Animated Bezier Flow of Particles

Create, edit and modify, animate your particles flow in Unity Editor Scene. No coding required.

Supports undo, points insertion, editing of individual Control Points, animation of points inside unity editor (and from code), points renaming.

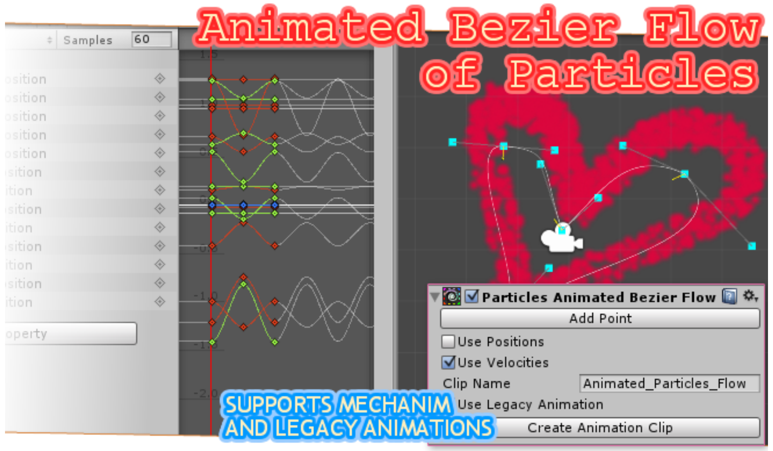
Created for Shuriken Particles Engine. Build for use from Editor UI for your integration needs no coding required. Provides clean pure .Net C# code for your developers happiness.

Main component is `ParticlesAnimatedBezierFlowController` which provides a curve with control points structured inside Unity3D scene hierarchy (meaning each controll point is a game object). Curve visual appearence is propagated across all control points, while all the behaviour and computational logic is kept inside a game object with `ParticlesAnimatedBezierFlowController` on it.

We created an experimental demo which runs only on WebGL 2 enabled browser such as Google Chrome 47

Tested on Unity versions:
- 5.3.1 - 4.7.0 - 3.5.7

Create an empty GameObject, Click on Component->Effects->MathArtCode->Animated Particles Bezier Flow is all the setup needed to strart editing your flow!
See video clips and embeded documentation for more installation and usage instructions.


Animated Bezier Flow of Particles
SUPPORTS MECHANIM AND LEGACY ANIMATIONS
Samples 60
Particles Animated Bezier Flow
Add Point
Create Animation Clip

Use Positions [ ]
Use Velocities [✓]
Clip Name Animated_Particles_Flow
Use Legacy Animation

ParticlesAnimatedBezierFlowInspector.cs

Particles Animated Bezier Flow Inspector
Provides a Unity Editor UI for `ParticlesAnimatedBezierFlowController`

License information: ASSET STORE TERMS OF SERVICE AND EULA

Developed by MathArtCode team, 2016

```
using System.Linq;
using Assets.Editor.BezierCurvedParticlesFlow.Utilities;
using UnityEditor;
using UnityEngine;

#if !UNITY_3_5
using Assets.Scripts.BezierCurvedParticlesFlow;
#endif

#if !UNITY_3_5
namespace Assets.Editor.BezierCurvedParticlesFlow {
#endif
```

ReSharper disable once UnusedMember.Global

```
    [CustomEditor(typeof (ParticlesAnimatedBezierFlowController))]
    public sealed class ParticlesAnimatedBezierFlowInspector : UnityEditor.Editor {
        private string _clipName = "Animated_Particles_Flow";
        private CurveDrawingHelper _curveDrawingHelperObject;
        private ParticlesAnimatedBezierFlowController _flowController;
        private bool _hasSelectedAnimation;
        private bool _useLegacyAnimations;
        private string _lastAnimationClipName;
        private int _videoLength = 2;
```

A safe ParticlesAnimatedBezierFlowController getter

```
        private ParticlesAnimatedBezierFlowController FlowController {
            get {
                if (_flowController == null) {
                    _flowController = target as ParticlesAnimatedBezierFlowController;
                }
                return _flowController;
            }
        }
```

A safe CurveDrawingHelper getter

```
        private CurveDrawingHelper CurveDrawingHelper {
            get {
                if (_curveDrawingHelperObject == null) {
                    _curveDrawingHelperObject =
                        new CurveDrawingHelper(this, FlowController);
                    CurveDrawingHelper.OnChangeInitiated += OnChangeStarted;
                    CurveDrawingHelper.OnChangeCommit += OnChangeCommit;
                }
                return _curveDrawingHelperObject;
            }
        }
```

