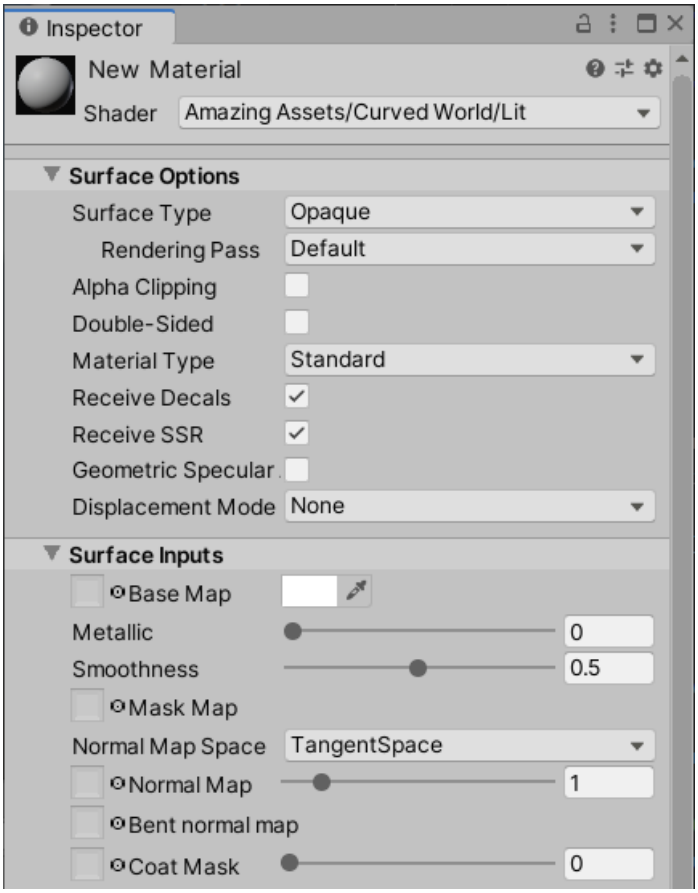


Default system GUI for High Definition RP

Using default system GUI may be problematic for adjusting material properties. It is possible to restore custom GUI for HDRP shaders: Open ***.shader*** file in any text editor and on the last line uncomment line with CustomEditor string. Save File.

```
            #include "ShaderPassDepthOnly.hlsl"

            #pragma vertex Vert
            #pragma fragment Frag

            ENDHLSL
        }
    }


    //Uncomment line below to render this shader using Unity default material editor
    //CustomEditor "Rendering.HighDefinition.LitGUI"
}
```

Material now will be displayed using HDRP inspector, but without Curved World properties.
For displaying Curved World properties comment CustomEditor string inside ***.shader*** file again.



Supporting Curved World by Asset Store Publishers

For making shader asset compatible with the Curved World, it is enough to follow steps described in chapter Custom Shader and add into a shader:

1. Curved World material property.

```
Properties
{
    [CurvedWorldBendSettings] _CurvedWorldBendSettings("0|1|1", Vector) = (0, 0, 0, 0)

    _MainTex ("Texture", 2D) = "white" {}
}
```

2. Initial bend types keywords and path the to the `CurvedWorldTransformation.cginc` file.

```
        #include "UnityCG.cginc"

#define CURVEDWORLD_BEND_TYPE_CLASSICRUNNER_X_POSITIVE
#define CURVEDWORLD_BEND_ID_1
#pragma shader_feature_local CURVEDWORLD_DISABLED_ON
#pragma shader_feature_local CURVEDWORLD_NORMAL_TRANSFORMATION_ON
#include "Assets/Amazing Assets/Curved World/Shaders/Core/CurvedWorldTransform.cginc"
```

String lines provided in steps 1 and 2 above should be added into the main ***.shader*** file, not in sub-shader or used cginc/hlsl files. As `CurvedWorldBendSettings` editor drawer script will modify this ***.shader*** file to add/remove bend definitions and keywords.

3. Curved World transformation to the vertex shader stage. This can be added into the main shader file or in any sub-shader or cginc/hlsl files.

```
v2f vert (appdata_full v)
{
    v2f o;

    #if defined(CURVEDWORLD_IS_INSTALLED) && !defined(CURVEDWORLD_DISABLED_ON)
        #ifdef CURVEDWORLD_NORMAL_TRANSFORMATION_ON
            CURVEDWORLD_TRANSFORM_VERTEX_AND_NORMAL(v.vertex, v.normal, v.tangent)
        #else
            CURVEDWORLD_TRANSFORM_VERTEX(v.vertex)
        #endif
    #endif

    o.vertex = UnityObjectToClipPos(v.vertex);
    o.uv = TRANSFORM_TEX(v.texcoord, _MainTex);
    UNITY_TRANSFER_FOG(o,o.vertex);

    return o;
}
```

Note, pay attention to the vertex data semantic and variable names provided in the transformation methods. In the example above `vertex`, `normal` and `tangent` variables are described in `appdata_full` class with provided names. In custom shaders those names may be different.

After making shader compatible with Curved World, string lines added in steps 1 and 2 can be commented. In this case Curved World's vertex transformation method in the vertex shader stage will not be compiled (even it is not commented) because `CURVEDWORLD_IS_INSTALLED` definition will become invalid, as it is defined in commented `CurvedWorldTransformation.cginc` file.

Curved World compatible package now can be published to the Asset Store.

After importing asset package into a project (with Curved World asset installed), shaders can be activated by Activator tool.