Computer Graphics

Curves and Splines

Fernando Geraci

Subtopics

Linear Interpolation
Quadratic and Cubic Bezier Curves
Splines
Enforcing Control Points Constraints
Catmull-Rom Splines
Loops

Linear Interpolation

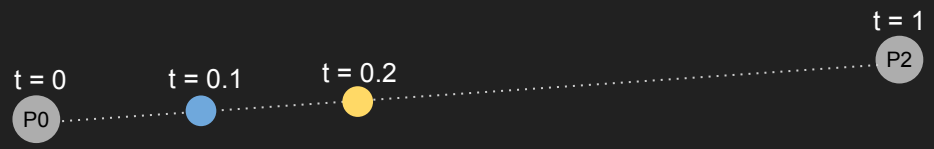
It is a parametric equation which, at any given value t, such that t >= 0 and t <= 1, there will correspond, geometrically speaking, to a value in between two points, P0 and Pn.

Said points are vectors in usually R2 or R3, hence the equation is a linear combination of said vectors multiplied by a scalar value t.

Formally:

$$Lerp(t) = \{ (1-t)\ P0 + t\ Pn \mid t >= 0\ \&\&\ t <= 1 \}$$

We usually will break t into “steps”, such that **t** → `for(i = 1; i <= steps; i++ ) { Lerp(i / steps); }`



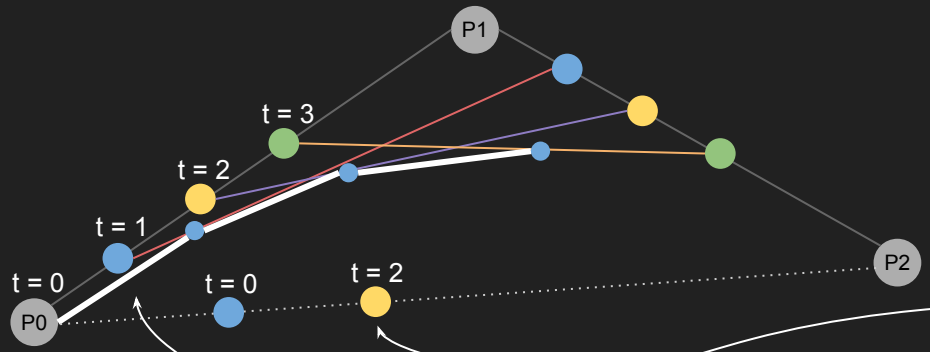
Granted:

Lerp(0) = (1) P0 + (0) Pn = P0

Lerp(1) = (0) P0 + (1) Pn = Pn

Fernando Geraci

Quadratic **Bezier Curve** using **Parametric Linear Interpolation**



Given a set of control points:

CP = { P0, … , Pn }

Any point of the curve can be calculated with a parameter *t* such that:

Linear:

LinearBezier(t) = (1 - t) P0 + t * P1

Quadratic:

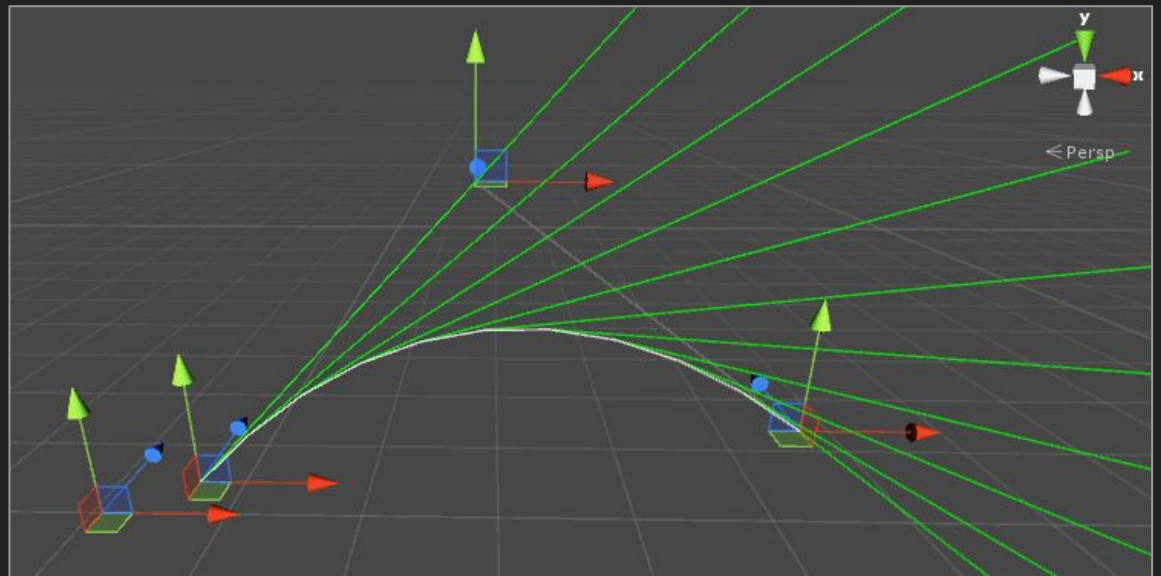
P’01 = (1-t) P0 + t P1
P’12 = (1-t) P1 + t P2
QuadBezier = (1 - t) P’01 + t P’02

