naru logs

- january 12
  - Jan 12
- january 12th raw- no ai
- splines plugin
  - splines plugin code default
- New Page

january 12

january 12

Jan 12

Exploring the World of Linux: My Personal Journey (I don't like this title but the ai gave it)

It all started on a cold January night. With my weird sleep schedule I ended up waking up at midnight, I found myself in the mood to program and decided to tackle a problem that had been bugging me for a while. The day before, I had been running stable diffusion, an open-source project that does the same thing as Dall-E, with custom training data and downloaded models. As I was running them locally on my Windows PC with a 3060 12GB GPU, I was unsatisfied with the overhead and wanted to find a solution. I considered several options, such as trying it in Linux, modifying the code myself, killing background tasks, or overclocking my GPU. After hours of trial and error, I decided to give up for the time being and refresh my Python skills.

To challenge myself, I decided to write a program that would give all possible variations of an 8x8 image with only black and white and output it into a video. It was fun but difficult because I had to learn how to multithreaded the task and ensure I didn't use all of my RAM or create too many files. But, I kept at it and enjoyed the process.

And then, the time came. I received a package containing a portable USB-C SSD and I knew that this was the perfect opportunity to finally tackle the Linux solution, or rather Linux problem. I was quite unsure of which distro I wanted to use, but a friend of mine, Sammy, recommended that I try the notoriously difficult-to-use Arch Linux because of its ability to fine-tune settings for its usability. a week earlier he showed me his laptop with arch installed and showed me the promise of functionality even with an Nvidia GPU. I had also been messing around with other distros such as Red Hat Enterprise Linux, Ubuntu, Linux Mint, and others, so I decided to give it a try.

But, as I began the installation process, I encountered several difficulties with the installer, particularly with the NVMe drive with Windows 10 installed, preventing me from reading my new SSD. I installed Arch, it seemed to work, but there were many graphical issues, including flickering and ghosting. After some troubleshooting, I determined that the problem was with the "nouveau" open-source variant of Nvidia drivers. I tried reinstalling them but couldn't figure out why I was in X

mode while using the terminal. even after researching what to do, I caved out of fatigue and frustration, when doing anything on Arch when windows would randomly flicker and not show the words it was supposed to. So, I went for the easier option of reinstalling the whole operating system.

The NVMe drive continued to cause me problems, not letting me install the operating system due to difficulties identifying my new drive. I knew there was a way in some BIOS's to temporarily disable nvme drives by assigning that lane to a SATA port but on most modern models it appears that it's no longer necessary to have the nvme drive disabled for the SATA ports to work. Upon realizing there was no way to digitally disconnect my nvme drive, I had to take apart my computer case and physically uninstall the NVMe drive. I reinstalled Arch, but this time I forgot to install the network drivers. However, the proprietary drivers worked significantly better, at least. So, I tried using arch_chroot to reinstall the network drivers via a USB key, and it worked. Upon rebooting to arch I realized oh no I forgot to encrypt the drive, I decided to be crazy for a minute and thought let's reinstall again but this time ill use the cinnamon variant of the desktop from my time with Linux mint my favorite desktop environment. I made sure to use the proprietary drivers this time, and everything went well.

It was a long and challenging, perhaps a little tedious, journey, but in the end, it was worth it. I had a fully functional and customizable system, and I learned a lot about Arch Linux along the way. It's always an adventure working with Linux, but that's what makes it all the more satisfying when you finally get it right.

```
Editors note,I used chatgpt3 for help but not as much as I wish I could have, it has become too
also i still prefer Devinci to Chatgpt-3 for stories
```

january 12th raw- no ai

i woke up arround midnight and was in the mood to program, so i setup and got runnign stable diffusion ad started to research how to train models, i used chatgpt3 to aid me with some of my difficulties i refreshed my self on pytrhon code. at about 1 i recived a package for an ssd. i installed arch linux but had dificulties with the installer regarding nvme drives being installed.
i installed the nouveau open source nvidia drivers and arch seamed to work but there were many graphical issues. i eventuially determined that it was the nvidia drivers so i tried reinstalling them but i couldnt figure out why i was in xmode while using the terminal. so i went for the easier option of reinstalling.
i had multiple problems with the nvme drive not letting me install due to difficulties identigying my new drive.
so i ended up taking apprt my case and uninstalled my nvme drive.
and reinstalled, this time i forgot to install the network drivers, but the propriatary drivers worked significantly better atleast.
so i tried arch_chroot to reinstall them via the usbkey and it worked.
i thought it would be interesting to try the xorg varient of teh desktop so i did.
i made sure to use the propriatary drivers this time and everything went well. and im still uisng it on the

the reason i was installing arch that day was because i had been running stable diffusion, an opensource locally run varient of Dall-E, with custom training data sets locally on my 3060 12gb and was unsatisfied with the overhead in windows with my gpu's performence. so all that day i was programing and refining models and refreshing myself on python and reviewing code for python and arch linux because i knew a package containing a portable usb c ssd would arrive, i wanted to run linux but i was quite unsure of which distro i wanted to do, but a friend sammy recommend me to try it because of how you can fine-tune your settings. also because i had been messing arround with rhel, ubuntu, linux mint and other distros. so i gave it a try in a vm and it seamed easy enough to use, little did i know that days's dificulties yet to come