Draws curve on in Scene window ReSharper disable once UnusedMember.Local ReSharper disable once InconsistentNaming

```
        private void OnSceneGUI() {
            CurveDrawingHelper.OnSceneGUI();
        }
```

Registers editor action checkpoint

```
        private void OnChangeStarted(string label) {
#if UNITY_3_5
        Undo.RegisterUndo(FlowController, label);
#else
            Undo.RecordObject(FlowController, label);
#endif
        }
```

Registers commits new checkpoint

```
        private void OnChangeCommit() {
            EditorUtility.SetDirty(FlowController);
        }
```

Draws inspector interface

```
        public override void OnInspectorGUI() {
            EditorGUI.BeginChangeCheck();
            CurveDrawingHelper.OnInspectorGUI();

#if UNITY_3_5
        EditorGUILayout.HelpBox(
            "You shall set this to the same value \"Particle System->Game Object->Max Particles\" is set. \nThis field is editable only for Unity 3.",
            MessageType.Warning);
        var maxParticles = EditorGUILayout.IntField("Max Particles", FlowController.MaxParticles);
        if(maxParticles != FlowController.MaxParticles) {
            OnChangeStarted("Max Particles count changed");
            FlowController.MaxParticles = maxParticles;
            OnChangeCommit();
        }
#endif
#if !UNITY_3_5
            _clipName = EditorGUILayout.TextField("Clip Name", _clipName);
            _useLagacyAnimations = GUILayout.Toggle(_useLagacyAnimations, "Use Legacy Animation");
#endif
```

```
            if (GUILayout.Button("Create Animation Clip")) {
#if !UNITY_3_5
                if (_useLagacyAnimations) {
#endif
                    UseLegacyAnimations();
#if !UNITY_3_5
                } else {
                    var clip = new AnimationClip();
                    FillAnimation(clip);
                    AssetDatabase.CreateAsset(clip, "Assets/" + _clipName + ".anim");
                    AssetDatabase.SaveAssets();
                }
#endif
            }
        }

        private void UseLegacyAnimations() {
            var anim = FlowController.GetComponent<Animation>();

            if (anim == null) {
                Debug.LogError("Legacy Animation Clip was not created/added because ther is no Animation component on " +
                               FlowController.gameObject.name);
                return;
            }


            _videoLength = EditorGUILayout.IntField("Animation Length", _videoLength);

            if (!_hasSelectedAnimation && anim != null && anim.clip != null) {
                _hasSelectedAnimation = true;
                _clipName = anim.clip.name;
            } else if (anim != null && anim.clip == null) {
                _hasSelectedAnimation = false;
            }

            _clipName = EditorGUILayout.TextField("Animation Clip Name", _clipName);

            if (GUILayout.Button("Prepare Animation Clip")) {
#if UNITY_3_5
            Undo.RegisterUndo(anim, "Create Animation Clip");
#else
                Undo.RecordObject(anim, "Create Animation Clip");
#endif
                AnimationClip clip = null;

                var updateAnimationClip = anim.clip != null && anim.clip.name == _clipName;
                clip = updateAnimationClip ? anim.clip : new AnimationClip();
                FillAnimation(clip);

                if (!updateAnimationClip) {
                    anim.AddClip(clip, _clipName);
                }

                EditorUtility.SetDirty(anim);
            }
        }

        private void FillAnimation(AnimationClip clip) {
#if !UNITY_3_5
            clip.legacy = _useLagacyAnimations;
#endif

            FlowController.GetInformers().ForEach( informer => {
                var localPosition = informer.transform.localPosition;
                var path = AnimationUtility.CalculateTransformPath( informer.transform, FlowController.transform );
                clip.SetCurve( path, typeof( Transform ), "localPosition.x",
                    AnimationCurve.Linear( 0, localPosition.x, _videoLength, localPosition.x ) );
                clip.SetCurve( path, typeof( Transform ), "localPosition.y",
                    AnimationCurve.Linear( 0, localPosition.y, _videoLength, localPosition.y ) );
                clip.SetCurve( path, typeof( Transform ), "localPosition.z",
                    AnimationCurve.Linear( 0, localPosition.z, _videoLength, localPosition.z ) );
            } );
        }
    }

#if !UNITY_3_5
}
#endif
```