Any point on this curve is given by: $B(t) = (1 - t)^2 P0 + 2t (1 - t) P1 + t^2 P2$

Rate of Change

By knowing the quadratic polynomial: $B(t) = (1 - t)^2 P0 + 2t (1 - t) P1 + t^2 P2$

We can get its first derivative: $B'(t) = 2(1 - t) (P1 - P0) + 2 t (P2 - P1)$



Things to consider:

- In the plane, the velocity (slope) is given by:

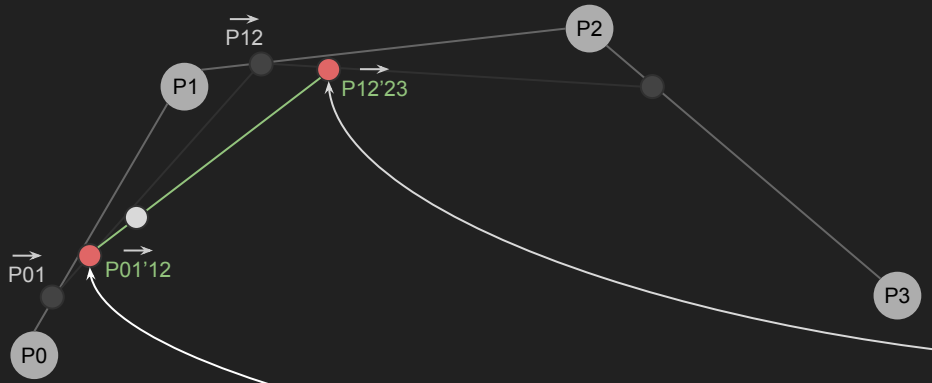
  B'(t) - transform.position

- Because of B'(t) 's nature, any line can be given by:

  Line (p0 , p0 + B'(t))

Cubic **Bezier Curves**

Following the same principle, and its **recursive nature**, the following illustrates a more complex interpolation of 4 points { P0 , … , P3 }



Recall:

Linear:

$(1 - t) P0 + (t) P1$

Quadratic:

$(1 - t) P01 + (t) P12$

$P01 = (1-t) P0 + (t) P1$

$P12 = (1-t) P1 + (t) P2$

Cubic:

$(1- t) P01'12 + (t) P12'23$

$(1 - t) P12 + (t) P23$

$P12 = (1 - t) P1 + (t) P2$

$P23 = (1 - t) P2 + (t) P3$

$(1 - t) P01 + (t) P12$

$P01 = (1-t) P0 + (t) P1$

$P12 = (1-t) P1 + (t) P2$

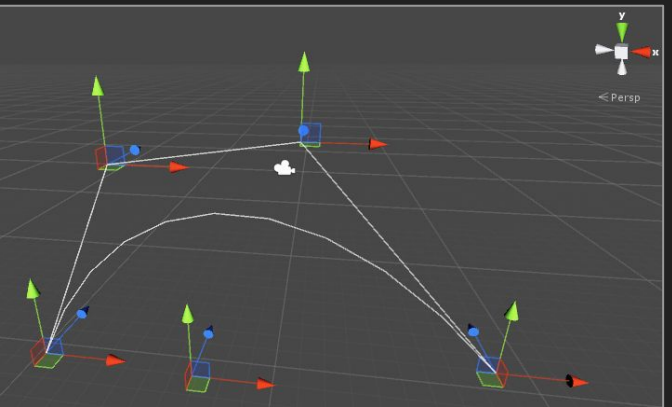
Cubic **Bezier Curves** - Cntd.

Summarizing, the equation follows the resulting binomial expansion:

$(s + t)^3 = s^3\ P0 + 3s^3t\ P1 + 3st^3\ P2 + t^3\ P3$

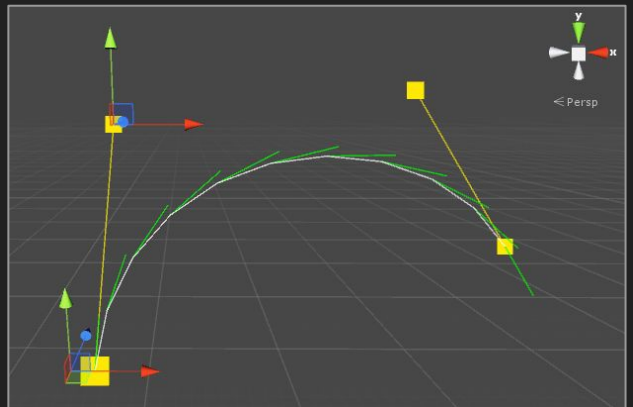
if we let $s = (1 - t)$ it follows:

$$B(t) = \{ (1 - t)^3\ P0 + 3\ (1 - t)^2\ t\ P1 + 3\ (1 - t)\ t^2\ P2 + t^3\ P3 \mid t >= 0 \text{ and } t <= 1 \}$$



For the curve on the left, we are only drawing control points $0 \rightarrow 1$ and $2 \rightarrow 3$.