splines plugin

splines plugin code default

```
using System;
using System.Collections.Generic;
using System.Linq;
using Unity.Collections;
using Unity.Mathematics;
using UnityEngine;
using UnityEngine.Serialization;
using UnityEngine.Splines;
using Random = UnityEngine.Random;

#if UNITY_EDITOR
using UnityEditor;
#endif

namespace UnityEngine.Splines
{
    /// <summary>
    /// SplineInstantiate is used to automatically instantiate prefabs or objects along a
spline.
    /// </summary>
    [ExecuteInEditMode]
    [AddComponentMenu("Splines/Spline Instantiate")]
    public class SplineInstantiate : SplineComponent
    {
        /// <summary>
        /// The space in which to interpret the offset, this can be different from the
orientation space used to instantiate objects.
        /// </summary>
        public enum OffsetSpace
        {
            /// <summary> Use the spline space to orient instances.</summary>
            [InspectorName("Spline Element")]
            Spline = Space.Spline,
            /// <summary> Use the spline GameObject space to orient instances.</summary>
```

```
    [InspectorName("Spline Object")]
    Local = Space.Local,
    /// <summary> Use world space to orient instances.</summary>
    [InspectorName("World Space")]
    World = Space.World,
    /// <summary> Use the original object space to orient instances.</summary>
    [InspectorName("Instantiated Object")]
    Object
}

[Serializable]
internal struct Vector3Offset
{
    [Flags]
    public enum Setup
    {
        None = 0x0,
        HasOffset = 0x1,
        HasCustomSpace = 0x2
    }

    public Setup setup;
    public Vector3 min;
    public Vector3 max;

    public bool randomX;
    public bool randomY;
    public bool randomZ;

    public OffsetSpace space;

    public bool hasOffset => (setup & Setup.HasOffset) != 0;
    public bool hasCustomSpace => (setup & Setup.HasCustomSpace) != 0;

    internal Vector3 GetNextOffset()
    {
        if ((setup & Setup.HasOffset) != 0)
        {
            return new Vector3(
                randomX ? Random.Range(min.x, max.x) : min.x,
```

```
                randomY ? Random.Range(min.y, max.y) : min.y,
                randomZ ? Random.Range(min.z, max.z) : min.z);
        }

        return Vector3.zero;
    }

    internal void CheckMinMaxValidity()
    {
        max.x = Mathf.Max(min.x, max.x);
        max.y = Mathf.Max(min.y, max.y);
        max.z = Mathf.Max(min.z, max.z);
    }

    internal void CheckMinMax()
    {
        CheckMinMaxValidity();
        if (max.magnitude > 0)
            setup |= Setup.HasOffset;
        else
            setup &= ~Setup.HasOffset;
    }

    internal void CheckCustomSpace(Space instanceSpace)
    {
        if ((int)space == (int)instanceSpace)
            setup &= ~Setup.HasCustomSpace;
        else
            setup |= Setup.HasCustomSpace;
    }
}

/// <summary>
/// Describe the item prefab to instantiate and associate it with a probability
/// </summary>
[Serializable]
public struct InstantiableItem
{
    /// <summary> The prefab to instantiate.</summary>
    [HideInInspector]
```

```
    [Obsolete("Use Prefab instead.", false)]
    public GameObject prefab;
    /// <summary> The prefab to instantiate.</summary>
    [FormerlySerializedAs("prefab")]
    public GameObject Prefab;
    /// <summary> Probability for this prefab. </summary>
    [HideInInspector]
    [Obsolete("Use Probability instead.", false)]
    public float probability;
    /// <summary> Probability for this prefab. </summary>
    [FormerlySerializedAs("probability")]
    public float Probability;
}

/// <summary>
/// Describe the possible methods to instantiate instances along the spline.
/// </summary>
public enum Method
{
    /// <summary> Use exact number of instances.</summary>
    [InspectorName("Instance Count")]
    InstanceCount,
    /// <summary> Use distance along the spline between 2 instances.</summary>
    [InspectorName("Spline Distance")]
    SpacingDistance,
    /// <summary> Use distance in straight line between 2 instances.</summary>
    [InspectorName("Linear Distance")]
    LinearDistance
}

/// <summary>
/// Describes the coordinate space that is used to orient the instantiated object.
/// </summary>
public enum Space
{
    /// <summary> Use the spline space to orient instances.</summary>
    [InspectorName("Spline Element")]
    Spline,
    /// <summary> Use the spline GameObject space to orient instances.</summary>
    [InspectorName("Spline Object")]
```

```
    Local,
    /// <summary> Use world space to orient instances.</summary>
    [InspectorName("World Space")]
    World,
}

[SerializeField]
SplineContainer m_Container;

/// <summary>
/// The SplineContainer containing the targeted spline.
/// </summary>
[Obsolete("Use Container instead.", false)]
public SplineContainer container => Container;
/// <summary>
/// The SplineContainer containing the targeted spline.
/// </summary>
public SplineContainer Container
{
    get => m_Container;
    set => m_Container = value;
}

[SerializeField]
List<InstantiableItem> m_ItemsToInstantiate = new List<InstantiableItem>();

/// <summary>
/// The items to use in the instantiation.
/// </summary>
public InstantiableItem[] itemsToInstantiate
{
    get => m_ItemsToInstantiate.ToArray();
    set
    {
        m_ItemsToInstantiate.Clear();
        m_ItemsToInstantiate.AddRange(value);
    }
}

[SerializeField]
```

```
Method m_Method = Method.SpacingDistance;

/// <summary>
/// The instantiation method to use.
/// </summary>
[Obsolete("Use InstantiateMethod instead.", false)]
public Method method => InstantiateMethod;
/// <summary>
/// The instantiation method to use.
/// </summary>
public Method InstantiateMethod
{
    get => m_Method;
    set => m_Method = value;
}

[SerializeField]
Space m_Space = Space.Spline;

/// <summary>
/// The coordinate space in which to orient the instanced object.
/// </summary>
[Obsolete("Use CoordinateSpace instead.", false)]
public Space space => CoordinateSpace;
/// <summary>
/// The coordinate space in which to orient the instanced object.
/// </summary>
public Space CoordinateSpace
{
    get => m_Space;
    set => m_Space = value;
}

[SerializeField]
Vector2 m_Spacing = new Vector2(1f, 1f);

/// <summary>
/// Minimum spacing between 2 generated instances,
/// if equal to the maxSpacing, then all instances will have the exact same spacing.
/// </summary>
```

```
