Dreamteck Splines – API Reference

http://dreamteck-hq.com

https://www.facebook.com/dtkhq/

Contact the team: team@dreamteck-hq.com

Contact support: support@dreamteck-hq.com

Contents

SplinePoint

Struct

Description

Representation of a control point. SplinePoint is used to define Splines. Editing the points of a Spline edits the spline too.

Public Properties

SplinePoint.Type type	The type of the point
Vector3 position	The position of the point
Color color	The color of the point
Vector3 normal	The normal direction of the point
float size	The size of the point
Vector3 tangent	The first tangent position for Bezier splines
Vector3 tangent2	The second tangent position for Bezier splines

Public Methods

SetPosition(Vector3 pos)	Sets the position of the point and moves its tangents too
SetTangentPosition(Vector3 pos)	Sets the tangent position of the point
SetTangent2Position(Vector3 pos)	Sets the second tangent's position of the point

Static Methods

SplinePoint Lerp(SplinePoint a, SplinePoint b, float t)	Interpolation between two spline points

Enumerations

Type {Smooth, Broken}	Smooth mirrors the tangents, Broken make the tangents independent

Constructors

SplinePoint(Vector3 p)	Creates a new smooth point
SplinePoint(Vector3 p, Vector3 t)	Creates a new smooth point
SplinePoint(Vector3 pos, Vector3 tan, Vector3 nor, float s, Color col)	Creates a new fully defined smooth point
SplinePoint(Vector3 pos, Vector3 tan, Vector3 tan2, Vector3 nor, float s, Color col)	Creates a new fully defined broken point

SplineResult

Class

Description

When a spline is evaluated, multiple values are returned. This is the result of spline evaluation.

Public Properties

Vector3 position	The position of the evaluation result
Vector3 normal	The normal of the evaluation result
Vector3 direction	The direction of the evaluation result
Color color	The color of the evaluation result
float size	The size of the evaluation result
double percent	The time (0-1) the spline was evaluated at

Read-only Properties

Quaternion rotation	Returns Quaternion.LookRotation(direction, normal)
Vector3 right	Returns a perpendicular vector to direction and normal

Public Methods

Lerp(SplineResult b, double t)	Interpolates between the current values and b's values
Lerp(SplineResult b, float t)	Interpolates between the current values and b's values

Static Methods

SplineResult Lerp(SplineResult a, SplineResult b, double t)	Interpolates between two spline results
SplineResult Lerp(SplineResult a, SplineResult b, float t)	Interpolates between two spline results

Constructors

SplineResult()	Creates a spline result with default values
SplineResult(Vector3 p, Vector3 n, Vector3 d, Color c, float s, double t)	Creates a fully-defined spline result
SplineResult(SplineResult input)	Creates a deep copy of input

Spline

Class

Description

A spline in world space. This class stores a single spline and provides methods for evaluation, length calculation, raycasting and more.

Example

```
using UnityEngine;
using System.Collections;
using Dreamteck.Splines; //Include the Splines namespace

public class SimpleSplineController : MonoBehaviour {
       void Start () {
        //Create a new B-spline with precision 0.9
        Spline spline = new Spline(Spline.Type.BSpline, 0.9);
        //Create 3 control points for the spline
        spline.points = new SplinePoint[3];
        spline.points[0] = new SplinePoint(Vector3.left);
        spline.points[1] = new SplinePoint(Vector3.up);
        spline.points[2] = new SplinePoint(Vector3.right);
        //Evaluate the spline and get an array of values
        SplineResult[] results = new SplineResult[spline.iterations];
        spline.Evaluate(results);
        //Display the values in the editor
        for (int i = 0; i < results.Length; i++)
        {
            Debug.DrawRay(results[i].position, results[i].normal, results[i].color);
        }
    }
}
```

Public Properties

SplinePoint[] points	The control points of the spline
Spline.Type type	The type of the spline which defines what interpolation should be used.
double precision	The approximation rate (0-0.9999) of the spline
AnimationCurve customValueInterpolation	Custom curve for size and color interpolation between points

AnimationCurve customNormalInterpolation	Custom curve for normal interpolation between points

Read-only Properties

bool isClosed	Whether or not the spline is closed
double moveStep	The step size of the percent incrementation when evaluating a spline (based on percision)
int iterations	The total count of samples for the spline (based on the precision)