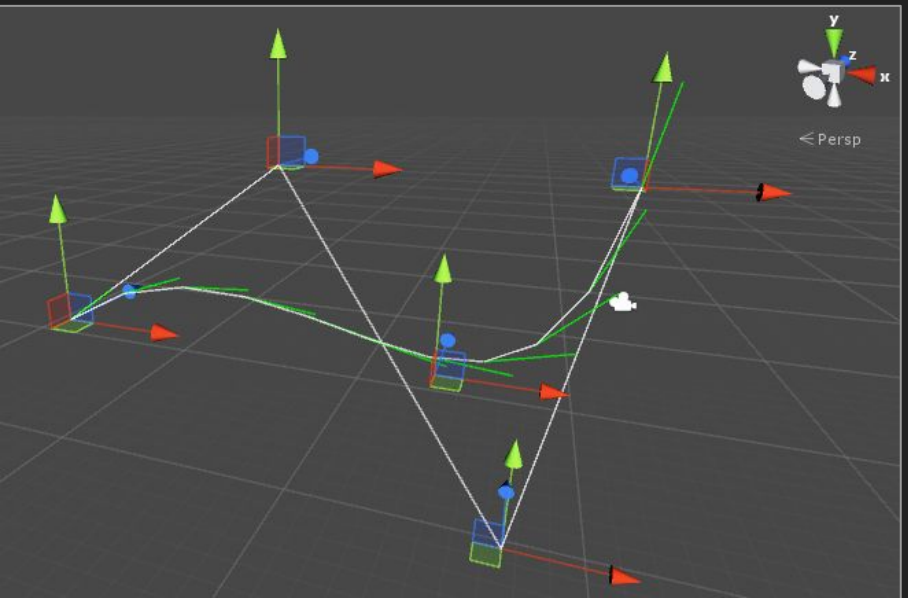
This will help us control the curve as we move into splines better.



Cubic **Bezier Curves** - Cntd.

The cubic equation first derivative is the following:

$$CubicB'(t) = 3\,(1 - t)^2\,(p1 - p0) + 6t\,(1 - t)\,(p2 - p1) + 3t^2\,(p3 - p2)$$

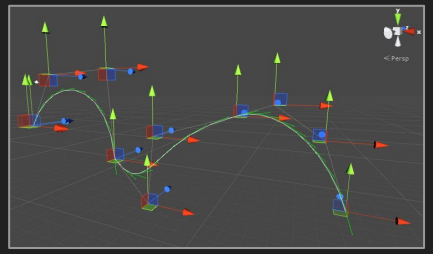


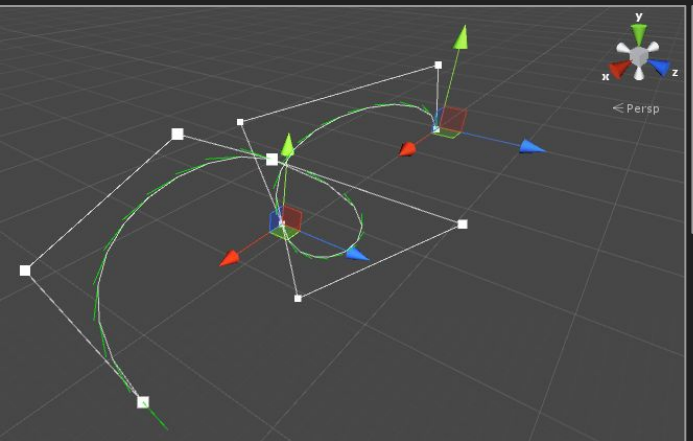
To avoid bothersome vectors, we could normalize the product of the equation by:

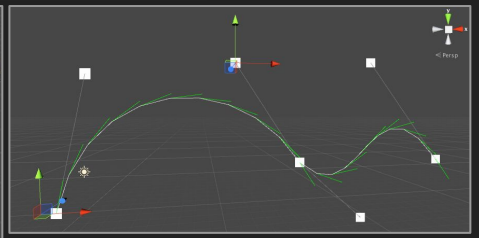
$$Norm(v) = \mathbf{v} / \|\mathbf{v}\|$$

Splines

The result of concatenating several curves is known as a **Spline**, also known as a “Path”







On the left, you can appreciate a spline composed of 3 cubic curves.