BezierCurvePointInspector.cs

Bezier Curve Point Inspector

Provides a Unity Editor UI for `BezierCurvePointInformer`

License information: ASSET STORE TERMS OF SERVICE AND EULA

Developed by MathArtCode team, 2016

```
using Assets.Scripts.BezierCurvedParticlesFlow.Utilities;
using UnityEditor;

#if !UNITY_3_5
namespace Assets.Editor.BezierCurvedParticlesFlow.Utilities {
#endif

[CustomEditor(typeof (BezierCurvePointInformer))]
class BezierCurvePointInspector : UnityEditor.Editor {
    private CurveDrawingHelper _drawer;
    private BezierCurvePointInformer _informer;

    private BezierCurvePointInformer informer {
        get {
            if (_informer == null) {
                _informer = target as BezierCurvePointInformer;
            }
            return _informer;
        }
    }

    private CurveDrawingHelper Drawer {
        get {
            if (_drawer == null) {
                _drawer = new CurveDrawingHelper(this, informer.Controller);
                _drawer.OnChangeInitiated += OnChangeStarted;
                _drawer.OnChangeCommit += OnChangeCommit;
            }
            return _drawer;
        }
    }

    private void OnSceneGUI() {
        Drawer.OnSceneGUI();
    }

    private void OnChangeStarted(string label) {
#if UNITY_3_5
        Undo.RegisterUndo(informer.Controller, label);
#else
            Undo.RecordObject( informer.Controller, label);
#endif
    }
```

Registers commits new checkpoint

```
    private void OnChangeCommit() {
        EditorUtility.SetDirty(informer.Controller);
    }

    public override void OnInspectorGUI() {
        if(Drawer.SelectedIndex != informer.Index) {
            Drawer.SelectedIndex = informer.Index;
        }
        Drawer.OnInspectorGUI();
    }
}

#if !UNITY_3_5
}
#endif
```

CurveDrawingHelper.cs

Curve Drawing Helper

This is an utilety class for all major calculations related to vectorized bezier curve scene UI representation.

License information: ASSET STORE TERMS OF SERVICE AND EULA

Developed by MathArtCode team, 2016

```
using Assets.Scripts.BezierCurvedParticlesFlow;

using UnityEditor;
using UnityEngine;

namespace Assets.Editor.BezierCurvedParticlesFlow.Utilities {
    public class CurveDrawingHelper {
        public delegate void ChangeChangeStart(string label);
```

Provides events for handling interactions

```
        public delegate void ChangeCommit();
```

UI style settings

```
        private const float HandleSize = 0.04f;
        private const float PickSize = 0.06f;

        private readonly ParticlesAnimatedBezierFlowController _flowController;
        private readonly Transform _handleTransform;

        private Quaternion _handleRotation;
```

Currently selected handle

```
        public int SelectedIndex = -1;

        public event ChangeChangeStart OnChangeInitiated;
        public event ChangeCommit OnChangeCommit;
```

Functions

Draws a dinamically movable UnityEditor point

```
        private Vector3 ShowPoint(int index) {
            var point = _handleTransform.TransformPoint(_flowController.BezierLogic.GetControlPoint(index));
            var size = HandleUtility.GetHandleSize(point);
```

```
    if (index == 0) {
        size *= 2f;
    }
    Handles.color = Color.cyan;

    if (Handles.Button(point, _handleRotation, size*HandleSize, size*PickSize, Handles.DotCap)) {
        SelectedIndex = index;
        var informer = _flowController.GetInformer(SelectedIndex);
        if (informer != null) {
            Selection.activeGameObject = informer.gameObject;
        } else {
            Debug.LogError(string.Format(
                "Particles Animated Bezier Flow Controller Informers are currupt! {0}", SelectedIndex));
        }
```

_client.Repaint();

```
    }
    if (SelectedIndex == index) {
        EditorGUI.BeginChangeCheck();
        point = Handles.DoPositionHandle(point, _handleRotation);
        if (EditorGUI.EndChangeCheck()) {
            if (OnChangeInitiated != null) {
                OnChangeInitiated("Move Point");
            }
            _flowController.BezierLogic.SetControlPoint(index, _handleTransform.InverseTransformPoint(point));
            if (OnChangeCommit != null) {
                OnChangeCommit();
            }
        }
    }
    return point;
}
```

Draws currently selected curve point Editor Inspector UI

```
private void DrawSelectedPointInspector() {
    GUILayout.Label("Selected Point");
    EditorGUI.BeginChangeCheck();
    var bezierLogic = _flowController.BezierLogic;

    var point = EditorGUILayout.Vector3Field("Position", bezierLogic.GetControlPoint(SelectedIndex));
    if (EditorGUI.EndChangeCheck()) {
        OnChangeInitiated("Move Point");
        bezierLogic.SetControlPoint(SelectedIndex, point);
        OnChangeCommit();
    }
    EditorGUI.BeginChangeCheck();
}
```