Public Methods

SplineResult Evaluate(double percent)	Evaluates the spline at the given time and returns a **SplineResult** object
void Evaluate(ref SplineResult[] samples, double from = 0.0, double to = 1.0)	Evaluates the spline using its precision and writes the results to the array
Vector3 EvaluatePosition(double percent)	Evaluates the spline and returns the position. This is simpler and faster than Evaluate.
void EvaluatePositions(Vector3[] positions, double from = 0.0, double to = 1.0)	Evaluates the spline using its precision and writes the result positions to the array.
public float CalculateLength(double from = 0.0, double to = 1.0, double resolution = 1.0)	Calculates the length of the spline
double Project(Vector3 point, int subdivide = 4, double from = 0.0, double to = 1.0)	Projects a point on the spline. Returns evaluation percent.
bool Raycast(out RaycastHit hit, out double hitPercent, LayerMask layerMask, double resolution = 1.0, double from = 0.0, double to = 1.0, QueryTriggerInteraction hitTriggers = QueryTriggerInteraction.UseGlobal)	Casts rays along the spline against all colliders in the scene
bool RaycastAll(out RaycastHit[] hits, out double[] hitPercents, LayerMask layerMask, double resolution = 1.0, double from = 0.0, double to = 1.0, QueryTriggerInteraction hitTriggers = QueryTriggerInteraction.UseGlobal)	Casts rays along the spline against all colliders in the scene and returns all hits. Order is not guaranteed.

double Travel(double start, float distance, Direction direction)	Returns the percent from the spline at a given distance from the start point
void Close()	Closes the spline (requires the spline to have at least 4 points)
void Break()	Breaks the closed spline
void Break(int at)	Breaks the closed spline at a given point

Constructors

Spline(Type t)	Creates a spline of a given type

SplineComputer : MonoBehaviour

Class

Description

The SplineComputer is attached to a Game Object in the scene and serves as a thread-safe MonoBehaviour wrapper for the Spline class. This is a component that holds a single Spline object, has all the public methods and properties the Spline class has and applies transformation to the spline according to its Game Object's Transform component.

Example

```
        //Get the SplineComputer component
        SplineComputer computer = GetComponent<SplineComputer>();
        //Make sure it's set to local space
        computer.space = SplineComputer.Space.Local;
        //Get the computer's control poitns
        SplinePoint[] points = computer.GetPoints();
        //if no contorl points are found - stop - there is nothing to do
        if (points.Length == 0) return;
        //Edit the first and the last point's positions
        points[0].SetPosition(points[0].position + Vector3.up);
        points[points.Length - 1].SetPosition(points[points.Length - 1].position +
Vector3.down);
        //Set the new points for the computer
        computer.SetPoints(points);
        //Transform the computer a little
        computer.transform.localScale *= 1.5f;
        computer.transform.Rotate(Vector3.one * 45f);
        //Get the evaluated results which will also be transformed
        SplineResult[] results = new SplineResult[computer.iterations];
        computer.Evaluate(ref results);
        //Display the values in the editor
        for (int i = 0; i < results.Length; i++)
        {
            Debug.DrawRay(results[i].position, results[i].normal, results[i].color);
        }
```

Public Properties

SplineComputer.Space space	The space in which the spline is evaluated.
Spline.Type type	The type of the spline which defines what interpolation should be used.

double precision	The approximation rate (0-0.9999) of the spline
AnimationCurve customValueInterpolation	Custom curve for size and color interpolation between points
AnimationCurve customNormalInterpolation	Custom curve for normal interpolation between points

Read-only Properties

bool isClosed	Whether or not the spline is closed
double moveStep	The step size of the percent incrementation when evaluating a spline (based on percision)
int iterations	The total count of samples for the spline (based on the precision)
int pointCount	The number of control points the spline is defined with
NodeLinks[] nodeLinks	The node links of the SplineComputer (used for connecting to Nodes)
SplineComputer.SplineMorph morph	The morph module of the SplineComputer
bool hasMorph	Does the SplineComputer have at least one morph channel?
Vector3 position	The position of the SplineComputer’s Transform (thread safe)
Quaternion rotation	The rotation of the SplineComputer’s Transform (thread safe)
Vector3 scale	The localScale of the SplineComputer’s Transform (thread safe)
int subscriberCount	The number of SplineUser that are subscribed to this SplineComputer