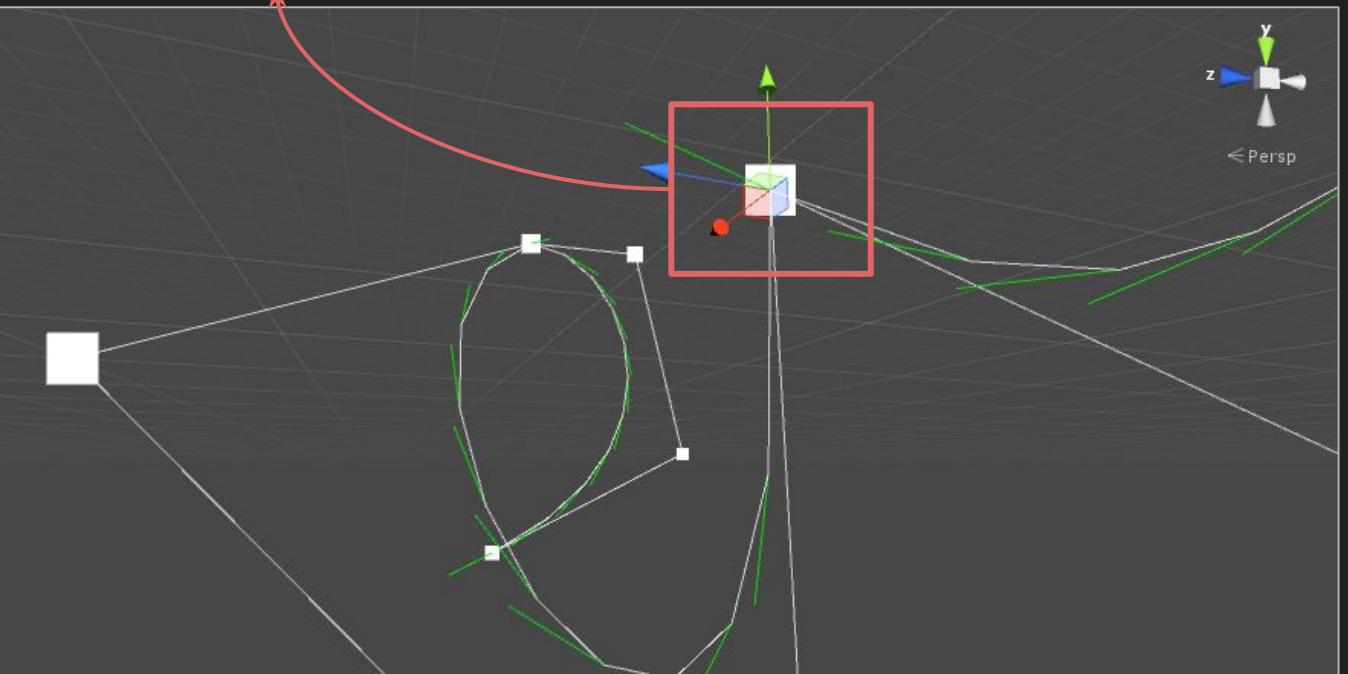
Note how the control points are being interpolated to create the curves as opposed as dictating the curves’ paths.

Any given Spline will have:

$$Spline_{control\ points} = Curves * 3 + 1$$

Splines

We still have a problem, the union point between concatenated curves, have different velocities, producing sharp vertices.



To solve this problem, and produce a soft curve Spline, the points enclosing said unions must mirror each other; these are the 3rd point from the curve before, and the 2nd point from the curve after.

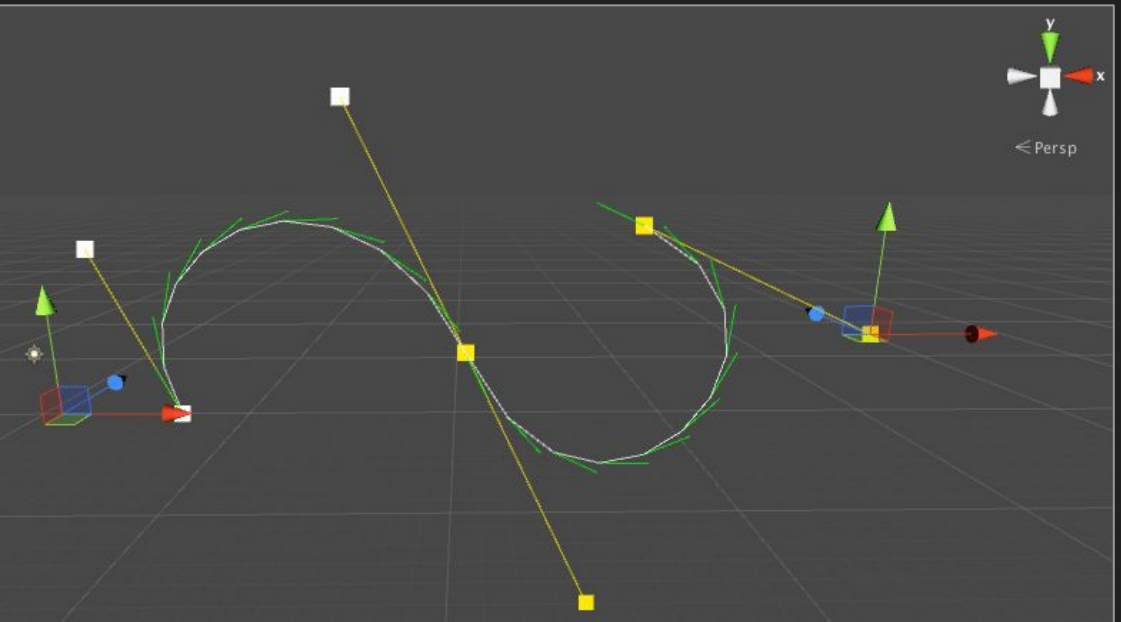
A “connector” control point can have 3 modes:

Free, Mirrored or Aligned

When we create a Curve, by default, we will set its first and last control point as FREE

Splines - Control Points Modes Grouping

Control points are grouped by mode, hence every two (or one in case of extremes) points surrounding a connector point of two curves, share the same mode.



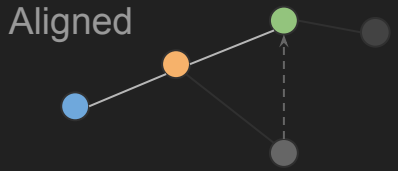
Keep in mind that when setting or getting a control point's mode, we access to it by:

(index + 1) / 3

This ensures we are always modifying the right point.