Draws a bezier curve in Scene space ReSharper disable once InconsistentNaming

```
public void OnSceneGUI() {
    if(Tools.pivotMode != PivotMode.Pivot) {
        Tools.pivotMode = PivotMode.Pivot;
    }

    _handleRotation = Tools.pivotRotation == PivotRotation.Local
        ? _handleTransform.rotation
        : Quaternion.identity;

    var p0 = ShowPoint(0);
    for(var i = 1; i < _flowController.BezierLogic.ControlPointCount; i += 3) {
        var p1 = ShowPoint(i);
        var p2 = ShowPoint(i + 1);
        var p3 = ShowPoint(i + 2);
```

```
                Handles.color = Color.gray;
                Handles.DrawLine(p0, p1);
                Handles.DrawLine(p2, p3);

                Handles.color = Color.yellow;
                Handles.ArrowCap(0, p0, Quaternion.LookRotation(p3 - p0), 0.25f);
#if !UNITY_3_5
                Handles.DrawBezier( p0, p3, p1, p2, Color.white, null, 1f );
#endif
                p0 = p3;
            }

#if UNITY_3_5
            var rasterize = 250;
            Handles.color = Color.gray;
            for(var i = 0; i < rasterize - 1; i++) {
                var pn = _handleTransform.TransformPoint(_flowController.BezierLogic.GetPoint(i / (float)rasterize));
                var pk = _handleTransform.TransformPoint(_flowController.BezierLogic.GetPoint((i + 1) / (float)rasterize));
                Handles.DrawLine(pn, pk);
            }
#endif

            var e = Event.current;
            switch(e.type) {
                case EventType.keyDown:
                    {
                        if(Event.current.keyCode == (KeyCode.Delete) || Event.current.keyCode == (KeyCode.Backspace)) {
                            if(SelectedIndex % 3 == 0 && SelectedIndex > 3 &&
                                SelectedIndex < _flowController.BezierLogic.ControlPointCount) {
                                e.Use();
                                if(OnChangeInitiated != null) {
                                    OnChangeInitiated("Remove Point");
                                }
                                _flowController.BezierLogic.RemovePoint(SelectedIndex, true);
                                if(OnChangeCommit != null) {
                                    OnChangeCommit();
                                }
                            }
                        }

                        break;
                    }
            }
        }

        public void OnInspectorGUI() {
            if(SelectedIndex >= 0 && SelectedIndex < _flowController.BezierLogic.ControlPointCount) {
                DrawSelectedPointInspector();
            }

            if(GUILayout.Button("Add Point")) {
                OnChangeInitiated("Add Point");
                _flowController.BezierLogic.AddPoint();
                OnChangeCommit();
            }

            var usePositions = GUILayout.Toggle(_flowController.UsePositions, "Use Positions");
            if(usePositions != _flowController.UsePositions) {
                OnChangeInitiated("Use Positions changed");
                _flowController.UsePositions = usePositions;
                OnChangeCommit();
            }
            var useVelocities = GUILayout.Toggle(_flowController.UseVelocities, "Use Velocities");
            if(useVelocities != _flowController.UseVelocities) {
                OnChangeInitiated("Use Velocities changed");
                _flowController.UseVelocities = useVelocities;
                OnChangeCommit();
            }
        }

        public CurveDrawingHelper(UnityEditor.Editor uiClient, ParticlesAnimatedBezierFlowController flowController) {
            _flowController = flowController;
            _handleTransform = _flowController.transform;
        }
    }
}
```

ParticlesAnimatedBezierFlowController.cs

Particles Animated Bezier Flow Controller

This is a component that controlls behaviour of particle system mounted to the same game object it is.

It controlls only particles path, it does not controll any other aspect ot their behaviour or loock.

License information: ASSET STORE TERMS OF SERVICE AND EULA

Developed by MathArtCode team, 2016

```
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;
using Assets.Scripts.BezierCurvedParticlesFlow.Utilities;
using UnityEngine;
#if UNITY_EDITOR
using UnityEditor;

#endif

#if !UNITY_3_5
namespace Assets.Scripts.BezierCurvedParticlesFlow {
#endif

[AddComponentMenu("Effects/MathArtCode/Animated Bezier Flow of Particles")]
[ExecuteInEditMode]
[RequireComponent(typeof (ParticleSystem))]
#if UNITY_3_5
[RequireComponent(typeof (Animation))]
#endif
public sealed class ParticlesAnimatedBezierFlowController : MonoBehaviour {
#if !UNITY_3_5
    [SerializeField]
    private bool _useLegacyAnimation = true;
#endif

    [SerializeField]
    private BezierLogic _bezierLogicLogic;

    private ParticleSystem.Particle[] _particles;

    private Dictionary<int, BezierCurvePointInformer> _informers = new Dictionary<int, BezierCurvePointInformer>();

    private ParticleSystem _system;

    [SerializeField]
    private bool _usePositions;

    [SerializeField]
    private bool _useVelocities;
```

```
#if UNITY_3_5
    [SerializeField]
    private int _maxParticles;
```