Public Methods

SplineResult Evaluate(double percent)	Evaluates the spline at the given time and returns a **SplineResult** object
void Evaluate(ref SplineResult[] samples, double from = 0.0, double to = 1.0)	Evaluates the spline using its precision and writes the results to the array
Vector3 EvaluatePosition(double percent)	Evaluates the spline and returns the position. This is simpler and faster than Evaluate.
void EvaluatePositions(Vector3[] positions, double from = 0.0, double to = 1.0)	Evaluates the spline using its precision and writes the result positions to the array.
public float CalculateLength(double from = 0.0, double to = 1.0, double resolution = 1.0)	Calculates the length of the spline
double Project(Vector3 point, int subdivide = 4, double from = 0.0, double to = 1.0)	Projects a point on the spline. Returns evaluation percent.
bool Raycast(out RaycastHit hit, out double hitPercent, LayerMask layerMask, double resolution = 1.0, double from = 0.0, double to = 1.0, QueryTriggerInteraction hitTriggers = QueryTriggerInteraction.UseGlobal)	Casts rays along the spline against all colliders in the scene
bool RaycastAll(out RaycastHit[] hits, out double[] hitPercents, LayerMask layerMask, double resolution = 1.0, double from = 0.0, double to = 1.0, QueryTriggerInteraction hitTriggers = QueryTriggerInteraction.UseGlobal)	Casts rays along the spline against all colliders in the scene and returns all hits. Order is not guaranteed.
double Travel(double start, float distance, Direction direction)	Returns the percent from the spline at a given distance from the start point
void Close()	Closes the spline (requires the spline to have at least 4 points)
void Break()	Breaks the closed spline
void Break(int at)	Breaks the closed spline at a given point
void Subscribe(SplineUser input)	Subscribes a SplineUser to the SplineComputer
void Unsubscribe(SplineUser input)	Unsubscribes a SplineUser from the SplineComputer

bool IsSubscribed(SplineUser user)	Checks if a SplineUser is subscribed to the computer
void AddNodeLink(Node node, int pointIndex)	Add a link to a node. This is used by the Node class
void RemoveNodeLink(int pointIndex)	Removes a link to a node.
SplinePoint GetPoint(int index, SplineComputer.Space getSpace = SplineComputer.Space.World)	Returns a spline control point by its index (Transformed)
SplinePoint[] GetPoints(SplineComputer.Space getSpace = SplineComputer.Space.World)	Returns all spline control points. (Transformed)
void SetPoint(int index, SplinePoint point,, SplineComputer.Space setSpace = SplineComputer.Space.World)	Sets the value of a control point. (Transformed)
void SetPoints(SplinePoint[] points, SplineComputer.Space setSpace = SplineComputer.Space.World)	Sets the points of the spline. (Transformed)
void Rebuild()	Forces the SplineComputer to rebuild all subscribed users on the next update cycle
void RebuildImmediately()	Forces the SplineComputer to rebuild all subscribed users immediately
int[] GetAvailableNodeLinksAtPosition (double percent, Spline.Direction direction)	Returns an array of node link indices
void SetMorphState(int index)	Set the selected morph's weight to 1 and all others to 0
void SetMorphState(string morphName)	Set the selected morph's weight to 1 and all others to 0
void SetMorphState(int index, float percent)	Set the selected morph's weight to percent and reduce all other morph weights automatically
void SetMorph(string morphName, float percent)	Set the selected morph's weight to percent and reduce all other morph weights automatically
void SetMorphState(float percent)	Automatically assigns a weight value to all morph states based on the percent. 0 will set the first morph’s weight to 1 and all others to 0. 1 will set the last morph’s weight to 1 and all others to 0.

List<SplineComputer> GetConnectedComputers()	Returns a list of all connected computers using Nodes. The list includes the current computer too.

Enumerations

Space {World, Local}	The space that the SplineComputer uses to transform the spline.

SplineComputer.Morph

Class

Description

A morph module for the SplineComputer component which stores and blends different spline paths with the same amount of control points. It's used for shape animations during runtime.