Note that "3" corresponds to cubic curves only.

Splines - Enforcing Constraints



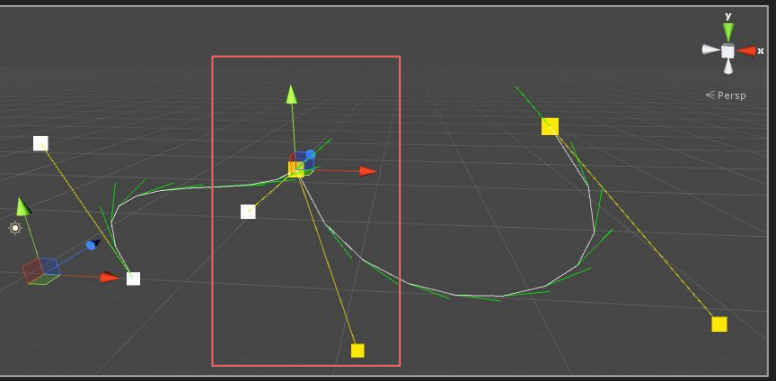
Given an index, we:

Find to which control point mode group belong to
    let current Mode ← (i + 1) / 3
Determine the index of its middle control point
    let middle ← current Mode * 3

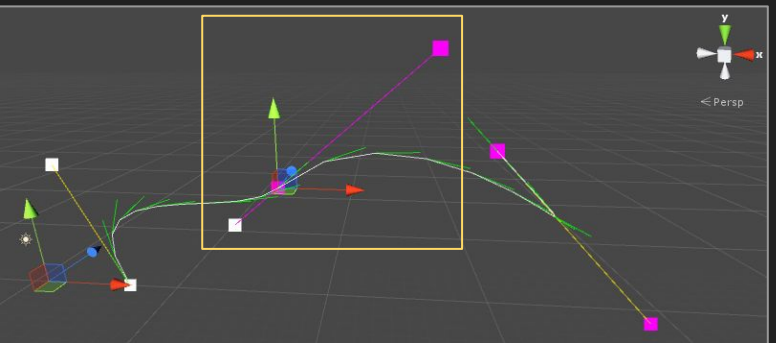
Determine which point will be fixed and which will get the enforced policy

If the selected index is the middle of the one before
    fix those and enforce the one after the middle
otherwise
    fix the selected one (after the middle) enforce the on the one before the middle, hence its opposite.

Splines - Aligned & Mirrored Constraint



Free → Aligned



For either case, we do the same with one difference:

1 - Get the direction vector from the middle to the fixed point

let direction ← middle - fixedPoint

1.a - If we are aligning these points, then we need to respect the distance from middle → toBeAdjusted

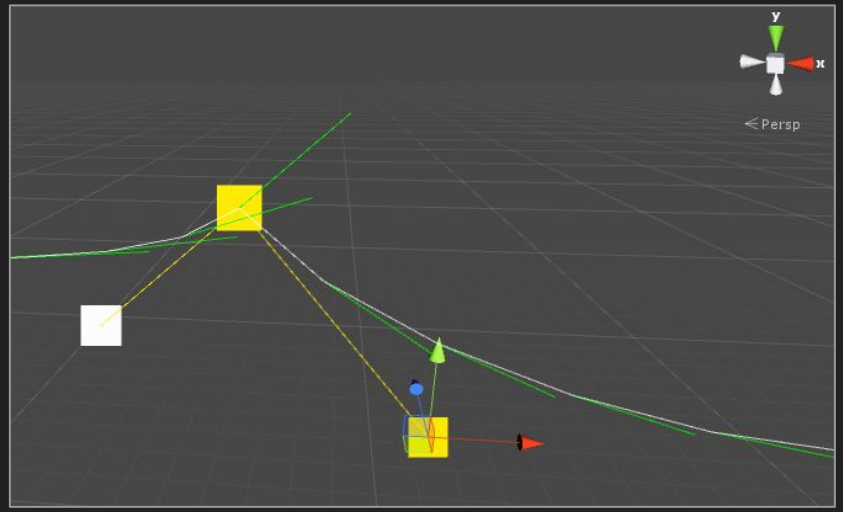
let direction← Normalized(direction)
* Distance(middle → origToBeAdjusted)

3 - Set the enforced point to be equal to the sum of the middle point and the obtained direction.