Unity3.5 ParticleSystem.MaxParticles

```
    public int MaxParticles {
        get { return _maxParticles; }
        set {
            _maxParticles = value;
            _particles = new ParticleSystem.Particle[_maxParticles];
        }
    }
#endif
```

Defines if pixel perfect positions as they are defined in bezier curve shall be used (which makes all particles follow one curve).

```
    public bool UsePositions {
        get { return _usePositions; }
        set { _usePositions = value; }
    }
```

Defines if each particle shall set its velocity relative to its life time (allowing you to use particle emmiter shape as flow form).

```
    public bool UseVelocities {
        get { return _useVelocities; }
        set { _useVelocities = value; }
    }
```

Hosts main curve related logic

```
    public BezierLogic BezierLogic {
        get {
            if (_bezierLogicLogic == null) {
                _bezierLogicLogic = new BezierLogic();
            }

            if (!_bezierLogicLogic.HasPointEventHandlers()) {
                _bezierLogicLogic.OnAdded += idx => {
                    var pointName = "";
                    var pointId = idx%3;
                    var isAngle = true;
                    var angleHost = idx;
                    if (pointId == 0) {
                        pointName = "control_point_" + angleHost/3;
                        isAngle = false;
                    } else if (pointId == 1) {
                        angleHost--;
                        pointName = "angle_out_" + angleHost/3;
                    } else if (pointId == 2) {
                        angleHost++;
                        pointName = "angle_in_" + angleHost/3;
                    }
                    var o = new GameObject {name = pointName};
#if UNITY_EDITOR
                    Undo.RegisterCreatedObjectUndo(o, pointName);
#endif
                    if (!isAngle) {
                        o.transform.position = transform.TransformPoint(BezierLogic.GetControlPoint(idx));
                        o.transform.parent = transform;
                    } else {
                        o.transform.position = transform.TransformPoint(BezierLogic.GetControlPoint(idx));
                        o.transform.parent = GetInformer(angleHost).transform;
                    }
#if UNITY_EDITOR
#if UNITY_3_5
```

```
                    Undo.RegisterSetTransformParentUndo(o.transform, o.transform.parent.transform, o.name);
#else
                    Undo.SetTransformParent(o.transform, o.transform.parent.transform, o.name);
#endif
#endif

                    var informer = o.AddComponent<BezierCurvePointInformer>();
                    informer.Controller = this;
                    informer.Index = idx;
                    _informers.Add(idx, informer);
#if UNITY_EDITOR
                    EditorUtility.SetDirty(informer.Controller);
#endif
                };

                _bezierLogicLogic.OnRemoved += idx => {
                    var bezierCurvePointInformer = GetInformer(idx);
                    if ((bezierCurvePointInformer != null) && (bezierCurvePointInformer.HasToBeDestroyed == null)) {
                        bezierCurvePointInformer.HasToBeDestroyed = true;
#if UNITY_EDITOR
                        DestroyImmediate(bezierCurvePointInformer.gameObject);
#else
                        Destroy(bezierCurvePointInformer.gameObject);
#endif
                    }
                };

                _bezierLogicLogic.OnMoved +=
                    idx => {
                        GetInformer(idx).transform.position = transform.TransformPoint(BezierLogic.GetControlPoint(idx));
                    };

                _bezierLogicLogic.OnRemoveCompleted += (start, count) => {
                    var range = Enumerable.Range(start, count);
                    var fixedInformers = _informers
                        .Where(pair => pair.Key > start && !range.Contains(pair.Key))
                        .ToDictionary(pair => pair.Key - count, pair => {
                            if (pair.Value != null) {
                                pair.Value.Index -= count;
                                var originalName = pair.Value.gameObject.name;

                                var id = Regex.Match(originalName, "(\\d+)(?!.*\\d)");
                                if (id.Success) {
                                    pair.Value.gameObject.name = Regex.Replace(originalName, "(\\d+)(?!.*\\d)",
                                        (int.Parse(id.Value) - 1).ToString());
                                }
                            }
                            return pair.Value;
                        });

                    _informers = _informers.Where(pair => pair.Key < start)
                        .Concat(fixedInformers)
                        .ToDictionary(pair => pair.Key, pair => pair.Value);
                };
            }
            return _bezierLogicLogic;
        }
    }
```

Allows to get a position which shall be taken by particle relative to its life time.

```
    private Vector3 GetPoint(float t) {
        return BezierLogic.GetPoint(t);
    }
```