Public Methods

void AddChannel(string name)	Creates a new morph channel which can be blended
void RemoveChannel(string name)	Removes a morph channel by name
int GetChannelCount()	Returns the channel count of the morph
string[] GetChannelNames()	Returns the channel names of the morph
float GetWeight(int index)	Gets the weight of a channel by index
float GetWeight(string name)	Gets the weight of a channel by name
void SetWeight(int index, float weight)	Sets the weight of a channel by index
void SetWeight(string name, float weight)	Sets the weight of a channel by name
void CaptureSnapshot(int index)	Saves the points of the spline into a channel (by index)
void CaptureSnapshot(string name)	Saves the points of the spline into a channel (by name)
SplinePoint[] GetSnapshot(int index)	Gets the points of the spline from a channel (by index)
SplinePoint[] GetSnapshot(string name)	Gets the points of the spline from a channel (by name)
void Clear()	Removes all morph channels

Node : MonoBehaviour

Class

Description

The Node class is used to bind the control points of SplineComputers to Game Objects in the scene and also to create junctions.

Public Properties

Node.Type type	Defines the way the node connects the points.
bool transformNormals	Should it transform the normals of the connected points?
bool transformSize	Should it transform the sized of the connected points?
bool transformTangents	Should it transform the tangents of the connected points?

Public Methods

void UpdateConnectedComputers(SplineComputer excludeComputer = null)	Forces the connected computers to rebuild their subscribed users. Can exclude one computer from update.
void UpdatePoint(SplineComputer computer, int pointIndex, SplinePoint point)	Updates the values of a connected point
void AddConnection(SplineComputer computer, int pointIndex)	Adds a new Spline Computer and point to the connections
void RemoveConnection(SplineComputer computer, int pointIndex)	Removes a Spline Computer and point from the connections
bool HasConnection(SplineComputer computer, int pointIndex)	Checks if a computer is connected at the given point
void ClearConnections()	Removes all connections from the node
Connection[] GetConnections()	Returns the connections the node has

Enumerations

Type {Smooth, Free}	Smooth makes the values of the connected points the same while free allows each point to retain its normal, size and color.

Node.Connection

Class

Description

A connection definition for the Node class. It stores a computer and the point index it's connected at.

Public Properties

SplineComputer computer	The computer that is connected
int pointIndex	The index of the point that the computer is connected at

SplineUser : MonoBehaviour

Class

Description

A base class that utilizes the SplineComputer component. It samples a single SplineComputer and provides useful sample information that can be used for path following, mesh generation, object positioning and everything else that imagination is capable of.

The SplineUser has a resolution multiplier [0-1] which can be used to reduce the sample rate for the SplineComputer as well as clip from and clip to values [0-1] which control what segment of the spline is sampled.

The SplineUser implements a multithreading framework which allows developers to easily make their code multithreaded.

Changing the public properties of a SplineUser causes it to automatically resample the Spline Computer the on the next update cycle.

Please refer to the User Manual for all available SplineUser-derived components.

Refer to BlankUser.cs in the Components folder for how to derive your own SplineUser classes.

All of the SplineUser's public properties update the user automatically when changed. For example, there is no need to call Subscribe if computer is set and there is no need to call Rebuild when resolution or clipFrom/clipTo are set.

Public Properties

UpdateMethod updateMethod	When should the user update?
SplineComputer computer	The computer that the SplineUser samples
SplineUser user	A SplineUser reference to use instead of a SplineComputer
double resolution	The resolution multiplier [0-1] to sample the SplineComputer with
double clipFrom	Where to start evaluating the spline from? [0-1]

double clipTo	Where to evaluate the spline to? [0-1]
bool averageResultVectors	Should normals and directions be averaged when sampling?
bool multithreaded = false;	Should the SplineUser use multithreading?
SplineAddress address	The junction address of the SplineUser. Used for taking different routes when there are junctions

Read-only Properties

double span	Always equal to clipTo - clipFrom

Protected Properties

SplineResult[] samples	The samples from the SplineComputer component
SplineResult[] clippedSamples	The clipped samples from the SplineComputer component defined by clipFrom and clipTo

Public Methods

void Rebuild(bool sampleComputer)	Forces a rebuild on the next update cycle. sampleComputer controls whether the user should re-evaluate the SplineComputer
void RebuildImmediate(bool sampleComputer)	Forces a rebuild immediately
void EnterAddress(Node node, int pointIndex, Spline.Direction direction = Spline.Direction.Forward)	Adds a new element to the junction address
void ExitAddress(int depth)	Removes the last (depth) elements from the junction address
void ClearAddress()	Clears the junction address
int GetSampleIndex(double percent)	Get a sample index from a percent
SplineResult Evaluate(double percent)	Evaluates the sampled spline

Vector3 EvaluatePosition(double percent)	Evaluates the sampled spline and returns the position. This is simpler and faster than Evaluate.
SplineResult Project(Vector3 point, double from = 0.0, double to = 1.0)	Projects a point on the sampled spline
double Travel(double start, float distance, Direction direction)	Returns the percent from the sampled spline at a given distance from the start point

Enumerations

enum UpdateMethod { Update, FixedUpdate, LateUpdate}	When should the SplineUser rebuild?