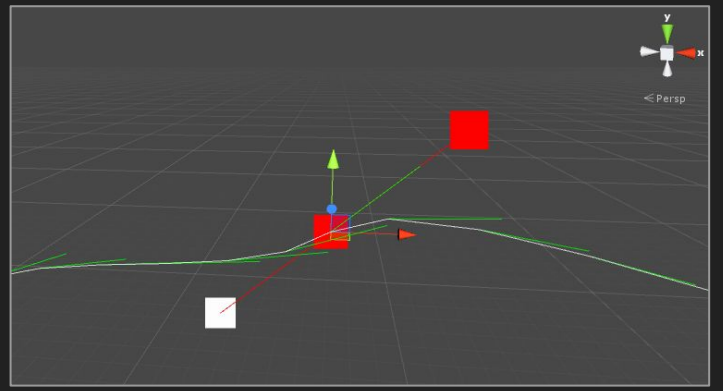
let toBeAdjusted ← middle + direction

Splines - Aligned & Mirrored Constraint

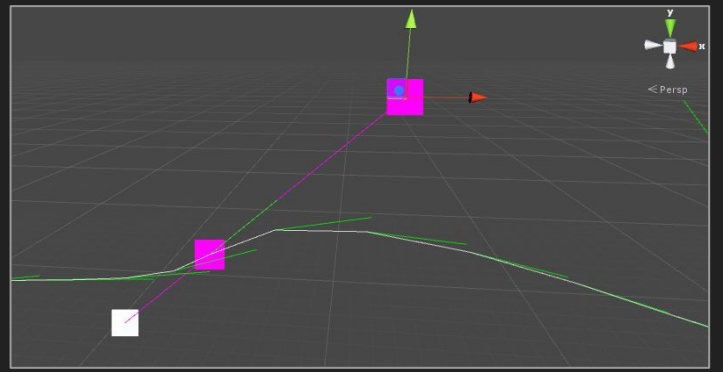
Free



Mirrored

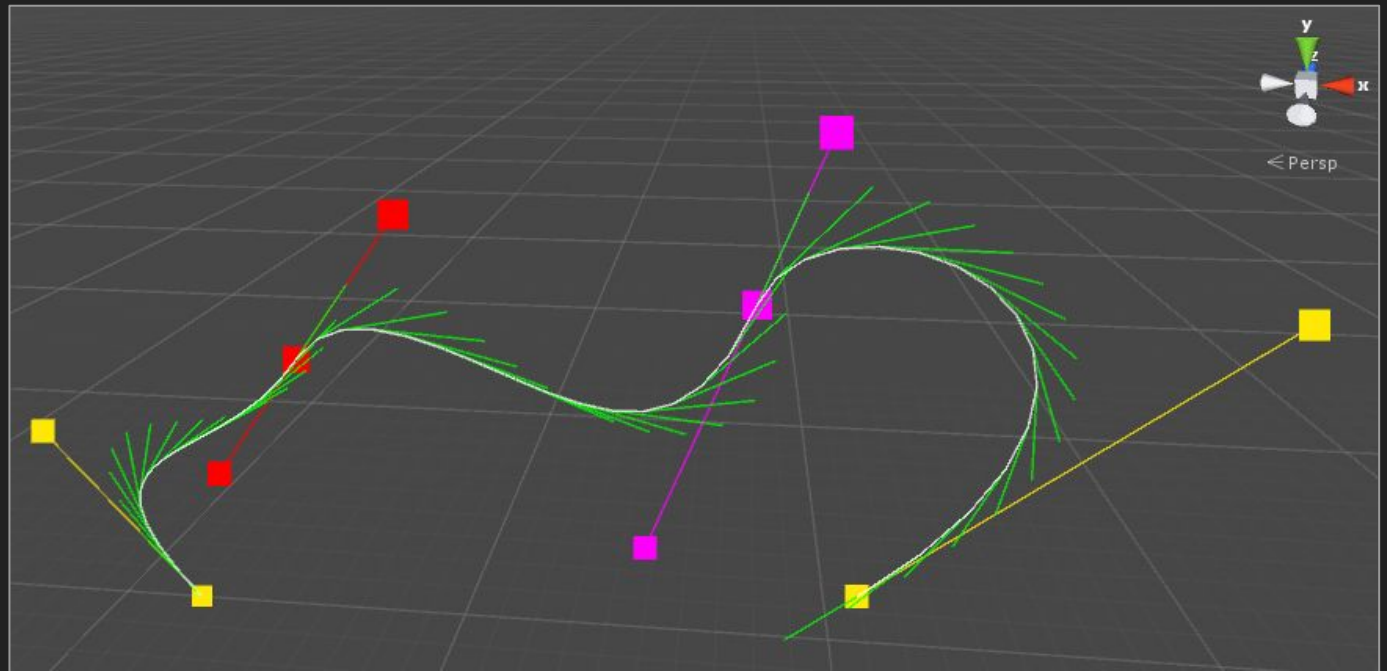


Aligned



Splines - Aligned & Mirrored Constraints

The result is a much smoother curve when the constraints are enforced. The first one from the left uses mirrored, while the the next one are aligned.



These constraints shall be enforced when:

A point is moved

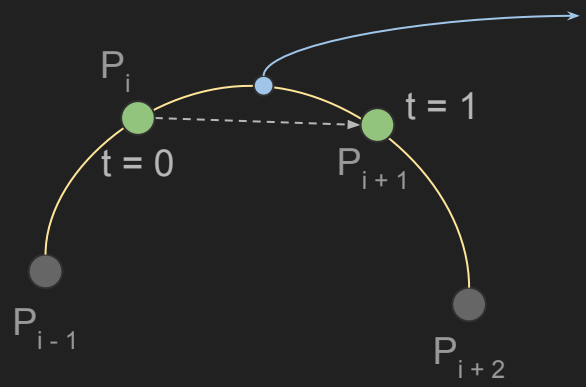
A mode is changed

A curve is added

Catmull-Rom Spline

We take cubic splines a step further to explain Catmull-Rom splines. These are, just as Bezier Splines, cubic interpolated curves, with the difference that the curve passes through each control point.