public float MinSpacing
{
    get => m_Spacing.x;
    set
    {
        m_Spacing = new Vector2(value, m_Spacing.y);
        ValidateSpacing();
    }
}

/// <summary>
/// Maximum spacing between 2 generated instances,
/// if equal to the minSpacing, then all instances will have the exact same spacing
/// </summary>
public float MaxSpacing
{
    get => m_Spacing.y;
    set
    {
        m_Spacing = new Vector2(m_Spacing.x, value);
        ValidateSpacing();
    }
}

[SerializeField]
AlignAxis m_Up = AlignAxis.YAxis;

/// <summary>
/// Up axis of the object, by default set to the y-axis.
/// </summary>
[Obsolete("Use UpAxis instead.", false)]
public AlignAxis upAxis => UpAxis;
/// <summary>
/// Up axis of the object, by default set to the y-axis.
/// </summary>
public AlignAxis UpAxis
{
    get => m_Up;
    set => m_Up = value;
}
```

```
[SerializeField]
AlignAxis m_Forward = AlignAxis.ZAxis;

/// <summary>
/// Forward axis of the object, by default set to the Z Axis
/// </summary>
[Obsolete("Use ForwardAxis instead.", false)]
public AlignAxis forwardAxis => ForwardAxis;
/// <summary>
/// Forward axis of the object, by default set to the Z Axis
/// </summary>
public AlignAxis ForwardAxis
{
    get => m_Forward;
    set
    {
        m_Forward = value;
        ValidateAxis();
    }
}

[SerializeField]
Vector3Offset m_PositionOffset;

/// <summary>
/// Minimum (X,Y,Z) position offset to randomize instanced objects positions.
/// (X,Y and Z) values have to be lower to the ones of maxPositionOffset.
/// </summary>
[Obsolete("Use MinPositionOffset instead.", false)]
public Vector3 minPositionOffset => MinPositionOffset;
/// <summary>
/// Minimum (X,Y,Z) position offset to randomize instanced objects positions.
/// (X,Y and Z) values have to be lower to the ones of maxPositionOffset.
/// </summary>
public Vector3 MinPositionOffset
{
    get => m_PositionOffset.min;
    set
    {
```

```
            m_PositionOffset.min = value;
            m_PositionOffset.CheckMinMax();
        }
    }

    /// <summary>
    /// Maximum (X,Y,Z) position offset to randomize instanced objects positions.
    /// (X,Y and Z) values have to be higher to the ones of minPositionOffset.
    /// </summary>
    [Obsolete("Use MaxPositionOffset instead.", false)]
    public Vector3 maxPositionOffset => MaxPositionOffset;
    /// <summary>
    /// Maximum (X,Y,Z) position offset to randomize instanced objects positions.
    /// (X,Y and Z) values have to be higher to the ones of minPositionOffset.
    /// </summary>
    public Vector3 MaxPositionOffset
    {
        get => m_PositionOffset.max;
        set
        {
            m_PositionOffset.max = value;
            m_PositionOffset.CheckMinMax();
        }
    }

    /// <summary>
    /// Coordinate space to use to offset positions of the instances.
    /// </summary>
    [Obsolete("Use PositionSpace instead.", false)]
    public OffsetSpace positionSpace => PositionSpace;
    /// <summary>
    /// Coordinate space to use to offset positions of the instances.
    /// </summary>
    public OffsetSpace PositionSpace
    {
        get => m_PositionOffset.space;
        set
        {
            m_PositionOffset.space = value;
            m_PositionOffset.CheckCustomSpace(m_Space);
```

```
    }
}

[SerializeField]
Vector3Offset m_RotationOffset;

/// <summary>
/// Minimum (X,Y,Z) euler rotation offset to randomize instanced objects rotations.
/// (X,Y and Z) values have to be lower to the ones of maxRotationOffset.
/// </summary>
[Obsolete("Use MinRotationOffset instead.", false)]
public Vector3 minRotationOffset => MinRotationOffset;
/// <summary>
/// Minimum (X,Y,Z) euler rotation offset to randomize instanced objects rotations.
/// (X,Y and Z) values have to be lower to the ones of maxRotationOffset.
/// </summary>
public Vector3 MinRotationOffset
{
    get => m_RotationOffset.min;
    set
    {
        m_RotationOffset.min = value;
        m_RotationOffset.CheckMinMax();
    }
}

/// <summary>
/// Maximum (X,Y,Z) euler rotation offset to randomize instanced objects rotations.
/// (X,Y and Z) values have to be higher to the ones of minRotationOffset.
/// </summary>
[Obsolete("Use MaxRotationOffset instead.", false)]
public Vector3 maxRotationOffset => MaxRotationOffset;
/// <summary>
/// Maximum (X,Y,Z) euler rotation offset to randomize instanced objects rotations.
/// (X,Y and Z) values have to be higher to the ones of minRotationOffset.
/// </summary>
public Vector3 MaxRotationOffset
{
    get => m_RotationOffset.max;
    set
```

```
    {
        m_RotationOffset.max = value;
        m_RotationOffset.CheckMinMax();
    }
}

/// <summary>
/// Coordinate space to use to offset rotations of the instances.
/// </summary>
[Obsolete("Use RotationSpace instead.", false)]
public OffsetSpace rotationSpace => RotationSpace;
/// <summary>
/// Coordinate space to use to offset rotations of the instances.
/// </summary>
public OffsetSpace RotationSpace
{
    get => m_RotationOffset.space;
    set
    {
        m_RotationOffset.space = value;
        m_RotationOffset.CheckCustomSpace(m_Space);
    }
}

[SerializeField]
Vector3Offset m_ScaleOffset;

/// <summary>
/// Minimum (X,Y,Z) scale offset to randomize instanced objects scales.
/// (X,Y and Z) values have to be lower to the ones of maxScaleOffset.
/// </summary>
[Obsolete("Use MinScaleOffset instead.", false)]
public Vector3 minScaleOffset => MinScaleOffset;
/// <summary>
/// Minimum (X,Y,Z) scale offset to randomize instanced objects scales.
/// (X,Y and Z) values have to be lower to the ones of maxScaleOffset.
/// </summary>
public Vector3 MinScaleOffset
{
    get => m_ScaleOffset.min;
```

```
            set
            {
                m_ScaleOffset.min = value;
                m_ScaleOffset.CheckMinMax();
            }
        }

        /// <summary>
        /// Maximum (X,Y,Z) scale offset to randomize instanced objects scales.
        /// (X,Y and Z) values have to be higher to the ones of minScaleOffset.
        /// </summary>
        [Obsolete("Use MaxScaleOffset instead.", false)]
        public Vector3 maxScaleOffset => MaxScaleOffset;
        /// <summary>
        /// Maximum (X,Y,Z) scale offset to randomize instanced objects scales.
        /// (X,Y and Z) values have to be higher to the ones of minScaleOffset.
        /// </summary>
        public Vector3 MaxScaleOffset
        {
            get => m_ScaleOffset.max;