Allows to get a velocity which shall be taken by particle relative to its life time.

```
    private Vector3 GetVelocity(float t) {
        return BezierLogic.GetVelocity(t);
    }
```

Gets GameObject particle system

```
    private void InitializeIfNeeded() {
        if(_system == null) {
            _system = GetComponent<ParticleSystem>();
        }

        if(_particles == null) {
#if !UNITY_3_5
                _particles = new ParticleSystem.Particle[_system.maxParticles];
#else
            _particles = new ParticleSystem.Particle[_maxParticles];
#endif
            _system.GetParticles(_particles);
        }
    }
```

Resets component to its default values ReSharper disable once UnusedMember.Local

```
    private void Reset() {
#if UNITY_3_5
        _maxParticles = 150;
#endif
        BezierLogic.Reset();

        _usePositions = true;
        _usePositions = true;
    }
```

ReSharper disable once UnusedMember.Local

```
    private void Start() {
        Awake();
    }

    private void Awake() {
        _informers = new Dictionary<int, BezierCurvePointInformer>();
    }
```

Updates `ParticleSystem` particles setting positions and velocities if needed ReSharper disable once UnusedMember.Local

```
    private void LateUpdate() {
        InitializeIfNeeded();

        var numParticlesAlive = _system.GetParticles(_particles);

        for (var i = 0; i < numParticlesAlive; i++) {
            var p = _particles[i];
            var t = (p.startLifetime - p.lifetime)/p.startLifetime;
            if (_useVelocities) {
                _particles[i].velocity = GetVelocity(t);
            }

            if (_usePositions) {
                _particles[i].position = GetPoint(t);
            }
        }

        _system.SetParticles(_particles, numParticlesAlive);
    }
```

ReSharper disable once UnusedMember.Local

```
    private void OnDestroy() {
        BezierLogic.CleanUp();
    }
```

Registers already existing controll point informer

```
    public void RegisterInformer(int idx, BezierCurvePointInformer informer) {
        if(_informers.ContainsKey(idx)) {
            _informers[idx] = informer; // All objects get recreated on GO Destruction Undo
        } else {
            _informers.Add(idx, informer);
        }
    }
```

Returns an BezierCurvePointInformer that correlates to given index, can be null.

```
    public BezierCurvePointInformer GetInformer(int idx) {
        BezierCurvePointInformer informer;
        _informers.TryGetValue(idx, out informer);
        return informer;
    }

    public List<BezierCurvePointInformer> GetInformers() {
        return _informers.Values.ToList();
    }

}

#if !UNITY_3_5
}
#endif
```

BezierCurvePointInformer.cs

Bezier Curve Point Informer

This bahaviour is an informer that tells `ParticlesAnimatedBezierFlowController` mounted on some parent about bezier curve controll point position

License information: ASSET STORE TERMS OF SERVICE AND EULA

Developed by MathArtCode team, 2016

```
using System;
using UnityEngine;

#if UNITY_EDITOR
#endif

#if !UNITY_3_5
namespace Assets.Scripts.BezierCurvedParticlesFlow.Utilities {
#endif
```

A Control Point proxy object that is animatable inside unity editor animator Must be on a child of a GameObject containing `ParticlesAnimatedBezierFlowController`

```
[ExecuteInEditMode]
public sealed class BezierCurvePointInformer : MonoBehaviour {
    private Vector3 _lastPosition;

    [SerializeField]
    public ParticlesAnimatedBezierFlowController Controller;

    public bool? HasToBeDestroyed;

    [SerializeField]
    public int Index;
```

Correlates positioning of current element with hast `ParticlesAnimatedBezierFlowController`

```
    private void UpdatePosition() {
        var position = Controller.transform.InverseTransformPoint(transform.position);
        try {
            if (Controller.BezierLogic.GetControlPoint(Index) != position) {
                Controller.BezierLogic.SetControlPoint(Index, position, false);
            }
        } catch (IndexOutOfRangeException) {}
        _lastPosition = transform.localPosition;
    }

    private void InitializeIfNeeded() {
        if(Controller != null) {
            var parent = transform.parent;
            while(parent != null) {
                var component = parent.GetComponent<ParticlesAnimatedBezierFlowController>();
                if(component != null) {
                    Controller = component;
                    break;
                }
                parent = parent.transform.parent;
            }
            if(Controller == null) {
                Debug.LogError("Could not find a ParticlesAnimatedBezierFlowController in object parents");
            } else {
                Controller.RegisterInformer(Index, this);
                UpdatePosition();
            }
        }
    }
```

ReSharper disable once UnusedMember.Local

```
    private void Start() {
        Awake();
    }
```