The tangent at each point P_n, is calculated using points P_{n-1} and P_{n+1}



Each point of the curve is given by two factors: t and the tangents described above.

Points are distanced equally and each tangent, is given by:

$$D_n = T\,(P_{i+1} - P_{i-1})\,,\; D_{n+1} = T\,(P_{i+2} - P_i)$$

Catmull-Rom Spline - Derivation

Let’s see what information we have:

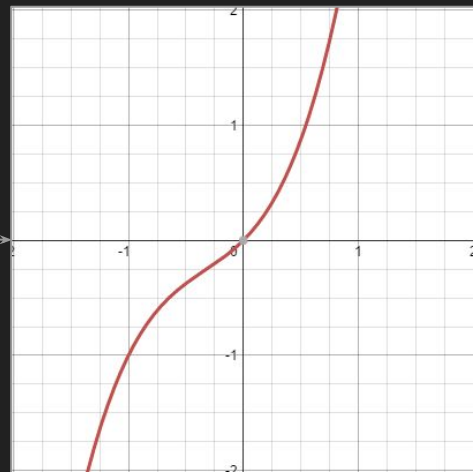
The curve is given by the cubic polynomial: $P(t) = a\,t^3 + b\,t^2 + c\,t + d$

We defined the following geometric constraints:

$P(0) = P_i$
$P(1) = P_{i+1}$
$P'(0) = T\,(P_{i+1} - P_{i-1})$
$P'(1) = T\,(P_{i+2} - P_{i-1})$

such that T is a value in R between [0,1]

Then, we obtain $P'(t) = 3at^2 + 2bt + c$



Catmull-Rom Spline - Derivation

Is we substitute the values we know this far:

$$P(0) = d \longrightarrow P(0) = P_i$$
$$P(1) = a + b + c + d \longrightarrow P(1) = P_{i+1}$$
$$P'(0) = c \longrightarrow P'(0) = T(P_{i+1} - P_{i-1})$$
$$P'(1) = 3a + 2b + c \longrightarrow P'(1) = T(P_{i+2} - P_i)$$

KEY! - note the identity

$$\begin{pmatrix} P(0) \\ P(1) \\ P'(0) \\ P'(1) \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -T & 0 & T & 0 \\ 0 & -T & 0 & T \end{pmatrix} \begin{pmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{pmatrix}$$

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \overbrace{\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}^{-1}}^{M} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -T & 0 & T & 0 \\ 0 & -T & 0 & T \end{pmatrix}$$

Catmull-Rom Spline - Derivation

Now… after we find M^{-1} ...

$$M^{-1}\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -T & 0 & T & 0 \\ 0 & -T & 0 & T \end{pmatrix} = \begin{pmatrix} -T & 2-T & T-2 & T \\ 2-T & T-3 & 3-2T & -T \\ -T & 0 & T & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

let T = 1/2 tension →

$$\begin{pmatrix} -\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ \frac{2}{2} & -\frac{5}{2} & \frac{4}{2} & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Now we can express our cubic curve:

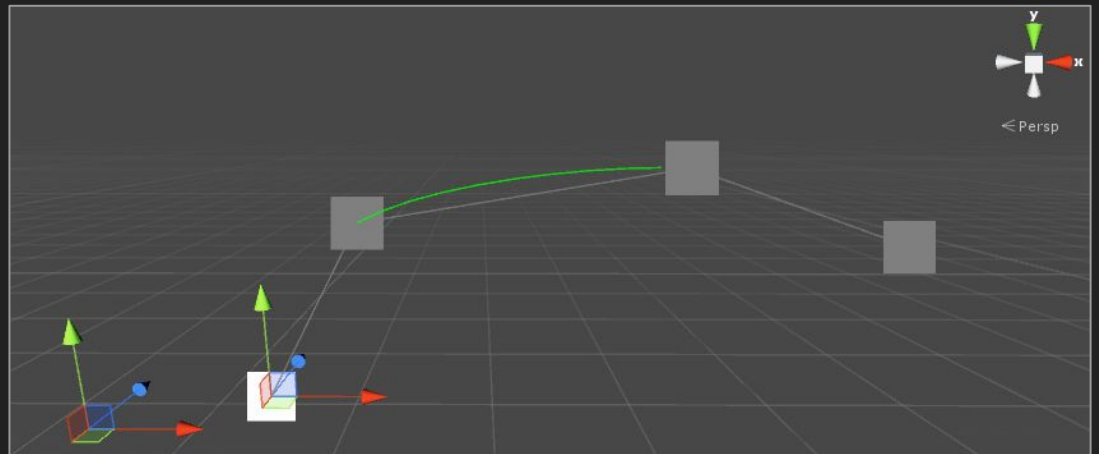
$$\frac{1}{2}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$CR(t) = \begin{pmatrix} t & t^2 & t^3 & 1 \end{pmatrix}\frac{1}{2}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{pmatrix}$$

Catmull-Rom Spline - Result

The result is a smooth curve which will derived, as we shown, from the tangents created from points (in this example) of index 0 and 3.

We will continue to close the gap between all the points.