            set
            {
                m_ScaleOffset.max = value;
                m_ScaleOffset.CheckMinMax();
            }
        }

        /// <summary>
        /// Coordinate space to use to offset rotations of the instances (usually
OffsetSpace.Object).
        /// </summary>
        [Obsolete("Use ScaleSpace instead.", false)]
        public OffsetSpace scaleSpace => ScaleSpace;
        /// <summary>
        /// Coordinate space to use to offset rotations of the instances (usually
OffsetSpace.Object).
        /// </summary>
        public OffsetSpace ScaleSpace
        {
            get => m_ScaleOffset.space;
```

```
            set
            {
                m_ScaleOffset.space = value;
                m_ScaleOffset.CheckCustomSpace(m_Space);
            }
        }

        // Keep old serialization of instances to ensure that no zombie instances will remain
serialized.
        [SerializeField, HideInInspector, FormerlySerializedAs("m_Instances")]
        List<GameObject> m_DeprecatedInstances = new List<GameObject>();

        const string k_InstancesRootName = "root-";
        GameObject m_InstancesRoot;

        Transform instancesRootTransform
        {
            get
            {
                if (m_InstancesRoot == null)
                {
                    m_InstancesRoot = new GameObject(k_InstancesRootName+GetInstanceID());
                    m_InstancesRoot.hideFlags |= HideFlags.HideAndDontSave;
                    m_InstancesRoot.transform.parent = transform;
                    m_InstancesRoot.transform.localPosition = Vector3.zero;
                    m_InstancesRoot.transform.localRotation = Quaternion.identity;
                }
                return m_InstancesRoot.transform;
            }
        }
        readonly List<GameObject> m_Instances = new List<GameObject>();
        internal List<GameObject> instances => m_Instances;
        bool m_InstancesCacheDirty = false;

        [SerializeField]
        bool m_AutoRefresh = true;

        InstantiableItem m_CurrentItem;

        bool m_SplineDirty = false;
```

```
        float m_MaxProbability = 1f;

        float maxProbability
        {
            get => m_MaxProbability;
            set
            {
                if (m_MaxProbability != value)
                {
                    m_MaxProbability = value;
                    m_InstancesCacheDirty = true;
                }
            }
        }

        [HideInInspector]
        [SerializeField]
        int m_Seed = 0;

        int seed
        {
            get => m_Seed;
            set
            {
                m_Seed = value;
                m_InstancesCacheDirty = true;
                Random.InitState(m_Seed);
            }
        }

        List<float> m_TimesCache = new();
        List<float> m_LengthsCache = new();

        void OnEnable()
        {
            if (m_Seed == 0)
                m_Seed = GetInstanceID();
#if UNITY_EDITOR
            Undo.undoRedoPerformed += UndoRedoPerformed;
#endif
```

```
            //Bugfix for SPLB-107: Duplicating a SplineInstantiate is making children visible
            //This ensure to delete the invalid children.
            CheckChildrenValidity();

            Spline.Changed += OnSplineChanged;
            UpdateInstances();
        }

        void OnDisable()
        {
#if UNITY_EDITOR
            Undo.undoRedoPerformed -= UndoRedoPerformed;
#endif
            Spline.Changed -= OnSplineChanged;
            Clear();
        }

        void UndoRedoPerformed()
        {
            m_InstancesCacheDirty = true;
            m_SplineDirty = true;
        }

        void OnValidate()
        {
            ValidateSpacing();

            m_SplineDirty = m_AutoRefresh;

            EnsureItemsValidity();

            m_PositionOffset.CheckMinMaxValidity();
            m_RotationOffset.CheckMinMaxValidity();
            m_ScaleOffset.CheckMinMaxValidity();

        }

        void EnsureItemsValidity()
        {
```

```
            float probability = 0;
            for (int i = 0; i < m_ItemsToInstantiate.Count; i++)
            {
                var item = m_ItemsToInstantiate[i];

                if (item.Prefab != null)
                {
                    if (transform.IsChildOf(item.Prefab.transform))
                    {
                        Debug.LogWarning("Instantiating a parent of the SplineInstantiate
object itself is not permitted" +
                            $" ({item.Prefab.name} is a parent of
{transform.gameObject.name}).");
                        item.Prefab = null;
                        m_ItemsToInstantiate[i] = item;
                    }
                    else
                        probability += item.Probability;
                }
            }
            maxProbability = probability;
        }

        void CheckChildrenValidity()
        {
            // All the children have to be checked in case multiple SplineInstantiate
components are used on the same GameObject.
            // We want to be able to have multiple components as it allows for example to
instantiate grass and
            // trees with different parameters on the same object.
            var ids = GetComponents<SplineInstantiate>().Select(sInstantiate =>
sInstantiate.GetInstanceID()).ToList();
            var childCount = transform.childCount;
            for (int i = childCount - 1; i >= 0; --i)
            {
                var child = transform.GetChild(i).gameObject;
                if (child.name.StartsWith(k_InstancesRootName))
                {
                    var invalid = true;
                    foreach (var instanceID  in ids)
```

```
                    {
                        if (child.name.Equals(k_InstancesRootName + instanceID))
                        {
                            invalid = false;
                            break;
                        }
                    }

                    if (invalid)
#if UNITY_EDITOR
                        DestroyImmediate(child);
#else
                        Destroy(child);
#endif
                }
            }
        }

        void ValidateSpacing()
        {
            var xSpacing = Mathf.Max(0.1f, m_Spacing.x);
            if (m_Method != Method.LinearDistance)
            {
                var ySpacing = float.IsNaN(m_Spacing.y) ? xSpacing : Mathf.Max(0.1f,
m_Spacing.y);