Registers Object in ParticlesAnimatedBezierFlowController

```
    private void Awake() {
        InitializeIfNeeded();
        _lastPosition = transform.localPosition;
        HasToBeDestroyed = null;
    }
```

ReSharper disable once UnusedMember.Local

```
    private void Update() {
        InitializeIfNeeded();
        if (transform.localPosition != _lastPosition) {
            UpdatePosition();
        }
    }
```

ReSharper disable once UnusedMember.Local

```
    private void OnDestroy() {
        InitializeIfNeeded();
        if (HasToBeDestroyed == null) {
            HasToBeDestroyed = false;
        }
#if UNITY_EDITOR
        if ((Event.current != null) && (!Event.current.isMouse) && (Event.current.commandName == "SoftDelete"))
```

Project Open Scene == false, Game Object delete key == true

```
        {
#endif
            if (!HasToBeDestroyed.Value) {
                if (Index == 0 || Index == Controller.BezierLogic.ControlPointCount - 1) {
                    Debug.LogError("One shall not remove Bezier curve End Points! Please Undo!", Controller);
                } else if (Index%3 != 0) {
                    Debug.LogError("One shall not remove Bezier curve Angle Points! Please Undo!", Controller);
                }
            }

            if (Index%3 == 0 && !HasToBeDestroyed.Value) {
                Controller.BezierLogic.RemovePoint(Index, true);
            }
#if UNITY_EDITOR
        }
#endif
    }

}

#if !UNITY_3_5
}
#endif
```

BezierLogic.cs

Bezier Logic
This is an utilety class for all major calculations related to vectorized bezier curve.

License information: ASSET STORE TERMS OF SERVICE AND EULA

Developed by MathArtCode team, 2016

```
using System;
using System.Linq;
using UnityEngine;

namespace Assets.Scripts.BezierCurvedParticlesFlow.Utilities {
    [Serializable]
    public sealed class BezierLogic {
        public delegate void ControllPointsChange(int index);

        public delegate void OnRemoveComplete(int begin, int count);

        [SerializeField]
        private Vector3[] _points;
```

All points are handeled inside a single array - positioning (end points) have indexes 0, 3, 6, ... - angles (control points) have indexes 1,2, 4,5, ...

```
        private Vector3[] Points {
            get {
                if (_points == null) {
                    Reset();
                }
                return _points;
            }
        }
```

Provides ammount of all points (end points + control pouints in AI terminolagy)

```
        public int ControlPointCount {
            get { return Points.Length; }
        }
```

Provides ammount of all curved segments of Bezier curve

```
        private int CurveCount {
            get { return (Points.Length - 1)/3; }
        }
```

Called on each controll point creation, first is called for main CP, then for angle related controll points

```
        public event ControllPointsChange OnAdded;
```

Called on CP move

Note: one event event is sent per controll point, meaning there are no events for angle points when main CP is moved

```
        public event ControllPointsChange OnMoved;
```

Called on each controll point removal, first is called for angles and then for main controll point

```
public event ControllPointsChange OnRemoved;
```

Sends a complete range of removed items

```
public event OnRemoveComplete OnRemoveCompleted;
```

Mathematical core of Bezier Logic

```
private static Vector3 GetPoint(Vector3 p0, Vector3 p1, Vector3 p2, Vector3 p3, float t) {
    t = Mathf.Clamp01(t);
    var oneMinusT = 1f - t;
    return
        oneMinusT * oneMinusT * oneMinusT * p0 +
        3f * oneMinusT * oneMinusT * t * p1 +
        3f * oneMinusT * t * t * p2 +
        t * t * t * p3;
}

private static Vector3 GetFirstDerivative(Vector3 p0, Vector3 p1, Vector3 p2, Vector3 p3, float t) {
    t = Mathf.Clamp01(t);
    var oneMinusT = 1f - t;
    return
        3f * oneMinusT * oneMinusT * (p1 - p0) +
        6f * oneMinusT * t * (p2 - p1) +
        3f * t * t * (p3 - p2);
}
```

Main Bezier Logic

Returns a coordinate point relative to curve 'time' where 0 is bezier curve beginning and 1 is end of the bezier curve (length vise)

```
public Vector3 GetPoint(float t) {
    if (_points.Length < 4) {
        return Vector3.one;
    }
    int i;
    if (t >= 1f) {
        t = 1f;
        i = _points.Length - 4;
    } else {
        t = Mathf.Clamp01(t)*CurveCount;
        i = (int) t;
        t -= i;
        i *= 3;
    }
    return GetPoint(_points[i], _points[i + 1], _points[i + 2], _points[i + 3], t);
}
```