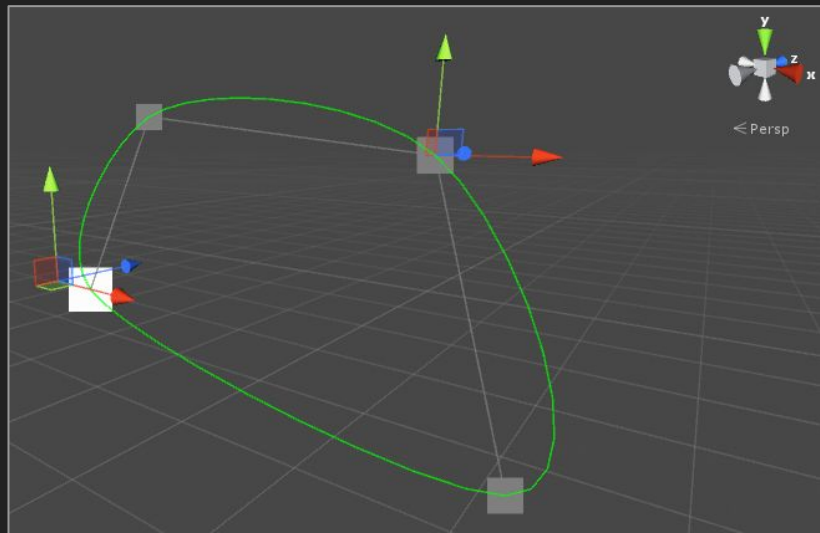
Catmull-Rom Spline - Using all points

Note how we can only draw our curve between points P_i and P_{i+1}

Still, we could simply loop through out all the points by clamping the lower boundary.

```
if index == 0
    iMinusOne ← controlPoints - 1;
iPlusOne ← (i + 1) % controlPoints;
iPlusTwo ← (i + 2) % controlPoints;
```

This way we always pick the right points despite of how many there are.



Loops - Cubic Bezier

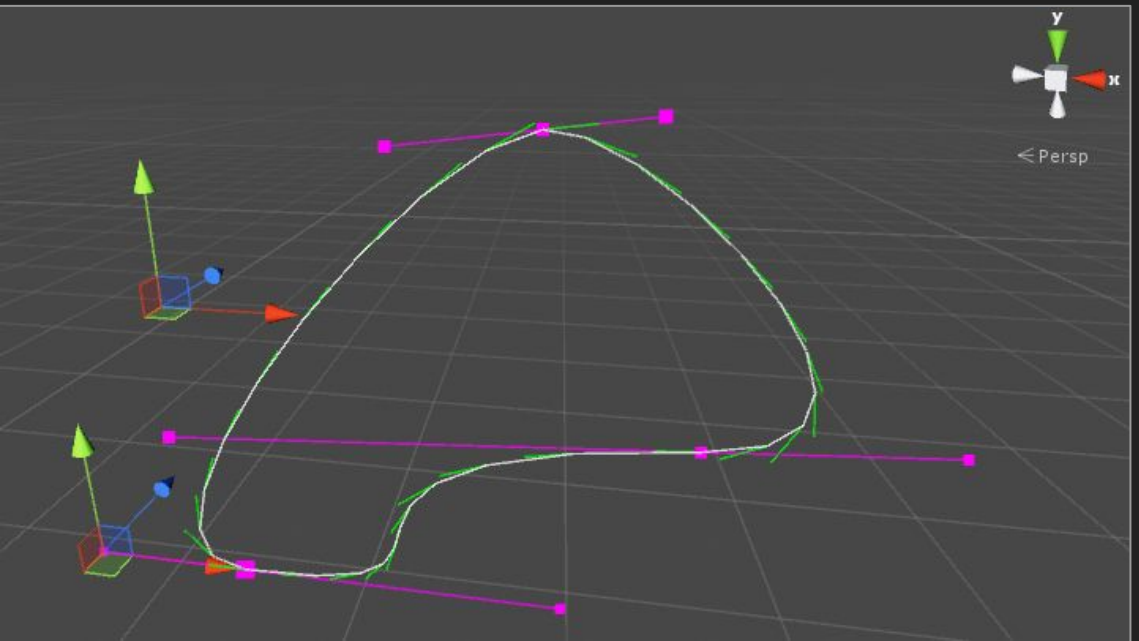
If we force the first and last control points of a curve to share the same position, what would we get? a loop!

To properly create a loop we must:

1) Ensure both ends share the same constraint type
    a) If loop already enabled, we need to ensure this remains true if constraints are changed on either of each end point of the spline
2) Enforce the constraint of the control point 0 when set
3) If loop selected, we need to make sure we take into account the delta (newPoint - actualPosition) and add it to the surrounding points to maintain the loop shape (remember to consider extreme cases)
4) When enforcing points, we need to ROLLOVER targets. If the middle point is either the first or the last, we need to select the ADJUST and FIXED accordingly.
5) Finally, when adding a curve, we need to match end-points and enforce policy

Loops - Cubic Bezier

The end product is a nicely closed loop which might save us some detailing work should we want to achieve this result.



Note how, differently from a Catmull Rom curve, we are enforcing constraints on the curves' control point.

In Catmull Rom curves, loops are determined by the tangents between each pair of points, while in cubic beziers, the curve is product of the interpolation of points.