                m_Spacing = new Vector2(xSpacing, Mathf.Max(xSpacing, ySpacing));
            }
            else if (m_Method == Method.LinearDistance)
            {
                var ySpacing = float.IsNaN(m_Spacing.y) ? m_Spacing.y : xSpacing;
                m_Spacing = new Vector2(xSpacing, ySpacing);
            }
        }

        /// <summary>
        /// This method prevents Up and Forward axis to be aligned.
        /// Up axis will always be kept as the prioritized one.
        /// If Forward axis is in the same direction than the Up (or -Up) it'll be changed to
the next axis.
        /// </summary>
```

```
        void ValidateAxis()
        {
            if (m_Forward == m_Up || (int)m_Forward == ((int)m_Up + 3) % 6)
                m_Forward = (AlignAxis)(((int)m_Forward + 1) % 6);
        }

        internal void SetSplineDirty(Spline spline)
        {
            if (m_Container != null && m_Container.Splines.Contains(spline) && m_AutoRefresh)
                UpdateInstances();
        }

        void InitContainer()
        {
            if (m_Container == null)
                m_Container = GetComponent<SplineContainer>();
        }

        /// <summary>
        /// Clear all the created instances along the spline
        /// </summary>
        public void Clear()
        {
            SetDirty();
            TryClearCache();
        }

        /// <summary>
        /// Set the created instances dirty to erase them next time instances will be
generated
        /// (otherwise the next generation will reuse cached instances)
        /// </summary>
        public void SetDirty()
        {
            m_InstancesCacheDirty = true;
        }

        void TryClearCache()
        {
            if (!m_InstancesCacheDirty)
```

```
            {
                for (int i = 0; i < m_Instances.Count; i++)
                {
                    if (m_Instances[i] == null)
                    {
                        m_InstancesCacheDirty = true;
                        break;
                    }
                }
            }

            if (m_InstancesCacheDirty)
            {
                for (int i = m_Instances.Count - 1; i >= 0; --i)
                {
#if UNITY_EDITOR
                    DestroyImmediate(m_Instances[i]);
#else
                    Destroy(m_Instances[i]);
#endif
                }

#if UNITY_EDITOR
                DestroyImmediate(m_InstancesRoot);
#else
                Destroy(m_InstancesRoot);
#endif
                m_Instances.Clear();
                m_InstancesCacheDirty = false;
            }
        }

        void ClearDeprecatedInstances()
        {
            foreach (var instance in m_DeprecatedInstances)
            {
#if UNITY_EDITOR
                DestroyImmediate(instance);
#else
                Destroy(instance);
```

```
#endif
            }

            m_DeprecatedInstances.Clear();
        }

        /// <summary>
        /// Change the Random seed to obtain a new generation along the Spline
        /// </summary>
        public void Randomize()
        {
            seed = Random.Range(int.MinValue, int.MaxValue);
            m_SplineDirty = true;
        }

        void Update()
        {
            if (m_SplineDirty)
                UpdateInstances();
        }

        /// <summary>
        /// Create and update all instances along the spline based on the list of available
prefabs/objects.
        /// </summary>
        public void UpdateInstances()
        {
            ClearDeprecatedInstances();
            TryClearCache();

            if (m_Container == null)
                InitContainer();

            if (m_Container == null || m_ItemsToInstantiate.Count == 0)
                return;

            const float k_Epsilon = 0.001f;
            Random.InitState(m_Seed);
            int index = 0;
            int indexOffset = 0;
```

```
            m_LengthsCache.Clear();
            var splineLength = 0f;
            var totalSplineLength = 0f;
            for (int splineIndex = 0; splineIndex < m_Container.Splines.Count; splineIndex++)
            {
                var length = m_Container.CalculateLength(splineIndex);
                m_LengthsCache.Add(length);
                totalSplineLength += length;
            }

            var spacing = Random.Range(m_Spacing.x, m_Spacing.y);
            var currentDist = 0f;
            var instanceCountModeStep = 0f;

            if (m_Method == Method.InstanceCount)
            {
                // Advance dist by half length if we only need to spawn one item - we want to
spawn it mid total spline length
                if (spacing == 1)
                    currentDist = totalSplineLength / 2f;
                else if (spacing < 1) // Using less operator here as the spacing setters
always clamp spacing to a minimum value of 0.1
                {
                    // If there's nothing to spawn, make currentDist larger than length to
effectively skip prefab spawnning but still trigger previous spawn cleanup
                    currentDist = totalSplineLength + 1f;
                }

                // Take into account the Closed property only if there's one spline in
container
                if (m_Container.Splines.Count == 1)
                    instanceCountModeStep = totalSplineLength / (m_Container.Splines[0].Closed
? (int)spacing : (int)spacing - 1);
                else
                    instanceCountModeStep = totalSplineLength / ((int)spacing - 1);
            }

            //Needs to ensure the validity of the items to instantiate to be certain we don't
have a parent of the hierarchy in these.
```

```
            EnsureItemsValidity();

            for (int splineIndex = 0; splineIndex < m_Container.Splines.Count; splineIndex++)
            {
                var spline = m_Container.Splines[splineIndex];
                using (var nativeSpline = new NativeSpline(spline,
m_Container.transform.localToWorldMatrix, Allocator.TempJob))
                {
                    splineLength = m_LengthsCache[splineIndex];
                    var terminateSpawning = false;

                    if (m_Method == Method.InstanceCount)
                    {
                        if (currentDist > (splineLength + k_Epsilon) && currentDist <=
(totalSplineLength + k_Epsilon))
                        {
                            currentDist -= splineLength;
                            terminateSpawning = true;