SplineFollower : SplineUser

Class

Description

A SplineUser class dedicated for following SplineComputers.

Public Properties

Wrap wrapMode	Defines what happens when the follower reaches the end of the spline
FollowMode followMode	The follow mode of the follower. It can either follow with uniform speed or follow based on a duration variable
bool autoFollow	If true, the SplineFollower will automatically move along the spline during runtime
double startPercent	The start position along the spline when autoFollow is checked
bool findStartPoint	If set to true and autoFollow is on, the follower will disregard the startPercent value and find the closest point on the spline
Spline.Direction direction	The follow direction of the follower. It can be either forward or backward.
float followSpeed	if followMode is set to FollowMode.Uniform, this will be the constant follow speed
float followDuration	if followMode is set to FollowMode.Time this will be the time in which the follower will reach the end of the spline
bool applyPosition	Whether or not to apply the position to the transform
bool applyRotation	Whether or not to orient the transform along the spline
bool applyDirectionRotation	If set to true and applyRotation is true, the orientation will point in the direction of movement

bool applyScale	Whether or not to apply the scale from the spline (if there are points with different sizes)
Vector3 baseScale	if applyScale is set to true, this will be the base scale used by the follower
Vector2 offset	Position offset along X and Y relative to the direction of the spline at the follow position
Vector3 rotationOffset	if applyRotation is true, this is offset the rotation of the transform

Read-only Properties

SplineResult followResult	The current spline result from the follow operation
SplineResult offsettedFollowResult	The current spline result from the follow operation with applied offset and rotationOffset

Public Methods

public void Move(double percent)	Move along the spline with percent
public void Move(float distance)	Move a certain distance along the spline
public void SetPercent(double distance)	Set the position of the follower along the spline immediately
public void SetDistance(float distance)	Set the follower position to a certain distance from the start of the spline
void AddTrigger(SplineTrigger.Type t, UnityAction call, double position = 0.0, SplineTrigger.Type type = SplineTrigger.Type.Double)	Add a trigger with a parameterless action
void AddTrigger(SplineTrigger.Type t, UnityAction<int> call, int value, double position = 0.0, SplineTrigger.Type type = SplineTrigger.Type.Double)	Add a trigger with an action with an integer parameter
void AddTrigger(SplineTrigger.Type t, UnityAction<float> call, float value, double position = 0.0, SplineTrigger.Type type = SplineTrigger.Type.Double)	Add a trigger with an action with an float parameter

void AddTrigger(SplineTrigger.Type t, UnityAction<double> call, double value, double position = 0.0, SplineTrigger.Type type = SplineTrigger.Type.Double)	Add a trigger with an action with an double parameter
void AddTrigger(SplineTrigger.Type t, UnityAction<string> call, string value, double position = 0.0, SplineTrigger.Type type = SplineTrigger.Type.Double)	Add a trigger with an action with an string parameter
void AddTrigger(SplineTrigger.Type t, UnityAction<bool> call, bool value, double position = 0.0, SplineTrigger.Type type = SplineTrigger.Type.Double)	Add a trigger with an action with an bool parameter
void AddTrigger(SplineTrigger.Type t, UnityAction<GameObject> call, GameObject value, double position = 0.0, SplineTrigger.Type type = SplineTrigger.Type.Double)	Add a trigger with an action with an GameObject parameter
void AddTrigger(SplineTrigger.Type t, UnityAction<Transform> call, Transform value, double position = 0.0, SplineTrigger.Type type = SplineTrigger.Type.Double)	Add a trigger with an action with an Transform parameter

Enumerations

enum FollowMode { Uniform, Time }	A follow mode enumeration defining whether the follower should use a uniform follow speed or use time for traversing
enum Wrap { Default, Loop, PingPong}	Wrap mode for the follower. Default stops when the follower reaches the end. Loop moves the follower to the opposite end of the spline when the end is reached and continues in the same direction. PingPong changes the direction when the end is reached.