Returns a velocity point relative to curve 'time' where 0 is bezier curve beginning and 1 is end of the bezier curve (length vise)

```
public Vector3 GetVelocity(float t) {
    if (_points.Length < 4) {
        return Vector3.one;
    }
    int i;
    if (t >= 1f) {
        t = 1f;
        i = _points.Length - 4;
    } else {
        t = Mathf.Clamp01(t)*CurveCount;
        i = (int) t;
        t -= i;
        i *= 3;
    }
    return
```

```
        GetFirstDerivative(_points[i], _points[i + 1], _points[i + 2], _points[i + 3], t);
}
```

Returns a normalized velocity point relative to curve 'time' where 0 is bezier curve beginning and 1 is end of the bezier curve (length vise)

```
public Vector3 GetDirection(float t) {
    return GetVelocity(t).normalized;
}

public void CleanUp() {
    if (_points != null) {
        for (var i = 0; i < _points.Length; i += 3) {
            RemovePoint(i, false);
        }
        if (OnRemoveCompleted != null) {
            OnRemoveCompleted(0, _points.Length);
        }
    }
}
```

Resets component to its default 2 points (forming a straight line)

```
public void Reset() {
    CleanUp();

    _points = new[] {
        new Vector3(0f, 0f, 0f),
        new Vector3(2f, 0f, 0f),
        new Vector3(3f, 0f, 0f),
        new Vector3(4f, 0f, 0f)
    };

    if (OnAdded != null) {
        OnAdded(0);
        OnAdded(1);
        OnAdded(3);
        OnAdded(2);
    }
}
```

Returns a point by its Index - positioning (end points) have indexes 0, 3, 6, ... - angles (control points) have indexes 1,2, 4,5, ...

```
public Vector3 GetControlPoint(int index) {
    if (Points.Length <= index) {
        throw new IndexOutOfRangeException();
    }

    return Points[index];
}
```

Sets a point value by its Index - positioning (end points) have indexes 0, 3, 6 ... - angles (control points) have indexes 1,2, 4,5, ...

```
public void SetControlPoint(int index, Vector3 point, bool fireEvent = true) {
    var delta = point - Points[index];

    if (index%3 == 0) {
        if (index > 0) {
            var cpIdxBack = index - 1;
            _points[cpIdxBack] += delta;
        }
        var cpIdxFront = index + 1;
        if (cpIdxFront < _points.Length) {
            _points[cpIdxFront] += delta;
        }
    }

    Points[index] = point;
    if (fireEvent && OnMoved != null) {
            OnMoved(index);
        }
    }
```

Unity Editor Undo clears event handlers

```
    public bool HasPointEventHandlers() {
        var result = OnAdded != null && OnMoved != null && OnRemoved != null && OnRemoveCompleted != null;
        return result;
    }
```

Adds a position point + 2 angle points

```
    public void AddPoint() {
        var length = Points.Length;
        var point = _points[length - 1];
        var velocity = GetDirection(1f);
        Array.Resize(ref _points, length + 3);
        length = Points.Length;

        point += velocity;
        _points[length - 3] = point;
        point += velocity;
        _points[length - 2] = point;
        point += velocity;
        _points[length - 1] = point;

        if (OnAdded != null) {
            OnAdded(length - 3);
            OnAdded(length - 1);
            OnAdded(length - 2);
        }
    }
```

Removes a position point + 2 angle points. gets an end point Index as an argument - positioning (end points) have indexes 0, 3, 6...

```
    public void RemovePoint(int selectedIndex, bool doRemoval) {
        var points = Points.ToList();
        var atEnd = false;
        var atStart = false;
        var removeRangeStart = selectedIndex;
        var removeRangeCount = 3;

        if (selectedIndex == 0) {
            atStart = true;
            removeRangeCount = 2;
        } else if (selectedIndex == _points.Length - 1) {
            atEnd = true;
            removeRangeStart -= 1;
            removeRangeCount = 2;
        } else {
            removeRangeStart = selectedIndex - 1;
        }

        if ((removeRangeStart < 0) || ((removeRangeStart + removeRangeCount) > _points.Length)) {
            return; // Undo specifics
        }

        if (doRemoval) {
            points.RemoveRange(removeRangeStart, removeRangeCount);
            _points = points.ToArray();
        }

        if (OnRemoved != null) {
            if (atStart) {
                OnRemoved(selectedIndex + 1);
            } else if (atEnd) {
                    OnRemoved(selectedIndex - 1);
                } else {
                    OnRemoved(selectedIndex - 1);
                    OnRemoved(selectedIndex + 1);
                }
                OnRemoved(selectedIndex);
            }

            if ((OnRemoveCompleted != null) && doRemoval) {
                OnRemoveCompleted(removeRangeStart, removeRangeCount);
            }
        }
    }
}
```