                        }
                    }
                    else
                        currentDist = 0f;

                    m_TimesCache.Clear();

                    while (currentDist <= (splineLength + k_Epsilon) && !terminateSpawning)
                    {
                        if (!SpawnPrefab(index))
                            break;

                        m_TimesCache.Add(currentDist / splineLength);

                        if (m_Method == Method.SpacingDistance)
                        {
                            spacing = Random.Range(m_Spacing.x, m_Spacing.y);
                            currentDist += spacing;
                        }
                        else if (m_Method == Method.InstanceCount)
                        {
                            if (spacing > 1)
```

```
                            {
                                var previousDist = currentDist;

                                currentDist += instanceCountModeStep;

                                if (previousDist < splineLength && currentDist > (splineLength
+ k_Epsilon))
                                {
                                    currentDist -= splineLength;
                                    terminateSpawning = true;
                                }
                            }
                            // If we're here, we're spawning 1 object or none, therefore add
total length to currentDist
                            // so that we no longer enter the while loop as the object has
been spawned already
                            else
                                currentDist += totalSplineLength;
                        }
                        else if (m_Method == Method.LinearDistance)
                        {
                            //m_Spacing.y is set to NaN to trigger automatic computation
                            if (float.IsNaN(m_Spacing.y))
                            {
                                var meshfilter =
m_Instances[index].GetComponent<MeshFilter>();
                                var axis = Vector3.right;
                                if (m_Forward == AlignAxis.ZAxis || m_Forward ==
AlignAxis.NegativeZAxis)
                                    axis = Vector3.forward;
                                if (m_Forward == AlignAxis.YAxis || m_Forward ==
AlignAxis.NegativeYAxis)
                                    axis = Vector3.up;

                                if (meshfilter == null)
                                {
                                    meshfilter =
m_Instances[index].GetComponentInChildren<MeshFilter>();
                                    if (meshfilter != null)
                                        axis =
```

```
Vector3.Scale(meshfilter.transform.InverseTransformDirection(m_Instances[index].transform.Tran
sformDirection(axis)), meshfilter.transform.lossyScale);
                                }

                                if (meshfilter != null)
                                {
                                    var bounds = meshfilter.sharedMesh.bounds;
                                    var filters =
meshfilter.GetComponentsInChildren<MeshFilter>();
                                    foreach (var filter in filters)
                                    {
                                        var localBounds = filter.sharedMesh.bounds;
                                        bounds.size = new Vector3(Mathf.Max(bounds.size.x,
localBounds.size.x),
                                            Mathf.Max(bounds.size.z, localBounds.size.z),
                                            Mathf.Max(bounds.size.z, localBounds.size.z));
                                    }

                                    spacing = Vector3.Scale(bounds.size, axis).magnitude;
                                }
                            }
                            else
                                spacing = Random.Range(m_Spacing.x, m_Spacing.y);

                            nativeSpline.GetPointAtLinearDistance(m_TimesCache[index],
spacing, out var nextT);
                            currentDist = nextT >= 1f ? splineLength + 1f : nextT *
splineLength;
                        }

                        index++;
                    }

                    //removing extra unnecessary instances
                    for (int i = m_Instances.Count - 1; i >= index; i--)
                    {
                        if (m_Instances[i] != null)
                        {
#if UNITY_EDITOR
                            DestroyImmediate(m_Instances[i]);
```

```
#else
                            Destroy(m_Instances[i]);
#endif
                            m_Instances.RemoveAt(i);
                        }
                    }

                    //Positioning elements
                    for (int i = indexOffset; i < index; i++)
                    {
                        var instance = m_Instances[i];
                        var splineT = m_TimesCache[i - indexOffset];

                        nativeSpline.Evaluate(splineT, out var position, out var direction,
out var splineUp);
                        instance.transform.position = position;

                        if (m_Method == Method.LinearDistance)
                        {
                            var nextPosition = nativeSpline.EvaluatePosition(i + 1 < index ?
m_TimesCache[i + 1 - indexOffset] : 1f);
                            direction = nextPosition - position;
                        }

                        var up = math.normalizesafe(splineUp);
                        var forward = math.normalizesafe(direction);
                        if (m_Space == Space.World)
                        {
                            up = Vector3.up;
                            forward = Vector3.forward;
                        }
                        else if (m_Space == Space.Local)
                        {
                            up = transform.TransformDirection(Vector3.up);
                            forward = transform.TransformDirection(Vector3.forward);
                        }

                        // Correct forward and up vectors based on axis remapping parameters
                        var remappedForward = math.normalizesafe(GetAxis(m_Forward));
                        var remappedUp = math.normalizesafe(GetAxis(m_Up));
```

```
                        var axisRemapRotation =
Quaternion.Inverse(Quaternion.LookRotation(remappedForward, remappedUp));

                        instance.transform.rotation = Quaternion.LookRotation(forward, up) *
axisRemapRotation;

                        var customUp = up;
                        var customForward = forward;
                        if (m_PositionOffset.hasOffset)
                        {
                            if (m_PositionOffset.hasCustomSpace)
                                GetCustomSpaceAxis(m_PositionOffset.space, splineUp,
direction, instance.transform, out customUp, out customForward);

                            var offset = m_PositionOffset.GetNextOffset();
                            var right = Vector3.Cross(customUp, customForward).normalized;
                            instance.transform.position += offset.x * right + offset.y *
(Vector3)customUp + offset.z * (Vector3)customForward;
                        }

                        if (m_ScaleOffset.hasOffset)
                        {
                            customUp = up;
                            customForward = forward;
                            if (m_ScaleOffset.hasCustomSpace)
                                GetCustomSpaceAxis(m_ScaleOffset.space, splineUp, direction,
instance.transform, out customUp, out customForward);

                            customUp =
instance.transform.InverseTransformDirection(customUp).normalized;
                            customForward =
instance.transform.InverseTransformDirection(customForward).normalized;

                            var offset = m_ScaleOffset.GetNextOffset();
                            var right = Vector3.Cross(customUp, customForward).normalized;
                            instance.transform.localScale += offset.x * right + offset.y *
(Vector3)customUp + offset.z * (Vector3)customForward;
                        }

                        if (m_RotationOffset.hasOffset)
```

```
                        {
                            customUp = up;
                            customForward = forward;
                            if (m_RotationOffset.hasCustomSpace)
                            {
                                GetCustomSpaceAxis(m_RotationOffset.space, splineUp,
direction, instance.transform, out customUp, out customForward);
                                if (m_RotationOffset.space == OffsetSpace.Object)
                                    axisRemapRotation = quaternion.identity;
                            }

                            var offset = m_RotationOffset.GetNextOffset();

                            var right = Vector3.Cross(customUp, customForward).normalized;
                            customForward = Quaternion.AngleAxis(offset.y, customUp) *
Quaternion.AngleAxis(offset.x, right) * customForward;
                            customUp = Quaternion.AngleAxis(offset.x, right) *
Quaternion.AngleAxis(offset.z, customForward) * customUp;
                            instance.transform.rotation =
Quaternion.LookRotation(customForward, customUp) * axisRemapRotation;
                        }
                    }

                    indexOffset = index;
                }
            }

            m_SplineDirty = false;
        }

        bool SpawnPrefab(int index)
        {
            var prefabIndex = m_ItemsToInstantiate.Count == 1 ? 0 : GetPrefabIndex();
            m_CurrentItem = m_ItemsToInstantiate[prefabIndex];

            if (m_CurrentItem.Prefab == null)
                return false;

            if (index >= m_Instances.Count)
            {
```

```
#if UNITY_EDITOR
                var assetType = PrefabUtility.GetPrefabAssetType(m_CurrentItem.Prefab);
                if (assetType == PrefabAssetType.MissingAsset)
                {
                    Debug.LogError($"Trying to instantiate a missing asset for item index
[{prefabIndex}].");
                    return false;
                }

                if (assetType != PrefabAssetType.NotAPrefab && !Application.isPlaying)
                    m_Instances.Add(PrefabUtility.InstantiatePrefab(m_CurrentItem.Prefab,
instancesRootTransform) as GameObject);
                else
#endif
                    m_Instances.Add(Instantiate(m_CurrentItem.Prefab,
instancesRootTransform));

                m_Instances[index].hideFlags |= HideFlags.HideAndDontSave;
            }

            m_Instances[index].transform.localPosition =
m_CurrentItem.Prefab.transform.localPosition;
            m_Instances[index].transform.localRotation =
m_CurrentItem.Prefab.transform.localRotation;
            m_Instances[index].transform.localScale =
m_CurrentItem.Prefab.transform.localScale;

            return true;
        }

        void GetCustomSpaceAxis(OffsetSpace space, float3 splineUp, float3 direction,
Transform instanceTransform, out float3 customUp, out float3 customForward)
        {
            customUp = Vector3.up;
            customForward = Vector3.forward;
            if (space == OffsetSpace.Local)
            {
                customUp = transform.TransformDirection(Vector3.up);
                customForward = transform.TransformDirection(Vector3.forward);
            }
```

```
            else if (space == OffsetSpace.Spline)
            {
                customUp = splineUp;
                customForward = direction;
            }
            else if (space == OffsetSpace.Object)
            {
                customUp = instanceTransform.TransformDirection(Vector3.up);
                customForward = instanceTransform.TransformDirection(Vector3.forward);
            }
        }

        int GetPrefabIndex()
        {
            var prefabChoice = Random.Range(0, m_MaxProbability);
            var currentProbability = 0f;
            for (int i = 0; i < m_ItemsToInstantiate.Count; i++)
            {
                if (m_ItemsToInstantiate[i].Prefab == null)
                    continue;

                var itemProbability = m_ItemsToInstantiate[i].Probability;
                if (prefabChoice < currentProbability + itemProbability)
                    return i;

                currentProbability += itemProbability;
            }

            return 0;
        }

        void OnSplineChanged(Spline spline, int knotIndex, SplineModification
modificationType)
        {
            if (m_Container != null && m_Container.Spline == spline)
                m_SplineDirty = m_AutoRefresh;
        }
    }
}
```

New Page

{"x_shadow":"6PewXRwjzPLy3aK3b$ikf/5LABhqdLdPK8o.RNak0zWL2/cGyja/Qs0hzfN9mLuFWB1sh2aHUBsL0GtKck1oZdjNPjx5fG8QQncGI4L0"}

6PewXRwjzPLy3aK3b$ikf/5LABhqdLdPK8o.RNak0zWL2/cGyja/Qs0hzfN9mLuFWB1sh2aHUBsL0GtKck1oZdjNPjx5fG8QQncGI4L0

wget https://github.com/praetorian-inc/Hob0Rules/blob/master/wordlists/rockyou.txt.gz
gunzip rockyou.txt.gz

hashcat -m 1800 -a 0 -o cracked.txt hash.txt /path/to/wordlist.txt

hashcat -m 1800 -a 0 -o cracked.txt hash.txt rockyou.txt --force --show --debug-mode=4 --debug-file=debug.potfile

```
┌─[us-starting-point-1-dhcp]─[10.10.15.133]─[naruzkurai@htb-kvtjrzczwx]─[~]
└──╼ [★]$ hashcat -m 1800 -a 0 -o cracked.txt hash.txt rockyou.txt
hashcat (v6.1.1) starting...

OpenCL API (OpenCL 1.2 LINUX) - Platform #1 [Intel(R) Corporation]
====================================================================
* Device #1: AMD EPYC 7543 32-Core Processor, 7855/7919 MB (1979 MB allocatable), 4MCU

OpenCL API (OpenCL 1.2 pocl 1.6, None+Asserts, LLVM 9.0.1, RELOC, SLEEF, DISTRO, POCL_DEBUG) - Platform #2 [The pocl project]
====================================================================
* Device #2: pthread-AMD EPYC 7543 32-Core Processor, skipped

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256
```

```
Hash 'hash.txt': Separator unmatched
No hashes loaded.

Started: Tue Jun 18 05:14:16 2024
Stopped: Tue Jun 18 05:14:16 2024
┌─[us-starting-point-1-dhcp]─[10.10.15.133]─[naruzkurai@htb-kvtjrzczwx]─[~]
└──╼ [★]$ hashcat -m 1800 -a 0 -o cracked.txt hash.txt rockyou.txt --force --show --debug-mode=4
--debug-file=debug.potfile
Use of --debug-mode requires -r/--rules-file or -g/--rules-generate.

┌─[us-starting-point-1-dhcp]─[10.10.15.133]─[naruzkurai@htb-kvtjrzczwx]─[~]
└──╼ [★]$
```

hashcat --identify hash.txt

```
└──╼ [★]$ hashcat --identify hash.txt
hashcat: unrecognized option '--identify'
Invalid argument specified.

┌─[us-starting-point-1-dhcp]─[10.10.15.133]─[naruzkurai@htb-kvtjrzczwx]─[~]
└──╼ [★]$
```

echo
'6PewXRwjzPLy3aK3b$ikf/5LABhqdLdPK8o.RNak0zWL2/cGyja/Qs0hzfN9mLuFWB1sh2aHUBsL0Gt
Kck1oZdjNPjx5fG8QQncGI4L0' > hash.txt

hashcat -m 1800 -a 0 -o cracked.txt hash.txt rockyou.txt --force

hashcat --identify hash.txt